# Extensible Resource Identifier (XRI) Syntax and Resolution Specification

## Working Draft 07, 29 July 2003

**Document identifier:**
wd-xri-specification-07

**Location:**
http://www.oasis-open.org/committees/xri

**Editors:**
Gabe Wachob, Visa International <gwachob@visa.com>
Drummond Reed, OneName <drummond.reed@onename.com>
Dave McAlpin, Epok <dave.mcalpin@epokinc.com>
Mike Lindelsee, Visa International <mlindels@visa.com>
Peter Davis, Neustar <peter.davis@neustar.biz>
Nat Sakimura, NRI <n-sakimura@nri.co.jp>

**Abstract:**
This document is the normative technical specification for XRI syntax and resolution. For an introduction to the uses and features of XRIs, see the non-normative *XRI Primer*.

**Status:**
This document is a working draft updated periodically on no particular schedule. Send comments to the editors.

Committee members should send comments on this specification to the xri@lists.oasis-open.org list. Others should subscribe to and send comments to the xri-comment@lists.oasis-open.org list. To subscribe, send an email message to xri-comment-request@lists.oasis-open.org with the word "subscribe" as the body of the message.

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 XRI TC web page (http://www.oasis-open.org/committees/xri/).

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

# Table of Contents

# Introduction

## 1.1 Overview of XRIs

An Extensible Resource Identifier (XRI) provides a standard means of abstractly identifying a resource independent of any given concrete representation of that resource (or, in the case of a completely abstract resource, independent of any representation at all). XRIs are defined similarly to URIs in *"Uniform Resource Identifiers (URI): Generic Syntax"* **[RFC2396]** but contain additional syntactical elements and extend the unreserved character set to include characters beyond those allowed in generic URIs. To accommodate applications that expect generic URIs, rules are defined that allow an XRI to be transformed into a conformant URI as defined by **[RFC2396]**. Since a revision of RFC 2396 is currently a work in progress, the XRI scheme also incorporates some simplifications and enhancements to generic URI syntax as proposed in **[RFC2396bis]**.

In addition, XRI syntax is internationalized following the recommendations in *"Guidelines for New URL Schemes*" **[RFC2718]** and " *Extensible Markup Language (XML) 1.0 (Second Edition)"* **[XML]**, and specifically the requirements of the "anyURI" datatype as specified in "XML Schema Part 2: Datatypes" **[XMLSchema2]**. To do this, the XRI scheme incorporates the syntax recommended in another work-in-progress, "*Internationalized Resource Identifiers (IRIs)*" **[IRI]**.

Although an XRI is not a Uniform Resource Name (URN) as defined in *"URN Syntax"* **[RFC2141]**, fully persistent XRIs are also designed to meet the requirements set out in *"Functional Requirements for Uniform Resource Names"* **[RFC1737]**.

This document specifies the ABNF that defines the XRI scheme. A valid XRI MUST conform to the ABNF specified in this document. In addition this document specifies a resolution framework for XRIs. An XRI MAY be resolved using one or more of mechanisms specified by this framework.

### 1.1.1 Generic Syntax

URI syntax is designed to be simple and extensible, and XRI syntax is very similar. A fully-qualified XRI consists of the scheme name "xri:" followed by the same four optional components as a generic URI.

```
   xri:  authority   / path   ? query   # fragment
```

One advantage of this approach is that the vast majority of HTTP URIs, which inherit directly from generic URI syntax, can be transformed to valid XRIs simply by changing the scheme from “http” to "xri". The relationship of HTTP URIs and XRIs and rules for this transformation are further discussed in [ref to Appendix C, "Transforming HTTP URIs to XRIs"].

XRI syntax extends this generic URI syntax in six ways by providing syntactic support for:

1. *Persistent and reassignable segments.* Generic URI syntax does not distinguish between persistent and reassignable identifiers. XRI syntax enables the top-level authority segment as well as any subsequent path segment to be expressed as either persistent or reassignable.
2. *Unlimited delegation.* Generic URI syntax supports delegated identifiers (i.e., DNS names or IP addresses) within the top-level authority segment. XRI syntax supports delegation of both persistent and reassignable identifiers at any level of the path.
3. *Cross-references.* Generic URI syntax does not provide for nesting of URIs in order to share identifiers across contexts. Since this is particularly useful with abstract identifiers (e.g., to establish the generic type of a resource, or to share identifier metadata such as versioning), XRI syntax allows URIs (including XRIs) to be nested inside parentheses.

4. *Internationalized character set.* Generic URI syntax limits legal characters to a subset of the repertoire of US-ASCII characters. XRI syntax allows the much wider repertoire of Unicode characters, greatly facilitating the use of XRIs in languages other than English.
5. *Global context symbols.* In addition to generic URI syntax for DNS and IP authorities, XRI syntax provides shorthand symbols for establishing the global context of an identifier.
6. *Non-resolvability.* Generic URI syntax does not provide a way to indicate whether or not a URI is resolvable. Since an XRI may itself be the full representation of a non-resolvable, abstract resource (e.g., a concept like "love", "honor", or "user-friendly") that is used only for the purposes of establishing equivalence, XRI syntax permits an XRI value to be expressed as explicitly non-resolvable.

### 1.1.2 Examples

The following examples illustrate XRI syntax. They have minimal annotation and are only intended to give a sense of the scope of XRI syntax. For details and the normative syntax, see Section 2.

```
xri://www.example.com/pages/index.html
       --standard HTTP URI used as an XRI

xri://[2010:836B:4179::836B:4179]/pages/index.html
       --using an IPv6 authority per RFC 2732

xri://www.example.com/inventory.parts/widget.subwidget.foobarator
       --delegation of reassignable identifiers

xri://www.example.com/:inventory:parts/:12:7:234
       --delegation of persistent identifiers

xri:@ExampleCorp
xri:@ExampleCorp.website
xri:=JohnDoe
xri:=JohnDoe.home
xri:=JohnDoe.work
xri:+flowers
xri:+flowers.rose
xri:+flowers.daisy
       --global context symbols

xri://www.example.com/(+management)/(+CEO)
xri:(urn:oasis:spec:2040)/(+tableofcontents)
xri:(mailto:john.doe@example.com)/(+email.address)
xri:=JohnDoe.home/(+email.address)
xri:=JohnDoe.home/(+email.address).($v/3)
       --cross-references

xri:(+flowers.rose)
xri:(//www.example.com/dictionary/flowers/rose)
       --non-resolvable XRIs
```

### 1.1.3 URI, URL, URN, and XRI

The evolution and interrelationships of the terms "URI", "URL", and "URN" are explained in a report from the Joint W3C/IETF URI Planning Interest Group, *"Uniform Resource Identifiers (URIs), URLs, and Uniform Resource Names (URNs): Clarifications and Recommendations"* **[RFC3305]**. This report states in Section 2.1:

> "During the early years of discussion of web identifiers (early to mid 90s), people assumed that an identifier type would be cast into one of two (or possibly more) classes. An identifier might specify the location of a resource (a URL) or its name (a URN), independent of location. Thus a URI was either a URL or a URN."

This view has since changed, as the report goes on to state in Section 2.2:

> "Over time, the importance of this additional level of hierarchy seemed to lessen; the view became that an individual scheme did not need to be cast into one of a discrete set of URI types, such as "URL", "URN", "URC", etc. Web-identifier schemes are, in general, URI schemes, as a given URI scheme may define subspaces."

This conclusion is shared by **[RFC2396bis]**, which states in Section 1.1.3:

> "An individual [URI] scheme does not need to be classified as being just one of "name" or "locator". Instances of URIs from any given scheme may have the characteristics of names or locators or both, often depending on the persistence and care in the assignment of identifiers by the naming authority, rather than any quality of the scheme."

The XRI scheme expressly embraces this precept. As an abstract URI, an XRI is explicitly intended to be used as a persistent identifier or long-term "name" for a resource. However XRIs are also resolvable and can be used a method of locating a resource (including another XRI). Since in certain contexts it may be important to distinguish whether an XRI is intended to be resolved vs. being used only for identification, the XRI scheme includes syntax for expressing this difference. See [ref to "Cross-references"].

## 1.2 Design Considerations

The full set of requirements for XRI syntax and resolution is documented in *"XRI Requirements and Glossary v1.0"* **[XRIReqs]**. A synopsis of the major design considerations is included here.

### 1.2.1 Abstraction and Independence

The preeminent requirement is that XRI syntax be fully abstract, i.e., independent of resource location, network, application, transport protocol, type, or security method. Although XRI syntax may be extended for specific uses, the generic XRI syntax is designed to represent pure UML-describable associations between resources (see **[UML]**) and thus to allow portability across all networks, directories, domains, and applications.

### 1.2.2 Persistence and Reassignability

As noted in Section 1.1.3 above, XRI syntax and resolution is designed to express and resolve fully persistent identifiers, fully reassignable identifiers, or any combination of persistent and reassignable identifier segments.

### 1.2.3 Human-friendliness and Machine-friendliness

XRI syntax and resolution is designed to support both human-friendly identifiers (HFIs—those optimized for human readability, memorability, and usability) and machine-friendly identifiers (MFIs—those optimized for machine processing and network efficiency). XRI syntax allows any combination of HFI and MFI components within a single XRI.

### 1.2.4 Internationalization

XRIs are designed to be rendered in the natural language of their intended consumer. They allow the Unicode range of characters **[Unicode]** and provide syntactical support for expressing optional language-dependent context metadata. As a result, XRIs extend the virtues of human readability, memorability, and usability to non-English speaking audiences.

### 1.2.5 Cross-Context Identification

XRI syntax and resolution is designed to allow the use of an absolute identifier in the context of another absolute identifier, i.e., for a URI (including an XRI) to be contained within another XRI. Such embedded identifiers are called *cross-references*, and they are key to XRI extensibility.

### 1.2.6 Authority, Delegation, and Federation

XRI syntax and resolution are designed to allow any resource to serve as a root authority, and for any authority to delegate to any other authority at any level of the path. Thus XRI design imposes no specific delegation model, network topology, or federation structure.

### 1.2.7 Security and Privacy

XRI syntax and resolution is designed to be adapted to any security model, method, or infrastructure, as well at to any privacy policy or framework. XRI design does not require sensitive data to be included in an identifier, and if such data is needed in an XRI, the syntax permits encryption and obfuscation of identifier segments for enhanced security and privacy.

### 1.2.8 Extensibility

Like XML, the XRI scheme is designed to be extended and specialized by different identifier authorities, and also like XML, these extensions and specializations are designed to be interoperable.

## 1.3 Terminology and Notation

### 1.3.1 Keywords

The key words “MUST“,“MUST NOT“,“REQUIRED“,“SHALL“,“SHALL NOT“,“SHOULD“,“SHOULD 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.

### 1.3.2 Syntax Notation

This specification uses the same syntax notation as **[RFC2396]**, namely, Augmented Backus-Naur Form (ABNF) as defined in **[RFC2234]**. As explained in RFC 2396, although the ABNF defines syntax in terms of the US-ASCII character encoding, XRI syntax should be interpreted in terms of the character that the ASCII-encoded octet represents, rather than the octet encoding itself. Like other URIs, how an XRI is represented in terms of bits and bytes on the wire is dependent upon the character encoding of the protocol used to transport it, or the charset of the document that contains it.

The following core ABNF productions are used by this specification as defined by Section 6.1 of **[RFC2234]**: ALPHA, CR, CTL, DIGIT, DQUOTE, HEXDIG, LF, OCTET, and SP. The complete XRI ABNF syntax is collected in Appendix A.

To simplify comparison between generic XRI syntax and generic URI syntax, the ABNF productions that are new to XRIs are shown with light green shading, while those inherited from **[RFC2396]** or **[RFC2396bis]** are shown with light yellow shading.

```
  This is an example of ABNF specific to XRI.

  This is an example of generic URI ABNF from RFC 2396 or 2396bis.
```

In addition, productions inherited from the IRI proposal **[IRI]** are prefixed with the letter "i" as they are in that document.

### 1.3.3 Glossary

A complete glossary of XRI-related terms is included in XRI Requirements and Glossary v1.0 **[XRIReqs]**. Following are the definitions central to this specification.

[Note: Are terms that need to be defined specific to internationalization?]

**Absolute Identifier**

An identifier that refers to a resource independent of the current context, i.e., using a global context. Mutually exclusive with "Relative Identifier".

**Abstract Identifier**

An identifier that is not directly resolvable to a resource, but is either: a) non-resolvable because it abstractly represents a non-network resource (see "Non-Resolvable Identifier"), or b) must be resolved to another identifier first (which may in turn be another abstract identifier, or a concrete identifier). A URN as described in **[RFC2141]** is an example of an abstract identifier. Abstract identifiers provide for additional levels of indirection in referencing resources which can be useful for a variety of purposes, including persistence, equivalence, human-friendliness, and data protection.

**Authority (or Identifier Authority)**

A resource that assigns identifiers to other resources. Note that in URI ABNF (and in the equivalence sections of XRI ABNF), the "authority" production refers explicitly to the top-level authority, i.e., the community root. However elsewhere in this specification the term "authority" refers more generally to the entity responsible for assigning and resolving identifiers at any level of delegation.

**Community (or Identifier Community)**

The set of resources that share a common identifier authority, typically a common root authority. Technically, the set of resources whose identifiers form a directed acyclic graph or tree.

**Concrete Identifier**

An identifier that can be directly resolved to a resource, rather than indirectly to another identifier. Examples include the MAC address of a networked computer, a phone number (that rings directly to a specific device), and a postal address (that is not a forwarding address). All concrete identifiers are intended to be resolvable identifiers. Contrast with "Abstract Identifier".

**Context (or Identifier Context)**

The backpointer of an identifier, i.e., the resource of which the identifier is an attribute. Context is the parent resource that assigns the identifier for the target resource. Since multiple resources may assign an identifier for a target resource, the resource can be said to be identified in multiple contexts. For absolute identifiers, the context is global, i.e., they have a known starting point. For relative identifiers, the context is local, i.e., it depends on the resource resolving the identifier.

**Cross-reference**

An absolute identifier assigned in one context that is reused in another context. Cross-references are used primarily to identify logically equivalent resources in different domains or physical locations. For example, a cross-reference may be used to identify the same logical invoice stored in two accounting systems (the originating system and the receiving system), the same logical Web page stored on multiple proxy servers, the same datatype used in multiple databases or XML schemas, or the same abstract concept used in multiple taxonomies or ontologies.

**Delegated Identifier**

A multi-segment identifier in which different segments are assigned by different identifier authorities. Mutually exclusive with "Local Identifier".

**Identifier**

Per **[RFC2396bis]**, anything that "embodies the information required to distinguish what is being identified from all other things within its scope of identification". In UML terms, an identifier is an attribute of a resource (the identifier context) that forms an association with another resource (the identifier target). The general term "identifier" does not specify whether the identifier is abstract or concrete, persistent or reassignable, human-friendly or machine-friendly, absolute or relative, local or delegated, or resolvable or non-resolvable.

**Local Identifier**

A single identifier, or any set of segments in a multi-segment identifier, that are assigned by the same identifier authority. Mutually exclusive with "Delegated Identifier".

**Non-Resolvable Identifier**

An identifier that does not directly reference a network resource or resource representation, but only abstractly represents a resource. A non-resolvable identifier is always an abstract identifier and does not have any corresponding data or metadata describing the resource it represents, thus it cannot be resolved in the conventional sense. From a machine perspective, the purpose of non-resolvable identifiers is to establish equivalence across contexts. Mutually exclusive with “Resolvable Identifier.”

**Persistent Identifier**

An identifier that is permanently assigned to a resource and that is intended never to be reassigned to another resource even if the original resource goes off the network, is terminated, or no longer exists. A URN as described in **[RFC2141]** is a persistent identifier. Mutually exclusive with "Reassignable Identifier".

**Reassignable Identifier**

An identifier that may be reassigned from one resource to another. Example: the domain name "example.com" may reassigned from ABC Company to XYZ Company, or the email address "john@example.com" may be reassigned from John Smith to John Jones. Reassignable identifiers tend to be human-friendly identifiers because they often represent the mapping of semantic relationships onto network resources or resource representations. Mutually exclusive with "Persistent Identifier".

**Relative Identifier**

An identifier that refers to a resource only in relationship to the current context, i.e., the context in which the identifier is being resolved. Mutually exclusive with "Absolute Identifier".

**Resolvable Identifier**

An identifier that references a network resource or resource representation and that can be resolved into data or metadata describing the target resource. Mutually exclusive with “Non-Resolvable Identifier.”

**Resource**

Per **[RFC2396bis]**, "anything that can be named or described". Resources are of two types: network resources (those that are network addressable) and non-network resources (those that exist entirely independent of a network). Network resources in turn contain a subtype, resource representations. A resource representation may represent either a network resource or a non-network resource.

**Resource Representation**

A network resource that represents the attributes of another resource. A resource representation may represent either a network resource (such as an application) or a non-network resource (such as a person, organization, or concept).

**Target (or Identifier Target)**

The resource referenced by an identifier. A target may be either a network resource (including a resource representation) or a non-network resource.

# 2 Syntax

## 2.1 Syntax Components

Generic XRI syntax consists of the scheme name "xri:" follow by the same hierarchical sequence of components as generic URI syntax. [ Need to highlight that we are not defining a URI scheme, but still using some of the URI ABNF productions ] Taken as a whole this sequence is referred to as the *XRI value*.

```
XRI             = "xri:" xri-value
xri-value       = [ xri-path ] [ "?" xri-query ] [ "#" xri-fragment ]
```

The path component can be hierarchical to any depth. A path can be globally absolute, relative to the local community, or relative to the current context as discussed in [ref Relative XRI section].

```
xri-path        = global-path / local-path / relative-path
global-path     = authority-part [ local-path ]
local-path      = "/" relative-path
relative-path   = *( [ "." ] "./" ) xri-segments
```

### 2.1.1 Authority

XRI syntax supports the same set of authorities as generic URI syntax, called a *URI authority*. In addition it supports an *XRI authority* which provides two other mechanisms of specifying the global context of an identifier, as defined in section 2.1.1.2.

```
authority-part  = URI-authority / XRI-authority
```

#### 2.1.1.1 URI Authority

In the context of an XRI, a URI authority is distinguished by the starting double slash ("//").

```
URI-authority   = "//" [ userinfo "@" ] host [ ":" port ]
```

The syntax following this starting delimiter is inherited directly from **[RFC2396bis]**, which simplifies the syntax in **[RFC2396]** and includes support for IPv6 addresses defined in **[RFC2732]**. First, the "userinfo" sub-component permits identifying a user in the context of a host.

```
userinfo        = *( unreserved / escaped / ";" /
                ":" / "&" / "=" / "+" / "$" / "," )
```

Next, the "host" sub-component has three options for identifying the host: a domain name, an IPv4 address, or an IPv6 literal.

```
host            = [ hostname / IPv4address / IPv6reference ]
```

Note that the host identifier may be omitted; if so a default may be defined by the semantics of a specific URI scheme. No default is specified by the XRI scheme.

A hostname, after the transformation described in step 4 of section 2.2.3.2, MUST meet the rules defined in section 3.2.2 of **[RFC2396]**. The productions for idomainlabel, qualified and hostname, therefore, have additional restrictions not reflected in the ABNF.

```
hostname        = idomainlabel qualified
qualified       = *( "." idomainlabel ) [ "." ]
idomainlabel    = 1*ucschar
domainlabel     = alphanum [ 0*61( alphanum / "-" ) alphanum ]
alphanum        = ALPHA / DIGIT
```

```
IPv4address     = dec-octet "." dec-octet "." dec-octet "." dec-octet
dec-octet       = DIGIT                                 ; 0-9
                / %x31-39 DIGIT                         ; 10-99
                / "1" 2DIGIT                            ; 100-199
                / "2" %x30-34 DIGIT                     ; 200-249
                / "25" %x30-35                          ; 250-255
```

Support for an IPv6 address literal was added by **[RFC2396bis]** following the syntax originally specified in **[RFC2732]**. Note that because IPv6 literals use colons as delimiters, they must be encapsulated within square brackets. This is similar to the use of parentheses in XRI cross-references (see [ref Xref section]).

```
IPv6reference   = "[" IPv6address "]"
IPv6address =   6( h4 ":" ) ls32
                / "::" 5( h4 ":" )                          ls32
                / [              h4 ] "::" 4( h4 ":" )      ls32
                / [ *1( h4 ":" ) h4 ] "::" 3( h4 ":" )      ls32
                / [ *2( h4 ":" ) h4 ] "::" 2( h4 ":" )      ls32
                / [ *3( h4 ":" ) h4 ] "::"    h4 ":"        ls32
                / [ *4( h4 ":" ) h4 ] "::"                  ls32
                / [ *5( h4 ":" ) h4 ] "::"                  h4
                / [ *6( h4 ":" ) h4 ] "::"
ls32            = ( h4 ":" h4 ) / IPv4address
                ; least-significant 32 bits of address
h4              = 1*4HEXDIG
```

Lastly, a host identifier can be followed by an optional port number. XRI does not define a default port, so if the port is omitted in an XRI it is undefined.

```
port            = *DIGIT
```

### 2.1.1.2 XRI Authority

In addition to the authorities supported in generic URI syntax, XRIs support two other mechanisms for specifying the global context of an identifier. The first is via global context symbols (GCS) and the second is via cross-references (abbreviated in the ABNF as "xref").

```
XRI-authority   = ( gcs-char xri-segment ) / xref-authority
```

### 2.1.1.2.1 Global Context Symbols (GCS)

In support of the human-friendly identifier (HFI) requirements, XRIs offer a compact syntax for indicating the global context of an identifier. This approach uses the minimal possible metadata—a single prefix character—to provide the context for an XRI authority segment.

```
gcs-char          = "+" / "=" / "@" / "$" / "*"
```

The global context symbol characters were selected from the set of symbol characters that are valid in a URI under **[RFC2396]** in order to represent the following global contexts:

| Symbol Character | Authority Type | Establishes global context for |
|---|---|---|
| + | General public | Identifiers for which there is no specific authority, i.e., that are established by public convention (e.g., in the English language, these would be the generic nouns). |
| = | Person | Identifiers that represent an individual person. |
| @ | Organization | Identifiers that represent any authority other than the general public or an individual person. |
| $ | OASIS XRI TC | Identifiers established by the XRI specification for specific types of identifier metadata (e.g., language, version syntax, query syntax, etc.). See [ref Appendix B: Special Identifiers Assigned by the XRI-specification] for a list of these identifiers. |
| * | User-relative | Identifiers for which the authority is relative to the current user (i.e., "user-shortcut XRIs"). |

Note that because the global context symbol precedes an xri-segment and the xri-segment production allows cross-references (below), the global context symbols can be used with any type of authority specified under any URI scheme.

### 2.1.1.2.2 Cross-References

Cross-references are the primary extensibility mechanism in XRI. A cross-reference is either: a) an absolute URI, or b) a global XRI value. Note these are syntactically distinct because the former must start with a legal URI scheme, and consequently an ALPHA, while the latter must start with a symbol character. In either case, a cross-reference is enclosed in parentheses the same way an IPv6 literal is encapsulated in square brackets as specified in **[RFC2732]** (see section 2.1.1.1).

```
xref-authority  = xref  ( "." sub-segment / ":" sub-segment) *( "."
                  sub-segment / ":" sub-segment)
xref            = "(" ( global-xri / URI ) ")"
global-xri      = global-path [ "?" xri-query ] [ "#" xri-fragment ]
```

A cross-reference may appear at any node of any XRI except within a URI authority segment. When a cross-reference is used as the very first segment in an XRI, it enables any globally-

unique identifier in any URI scheme to specify an authority, e.g., an HTTP URI, mailto URI, URN, etc.

A cross-reference is also the means by which a XRI can be expressed as non-resolvable. To do this, the entire XRI is enclosed in parentheses. Note that this is the equivalent in the English language of putting a word or phrase in quotes to express that the author is referring to the word or phrase itself and not to its normal meaning. Examples:

```
The term "user-friendly" is used frequently in computing.
       --English-language usage of a quoted term

xri:(+user-friendly)
       --XRI equivalent of expressing this abstract concept
```

### 2.1.2 Path

As with URIs in general, the XRI path component is a hierarchal sequence of path segments separated by a slash ("/") character and terminated by the first question-mark ("?") or number sign ("#") character, or by the end of the XRI. The key difference is that while a URI path segment is considered opaque, an XRI path segment can have two types of sub-segments: dot-sub-segments and colon-sub-segments.

```
xri-segments     = xri-segment *( "/" xri-segment )
xri-segment      = ( [ "." ] sub-segment / ":" sub-segment )
                 *( "." sub-segment / ":" sub-segment )
sub-segment      = *xri-pchar / xref
```

Dot-sub-segments specify reassignable identifiers and colon-sub-segments specify persistent identifiers (following the lead of URN syntax in **[RFC2141]**). The default is a reassignable identifier, so no leading dot is required if this is the first (or only) sub-segment. [ Does this distinction between reassignable and persistent “segments” need to be spun on out a bit more? ]

An XRI path segment can contain the same characters as a URI path segment with the exception of the dot (".") and the colon (":"), which if used will be interpreted as described above. If this interpretation is not desired for these characters, or for any other special XRI delimiters, these characters MUST be escaped when they appear in the path segment. See [Ref to Escaping section].

```
xri-pchar        = xri-unreserved / escaped / ";" / "!" / "*"
                 "@" / "&" / "=" / "+" / "$" / ","
```

Other than dot-sub-segments and colon-sub-segments (and cross-references within these), an XRI path segment is considered opaque by generic XRI syntax. As with URIs in general, XRI extensions or generating applications may define special meanings for other URI reserved characters for the purpose of delimiting extension-specific or generator-specific sub-components. For example, section 3.4 of **[RFC2396]** specifies the set of URI reserved characters that can be used within a query segment.

### 2.1.3 Query

The XRI query component is identical to the URI query component as described in Section 3.4 of **[RFC2396]** with one exception: it may begin with a cross-reference. This permits the incorporation of metadata in XRI syntax describing the query string syntax. See [ref Appendix B: Special Identifiers Assigned by the XRI-specification] for more about query syntax identifiers.

```
xri-query       = [ xref ] * ( pchar / "/" / "?" )
```

The characters permitted in a query segment are the full set allowed in a URI path segment.

```
pchar           = unreserved / escaped / ";" /
                  ":" / "@" / "&" / "=" / "+" / "$" / ","
```

### 2.1.4 Fragment

XRI syntax also supports fragments as described in Section 4.1 of **[RFC2396]** with the exception that it may begin with a cross-reference.

```
xri-fragment    = [ xref ] * ( pchar / "/" / "?" )
```

Fragments are supported primarily for compatibility with generic URI syntax, as XRI syntax can directly address attributes or secondary representations of a primary resource to any depth. XRIs can also use cross-references to identify media types or other alternative representations of a resource.

## 2.2 Characters

The character set and encoding of an XRI is primarily inherited from generic URI syntax as defined in **[RFC2396]** and clarified in **[RFC2396bis]**, however it also includes the expanded character set defined in **[IRI]**. XRI characters fall into the same three subsets as URI characters.

```
xri-characters  = xri-reserved / xri-unreserved / escaped
```

### 2.2.1 Reserved Characters

XRI reserved characters are used to delimit XRI syntax components and thus are a superset of the URI reserved character set. Specifically, four characters have been added: opening parentheses ("("), closing parentheses (")"), dot ("."), and asterisk ("*").

```
xri-reserved    = "/" / "?" / "#" / "[" / "]" / "(" / ")" / ";" / ":" /
                  "," / "." / "&" / "@" / "=" / "+" / "*" / "$"
```

If the use of an unescaped XRI reserved character as a data character would cause the interpretation of the XRI to be ambiguous, the character MUST be escaped as per the rules in [ref Escaping section].

### 2.2.2 Unreserved Characters

With the exception of the expanded UCS character set described in **[IRI]**, the unreserved character set for XRIs is the same as that of URIs after the subtraction of the four characters noted above (all of which are in of the "mark" production of **[RFC2396]** and **[RFC2396bis]**).

```
xri-unreserved  = ALPHA / DIGIT / ucschar / xri-mark
xri-mark        = "-" / "_" / "!" / "~" / "'"
```

The principle difference between XRI and URI reserved character sets is the inclusion of the UCS character set.

```
ucschar         = %xA0-D7FF / %xF900-FDCF / %xFDF0-FFEF /
                  %x10000-1FFFD / %x20000-2FFFD / %x30000-3FFFD /
                  %x40000-4FFFD / %x50000-5FFFD / %x60000-6FFFD /
                  %x70000-7FFFD / %x80000-8FFFD / %x90000-9FFFD /
                  %xA0000-AFFFD / %xB0000-BFFFD / %xC0000-CFFFD /
                  %xD0000-DFFFD / %xE1000-EFFFD
```

Escaping unreserved characters in an XRI does not change what resource is identified by that XRI. However, it may change the result of a URI comparison (see [ref Normalization and Comparison]), so unreserved characters should not be escaped unless necessary.

### 2.2.3 Escaped Characters

XRIs follow the same rules for escaping characters as URIs, i.e., any data in an XRI MUST be escaped if: a) it does not have a representation using an unreserved character, and b) using a reserved character would cause the XRI to be misinterpreted. An XRI thus escaped is said to be in “escaped normal form”. For consistency, all characters that are not in the ‘xri-unreserved’ production and that are not used as syntactical elements as defined in this specification SHOULD be escaped. In this context, misinterpretation applies to XRIs used directly (i.e. not as URIs). Rules for converting an XRI into a legal URI are discussing in section 2.2.3.2. [ This last sentence is somewhat confusing. Could use some examples here. ]

#### 2.2.3.1 Escaped Encoding

XRIs use the same percent-encoding as URIs as per section 2.4.1 of **[RFC2396]** and **[RFC2396bis]**. An escaped octet is encoded as a character triplet consisting of the percent character "%" followed by the two hexadecimal digits representing that octet's numeric value.

```
escaped         = "%" HEXDIG HEXDIG
```

The uppercase hexadecimal digits 'A' through 'F' are equivalent to the lowercase digits 'a' through 'f', respectively. XRIs that differ only in the case of hexadecimal digits used in escaped octets are equivalent. For consistency, uppercase digits SHOULD be used by XRI generators and normalizers.

#### 2.2.3.2 Converting XRIs to URIs

Although XRIs can be used directly, there may be times when it is desirable to use an XRI in a context that expects a URI reference as defined by **[RFC2396]**. In other cases it may be desirable to use an XRI in a context that allows an identifier containing characters disallowed by **[RFC2396]** but which provides a simple mapping into a legal URI. The anyURI tag in defined in **[XMLSchema2]** is an example of the second case, where an escaping procedure is defined for characters that would otherwise be illegal under **[RFC2396]**. Additionally, **[IRI]** is a work-in-progress that proposes a new protocol element - an Internationalized Resource Identifier, or IRI- and defines the process for converting an IRI to a URI. IRI to URI conversion differs from the conversion defined for anyURI in **[XMLSchema2]** primarily in that it includes an algorithm appropriate for internationalized domain names. There may be cases in which it is desirable to use an XRI in a context that expects an IRI.

This specification defines the process for transforming an XRI into a legal URI. Depending on the target application, it may be appropriate to terminate the transformation process before the final step. If the target application expects an identifier defined as anyURI in **[XMLSchema2]**, for

example, the transformation may terminate at the point at which the XRI has reached the threshold defined for protocol elements allowed under that specification. Where appropriate, the transformation steps below note such thresholds. Except for transformations specific to XRI syntax, these steps closely follow the algorithm proposed in **[IRI]**.

Applications MUST map XRIs to URIs using the following steps (or any equivalent process that achieves the same result).

1. If the XRI is not encoded in UTF-8, convert the XRI to a sequence of characters encoded in UTF-8, normalized according to Normalization Form C (NFC) as defined in **[UTR15]**.
2. Optionally add font and language metadata (see note below).
3. Perform XRI-specific conversion defined in section 2.2.3.3. At this point the identifier may be used as an IRI.
4. If the XRI has a 'hostname' component, replace it with the 'hostname' component converted using the ToASCII operation defined in section 4.1 of **[RFC3490]**, with the UseSTD3ASCIIRules flag set to true and the AllowUnassigned flag set to false. At this point the identifier may be used as anyURI defined in **[XMLSchema2]** or in a comparable context.
5. Replace each character that is disallowed in URI references with escaped triplet(s) as described in section 2.2.3.1, one escaped triplet for each octet in the UTF-8 encoding of the disallowed character. At this point the identifier may be used as a generic URI.

A note on step two above. In some languages, a UTF-8 encoded string (i.e. a sequence of UTF-8 encoded characters) does not contain enough information to determine how to properly render that string in the intended language. Specifically, to represent the glyph of a UTF-8 encoded character, language information and font information may be required. On the other hand, local language encoding always has the language and font information associated with it. To make it possible to revert back to the local language representation of an XRI, it may be necessary to record the language and font context of an XRI when converting to UTF-8. If UTF-8 encoding would lose information required to transform the XRI back into human readable form in the intended language and font, the transformation MAY include mark up by use of cross references containing the $l and/or $f identifier defined in Appendix B. Once the language and font context is declared it will be valid until it is reset by another $l/$f declaration.

The XRI-specific conversion described in step three is not idempotent, i.e. each time this step is applied it may yield different results. It is very important, therefore, that implementers are careful not to apply this step more than once since doing so may change the semantics of the identifier. In general, an application SHOULD use the least escaped version appropriate for the context in which the identifier appears. If the context, for example, allows an XRI directly, the identifier SHOULD be in escaped normal form described in section 2.2.3. If the context allows an IRI but not a XRI, the identifier SHOULD be in the form that results from step three, and so on.

The form of the XRI that results from each step in this section is equivalent to the result of any other step. In other words, applying this conversion does not change the equivalence of the identifier.

### 2.2.3.3 XRI-specific conversion for use in URIs

This section describes issues that can arise when an XRI is converted to URI. It looks only at issues specific to XRI syntax and not, for example, at international character issues. It also defines a conversion operation that performs the XRI-specific conversions required during the conversion of an XRI into a generic URI. This conversion operation must be done in conjunction with the steps defined in section 2.2.3.2 in order to effect a complete conversion from an XRI to a URI. In other words, the conversion in this section has very limited utility on its own. It is intended to be used as part of the larger conversion process described in section 2.2.3.2.

XRIs can contain other URIs as cross-references (see section [ref to cross-reference section]). These URIs can contain characters that, if unescaped, would cause misinterpretation when the XRI is converted to a URI. Consider the following XRI.

```
xri:@example/(xri:@example2/abc?id=1)
```

The generic parsing algorithm described in **[RFC2396]** would separate the above XRI into the following components

```
scheme = xri
authority = <undefined>
path = @example/(xri:@example2/abc?
query = id=1)
```

The desired separation is

```
scheme = xri
authority = <undefined>
path = @example/(xri:@example2?id=1)
query = <undefined>
```

To avoid this type of misinterpretation, certain characters in a cross-reference must be escaped when converting an XRI to a URI. In particular, cross-references must be converted such that the question mark "?" character is escaped as "%3F", the number sign "#" character is escaped as "%28", and the colon ":" character is escaped as "%3A".

The example above, then, would be expressed as

```
xri:@example/(xri%3A@example2%3Fid=1)
```

A slash "/" character in a cross-reference can also be misinterpreted when the XRI is converted into a URI. Consider

```
xri://example.com/(@example/abc)
```

If this were used as a base URI as defined in section 5 of [RFC2396], the algorithm described in section 5.2 of [RFC2396] would append a relative-path reference to

```
xri://example.com/(@example/
```

instead of the intended

```
xri://example.com/
```

because the algorithm is defined in terms of the last (right-most) slash character. This problem is avoided by escaping slashes within cross-references as '%2F'. The above example, then, would be expressed as

```
xri://example.com/(@example%2Fabc)
```

Note that ambiguity is possible if an XRI in escaped normal form contains characters that have been escaped to indicate that they should not be interpreted in their normal syntactical sense. For example, consider the following XRI in escaped normal form

```
xri://example.com/(@example/abc%2Fd/ef)
```

This slash character between ‘c’ and ‘d’ is escaped to show that it’s not a syntactical element of the XRI, i.e. that it should be interpreted literally and not as a path separator. To preserve this type of distinction when converting an XRI to a URI, the percent “%” character must be escaped as “%25”. The above example, fully converted, would be

```
xri://example.com/(@example%2Fabc%252Fd%2Fef)
```

The following, then, are the XRI-specific steps required to convert an XRI into a URI.

1. Escape all percent “%” characters as “%25” across the entire XRI.
2. Escape all number sign “#” characters that appear within a cross-reference as “%23”.
3. Escape all question mark “?” characters that appear within a cross-reference as “%3F”.
4. Escape all colon “:” characters that appear within a cross-reference as “%23”.
5. Escape all slash “/” characters that appear within a cross-reference as “%2F”.

Note that the XRI must be in escaped normal form and all URIs in cross-references must be in an escaped form appropriate to their schemes before the above rules are applied.

### 2.2.3.4 Converting URIs to XRIs

There may be times when it is desirable to convert an XRI in URI escaped form into an XRI in escaped normal form. This section gives a procedure to do such a conversion. Except for steps specific to XRIs, this procedure very closely follows the algorithm proposed by **[IRI]**.

Conversion from an XRI in URI escaped form into an XRI in escaped normal form MUST use the following steps (or any equivalent process that achieves the same result).

1. If the identifier is not encoded in US-ASCII, convert it to a sequence of octets in US-ASCII.
2. If the identifier has a ‘hostname’ component, replace it with the UTF-8 encoded ‘hostname’ component converted using the ToUnicode operation defined in section 4.2 of **[RFC3490]**, with the UseSTD3ASCIIRules flag set to true and the AllowedUnassigned flag set to false.
3. Convert all escaped characters (as defined in section 2.2.3.1) with their corresponding octets, except for the percent “%” character, those characters in the ‘reserved’ production of **[RFC2396]** and US-ASCII characters disallowed in URIs by section 2.4.3 of **[RFC2396]**.
4. Re-escape any octet produced in step 3 that is not part of a strictly legal UTF-8 octet sequence. [NOTE: This is verbatim from IRI. Is this ok? Should we elaborate?]
5. Perform the following XRI-specific conversions
    a. Convert all escaped slash “/” characters to their corresponding octets.
    b. Convert all escaped colon “:” characters to their corresponding octets.
    c. Convert all escaped question mark “?” characters to their corresponding octets.
    d. Convert all escaped number sign “#” characters to their corresponding octets.
    e. Convert all escaped percent “%” characters to their corresponding octets.

6. Encode the resulting sequence in UTF-8 (except for that portion already converted by step 3).

### 2.2.4 Excluded Characters

XRI syntax excludes the same characters as URI syntax for the same reasons as described in section 2.5 of **[RFC2396]** and **[RFC2396bis]**. Data octets corresponding to these characters must be escaped in order to be represented within an XRI.

```
excluded        = invisible / delims / unwise
invisible       = CTL / SP / %x80-FF
delims          = "<" / ">" / "%" / DQUOTE
unwise          = "{" / "}" / "|" / "\" / "^" / "`"
```

### 2.2.5 Legal Character Sequence

Not all ASCII sequences can be derived from UTF-8 sequences. A valid XRI character sequence MUST be derivable by escaping an equivalent UTF-8 sequence. [NOTE: This needs review/expansion.]

## 2.3 Character Encoding and Internationalization

The basic character encoding of XRI is UTF-8, as recommended by **[RFC2718]**. When an XRI is used as a human readable identifier, the representation of the XRI on the underlying document should use the character encoding of the underlying document. However, this string must be converted to UTF-8 before any further processing.

## 2.4 Relative XRIs

The authority component, as defined in 2.1.1, may be either a URI-authority (section 2.1.1.1) or an XRI-authority (section 2.1.1.2). In this section, “authority” should be understood as defined by section 2.1.1 of this specification and not in the narrower sense of section 3.2 of **[RFC2396]**.

For a relative XRI reference that does not contain an authority component but whose base XRI contains an authority component that matches the URI-authority production, the rules for resolving relative references defined in section 5.2 of **[RFC2396]** apply.

For a relative XRI reference that does not contain an authority component but whose base XRI contains an authority component that matches the XRI-authority production, the rules defined in section 5.2 of **[RFC2396]** need modification because an XRI authority is considered opaque by generic URI syntax.

The following sections, therefore, define the process for resolving a relative XRI reference into a string that matches the XRI production defined in section 2.1 for all XRIs, including those relative references that would otherwise be unresolvable because they are considered opaque by **[RFC2396]**.

### 2.4.1 Establishing a Base XRI

A base XRI is established according to the rules defined in section 5.1 of **[RFC2396]**. In other words, there is no difference between establishing a base XRI and establishing the base of any generic URI. [ Need to mention that XRIs are not URIs anyway, unless they are converted to URI form. ]

### 2.4.2 Obtaining the Referenced XRI

Section 5.2 of **[RFC2396]** describes rules for resolving relative references to absolute forms of URIs. For XRIs matching the XRI Authority production in [ref XRI Authority section], these same rules apply with the following modifications:

- In step 1, the XRI reference is parsed using an XRI aware parser such that the “authority component” is interpreted as the "authority-part" production defined in section 2.1.1 of this specification.
- Step 4 states, “If the authority component is defined, then the reference is a network-path and we skip to step 7”. For XRIs, the presence of an authority component does not imply that the reference is a network-path as defined by **[RFC2396]** because it may be an XRI-authority component. However, the instruction to skip to step 7 is still valid for XRIs. In other words, the processing instruction is correct, but the inference as to the type of reference is invalid.
- In step 4, the base XRI is parsed using an XRI aware parser such that the “authority component” is interpreted as the authority-part production defined in section 2.1.1 of this specification.
- In step 7, the block that reads

```
if authority is defined then
   append "//" to result
   append authority to result
```

  is replaced by

```
if authority is defined then
   if type-of(authority) == URI-authority
      append "//" to result
   append authority to result
```

It is important to note that the algorithm described in section 5.2 of **[RFC2396]** will generally produce incorrect results when applied to relative XRI references in which the authority component matches the XRI-authority production. This type of relative XRI reference, therefore, should only be used in contexts in which the above algorithm is known to be employed. [ Example would be useful. ]

## 2.5 Normalization and Comparison

The scheme component is case-insensitive for comparison for XRIs and all URIs used as cross-references.

Comparison of authority components of two XRIs, as defined in 2.1.1, is case-insensitive for all characters in the ALPHA production.

Two XRI authority components, as defined in 2.1.1, are equivalent if they match using a case-insensitive comparison after applying steps one and three of the process described in section 2.2.3.2.

Two XRIs MUST be equivalent if they are character-for-character equivalent. It follows, then, that they are equivalent if they are byte-for-bye equivalent when both XRIs use the same character encoding.

All forms of the XRI during the conversion process described in section 2.2.3.2 are equivalent.

Two XRIs that differ only in escaped unreserved characters are equivalent.

Each application that uses XRIs MAY define additional equivalence rules as appropriate.

Section 6 of **[RFC2396bis]** offers advice on more aggressive strategies for normalization and comparison as well as best practices for canonicalization of generic URIs. Implementers may find

891 this information useful in developing a strategy for establishing equivalence, particularly with
892 respect to non-XRI cross-references.

# 3 Resolution

## 3.1 Introduction to Resolution Architecture

Resolution is the process of converting an XRI into data and metadata about the resource identified by the XRI.

Because XRIs will be used in a wide variety of deployments, communities, and applications, no single resolution mechanism is appropriate for all XRIs. Thus, a resolution framework and concrete implementations of that framework are defined. This framework MAY be required by communities to allow resolution of XRIs that they define. Other resolution mechanisms MAY be defined on a per-community basis.

It is important to note that XRIs can be "resolved" in a variety of ways. For example, they may be used as keys in a database, or used as filenames in a filesystem. The intent of this framework is to define an interoperable process for discovering and accessing data in an open system such as the Internet where such data may be distributed across a number of systems.

Policies for management of identifiers are defined on a community-by-community basis. Each community is identified via the authority portion of an XRI (which can be either a URI authority or an XRI authority as defined in Section 2.1.1). When a community chooses to create a new identifier authority, it SHOULD define a policy for how identifiers under this authority are assigned and managed. Furthermore, it SHOULD define what resolution scheme should be used for resolving those identifiers.

Resolution is defined as a set of interactions between a client and a series of “endpoints.” Endpoints are networked systems that participate in XRI resolution using one or more of the implementations of the framework. These endpoints are usually discovered through processes defined in the framework, except for those which are defined by XRI communities as being the “community root” (which is effectively the starting place for the resolution process). These endpoints are advertised “out of band” to entities wishing to resolve identifiers in the identifier community.

The resolution framework here expects the XRI being resolved to have been converted into a URI-compatible form, following the rules in Section 2.2.3.2, "Converting XRIs to URIs.”

### 3.1.1 Assumptions

XRI resolution makes several minimal assumptions about XRIs:

- The endpoints representing the top-level authority for any globally unique XRI are identified with the "uri-authority" or "xri-authority" part of the XRI.
- Data corresponding to a single XRI may be retrieved or manipulated by multiple protocols at multiple endpoints.
- Each endpoint and protocol may present a different subset, type, or representation of the data and metadata associated with the identified resource.

### 3.1.2 Phases of Resolution

The XRI resolution framework is designed to be as flexible as possible given the assumptions described above and the wide number of anticipated uses for XRIs. The framework reflects the structure of XRIs, and consists of two phases:

- Authority Resolution
- Local Access

Authority Resolution is the process of finding the endpoint or endpoints representing the authority that controls the community of identifiers in which the XRI is defined. Authority Resolution results

in a list of local access descriptors, each describing an endpoint providing “local access” service. These Local Access Descriptors are defined in section 3.3.1. Resolving clients choose an endpoint described by one of these descriptors and select a local access protocol with which to access that endpoint.

Figure 1 demonstrates the phases of XRI resolution:

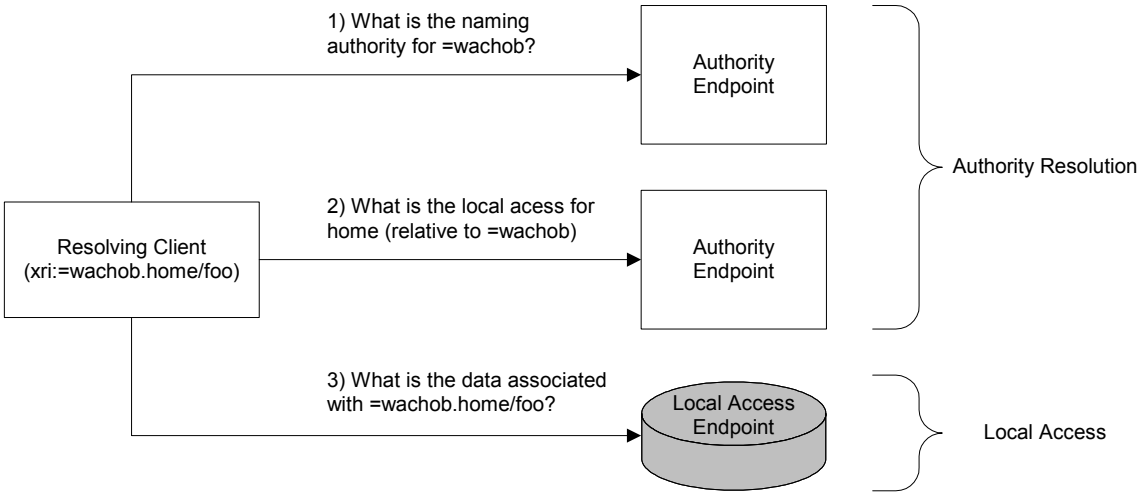



Figure 1: Phases of Resolution

## 3.2 Phase 1: Authority Resolution

### 3.2.1 General Description

Authority Resolution is the process of finding a system representing an authority identified by the “authority” component of the XRI. That component of the XRI can either be in the form of a DNS name, an IP address, a GCS identifier, or a cross-reference. Each type of authority has a separate method for resolution. DNS-specified authorities are resolved using DNS-based Authority Resolution (DNAR) described in section 3.2.2 below. Authorities identified by GCS identifiers or cross-references are resolved using the XRI-Authority Resolution Framework (XARF) defined in section 3.2.3 below. Finally, authorities identified by an IP address are resolved according to the process in section 3.2.4 below.

| Type of Authority | XRI BNF Production | Description | Example | Method of Resolution |
|---|---|---|---|---|
| DNS- or IP-Address Specified | URI-authority | An authority identified by a DNS name or IP address. | **xri://example.com**/foo.bar | DNAR |
| Abstract | XRI-authority | An authority identified by a resolvable GCS | **xri:+example**/foo.bar | XARF |

| | | | | |
|---|---|---|---|---|
| | | identifier or a cross-reference. | | |

957 Table 1: Types of Authority Identifiers

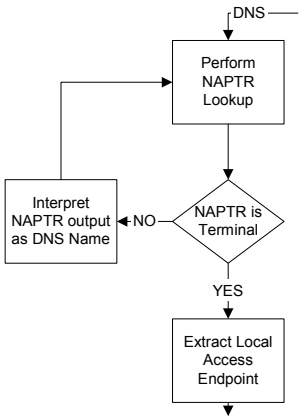
958 Whether DNAR or XARF is used, the result is a list of descriptors of local access endpoints.
959 These local access descriptors are defined in section 3.3.1 below. The resolving client can then
960 choose which endpoint it wishes to use to access data, attributes, or services associated with the
961 XRI.

## 962 3.2.2 DNS-Specified Authority Resolution (DAR)

963 The process for resolving DNS-specified authorities is a DDDS "application" which takes
964 advantage of NAPTR and SRV records associated with DNS names.

965 DAR resolution is formally defined as a DDDS application with the following attributes:

| | |
|---|---|
| **Application Unique String:** | The XRI being resolved |
| **First Well Know Rule:** | The DNS name specified in the Authority Name segment |
| **Flags:** | "s" – a terminal flag which indicates that the result of the application of the regular expression (or replacement) is a DNS name pointing to an SRV record<br>"u" – a terminal flag which indicates that the result of the application of the regular expression (or replacement) is a URI |
| **Service Parameters:** | This field is empty unless a terminal flag is present.<br><br>This field contains a local access protocol descriptor as described in section 3.3.2 ("Format of Local Access Protocol Descriptors"). Initially, the only protocol descriptors using the 'thttp' local access service descriptor (defined in section 3.3.5.1 below) may be used. Local access service descriptors are unique to each use of a network protocol, and are described more fully in section 3.3.3 below. |
| **Valid Databases:** | DNS from RFC 3403 |




966
967 Figure 2: DAR Algorithm

968 The DAR algorithm is as follows:

1) The DNS name that is the authority in the XRI is the “initial key” for the DDDS algorithm. The full XRI being resolved is the application unique identifier.
2) The DNS name is resolved using a query type of “NAPTR”.
3) The identifier is applied to the NAPTR record (either the regular expression or the replacement, following RFC 3402 rules).
4) If the NAPTR record does not contain a terminal flag, then the result of step 3 is interpreted as a DNS name. The algorithm loops back to step 2 with the result of step 3 as the DNS name to be resolved.
5) If the NAPTR record contains a “u” flag, then the result of step 3 is interpreted as a URI. This URI is the network location of the local access service, and the “service” field is the local access protocol descriptor used for local access. DAR terminates.
6) If the NAPTR record contains a “s” flag, then the result of step 3 is interpreted as a DNS name. This DNS name is resolved into a set of SRV records, each of which describes a network location of the local access service using a hostname and port number. The local access protocol descriptor is copied from the service field. These host and IP pairs corresponding to each SRV record are then reconstructed back into a URI based on the local access protocol descriptor and the SRV hostname. For example, if the DNS name that is the result of step 3 is “_thttp._tls._tcp” and the local access protocol descriptor is “thttps+I2R”, and an SRV record contains a target of “thttp.xri.example.com” and a port of 443, then the resulting network location would be https://thttp.xri.example.com/ as defined in section 3.3.5.1. After this network location is extracted, DAR terminates.
[ The construction of a network endpoint from an SRV record probably needs more explanation and/or examples ]

### 3.2.3 XRI Authority Resolution Framework (XARF)

#### 3.2.3.1 Introduction

The process for resolving abstractly-identified authorities (corresponding to the XRI-Authority BNF production) is comprised of an initialization step and an iterative series of operations. One of these iterations is performed for each node in the Authority component of the XRI. Each iteration consists of an invocation of a “relative lookup mechanism”. The term “relative” emphasizes the fact that only one node of the Authority component is being resolved at a time, from left to right.

This specification defines two of these relative lookup mechanisms: an HTTP-based mechanism (“XRI-HTTP Relative Lookup Mechanism (NA1)”) and one derived from DDDS (“RDDDS Relative Lookup Mechanism (NA2)”).

The XARF algorithm results in a set of one or more Local Access Descriptors, the form of which is defined in section 3.3.1.

#### 3.2.3.2 User Relative XRIs

XRIs beginning with the user-relative community symbol (‘*’) are a special case for resolution. The authority for these identifiers is defined by the user of the XRI, and not uniquely specified in the XRI itself. Thus, these XRIs are not resolvable without the establishment of an authority for the XRI from some source other than the characters in the XRI.

XRIs beginning with the User-Relative community identifiers MUST be transformed into XRIs with an explicit Authority identifier (other than one based on the user relative community) before they can be resolved using the resolution mechanisms defined in this specification.

Note that in most cases, this transformation is simply the replacement of the ‘*’ character with a prefix corresponding to an Authority identifier. For example, if a client is configured with a default community of “@employer”, then the xri “xri:*workstation/identifier” would be converted into “xri:@employer.workstation/identifier”.

### 3.2.3.3 Authority Descriptors

Endpoints that act as Authorities are described with Authority Descriptors, as defined below:

| Data Item | Description | Example |
|---|---|---|
| Location | The network address where this endpoint can be reached. This is in the form of a URI, for “xri-http” authorities and in the form of a DNS URI for “rddds” authorities. | dns:authority.example.com |
| Protocol Descriptor | The protocol to use when communicating with this authority endpoint. See the definition of Authority Protocol Descriptor at section 3.2.3.4 | na2 |

Table 2: Authority Descriptor

### 3.2.3.4 Authority Protocol Descriptors

Each Authority Relative Lookup Mechanism defines a short string to uniquely identify the protocol used to perform a lookup at the Authority. There are currently two Authority Protocol Descriptors defined:

| Protocol | Authority Protocol Descriptor |
|---|---|
| XRI-HTTP | na1 |
| RDDDS | na2 |

Table 3: Authority Protocol Descriptors

### 3.2.3.5 Algorithm

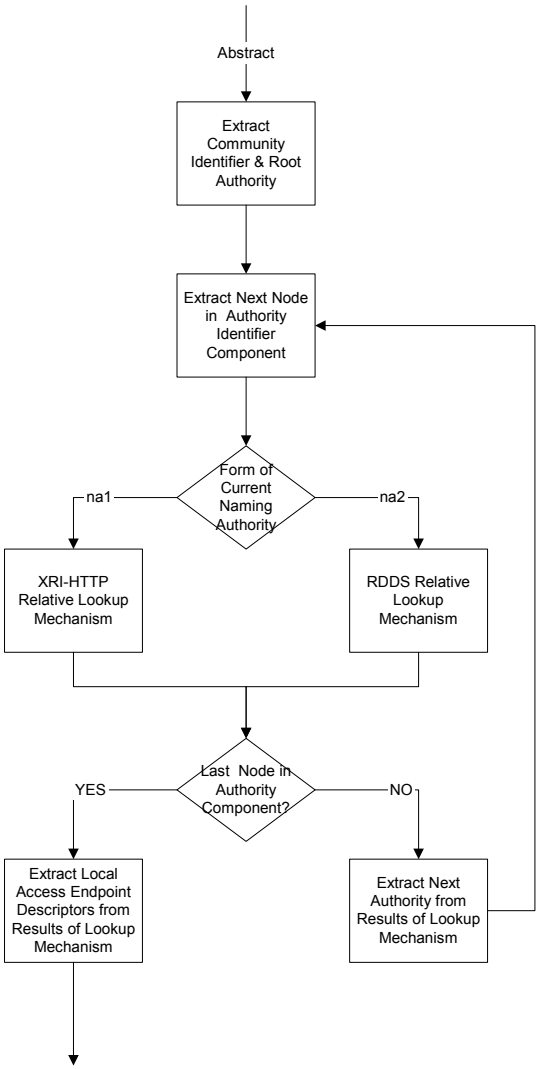



Figure 3: XARF Algorithm

The XARF algorithm consists of an initialization step and repeated invocations of a relative lookup mechanism. Each invocation of the relative lookup mechanism operates on a specific node (the "current node") of the XRI Name Authority component, progressing from left to right.

The initialization step is the extraction of the community identifier from the Authority component of the XRI. This first node may specify a global community (using a global community identifier) or a privately defined community (using a cross reference). In the case of a global community identifier, the global community identifier is treated as a separate node of the authority identifier. Also, every global community identifier has one or more associated Authority Descriptors. In the case of a cross reference, there must be one or more Authority Descriptor associated with that cross reference. The discovery of these associated Authority Descriptors is an out of band process that is assumed to have taken place when the resolving framework is deployed.

After the initial Authority Descriptor is selected, the next node of the Authority becomes the initial value for the "current node" in the iteration described below.

| | |
|---|---|
| **XRI** | xri:@example.internal/foo |
| **Authority** | @example.internal |
| **Community Identifier** | @ |
| **First Node Resolved** | .example |

Table 4: Global Community Identifiers

| | |
|---|---|
| **XRI** | xri:(http://www.example.com).internal/foo |
| **Authority** | (http://www.example.com).internal/foo |
| **Community Identifier** | (http://www.example.com) [Would this be escaped in URI form?] |
| **First Node Resolved** | .internal |

Table 5: Cross-Reference Community Identifiers

Each XARF iteration begins with an Authority Descriptor, and results in a set of Authority Descriptors or Local Access Descriptors.

If the node being operated on is the last node in the XRI Name Authority component, then the results should be a list of Local Access Descriptors.

If the node being operated on is not the last node in the XRI Name Authority component, then the results of the iteration should be a list of Authority Descriptors. The next iteration will use one of these Authority Descriptors from the pervious iteration to perform the lookup on the next node in the XRI Authority.

Each iteration consists of the following steps:

1) Select a Authority Descriptor which is available from the previous iteration, or, if this is the first iteration, select a Authority Descriptor from those configured to correspond to the first node of the XRI Authority component as described above.
2) Perform the relative lookup mechanism specified in the Authority Descriptor using the current node as the query for the lookup. Currently, there are two of these relative lookup mechanisms: XRI-HTTP and RDDDS.
3) If the current node is not the last node in the XRI Authority component, then the results of step 2 are a set of Authority Descriptors. The loop repeats at step 1, using the next node of the XRI Authority component, and using the Authority Descriptors just retrieved from step 2.
4) If the current node is the last node in the XRI Authority component, then the results of step 2 must be a set of Local Access Descriptors. XARF terminates with these Local Access Descriptors as the result.

If step 2 results in an empty list of Descriptors, this is equivalent to an unresolvable XRI Authority component, and SHOULD reported to the user of the resolver as an error.

### 3.2.3.6 XRI-HTTP Relative Lookup Mechanism (NA1)

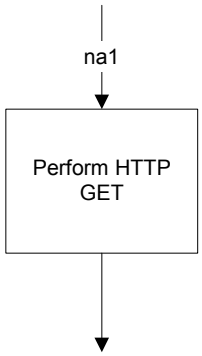



Table 6: XRI-HTTP Algorithm

The XRI-HTTP relative lookup mechanism performs a simple HTTP GET to resolve a node of an XRI Authority component. This relative lookup mechanism has the Authority Protocol Descriptor "na1". Any Authority Descriptor with the Authority Protocol Descriptor "na1" MUST contain a network location of the form of a HTTP or HTTPS URI.

A resolver performing the XRI-HTTP relative lookup mechanism constructs a request URL from the Authority Descriptor location field and the current node (from the XARF iteration). This URL is a concatenation of the location field and the current node as described below:

```
na1-url = authority-location ?("/") url-escape(current-node)
```

The separator "/" is inserted between the authority-location in the descriptor and the current node only if the authority location does not end in a "/". The current node must be "URL-escaped" [reference] before being inserted into the request URL.

The resolving client MUST use the HTTP/1.1 protocol. All HTTP semantics are available to the resolving client and Authority endpoint. Specifically, redirects, security, caching and other HTTP-defined semantics should be employed where necessary. However, for the purposes of interoperability and ease of implementation, use of such features should be minimized to the extent possible.

The content of the result of the HTTP request is a document that contains a list of newline-separated [better way to say this?] lines of plain text. The document is of content-type "text/plain". Each line is of the following form:

```
na1-result = protocol-descriptor space location newline
```

The value of every service-descriptor field is MUST be a legal value defined for Authority Protocol Descriptors (section 3.2.3.4) or Local Access Protocol Descriptors (section 3.3.2). The value of every location field is constrained by the definition of the service descriptor. For example, if the service descriptor is "na2", then the location must be a DNS URI.

Each line is parsed and transformed into an Authority Descriptor or Local Access Descriptor depending on whether the current node is the last node of the XRI Authority component. The result of each XRI-HTTP invocation is a list of descriptors.

```
Non-Terminal Results Example:
```

```
The name authority is available at a HTTP na1 endpoint, an HTTPS na2
endpoint, and at a domain name for use with na2 (DDDS).

na1 http://next.step.example.com/resolve/b/
na1 https://alternate.step.example.com/resolve/c/
na2 dns:my.nazone.example.com
```

```
Terminal Results Example:

There is a thttp-based I2R service for accessing wsdl associated with
the XRI and a ldap-based I2R service for accessing wsil associated with
the XRI.

thttp+I2R/wsdl http://authority.example.com/uri-res/
ldap+I2R/wsil ldap://user@foo.ldap.example.com/
```

```
Complete HTTP Request-Response Example:
The resolving client is peforming an "na1" request on the URL
http://xrib.example.com/naresolve with the relative identifier "c"

Client to Server:
GET /naresolve/c HTTP/1.1
Host: xrib.example.com

Response to Client from Server:
HTTP/1.1 200 OK
Content-Type: text/plain

na1 http://next.step.example.com/resolve/b/
na1 https://alternate.step.example.com/resolve/c/
na2 dns:my.nazone.example.com
```

### 3.2.3.7 RDDDS Relative Lookup Mechanism (NA2)

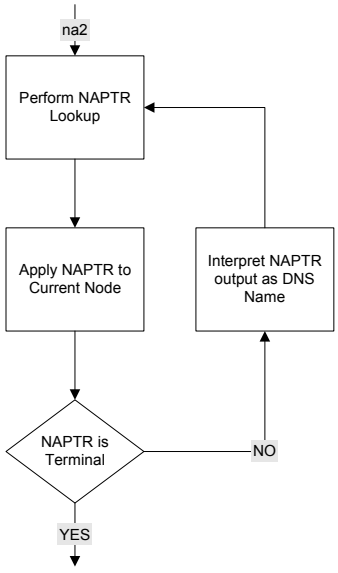



Figure 4: RDDS Algorithm

#### 3.2.3.7.1 Introduction

This lookup mechanism is a DDDS application as described in RFC 3401, and has the Authority Protocol Descriptor "na2". Each invocation of RDDDS on a node of the XRI Authority component is a complete run of the DDDS algorithm as defined in RFC 3402. This algorithm takes an identifier and finds an Authority for that identifier.

Note that while RDDDS is technically compliant with RFC 3402, it is not in conformance to the original intent of the DDDS specification because it performs resolution on only part of the XRI (i.e. a single node in the XRI Authority component), and does not apply the entire XRI to the NAPTR records retrieved in the DDDS algorithm.

RDDDS uses the DDDS algorithm straightforwardly, but there are some concepts that must be mapped from this specification to the DDDS suite of specifications. The following table describes the conceptual mapping:

| DDDS Concept | XRI Concept | Description |
|---|---|---|
| Application-unique string | Current node | Each invocation of RDDDS corresponds to a single node in the XRI Authority component, and thus appears to the DDDS algorithm as the entire application unique string. |
| First Well-Known Rule | Location from the current Authority Descriptor | Each invocation of RDDDS begins with an Authority Descriptor that contains a DNS name. This DNS name is how the DDDS algorithm initially queries DNS for the first iteration of the DDDS algorithm. |

Table 7: Mapping DDDS and XRI Concepts for RDDDS

The RDDDS relative lookup mechanism is formally defined here as an DDDS application:

| | |
|---|---|
| **Application Unique String:** | The current node from the XRI Authority Resolution Framework |
| **First Well Know Rule:** | The first key is AuthorityLocation |
| **Flags:** | *"s" (a terminal flag) which signifies that the result of the NAPTR regex/replacement is a domain name which has one or more SRV records associated with it.* The use of SRV records needs further specification – also see SRV records in section 3.2.2.<br>"u" (a terminal flag) which signifies that the result of the NAPTR regex/replacement is a URI (see RFC 3404). *This needs further fleshing out – see description above.*<br>The absence of a terminal flag means that the DDDS resolution continues with the domain name that is a result of applying the regex/replacement to the current node. In this case, DDDS continues with the key equal to this domain name. |
| **Service Parameters:** | If this DDDS resolution is NOT the last step in the XARF protocol, then the service must indicate a XARF resolution mechanism (e.g. 'na1' or 'na2' that are defined in this document). This indicates that the client resolver may use the indicated resolution procotol and endpoint for the next iteration in the XARF protocol.<br>If this is the last iteration in the XARF protocol, the service types must indicate a local access protocol. In this case, see Section |

| | |
|---|---|
| | 3.3.2, “Format of Local Access Protocol Descriptors” for the format of the service parameters. |
| **Valid Databases:** | DNS from RFC 3403 |

Table 8: Formal RDDDS Definition

### 3.2.3.7.2 Algorithm

As an implementation of the DDDS algorithm, each invocation of RDDDS consists of the execution of an iterative step one or more times. Each iteration of the loop begins with a current DNS name, along with the current node of the XRI Authority component that is being resolved. The “current node” does not change across the entire invocation of RDDDS; this algorithm operates on one node at a time. The initial DNS name is extracted from the current Authority Descriptor. The result of the RDDDS invocation is a set of Local Access Descriptors or Authority Descriptors as described in Section 3.2.3, “XRI Authority Resolution Framework (XARF)”.

The steps for the loop inside the RDDDS lookup mechanism are:

1) A NAPTR DNS query is performed using the current DNS name, resulting in a set of NAPTR records.
2) The set of NAPTR records is iterated through, according to the algorithm in RFC 3402, section 3.3 (??) until a NAPTR record that matches the current node (using the regex or replacement fields) and matches any requirements for service type defined by the resolver. (*Needs more formalization, but the idea is that the first matching NAPTR is used, based on the order and preference fields of the NAPTR records*)
3) The current node is applied to the regex or replacement field (as the application unique string) to the first NAPTR record that matches from step 3.
4) If the matched NAPTR record from step 3 does NOT contain any flags, then the result of the regex substitution (or replacement) from step 3 is a DNS name. The current DNS name is set to the result of the regex substitution (or replacement) from step 3. The algorithm jumps back to step 1.
5) If the NAPTR record has the “u” flag, then the result of applying the CurrentName to the regex in the NAPTR is a URI describing a network location. For each service listed in this NAPTR record, a new descriptor is created. This descriptor may be a Local Access Descriptor (if the current node is the last node of the XRI Authority component), or a Authority Descriptor (if the current node is not the last node of the XRI Authority component). Both Authority Descriptors and Local Access Descriptors have Location and Protocol Descriptor fields. The Protocol Descriptor field is set to the content of the NAPTR record’s service field. The Location field is set to the URI value that results from the regex substation (or replacement). RDDDS then terminates with this new descriptor as a result.
6) If the NAPTR record contains a “s” flag, then the result of step 3 is interpreted as a DNS name. This DNS name is resolved into a set of SRV records, each of which describes a network location using a hostname and port number. As with step 5, a new descriptor is created from each SRV record. The protocol descriptor is copied from the service field of the matching NAPTR record. The host and IP pairs corresponding to each SRV record are then reconstructed back into a URI based on the protocol descriptor and the SRV hostname. For example, if the DNS name that is the result of step 3 is “_thttp._tls._tcp” and the local access protocol descriptor is “thttps+I2R”, and an SRV record contains a target of “thttp.xri.example.com” and a port of 443, then the resulting network location would be https://thttp.xri.example.com/ as defined in section 3.3.5.1. RDDDS then terminates with this set of descriptors as a result. The use of SRV records needs further specification – also see SRV records in section 3.2.2.

### 3.2.4 IP-Address Authority Resolution (IAR)

[Its not clear what the use case is here, but for consistency this section defines a way of resolving identifiers at a “IP-address specified” authority.]

- IP address defines the local access endpoint
- Use Local Access protocol “thttp” binding below

## 3.3 Phase 2: Local Access

Local Access is the process of asking an authoritative endpoint to do something with the identifier. Local Access protocols typically are instances of data access or directory lookup protocols.

After performing the Authority Resolution step, a resolving client will choose which of the Local Access protocols and endpoints it wishes to use for the Local Access phase of resolution. This decision will be based on several factors:

- The type of data the client is looking for. This is akin to query types in DNS. A client is assumed to be looking for a particular type of information about the identifier. The data type associated with an endpoint is sometimes available as part of the Local Access Protocol Descriptor as described in section 3.3.2.
- The protocol through which each endpoint can be accessed. Clients may only implement a subset of Local Access protocols, or have preferences for certain Local Access protocols. Clients SHOULD implement at least the protocols described in this document.
- The identity of the network endpoint. Clients may choose different network endpoints because they have other knowledge about those endpoints, such as previous failed attempts to access the endpoint, or the security features associated with a particular endpoint (ie HTTPS vs. HTTP).

```
Example of Choosing an Endpoint:
      1. A resolving client is given the option of accessing a Local
         Access endpoint using LDAP or HTTP. Because the client has not
         implemented LDAP, it chooses the HTTP endpoint
      2. A resolving client is given the option of accessing a Local
         Access endpoint that provides WSDL data and one that provides
         RDDL data. Because the client is interested in discovering
         SOAP messaging endpoints, it chooses the Local Access endpoint
         that provides WSDL.
```

### 3.3.1 Format of Local Access Descriptors

All Local Access endpoints must be described with sufficient detail to allow resolving clients to make the sort of decisions described above. To do that, all Local Access endpoints are described with Local Access Descriptors.

A **Local Access Descriptor** is a pair of data items as described in the following table:

| Data Item | Description | Example |
|---|---|---|
| Location | The network address where this endpoint can be reached. This is in the form of a URI | http://xri.example.com/I2R |
| Protocol Descriptor | The protocol to communicate with. This field optionally includes the type of data available at this endpoint. See the definition of “Local Access Protocol Descriptor” at section 3.3.2 | thttp+i2r/wsdl |

Table 9: Local Access Descriptor

### 3.3.2 Format of Local Access Protocol Descriptors

The format for a local access protocol descriptor follows that from RFC 3404, but with a modification to include the data type associated with the service (if needed).

```
service_field        = protocol *("+" rs)
protocol             = ALPHA *31ALPHANUM
rs                   = ALPHA *31ALPHANUM *("/" type)
type                 = ALPHA *31ALPHANUM
```

The protocol and type fields are limited to 32 characters. The protocol, rs, and type fields must start with an alphabetic character.

Note that the protocol element is always required for local access protocol descriptors. This specification does not enumerate legal data "types". Communities wishing to use XRI identifiers SHOULD enumerate which, if any, data "types" are legal for that community.

```
Examples (the type descriptors here are hypothetical and not yet
defined):

To describe a thttp-based service to return a wsdl document
corresponding to an XRI:

thttp+I2R/wsdl

To describe a thttp-based service to return a canonical XRI for a XRI:

thttp+I2I

To describe a ldap-based service to return a WSIL document
corresponding to an XRI:

ldap+I2R/wsil
```

### 3.3.3 Local Access Service Descriptors

Local access service descriptors are short strings that unambiguously identify the use of a network protocol as a XRI Local Access mechanism, including the selection of any options that the protocol may otherwise provide. This use of a protocol is called a "Local Access binding". Usually, there is one Local Access Service Descriptor for each Local Access binding (as described in section 3.3.4 below). Local Access bindings may define multiple Local Access Service Descriptors to provide different options on using the protocol described in the Local Access binding. For example, if a local access service provides an unauthenticated version and an authenticated version, there should be separate service descriptors for each. Local Access service descriptors make up part of a Local Access protocol descriptor, as described in section 3.3.2

### 3.3.4 Requirements for Local Access Bindings

Local Access bindings are required to unambiguously describe the use of a network protocol for local access. It is expected that most Local Access bindings will refer largely to underlying network protocols such as HTTP, LDAP, or SOAP. However, in all cases, there are aspects of using such an underlying network protocol that must be explicitly specified for use with XRIs.

Thus, an XRI local access binding specifies:

- The Local Access Service Descriptors that identify use of this binding. For example, the THTTP Local Access binding defines "thttp" to refer to the THTTP binding.

- The underlying network protocol used to access the Authority. Examples would be HTTP and LDAP.
- How the XRI is mapped to protocol-specific fields. For example, if using LDAP as the underlying network protocol, the binding must describe how is the XRI mapped to a LDAP query.
- What sorts of interaction are possible with the Authority using this binding. Some Local Access protocols will be "read-only", while others will be "read/write".
- The range of data types the Local Access protocol can accommodate. Some bindings may deal only with a certain type of data, but usually the type of data a particular binding supports is unlimited. For example, if a Local Access binding uses LDAP, then potentially any type of data can be accessed via that binding.

### 3.3.5 Local Access Bindings

This specification defines one useful Local Access binding using RFC2438. It is expected that other Local Access bindings will be defined in separate specifications.

#### 3.3.5.1 THTTP Local Access Binding

The functionality and content of the response of this local access protocol is defined in RFC 2438. Generally, this protocol defines a method for getting a variety of types of data corresponding to an URI. In its use here, it can be used to retrieve data about an XRI.

THTTP is based on a simple HTTP 1.1 GET request. THTTP defines the Local Access Service Descriptors “thttp” for use with HTTPS URLs and “thttps” for use with HTTPS URLs in this binding specification.

The actual GET request is quite simple, and is constructed from several pieces of data. All semantics of the GET are inherited from RFC 2169 and RFC 2438, except as described below.

The HTTP URL is constructed from the Location field of the Local Access Protocol Descriptor. The Local Access Protocol Descriptor field is also used in constructing the request URL. The Local Access Protocol Descriptor is broken up into “protocol”, and a series of “rs”, and “type” fields (as described in section 3.3.2). A single pair of rs and type fields are chosen which specify the THTTP service type and the data type the client wishes to perform.

```
thttp-url = location ?(”/”) rs “/”  ?(type “/”) url-encode(xri-local-
path)
```

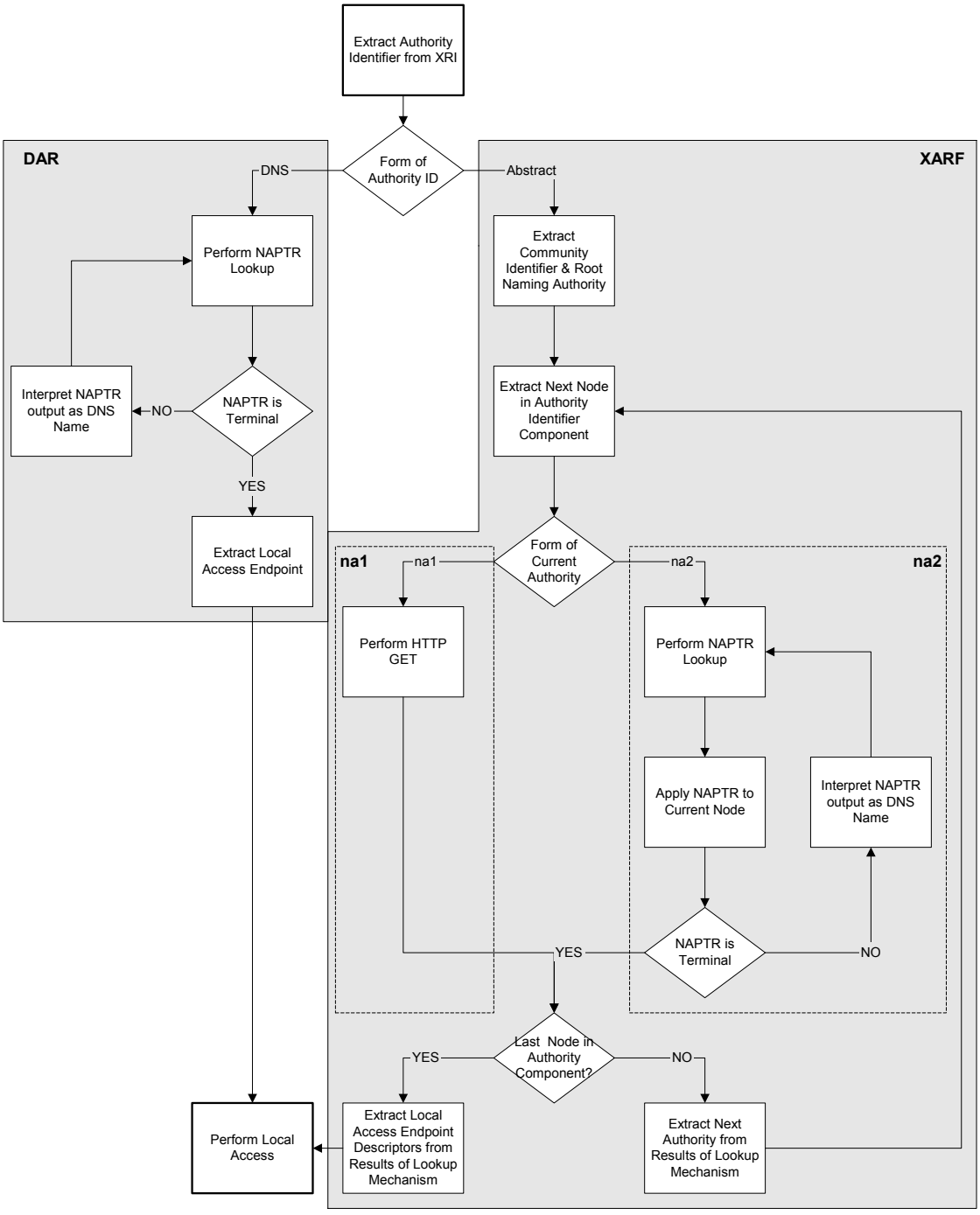
The separator “/” is inserted between the authority-location in the descriptor and the current node only if the authority location does not end in a “/”. The type field (and trailing slash) are inserted only if there is a type that corresponds to the selected rs in the Local Access Protocol Descriptor field. The local path node must be “URL-escaped” (reference) before being inserted into the request URL.

The result of the THTTP request is defined by RFC 2438 (RFC 2169?)

```
Example:

Suppose the local access phase is begun with a Local Access Descriptor
the xri “xri://a.b.c/1.2.3” containing the following fields:

Location: http://xriaccess.example.com
ProtocolDescriptor: thttpd+I2R/wsil+I2R/wsdl

Then a local access request for wsdl associated with the XRI using the
thttpd I2R service would be invoked by performing a HTTP GET to the
following URI:
```

```
http://xriaccess.example.com/I2R/wsdl/1.2.3

The result would be a WSDL document associated with xri://a.b.c/1.2.3
```

## 3.4 Flowchart of Authority Resolution



# 4 Security and Data Protection

## 4.1 XRI Usage in Legacy Infrastructure

Where XRIs are used within the legacy (pre-XRI) Internet and computing infrastructure, the security and data protection considerations relating to XRIs are similar to those of other URI schemes. In this context the material in section 7, *Security Considerations*, of **[RFC2396bis]** is informative. It include a discussion of the following topics:

- Reliability and Consistency
- Malicious Construction
- Rare IP Address Formats
- Sensitive Information
- Semantic Attacks

This material notes that "a URI does not in itself pose a direct security threat." This statement remains true only for the use of XRIs in legacy environments, and may not be accurate as new infrastructure evolves that takes full advantage of the extensibility of XRI architecture.

## 4.2 Secure Resolution

The resolution mechanisms described in section 3 are not intrinsically trustworthy. It is expected that, in practice, some combination of DNSSEC, SSL and other existing technologies will be employed to increase the security of the resolution process. Such considerations are outside the scope of this document, although follow-on work may be done to define best practices and facilitate inoperability.

## 4.3 XRI Usage in Evolving Infrastructure

As XRIs are adopted as abstract identifiers, it is anticipated that new services will be developed that take advantage of their extensibility. In particular, XRIs may enable new solutions to security and data protection problems that are not possible using existing URI schemes.

For example, XRI cross-reference syntax permits the inclusion of identifier metadata such as an encrypted or integrity-checked path, query, or fragment. Cross-references can also be used to indicate methods of obfusticating, proxying, or redirecting resolution to prevent the exposure of private or sensitive data. These capabilities may enable new security and data protection features at the fundamental level of resource identifiers.

A complete discussion of this topic is out of scope for this document. However, as a consequence of the extensibility of XRIs, it is not possible to make definitive statements regarding all security and data protection considerations relating to XRIs.

# 5 References

## 5.1 Normative


**[RFC2396]** T. Berners-Lee, R. Fielding, L. Masinter, *Uniform Resource Identifiers (URI): Generic Syntax,* http://www.ietf.org/rfc/rfc2396.txt, RFC 2396, August 1998.

**[XMLSchema2]** P. Biron, A. Malhotra, *XML Schema Part 2: Datatypes W3C Recommendation,* http://www.w3.org/TR/xmlschema-2/, May 2001.

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

**[XML]** T. Bray, J. Paoli, C.M. Sperberg-McQueen, E. Maler, *Extensible Markup Language (XML) 1.0 (Second Edition) W3C Recommendation*, http://www.w3.org/TR/REC-xml, October 2000.

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

**[UTR15]** M. Davis, M. Duerst, *Unicode Normalization Forms*, http://www.unicode.org/unicode/reports/tr15/tr15-23.html, April 17, 2003.

**[RFC3490]** P. Faltstrom, P. Hoffman, A. Costello, *Internationalizing Domain Names in Applications (IDNA)*, http://www.ietf.org/rfc/rfc3490, RFC 3490, March 2003.

**[RFC2732]** R. Hinden, B. Carpenter, L. Masinter, *Format for Literal IPv6 Addresses in URL's,* http://www.ietf.org/rfc/rfc2732.txt, RFC 2732, December, 1999.

**[RFC2718]** L. Masinter, H. Alvestrand, D. Zigmond, R. Petke, *Guidelines for New URL Schemes*, http://www.ietf.org/rfc/rfc2718.txt, RFC 2718, November 1999.

**[RFC3305]** M. Mealing, R. Denenberg, *Uniform Resource Identifiers (URIs), URLs, and Uniform Resource Names (URNs): Clarifications and Recommendations*, http://www.apps.ietf.org/rfc/rfc3305.html, RFC 3305, August 2002.

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

**[UML]** Object Management Group, *Unified Modeling Language (UML) Version 1.5*, http://www.omg.org/technology/documents/formal/uml.htm, March 1, 2003.

**[RFC1737]** K. Sollins, L. Masinter, *Functional Requirements for Uniform Resource Names*, http://www.ietf.org/rfc/rfc1737.txt, RFC 1737, December 1994.

**[Unicode]** The Unicode Consortium, *The Unicode Standard, Version v3.0*, Addison-Wesley Pub Co; ISBN: 0201616335, February, 2000.


## 5.2 Informative


**[IRI]** M. Duerst, M. Suignard, *Internationalized Resource Identifiers (IRIs)*, http://www.ietf.org/internet-drafts/draft-duerst-iri-04.txt, Work-In-Progress, June 2003.

**[RFC2396bis]** R. Fielding, *Uniform Resource Identifiers (URI): Generic Syntax*, Internet Draft draft-fielding-uri-rfc2396bis-03, http://www.apache.org/~fielding/uri/rev-2002/rfc2396bis.html, Work-In-Progress, June 2003.

**[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. Collected ABNF for XRI

This section contains the complete ABNF for XRI, which includes the complete ABNF for URI from **[RFC2396bis]** since XRI syntax is a superset. XRI productions use green shading and URI productions yellow shading. A valid XRI MUST conform to this ABNF.

```
abs-path        = "/"  path-segments

alphanum        = ALPHA / DIGIT

authority       = [ userinfo "@" ] host [ ":" port ]

authority-part  = URI-authority / XRI-authority

dec-octet       = DIGIT                                     ; 0-9
                / %x31-39 DIGIT                             ; 10-99
                / "1" 2DIGIT                                ; 100-199
                / "2" %x30-34 DIGIT                         ; 200-249
                / "25" %x30-35                              ; 250-255

delims          = "<" / ">" / "%" / DQUOTE

domainlabel     = alphanum [ 0*61( alphanum / "-" ) alphanum ]

escaped         = "%" HEXDIG HEXDIG

excluded        = invisible / delims / unwise

fragment        = *( pchar / "/" / "?" )

gcs-char        = "+" / "=" / "@" / "$" / "*"

global-path     = [ "!" ] authority-part [ local-path ]

global-xri      = global-path [ "?" xri-query ] [ "#" xri-fragment ]

h4              = 1*4HEXDIG

hier-part       = net-path / abs-path / rel-path

host            = [ hostname / IPv4address / IPv6reference ]

hostname        = idomainlabel qualified

idomainlabel    = 1*ucschar

invisible       = CTL / SP / %x80-FF

IPv4address     = dec-octet "." dec-octet "." dec-octet "." dec-octet

IPv6address =   6( h4 ":" ) ls32
                / "::" 5( h4 ":" )                          ls32
                / [              h4 ] "::" 4( h4 ":" )      ls32
                / [ *1( h4 ":" ) h4 ] "::" 3( h4 ":" )      ls32
                / [ *2( h4 ":" ) h4 ] "::" 2( h4 ":" )      ls32
                / [ *3( h4 ":" ) h4 ] "::"    h4 ":"        ls32
                / [ *4( h4 ":" ) h4 ] "::"                  ls32
                / [ *5( h4 ":" ) h4 ] "::"                  h4
```

```
                / [ *6( h4 ":" ) h4 ] "::"

IPv6reference   = "[" IPv6address "]"

local-path      = "/" relative-path

ls32            = ( h4 ":" h4 ) / IPv4address
                ; least-significant 32 bits of address

mark            = "-" / "_" / "." / "!" / "~" / "*" / "'" / "(" / ")"

net-path        = "//" authority [ abs-path ]

path-segments   = segment *( "/" segment )

pchar           = unreserved / escaped / ";" /
                ":" / "@" / "&" / "=" / "+" / "$" / ","

port            = *DIGIT

qualified       = *( "." idomainlabel ) [ "." ]

query           = *( pchar / "/" / "?" )

relative-path   = *( [ "." ] "./" ) xri-segments

rel-path        = path-segments

reserved        = "/" / "?" / "#" / "[" / "]" / ";" /
                ":" / "@" / "&" / "=" / "+" / "$" / ","

scheme          = ALPHA *( ALPHA / DIGIT / "+" / "-" / "." )

segment         = *pchar

sub-segment     = *xri-pchar / xref

ucschar         = %xA0-D7FF / %xF900-FDCF / %xFDF0-FFEF /
                %x10000-1FFFD / %x20000-2FFFD / %x30000-3FFFD /
                %x40000-4FFFD / %x50000-5FFFD / %x60000-6FFFD /
                %x70000-7FFFD / %x80000-8FFFD / %x90000-9FFFD /
                %xA0000-AFFFD / %xB0000-BFFFD / %xC0000-CFFFD /
                %xD0000-DFFFD / %xE1000-EFFFD

unreserved      = ALPHA / DIGIT / mark

unwise          = "{" / "}" / "|" / "\" / "^" / "`"

URI             = scheme ":" hier-part [ "?" query ] [ "#" fragment ]

URI-authority   = "//" [ userinfo "@" ] host [ ":" port ]

uric            = reserved / unreserved / escaped

userinfo        = *( unreserved / escaped / ";" /
                ":" / "&" / "=" / "+" / "$" / "," )

xref            = "(" ( global-xri / URI ) ")"

xref-authority  = xref  ( "." sub-segment / ":" sub-segment) *( "."
                sub-segment / ":" sub-segment)

XRI             = "xri:" xri-value
```

```
XRI-authority   = ( gcs-char xri-segment ) / xref-segment

xri-characters  = xri-reserved / xri-unreserved / escaped

xri-fragment    = [ xref ] * ( pchar / "/" / "?" )

xri-mark        = "-" / "_" / "~" / "'"

xri-path        = global-path / local-path / relative-path

xri-pchar       = xri-unreserved / escaped / ";" / "!" / "*"
                "@" / "&" / "=" / "+" / "$" / ","

xri-query       = [ xref ] * ( pchar / "/" / "?" )

xri-reserved    = "/" / "?" / "#" / "[" / "]" / "(" / ")" / ";" / ":" /
                "," / "." / "&" / "@" / "=" / "+" / "*" / "$" / "!"

xri-segment     = ( [ "." ] sub-segment / ":" sub-segment )
                *( "." sub-segment / ":" sub-segment )

xri-segments    = xri-segment *( "/" xri-segment )

xri-unreserved  = ALPHA / DIGIT / ucschar / xri-mark

xri-value       = [ xri-path ] [ "?" xri-query ] [ "#" xri-fragment ]
```

# Appendix B. Special Identifiers Assigned by the XRI Specification

As defined in Section 2.1.1.2.1, Global Context Symbols (GCS), the GCS character "$" is reserved for identifiers for which the XRI specification is the authority. The purpose of this special set is to define metadata that is specific to identifiers and the act of identification (resolution). Establishing these identifiers at the level of the XRI specification enables interoperability of this metadata among XRI implementations. Specifically this includes:

- *Human-readable metadata* that allows free text comments to be embedded in an XRI.
- *Versioning metadata* that identifies the syntax of a version identifier.
- *Linquistic metadata* that identifies the language or font of an internationalized identifier.
- *Internationalization encoding metadata* that identifies the level of encoding of an XRI. (See section [ref I18N section].
- *Query metadata* that identifies the syntax of a query string.

[DSR: I ran out of time to complete this section by turning the portion below into a 3-column table: Identifier, Identifer Purpose, Comments and Requirements. I also intend to prefix this with a set of requirements for the $ namespace as a whole, include terseness, the use of URI-legal chars, and the use of all-lowercase.]

| | |
|---|---|
| $! | = non-resolvable free text human comment |
| $v | = version (default is standard xri-segment syntax) |
| $v.d | = version in XML datetime format |
| $l | = language (when necessary for disambiguation of internationalized XRIs) |
| $f | = font (when necessary for disambiguation of internationalized XRIs) |
| $i | = internationalization encoding (when necessary for equivalence) |
| $q | = query (default is standard xri-segment syntax) |
| $q.xpath | = query in XPath syntax |

Note that like all authority segments, a slash delimits the end of the segment.

[DSR note: should also discuss the use of cross-references using "+" syntax for common names.]

# Appendix C. Transforming HTTP URIs to XRIs

[This section should discuss:
a) relationship of HTTP URIs and XRIs (e.g., answer the questions brought up on the list), and
b) specify the non-normative rules required to transform an HTTP URI into a legal XRI.]

# Appendix D. Acknowledgments

The following individuals were members of the committee during the development of this specification:

- Numerous people

In addition, the following people made contributions to this specification:

- Other people

# Appendix E. Revision History

*[This appendix should be removed for specifications that are at OASIS Standard level.]*

| Rev | Date | By Whom | What |
|---|---|---|---|
| wd-01 | 2003-06-24 | Drummond Reed | Initial version to review structure and Section 1 with other editors |
| wd-02 | 2003-06-25 | Drummond Reed | Reorganized overall structure and drafted first portion of Section 2 |
| wd-03 | 2003-06-30 | Drummond Reed | Reorganized level two headings; edited Section 1; drafted all ABNF portions of Section 2; added collected ABNF to Appendix A; added Appendix B with initial $ identifiers; added Appendix C. |
| wd-04 | 2003-07-02 | Dave McAlpin | Editorial changes; new text in 2.2.3.2, 2.4, 2.4.*, 2.5, 2.5.*, 4.* |
| wd-05 | 2003-07-03 | Dave McAlpin | Editorial changes; added resolution text (Section 3) |
| wd-06 | 2003-07-03 | Dave McAlpin | Minor edits; removed inline notes and created issues section as Appendix G. |
| wd-07 | 2003-07-24 | Dave McAlpin | Internationalization. Major revisions to 2.2 – 2.5. Harmonization of section 3. |

# Appendix F. 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 Executive Director.

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 Executive Director.

# Appendix G. Issues

| Issue | Section | Status |
|---|---|---|
| Need to remove this section. | Appendix G | Open |
| Need to add link to XRI Primer in Abstract when it exists | Abstract | Open |
| Need to add link to errata page in the Status section | Status | Open |
| Make sure internationalization text satisfies the HFI internationalization requirement or note that it's not supported in the current spec. | 1.2.3 | Addressed |
| Definition for concrete identifier is unclear. An HTTP URI resolves to an IP address, an IP address resolves to a MAC address, etc. Why aren't they abstract by this definition? | 1.3.3 | Addressed |
| Definition of abstract identifier may need to be revised to be consistent with clarified definition of concrete identifier. | 1.3.3 | Addressed |
| Definition of non-resolvable identifier raised this question, "In my mind, xri:!@IETF/rfc.2396 is non-resolvable not because there's no data and/or metadata about it but because it represents the abstract notion of the RFC rather than a particular digital representation of the text. Does this idea match the definition of non-resolvable identifier?" | 1.3.3 | Addressed |
| Need text for Character Encoding and Internationalization | 2.3 | Addressed |
| Relative resolution is broken by the ! (non-resolvable) symbol. Need to figure out how broken it is and how to fix. | 2.4 | Addressed |
| Need text for Internationalized XRI Equivalence | 2.5.3 | Addressed |
| Probably need to rename "THTTP Local Access Binding" from thttp to something XRI specific, since its not really RFC2169 compliant | 3.3.5.1 | Open |
| Need text for Privacy Considerations. | 4.3 | Addressed |
| References to RFC2277 and Unicode aren't used. If they aren't needed by internationalization text, they should be removed. | 5.1 | Addressed |
| Need to discuss the vocabulary of the $ namespace in appendix B. The list there is just the current candidates. They should be approved or removed from the spec. | Appendix B | Open |
| Appendix C "Transforming HTTP URIs to XRIs" needs text | Appendix C | Open |
| Appendix D "Acknowledgements" needs to be filled out with the current membership list. | Appendix D | Open |
| Resolution section needs thorough review | 3 | Open |
| Need to review BNF for completeness and correctness (i.e. need to prove the grammar) | Appendix A | Open |

| | | |
|---|---|---|
| Need a section similar to Appendix B of RFC2396 where we provide tools and guidance for parsing XRIs | None | Open |
| Security section should comment on lack of secure resolution. | 4 | Addressed |
| Hyperlinks in doc aren't all enabled. Need to make a pass through the doc and correct links to references and other sections of this document | All | Open |
| Need to review use of normative keywords ("MUST", "SHOULD", etc) for consistency and correctness. | All | Open |
| There's a possible terminology issue with section 4.3 "Privacy Considerations". In Europe, "data protection" is the code-word for "privacy". Since we already have the section title as Security and Data Protection, a separate section on Privacy Considerations appears redundant. | 4.3 | Addressed |
| Examples and tables don't have or have lost captions | 3 | Open |
| Clear up the intent of having multiple resolution mechanisms | 3 | Open |
| Mention the fact that resolution is currently only defined on URI-legal character strings, and confirm that this is a reasonable approach. | 3 | Open |
| Conversion of SRV records to Authority Descriptors needs fleshing out | 3 | Open |
| Use of IP addresses as Authority identifiers needs fleshing out | 3 | Open |
| Step 2 of section 2.2.3.2, combined with the last paragraph of that section implies that font and language tags are irrelevant for establishing equivalence. Are we ok with this? | 2.2.3.2 | Open |

1641