The XDI RDF Model

V12, 2009-01-28

This document is a work-in-progress from the OASIS XDI Technical Committee. It's purpose is to reflect the overall XDI RDF model that is formally defined in the XDI 1.0 specifications currently underway at the TC. Editors include:

- Drummond Reed, Cordance
- Markus Sabadello, XDI.org
- Paul Trevithick, Higgins Project

A link to the current version of this document is maintained on this XDI TC wiki page:

http://wiki.oasis-open.org/xdi/XdiRdfModel

Table of Contents

Introduction ... 3
Motivations ... 3
About the Proposed XRI 3.0 Syntax Used in this Document ... 4
Cross-References ... 4
Global Context Symbols ... 4
Compound XRIs ... 5
The XDI RDF Model ... 5
RDF Graph Structure and Addressing ... 5
Contexts, Context References, and Context Descriptors ... 6
Base Grammar – The XDI RDF Metagraph Predicates ... 7
Base Operations ... 8
The Type Dictionary ... 8
Variables ... 9
Typed Operations ... 9
Link Contracts ... 9
X3 Serialization Format ... 10
Standard X3 ... 10
X3 Whitespace ... 10
X3 Simple ... 11
Comments ... 11
Recommended Usage ... 11
Validation and Conversion ... 12
Example XDI RDF Documents ... 13
Single Subject ... 13
Multiple Subjects ... 14
Contexts and Subcontexts ... 15
Cross-References ... 16
Context Descriptors and Context References ... 17
Messages ... 22
Versioning ... 23
Version Logging ... 23
Version Snapshots ... 23
Subject Versioning Example ... 24
Predicate Versioning Example ... 31
Link Contracts ... 33
Dictionaries ... 36
Appendix A: Overview of the XDI RDF Metagraph Model ... 39
Appendix B: ABNF for Proposed XRI 3.0 Syntax ... 40
Appendix C: ABNF for X3 Standard ... 41
Base ... 41
Characters ... 41
Comment Characters ... 41
Typed Data Formats ... 42
Appendix D: X3 Whitespace Formatting Rules ... 42
Appendix E: X3 Simple Formatting Rules ... 43
Appendix F: XML Schema for XDI RDF ... 43
Appendix G: Example XML Instance Document ... 43
Appendix E: Revision History ... 44

Introduction

XDI (XRI Data Interchange) is an open standard structured data sharing format and protocol under development by the OASIS XDI Technical Committee. XDI is an application of XRI structured identifiers, specified by the OASIS XRI Technical Committee, to the problem of sharing, linking, and synchronizing data independent of any particular domain, application, or schema.

The XDI TC, which began its work in 2004, originally developed a data model called the ATI (Authority/Type/Instance) model. In early 2007 a new model was developed based on the RDF graph model from the W3C Semantic Web activity. It is proposed that this XDI RDF model become the basis for the XDI 1.0 specifications in conjunction with the OASIS XRI Technical Committee completing the third generation of the XRI specifications (XRI 3.0 and XRD 1.0).

This document provides an overview and technical definition of the XDI RDF model.

Motivations

The motivations for development of the XDI RDF data model are:

- *Simplicity*. While the XDI ATI model was relatively simple, consisting of only eight elements and one attribute in the XML serialization, the XDI RDF model is even simpler, using only five elements and one attribute in the XML serialization. It is so simple that the compact X3 serialization format can be expressed in just six lines of base ABNF.
- *Full fidelity with the RDF graph model.* Although the XDI ATI model was inspired by the core concepts of RDF, it did not strictly follow the RDF graph model. It permitted both data literals and data references to be associated with subject, predicate, and object nodes. The XDI RDF model follows the RDF graph model, so a data literal may only appear as an XDI RDF object.
- *Additional relation types*. The XDI ATI model included on two fundamental relation types—*refs* for equivalence and *links* for hierarchy. The XDI RDF model adds additional relation types and their inverses based on a simple metagraph model (see Appendix A).
- *Explicit XRI representation of all relations*. In the ATI model, refs and links were not fully representable as XRIs. In the XDI RDF model, all relations are expressed as XRIs, so every relation has its own explicit XDI address, and all XDI operations can be performed on relations just like on all other XDI RDF statements.
- *Simplied link contract structure.* The XDI RDF model simplifies and generalizes the model for XDI *link contracts*—XDI documents that express rights management for other XDI documents (or the resources they reference).
- *Simplified human understanding.* While the XDI RDF model can be graphed and serialized in conventional RDF formats, it can also be expressed in more compact and human-friendly formats, including a very simple notation called X3 (inspired by RDF N3 notation). See Appendix B.

About the Proposed XRI 3.0 Syntax Used in this Document

The XRI addressing used in the XDI RDF model is based on the proposed ABNF for XRI Syntax 3.0 included as Appendix A. The key new features of this syntax are described in this section.

Cross-References

The first key XRI feature used by XDI, called *cross-references*, has been part of XRI syntax since XRI 1.0. As a language for structured identifiers, XRI has always needed the ability to encapsulate and describe other identifiers from other identifier syntaxes and namespaces very much the same way XML can encapsulate and "tag" data from different native data sources. In XRI syntax, parentheses are used to syntactically encapsulate the cross-referenced identifier. This feature of XRI syntax is vital to XDI RDF because it enables any URI to be included in an XDI RDF statement. It also enables conventional RDF documents to be expressed and addressed in XDI RDF.

For example, following is an N3 relationship expressed using URIs:

```
<http://equalsdrummond.name> <http://dc.org/tag/author> <http://example.com/dsr.html>
```

Following is the same statement rendered as a single XRI using XDI RDF addressing syntax:

```
=(http://equalsdrummond.name)/+(http://dc.org/tag/author)/=(http://example.com/dsr.html)
```

Global Context Symbols

A second key feature of XRI syntax is symbols that represent abstract global contexts—shared root nodes for the XRI graph of identifiers. In XRI 2.0 there were five such symbols. The proposed ABNF for XRI 3.0 reduces that set to the four shown below. (In XRI 2.0, the "!" symbol was used as both a global context symbol and a local context symbol. In XRI 3.0 it will be deprecated as a global context symbol.)

GCS Char	Applies To	Description
$	Classes (dictionaries)	The self-context. $ is the root of the XRI dictionary specified by the OASIS XRI Technical Committee and OASIS XDI Technical Committee.
+	Classes (dictionaries)	The generic context—the root of identifiers that have no specified authority but evolve by shared consensus.
=	Instances (registries)	A personal context (i.e., the XDI authority is an individual person).
@	Instances (registries)	An organizational context (i.e., the XDI authority is a group or organization).

Compound XRIs

The third key feature of XRI 3.0 is the ability to directly concatenate two valid XRI 3.0 absolute XRIs (excluding a query or fragment) to form a third valid absolute XRI 3.0 XRI. The result, called a compound XRI, is central to the XDI RDF addressing algorithm.

For example, following are three different absolute XRIs representing an organization, a tag, and a person, respectively:

```
@example.company        +human.resources        =example.person.name
```

Using global cross-references, these three XRIs can be joined into a single compound XRI structured identifier.

```
@example.company+human.resources=example.person.name
```

In this structured identifier, each component XRI appears in the context of its predecessor. Like XML, such an identifier has a machine- and human-understandable structure that supports introspection, discovery, mapping, and other benefits not available from opaque identifiers.

The XDI RDF Model

RDF Graph Structure and Addressing

In the XDI RDF model, all data is structured and addressed using RDF statements encoded as composite XRIs (XRIs composed of other XRIs). These XRIs form paths in the XDI RDF directed graph, making the entire graph addressable.

The structure of the XDI RDF graph can thus be expressed in a very small set of ABNF statements defining an XDI address (these build on the *xri-segment* rule defined in the proposed ABNF for XRI Syntax 3.0 —see Appendix A):

```
xdi-address     = xdi-subject [ "/" xdi-predicate [ "/"  xdi-object ] ]
xdi-subject     = xri-segment
xdi-predicate   = xri-segment
xdi-object      = xri-segment
                / "/" [ xdi-address ]
```

Unlike conventional RDF, where statements consist only of subject/predicate/object triples, an XDI RDF statement can consist of four different types of addresses.

Address	Example	Addresses/Describes
Subject address	=drummond	The XDI subject node identified by this XRI.
Predicate address	=drummond/+friend	The set of XDI nodes which are the object of this predicate. This may be either: b) a data literal, b) a set of XDI subjects, or c) another XDI context.
Object address	=drummond/+friend/=markus	The XDI subject node identified by this XRI and described by this XDI RDF statement.
Context address	=drummond/+friend//	The XDI context (subgraph) identified by this XRI and described by this XDI RDF statement.

Note that while the structure of any single XDI address is a hierarchy (a path through a directed acyclic graph of nodes), the full XDI RDF graph is inherently a heterarchy, because any node may be referenced from any other node.

Contexts, Context References, and Context Descriptors

The XDI RDF addressing model supports addressing of *contexts*—subgraphs of the logical XDI RDF graph, each with its own unique XRI addressing space. This is analogous to the RDF concept of a *named graph*. One key difference is that XDI has the ability to address resources across XDI contexts, i.e., a single XDI address can span multiple contexts. Here's an example that spans three contexts:

```
=drummond/+friend//=markus/+friend//=paul.trevithick
```

This is roughly equivalent to the English statement, "Drummond has a friend Markus who has a friend Paul Trevithick". Note the double forward slashes that separate each context; they delimit the start of a new XRI addressing space.

Every XDI RDF document represents a context. If this context is accessible on the Web, it is available at at a service endpoint with at least one concrete URI (such as an http(s): URI) accessible via at least concrete interaction protocol (such as HTTPS or HTTPS). Like RDF, it is a core XDI design principle that any XDI subject may be identified and described in any number of XDI contexts. In each context the XDI subject may be addressable either absolutely or on relatively within that context or both. This is important from a privacy perspective:

- To *enable* correlation across contexts, one or more absolute XRIs for the subject may be shared across these contexts.
- To *prevent* correlation across contexts, one or more relative XRIs may be assigned to the subject in within each context and not shared across contexts.

Any combination of these two approaches may be used to fulfill the security and privacy requirements that may apply in each context.

An XDI statement that identifies another XDI context is called a *context reference*. Context references can be resolved by requesting a *context descriptor*—an XDI subject stored in one context that contains the set of metadata necessary to traverse to another XDI context. See the *Context Descriptors and Context References* section below for more.

Base Grammar – The XDI RDF Metagraph Predicates

The XDI RDF model uses four predicates defined in the XDI $ dictionary to express fundamental relations between XDI subjects (an XDI dictionary is a self-describing ontology expressed entirely in XRIs). Each of these has a corresponding inverse predicate that expresses the inverse RDF arc. The basis for these predicates is the very simple XDI RDF metagraph model described briefly in Appendix A.

Predicate	Inverse	Relation Type	Related Predicates in RDF/RDFS/OWL
$is	$is	Equivalence	`owl:equivalentClass`, `owl:sameAs`
$a	isa	Inheritance	`rdfs:subClassOf`, `rdf:type`
$has	ishas	Hierarchy	`rdfs:subClassOf`, `rdfs:range`
hasa	ishas$a	Property	`owl:ObjectProperty`, `rdfs:domain`

Following are examples of XRIs representing XDI RDF statements using these predicates:

Predicate	Statement	Inverse Statement
$is	+car/$is/+auto	+auto/$is/+car
	=drummond/$is/=drummond.reed	=drummond.reed/$is/=drummond
$a isa	+car/$a/+sedan	+sedan/isa/+car
	+person/$a/=drummond	=drummond/isa/+person
$has ishas	+apple/$has/+core	+core/ishas/+apple
	=drummond/$has/+work	+work/ishas/=drummond
hasa ishas$a	+car/hasa/+engine	+engine/ishas$a/+car
	=drummond/hasa/+hair+color	+hair+color/ishas$a/=drummond

The **$has** relation plays a special role in XDI addressing. Any two XDI subjects with a **$has** relationship can be concatenated into a compound XRI representing this relationship. If an XDI client has the appropriate access rights, it can perform XDI discovery by traversing **$has** relations between XRI subsegments using context descriptors. See the examples in *Context Descriptors and Context References* below.

Base Operations

Following the REST model, the XDI protocol supports four atomic operations on the XDI RDF graph itself and one abstract operation on XDI data described by the graph.

XDI RDF Graph Operation	CRUD Equivalent	Description
$get	read	Read one or more statements from the graph.
$add	create	Write one or more new statements to the graph.
$mod	update	Modify one or more existing statements in the graph.
$del	delete	Delete one or more existing statements from the graph.

Declaring XRIs for these explicit XDI protocol operations establishes the basis for permissioning in XDI link contracts (see the *Examples* section).

Another abstract XDI RDF operation, **$do**, serves as the root of an extensible dictionary of RPC-style operations. This topic will be covered in more detail in the XDI 1.0 specifications.

The Type Dictionary

To enable the XDI RDF graph to be entirely self-describing, the XDI RDF type definition dictionary is rooted in the inheritance predicate **$a.** A base **$a** dictionary will be specified by the XDI TC, however the **$a** dictionary is infinitely extensible by all XDI users.

Two main branches of the XDI TC-specified type dictionary have been proposed:

- **$a$mime**, which would encompass the IANA-specified MIME media types, and
- **$a$xsd**, which would encompass the W3C-specified XML Schema datatypes.

Each of these namespaces can then be further specialized using simple conventions for assigning http: URI fragments to XRIs. Following are some examples:

```
$a$mime$text$html
$a$mime$application$atom+xml
$a$xsd$string
$a$xsd$boolean
```

In addition, the **$a** dictionary can also be used to identify data serialization formats. This is particularly useful in X3 serialization syntax, which can carry data in other formats identified by **$a** tags. Two examples are **$a$json** for JSON format, and **$a$xml** for XML 1.0 format.

Variables

Operations on the XDI RDF graph often need to refer to nodes in the graph for which the client does not yet know an XRI. Such operations need a special XRI so the server can recognize the client is referring to a variable and not a literal XRI.

The XDI variable identifier **$$** is used for this purpose. Variables in any XDI document follow the same rule as all other XRIs in XDI RDF documents: they must be unique within their context. If more than one variable is needed in the same context, an XDI author can assign unique variable identifiers as needed. A convention is to use digits, e.g., **$$1**, **$$2**, **$$3**, however any unique XRI subsegment value may be used.

Typed Operations

In the Architecture of the World Wide Web (AWWW), a Web client can request different representations of a resource by specifying different media types in the HTTP Accept header. XDI offers the same capability by using global cross-references to type standard XDI operations. Examples:

Operation XRI	Description
geta$mime$text$html	Return the requested XDI resource as an HTML document.
getaxsdboolean	Returns a boolean ($true or $false) asserting whether or not the requested XDI resource exists.
adda$$	Add an XDI resource that contains one or more variables and return the variable assignments.

The **adda$$** typed operation is particular useful when an XDI client wishes to add a XDI resource to an XDI server but wants the server to assign the XRI to the new resource (especially assignment of a persistent i-number).

Link Contracts

One of the core design goals of XDI is for controls over XDI data sharing—everything from authorization and access control to data usage and termination—to be expressed in XDI itself so they can be viewed, shared, managed, and operated on just like any other part of the XDI RDF graph. This permits XDI rights management to be portable across XDI service providers, but it enables standard XDI operations to be used to create, manage, and share XDI permissions.

An XDI document used to express such data sharing controls is called a *link contract*. See the *Link Contract* section below.

X3 Serialization Format

XDI documents may be serialized in conventional XML using the XML schema in Appendix E. An example instance document is shown in Appendix F. However the very simple structure of the XDI RDF data model means XDI documents can be serialized in a more compact text format that does not require an XML parser. This format is called X3 because it was originally inspired by the N3 (Notation 3) format for RDF. There are three flavors of X3.

Standard X3

The ABNF for standard X3 format—the format actually transmitted across the wire—uses a very simple pattern that can be expressed (exclusive of character escaping) in six lines of ABNF:

```
   X3       = *( "[" sub *( "[" pred *( "[" obj "]" ) "]" ) "]" )
   sub      = [ comment ] xri-reference [ comment ]
   pred     = [ comment ] xri [ comment ]
   obj      = [ comment ] ( xri-reference / literal / X3 ) [ comment ]
   literal  = """ *char """
   comment  = "<--" *c-char "-->"
```

The full ABNF is included in Appendix B. The core concept is the use of nested square brackets to encode the XDI RDF subject/predicate/object relationships. Whitespace of any kind outside of quoted literals is ignored.

Following is a short example of a standard X3 document describing a single XDI subject. (Note: **=drummond** is a personal XRI registered by Drummond Reed, co-chair of the XDI TC. For privacy purposes, all data literals in the XDI examples in this document are ficticious.)

```
[=drummond[$is[=!f83.62b1.44f.2813]][$is$a[+person]][+person+name
["Drummond Reed"]][+email["drummond.example@cordance.net"]]]
```

X3 Whitespace

The lack of whitespace in standard X3 makes it hard for humans to read—similar to an XML document that does not include indenting. Appendix C defines a trivial transformation into X3 whitespace format that includes whitespace for readability. X3 Whitespace is still technically valid X3 because all whitespace is ignored. The rules for X3 Whitespace are included in Appendix C.

Following is the same X3 example above displayed in X3 Whitespace format.

```
[=drummond
   [$is
      [=!f83.62b1.44f.2813]
   ]
   [$is$a
      [+person]
   ]
```

```
   [+person+name
      ["Drummond Reed"]
   ]
   [+email
      ["drummond.example@cordance.net"]
   ]
]
```

X3 Simple

X3 Whitespace is relatively easy to read, but it is still harder than necessary to write because the author must keep track of nested square brackets. Appendix D defines another trivial transformation from X3 Whitespace into X3 Simple format, which eliminates square brackets altogether. This version is simple to read *and* write.

Following is the same example X3 document displayed in X3 Simple.

```
=drummond
   $is
      =!f83.62b1.44f.2813
   $is$a
      +person
   +person+name
      "Drummond Reed"
   +email
      "drummond.example@cordance.net"
```

Comments

Like XML, X3 supports human-readable comments. They are not normatively part of the XDI RDF graph, but they are are programmatically accessible in the graph model because exactly zero-or-one comment can prepended or postpend to an XDI subject, predicate, or object.

In X3 Simple the rule is that prepended comments appear on the line above the target, and postpended comments appear after the target on the same line, separated by a tab. Following is an example.

```
<-- prepended comment before a subject -->
=drummond   <-- postpended comment after a subject -->
   $is
      =!f83.62b1.44f.2813  <-- postpended comment after an object -->
   $is$a <-- postpended comment after a predicate -->
      +person
   <-- prepended comment before a predicate -->
   +person+name
      "Drummond Reed"
   +email
      <-- prepended comment before an object -->
      "drummond.example@cordance.net"
```

Recommended Usage

While all three forms of X3 are machine-readable, it is recommended to use:

- X3 Simple for all forms of human-readable X3 documents including most technical documentation.
- X3 Whitespace for highly technical XDI documentation and debugging where it is helpful to explicitly include all delimiters.
- Standard X3 for on-the-wire transmissions.

Validation and Conversion

Two online tools called *X3 Validator* and *X3 Converter* have been developed by XDI TC member Markus Sabadello for validating and converting between XDI RDF documents in all formats (X3, X3 Whitespace, X3 Simple, XDI/XML, XDI/JSON). They are publicly available at:

http://graceland.parityinc.net/xdi-validator/XDIValidator

http://graceland.parityinc.net/xdi-converter/XDIConverter

Example XDI RDF Documents

This section is a basic tutorial on the XDI RDF concepts in the previous sections using examples written in X3 Simple. All examples can be cut-and-pasted into the *XDI Validator* or *XDI Converter* (above) to validate them and convert them into other formats.

Single Subject

We'll start with the example from the previous section and expand it to contain more data typical of a standard business card (plus some identifiers and metadata that might make sense in an XDI business card context). All of this describes a single XDI subject identified by the XDI address **=drummond**.

```
=drummond
   $is
      =!f83.62b1.44f.2813
      =drummond.reed
      =(http://equalsdrummond.name)
   $is$a
      +person
   +person+name
      "Drummond Reed"
   +person+name+first
      "Drummond"
   +person+name+last
      "Reed"
   +email
      "drummond.example@cordance.net"
   +tel+office
      "+1.206.364.0992"
   +tel+mobile
      "+1.206.618.8530"
```

The first predicate, **$is**, asserts that **=drummond** is the same resource identified by two synonymous XRIs, **=!f83.62b1.44f.2813** (a persistent i-number) and **=drummond.reed** (another reassignable i-name), plus one http: URI "cast" as an XRI, **=(http://equalsdrummond.name)** (the address of Drummond's blog). The second predicate, **isa**, asserts that the subject is a subtype of the generic **+person** class.

All the other example predicates express attributes (XDI RDF predicates) of **=drummond** that have literal values (again, largely ficticious in these examples). These predicates are defined in the general XDI dictionary space denoted with the XRI global context symbol **+**. This dictionary is outside the scope of the XDI TC; it is intended to be an open community consensus-driven dictionary service in the same model as Wikipedia. The Identity Schemas Working Group at Identity Commons has begun work on a project called the Community Dictionary Service to implement this service.

Any specific XDI object can be requested from this XDI context using its XDI address of. For example, sending an XDI **$get** request to the XDI address…

```
   =drummond/+email
```

…would (assuming the requester has **$get** permission) return the following XDI document in response:

```
=drummond
   +email
      "drummond.example@cordance.net"
```

Multiple Subjects

The example above has only one XDI subject. An XDI document can contain any number of subjects, with or without linkages (**$has** relationships) between them. The next example shows how Drummond might creating multiple "personas" for **=drummond**, e.g., **=drummond+home**, **=drummond@cordance**, and **=drummond@(http://oasis-open.org),** each containing sets of attributes/values appropriate to specific contexts.

```
=drummond
   $has  <-- forms compound XRIs -->
      +home
      @cordance
      @(http://oasis-open.org)
   $is
      =!f83.62b1.44f.2813
      =drummond.reed
      =(http://equalsdrummond.name)
   $is$a
      +person
   +person+name
      "Drummond Reed"
   +person+name+first
      "Drummond"
   +person+name+last
      "Reed"
   +tel+mobile
      "+1.206.618.8530"
=drummond+home
   $is$a
      +person+home
   +tel
      "+1.206.362.5848"
   +tel+home+office
      "+1.206.364.0992"
   +email
      "drummond@example.com"
=drummond@cordance
   $is$a
      +person+work
      +director
   +email
      "drummond.example@cordance.net"
=drummond@(http://oasis-open.org)
   $is$a
      @oasis+member
      @oasis+technical+committee+chair
   +email
      "drummond.example@cordance.net"
```

Of the three new XDI subjects added to this example, one represents a generic context (**+home**), and two are specific organizational contexts (**@cordance** and **@(http://oasis-open.org**). The XDI addresses of each these subjects are formed via their **$has** relation to the subject **=drummond**. This relation can be queried just like any other predicate. For example, if an XDI **$get** request was sent to the following address…

```
   =drummond/$has
```

…it would (assuming the requester has **$get** permission) return the following XDI document in response:

```
=drummond
   $has
      +home
      @cordance
      @(http://oasis-open.org)
```

You can experiment with this yourself at the XDI Addresser application:

http://graceland.parityinc.net/xdi-addresser/XDIAddresser

This response informs the XDI client that more information is contained in the linked subjects **=drummond+home**, **=drummond@cordance**, and **=drummond@(http://oasis-open.org)**, all available in the same XDI context as **=drummond**.

Contexts and Subcontexts

Every XDI document represents its own its own XDI addressing context, called the *document context*. However the object of XDI statement can also be another XDI context, called a *subcontext*. One frequent use of subcontexts is XDI cross-references. For instance, the previous example included two instances of the same XDI object—the literal **"drummond.example@cordance.net"**. Following the best practice of not duplicating data, one of the instances should be a reference to the other. To do this, the second instance can contain a subcontext with a cross-reference to the first instance, as shown in bold below.

```
=drummond
   $has
      +home
      @cordance
      @(http://oasis-open.org)
   $is
      =!f83.62b1.44f.2813
      =drummond.reed
      =(http://equalsdrummond.name)
   $is$a
      +person
   +person+name
      "Drummond Reed"
   +person+name+first
      "Drummond"
   +person+name+last
```

```
      "Reed"
   +tel+mobile
      "+1.206.618.8530"
=drummond+home
   $is$a
      +person+home
   +tel
      "+1.206.362.5848"
   +tel+home+office
      "+1.206.364.0992"
   +email
      "dsr@example.org"
=drummond@cordance
   $is$a
      +person+work
      +director
   +email
      "drummond.example@cordance.net"
=drummond@(http://oasis-open.org)
   $is$a
      @oasis+member
      @oasis+technical+committee+chair
   +email
      /
         =drummond@cordance
            +email
```

Assuming the requester had **$get** permission, a XDI **$get** request for the following XDI address…

```
   =drummond@(http://oasis-open.org)/+email
```

…would return the following response:

```
=drummond@(http://oasis-open.org)
   +email
      /
         =drummond@cordance
            +email
=drummond@cordance
   +email
      "drummond.example@cordance.net"
```

The second XDI subject is included because an XDI service endpoint will resolve all cross-references within the same XDI context (similar to the way a web server returns all `<img>` tags within an HTML document, but does not resolve external links).

Cross-References

A reference across XDI contexts can also be made entirely within a single XDI context using XRI cross-reference syntax (i.e., the XRI being referenced is encapsulated in parentheses). For example, the reference in the final subject of the previous example could be also expressed as follows:

```
=drummond@(http://oasis-open.org)
   $is$a
      @oasis+member
      @oasis+technical+committee+chair
   +email
      (=drummond@cordance/+email)
```

This form is logically equivalent to the long form, i.e., if a requester had **$get** permission, a XDI **$get** request for the following XDI address…

```
   =drummond@(http://oasis-open.org)/+email
```

…would return the following response:

```
=drummond@(http://oasis-open.org)
   +email
      (=drummond@cordance/+email)
=drummond@cordance
   +email
      "drummond.example@cordance.net"
```

Context Descriptors and Context References

The XRI global context symbol **$** is reserved for context descriptors—an XDI subject that is a self-description of the containing context. Every XDI context can have zero-or-one context descriptor. A context descriptor can describe literal attributes of the context in which it appears, such the datetime it was first created (**$d$first**) or last modified (**$d$last**), or its http(s): URIs (**urihttp** or **urihttps**). It can also describe its context type (**isa**) using one or more XRIs rooted in the **$$** dictionary space which is assigned to contexts. And it can have backwards references to other XDI contexts that reference it, called its *supercontexts*.

The following shows a context descriptor added to our example XDI document.

```
$
   $is$a
      $$xdi
   $is$$xdi
      =drummond
      =!f83.62b1.44f.2813
   $d$first
      "2008-01-14T12:13:14Z"
   $d$last
      "2008-01-28T09:10:11Z"
   $uri$http$1 <-- first instance of this predicate -->
      "http://xdi.example.com/=!f83.62b1.44f.2813"
   $uri$http$2 <-- second instance of this predicate -->
      "http://xdi-backup.example.com/=!f83.62b1.44f.2813"
   $uri$https
      "https://xdi-secure.example.com/=!f83.62b1.44f.2813"
=drummond
   $has
      +home
      @cordance
      @(http://oasis-open.org)
   $is
```

```
      =!f83.62b1.44f.2813
      =drummond.reed
      =(http://equalsdrummond.name)
   $is$a
      +person
   +person+name
      "Drummond Reed"
   +person+name+first
      "Drummond"
   +person+name+last
      "Reed"
   +tel+mobile
      "+1.206.618.8530"
=drummond+home
   $is$a
      +person+home
   +tel
      "+1.206.362.5848"
   +tel+home+office
      "+1.206.364.0992"
   +email
      "drummond@example.com"
=drummond@cordance
   $is$a
      +person+work
      +director
   +email
      "drummond.example@cordance.net"
=drummond@(http://oasis-open.org)
   $is$a
      @oasis+member
      @oasis+technical+committee+chair
   +email
      (=drummond@cordance/+email)
```

Note the pattern used to include multiple instances of the **urihttp** predicate. The appending of **$1** and **$2** reflects the following implicit XDI statement about this attribute:

```
$uri$http
   $has
      $1
      $2
```

Although helpful for introspection, having a context descriptor in the XDI document for **=drummond** doesn't help someone trying to locate this XDI document—this is the very information the requester needs to discover. Therefore, copies of this context descriptor need to appear in other XDI contexts that reference this context.

For example, if **=drummond** has a friend **=web*markus** who knows **=drummond**'s XDI context, then **=web*markus** might have the following copy of **=drummond**'s context descriptor in his XDI document:

```
$
   $is$a
      $$xdi
   $is$$xdi
      =web*markus
      =!91f2.8153.f600.ae24
   $d$first
      "2007-09-10T12:13:14Z"
   $d$last
      "2008-01-27T07:08:09Z"
   $uri$http
      "http://freexri.com/xdi/user/=!91f2.8153.f600.ae24"
   $uri$https
      "https://freexri.com/xdi/user/=!91f2.8153.f600.ae24"
=drummond
   $$xdi
      /
         $
            $is$a
               $$xdi
            $is$$xdi
               =drummond
               =!f83.62b1.44f.2813
            $uri$http$1
               "http://xdi.example.com/=!f83.62b1.44f.2813"
            $uri$http$2
               "http://xdi-backup.example.com/=!f83.62b1.44f.2813"
            $uri$https
               "https://xdi-secure.example.com/=!f83.62b1.44f.2813"
```

Note that the subgraph with the address **=drummond/$$xdi//$** is an exact copy of the XDI RDF graph that has the address **$** in the previous example. It has just been replicated in a different context.

(Note: For readers familiar with the XRDS discovery architecture in XRI Resolution 2.0, the **$$xdi** predicate on **=drummond** above is the XDI equivalent of an XRDS *service endpoint*. The **isa** predicate in the context descriptor is the equivalent of the **<xrd:Type>** element, and the **urihttp** and **urihttps** predicates are the equivalent of the **<xrd:URI>** element.)

Assuming a requester knows the XDI service endpoint URI for of **=web*markus** and has **$get** permission, the requester can perform XDI discovery by doing an XDI **$get** for…

```
   =drummond/$$xdi
```

…and receive the following response:

```
=drummond
   $$xdi
      /
         $
            $is$a
               $$xdi
            $is$$xdi
               =drummond
               =!f83.62b1.44f.2813
            $uri$http$1
               "http://xdi.example.com/=!f83.62b1.44f.2813"
            $uri$http$2
               "http://xdi-backup.example.com/=!f83.62b1.44f.2813"
            $uri$https
               "https://xdi-secure.example.com/=!f83.62b1.44f.2813"
```

But what if the requester doesn't know anyone who knows **=drummond**? This is where we can leverage the fact that every XDI address is itself XDI RDF statement. For example, the XDI address **=drummond** asserts the following XDI statement:

```
=
   $has
      drummond
```

= is the XRI global context registry, whose job it is to provide public XRD, XRDS, and XDI discovery services for registrants. This registry record for **=drummond** can include context descriptors for the different XDI contexts in which **=drummond** wants to be publicly discoverable. For example, the portion of the = registry pertaining to the registration of =drummond might look like:

```
=
   $has
      drummond
      !F83.62B1.44F.2813
=drummond
   $d$first
      "2007-09-10T12:13:14Z"
   $d$last
      "2008-01-27T07:08:09Z"
   $is
      =!f83.62b1.44f.2813
   $$xdi
      /
         $
            $is$a
               $$xdi
            $is$$xdi
               =drummond
               =!f83.62b1.44f.2813
            $uri$http$1
               "http://xdi.example.com/=!f83.62b1.44f.2813"
            $uri$http$2
               "http://xdi-backup.example.com/=!f83.62b1.44f.2813"
            $uri$https
               "https://xdi-secure.example.com/=!f83.62b1.44f.2813"
   $$openid
      /
         $
            $is$a
               $$openid$v$2
               $$(http://specs.openid.net/auth/2.0/signon)
            $is$$openid
               =drummond
               =!f83.62b1.44f.2813
            $uri$http
               "http://openid.example.com/=!f83.62b1.44f.2813"
            $uri$https
               "https://xdi-secure.example.com/=!f83.62b1.44f.2813"
```

A requester could send the following XDI **$get** request to the XDI service endpoint for the = registry…

```
   =drummond
```

….to receive all of =drummond's publicly available discovery information. Or a requester could qualify the request to receive only discovery information about a specific context type, e.g.:

```
   =drummond/$$openid
```

Messages

Because XDI is a globally addressable RDF graph ("giant global graph"), XDI messages are themselves XDI documents within that graph.

The basic pattern of an XDI message is:

- The sender of the message is the XDI subject,
- One or more XDI operations are predicates, and
- The object of each predicate is a subcontext expressing the portion of the XDI RDF graph being acted upon by the operation.

For example, if **=web*markus** wants to request a telephone number and email address from **=drummond**, he could send the following XDI message to the XDI service endpoint for **=drummond**:

```
=markus
   $d
      "2008-01-01T12:13:14Z"  <-- datestamp of message -->
   $get
      /
         =drummond
            +tel
            +email
```

This same pattern applies for all XDI operations. For instance, **=drummond** could send the following XDI message to his own XDI service endpoint to add two attributes and attribute values to his own XDI document:

```
=drummond
   $d
      "2008-01-01T12:13:14Z"
   $add
      /
         =drummond
            +person+name
               "Drummond Reed"
            +email
               "drummond.example@cordance.net"
```

Multiple operations can be requested in the same XDI message—each will be performed sequentially in document order. (Transactional integrity is currently an open issue under discussion in the XDI TC.) For example, **=drummond** could both add and modify two attributes in his own XDI document:

```
=drummond
   $d
      "2008-01-01T12:13:14Z"
   $add
      /
         =drummond
            +tel+mobile
               "+1.206.618.8530"
   $mod
      /
         =drummond
            +email
               "dsr@example.org"
```

XDI TC member Markus Sabadello has developed a basic XDI message validation and testing service called XDI Messenger. It is publicly available at:

http://graceland.parityinc.net/xdi-messenger/XDIMessenger

Versioning

To support data synchronization between XDI service endpoints, XDI must be able to express versions of any portion of the XDI RDF graph. Since the versioning pattern exercises all the patterns discussed above, this section will provide a very detailed example.

In many ways, XDI versioning is very similar to how wiki servers maintains diffs on a wiki page over time, or how a file system tracks changes using a journaling file system or change logs. Versioning can be applied by an XDI service endpoint at either:

- The document level (all XDI subjects in that document).
- The subject level (specific XDI subjects in the document).
- The predicate level (specific predicates in an XDI subject).

There are two versioning policies that can be applied at each of these points.

Version Logging

With this policy, the XDI service simply maintains a log (in XDI) of the changes made with each XDI message. Because XDI version logs are themselves XDI documents, they can be addressed, shared, linked, and synchronized just like any other XDI document.

Version Snapshots

A version “snapshot” is like a backup copy of a specific version, with its own XDI address. This way any XDI subscriber can request a copy of a specific version from the publisher without having to store the copy themselves.

Note that every XDI service endpoint that uses version logging can provide “virtual” version snapshots, because any version snapshot can be completely reconstructed from the version log. However storing actual version snapshots can provide faster performance if they are frequently requested.

Subject Versioning Example

This first example will start with a blank XDI document, and then step by step, perform five operations on the document and show the results. For simplicity, this example will only show versioning at the subject level. Predicate versioning will be shown in the second example.

Each operation is shown in the form of the XDI message received by the XDI service endpoint (exclusive of any security tokens or signatures used to authenticate the requester).

Note that for readability, the subject XRI ***=drummond*** *is a reassignable i-name, however in practice most versioned subjects will be identified with an immutable identifier such as an XRI i-number because version references need to be persistent.*

Operation #1: The following message will create a new XDI subject (version #1) with the following predicates and literal values.

```
=drummond
   $d
      "2008-01-01T12:13:14Z"
   $add
      /
         =drummond
            +person+name
               "Drummond Reed"
            +email
               "drummond.example@cordance.net"
```

Operation #2: Change **=drummond**'s e-mail address (creates version #2).

```
=drummond
   $d
      "2008-01-02T12:13:14Z"
   $mod
      /
         =drummond
            +email
               "dsr@example.org"
```

Operation #3: Add a i-number synonym (creates version #3):

```
=drummond
   $d
      "2008-01-03T07:22:59Z"
   $add
      /
         =drummond
            $is
               =!f83.62b1.44f.2813
```

Operation #4: Change **=drummond**'s e-mail address again (creates version #4):

```
=drummond
   $d
      "2008-01-04T11:28:11Z"
   $mod
      /
         =drummond
            +email
               "drummond.reed@another.example.net"
```

Operation #5: Delete **=drummond**'s name (creates version #5).

```
=drummond
   $d
      "2008-01-05T03:34:52Z"
   $del
      /
         =drummond
            +person+name
```

Version #1

Following is the complete XDI document created with operation #1.

```
=drummond
   $has
      $v <-- this subject is versioned -->
   $v
      $1 <-- current version -->
   +person+name
      "Drummond Reed"
   +email
      "drummond.example@cordance.net"
=drummond$v <-- subject exists only if versioning is on -->
   $has
      $1 <-- version snapshots is on -->
   $has$a
      =drummond$v$1  <-- version logging is on -- predicates below this
line only exist if version logging is on -->
   =drummond$v$1
      /  <-- this is all a copy of the operation performed -->
         =drummond
            $d
               "2008-01-01T12:13:14Z"
            $add
               /
                  =drummond
                     +person+name
                        "Drummond Reed"
                     +email
                        "drummond.example@cordance.net"
<-- subjects below this line only exist if version snapshots is turned
on -->
=drummond$v$1
   +person+name
      "Drummond Reed"
```

```
   +email
      "drummond.example@cordance.net"
```

Version #2

```
=drummond
   $has
      $v
   $v
      $2
   +person+name
      "Drummond Reed"
   +email
      "dsr@example.org"
=drummond$v
   $has
      $2
   $has$a
      =drummond$v$2
   =drummond$v$1
      /
         =drummond
            $d
               "2008-01-01T12:13:14Z"
            $add
               /
                  =drummond
                     +person+name
                        "Drummond Reed"
                     +email
                        "drummond.example@cordance.net"
   =drummond$v$2
      /
         =drummond
            $d
               "2008-01-02T07:22:59Z"
            $mod
               /
                  =drummond
                     +email
                        "dsr@example.org"
=drummond$v$1
   +person+name
      "Drummond Reed"
   +email
      "drummond.example@cordance.net"
=drummond$v$2
   +person+name
      "Drummond Reed"
   +email
      "dsr@example.org"
```

Version #3

```
=drummond
   $has
      $v
   $v
      $3
   +person+name
      "Drummond Reed"
   +email
      "dsr@example.org"
   $is
      =!f83.62b1.44f.2813
=drummond$v
   $has
      $3
   $has$a
      =drummond$v$3
   =drummond$v$1
      /
         =drummond
            $d
               "2008-01-01T12:13:14Z"
            $add
               /
                  =drummond
                     +person+name
                        "Drummond Reed"
                     +email
                        "drummond.example@cordance.net"
   =drummond$v$2
      /
         =drummond
            $d
               "2008-01-02T07:22:59Z"
            $mod
               /
                  =drummond
                     +email
                        "dsr@example.org"
   =drummond$v$3
      /
         =drummond
            $d
               "2008-01-03T11:28:11Z"
            $add
               /
                  =drummond
                     $is
                        =!f83.62b1.44f.2813
=drummond$v$1
   +person+name
      "Drummond Reed"
   +email
      "drummond.example@cordance.net"
=drummond$v$2
   +person+name
      "Drummond Reed"
   +email
```

```
      "dsr@example.org"
=drummond$v$3
   +person+name
      "Drummond Reed"
   +email
      "dsr@example.org"
   $is
      =!f83.62b1.44f.2813
```

Version #4

```
=drummond
   $has
      $v
   $v
      $4
   +person+name
      "Drummond Reed"
   +email
      "drummond.reed@another.example.net"
   $is
      =!f83.62b1.44f.2813
=drummond$v
   $has
      $4
   $has$a
      =drummond$v$4
   =drummond$v$1
      /
         =drummond
            $d
               "2008-01-01T12:13:14Z"
            $add
               /
                  =drummond
                     +person+name
                        "Drummond Reed"
                     +email
                        "drummond.example@cordance.net"
   =drummond$v$2
      /
         =drummond
            $d
               "2008-01-02T07:22:59Z"
            $mod
               /
                  =drummond
                     +email
                        "dsr@example.org"
   =drummond$v$3
      /
         =drummond
            $d
               "2008-01-03T11:28:11Z"
            $add
```

```
               /
                  =drummond
                     $is
                        =!f83.62b1.44f.2813
   =drummond$v$4
      /
         =drummond
            $d
               "2008-01-04T02:30:41Z"
            $mod
               /
                  =drummond
                     +email
                        "drummond.reed@another.example.net"
=drummond$v$1
   +person+name
      "Drummond Reed"
   +email
      "drummond.example@cordance.net"
=drummond$v$2
   +person+name
      "Drummond Reed"
   +email
      "dsr@example.org"
=drummond$v$3
   +person+name
      "Drummond Reed"
   +email
      "dsr@example.org"
   $is
      =!f83.62b1.44f.2813
=drummond$v$4
   +person+name
      "Drummond Reed"
   +email
      "drummond.reed@another.example.net"
   $is
      =!f83.62b1.44f.2813
```

Version #5

```
=drummond
   $has
      $v
   $v
      $5
   +email
      "drummond.reed@another.example.net"
   $is
      =!f83.62b1.44f.2813
=drummond$v
   $has
      $5
   $has$a
      =drummond$v$5
```

```
   =drummond$v$1
      /
         =drummond
            $d
               "2008-01-01T12:13:14Z"
            $add
               /
                  =drummond
                     +person+name
                        "Drummond Reed"
                     +email
                        "drummond.example@cordance.net"
   =drummond$v$2
      /
         =drummond
            $d
               "2008-01-02T07:22:59Z"
            $mod
               /
                  =drummond
                     +email
                        "dsr@example.org"
   =drummond$v$3
      /
         =drummond
            $d
               "2008-01-03T11:28:11Z"
            $add
               /
                  =drummond
                     $is
                        =!f83.62b1.44f.2813
   =drummond$v$4
      /
         =drummond
            $d
               "2008-01-04T02:30:41Z"
            $mod
               /
                  =drummond
                     +email
                        "drummond.reed@another.example.net"
   =drummond$v$5
      /
         =drummond
            $d
               "2008-01-05T03:34:52Z"
            $del
               /
                  =drummond
                     +person+name
=drummond$v$1
   +person+name
      "Drummond Reed"
   +email
      "drummond.example@cordance.net"
=drummond$v$2
```

```
   +person+name
      "Drummond Reed"
   +email
      "dsr@example.org"
=drummond$v$3
   +person+name
      "Drummond Reed"
   +email
      "dsr@example.org"
   $is
      =!f83.62b1.44f.2813
=drummond$v$4
   +person+name
      "Drummond Reed"
   +email
      "drummond.reed@another.example.net"
   $is
      =!f83.62b1.44f.2813
=drummond$v$5
   $is
      =!f83.62b1.44f.2813
   +email
      "drummond.reed@another.example.net"
```

Predicate Versioning Example

Versioning may also be applied at the predicate level. Following is an example of what the final XDI document would look like (after all 5 operations) if the **+email** predicate had predicate versioning turned on when it was first added to the document. Turning on predicate versioning can be accomplished on a per-predicate basis by adding a "versioning predicate" as shown below.

Note that the XDI author requesting the change does need to have any knowledge of whether predicate versioning is turned on for a predicate – it is handled automatically by an XDI service endpoint on the basis of the presence or absence of the versioning predicate.

```
=drummond
   $has
      $v
   $v
      $5
   +email
      "drummond.reed@another.example.net"
   +email$v <-- this predicate is versioned -->
      $3 <-- the current version -->
   $is
      =!f83.62b1.44f.2813
=drummond$v
   $has
      $5
   $has$a
      =drummond$v$5
   =drummond$v$1
```

```
      /
         =drummond
            $d
               "2008-01-01T12:13:14Z"
            $add
               /
                  =drummond
                     +person+name
                        "Drummond Reed"
                     +email
                        "drummond.example@cordance.net"
                     +email$v <-- predicate versioning is turned on -->
   =drummond$v$2
      /
         =drummond
            $d
               "2008-01-02T07:22:59Z"
            $mod
               /
                  =drummond
                     +email
                        "dsr@example.org"
   =drummond$v$3
      /
         =drummond
            $d
               "2008-01-03T11:28:11Z"
            $add
               /
                  =drummond
                     $is
                        =!f83.62b1.44f.2813
   =drummond$v$4
      /
         =drummond
            $d
               "2008-01-04T02:30:41Z"
            $mod
               /
                  =drummond
                     +email
                        "drummond.reed@another.example.net"
   =drummond$v$5
      /
         =drummond
            $d
               "2008-01-05T03:34:52Z"
            $del
               /
                  =drummond
                     +person+name
=drummond$v$1
   +person+name
      "Drummond Reed"
   +email
      "drummond.example@cordance.net"
   +email$v
```

```
      $1 <-- +email version number -->
=drummond$v$2
   +person+name
      "Drummond Reed"
   +email
      "dsr@example.org"
   +email$v
      $2 <-- +email version number -->
=drummond$v$3
   +person+name
      "Drummond Reed"
   +email
      "dsr@example.org"
   $is
      =!f83.62b1.44f.2813
=drummond$v$4
   +person+name
      "Drummond Reed"
   +email
      "drummond.reed@another.example.net"
   +email$v
      $3 <-- +email version number -->
   $is
      =!f83.62b1.44f.2813
=drummond$v$5
   +person+name
      "Drummond Reed"
   $is
      =!f83.62b1.44f.2813
```

Link Contracts

Many of the preceeding examples of XDI requests included the caveat "*if the requester has permission*". A key design goal of XDI is to store these permissions—and any other controls over sharing of XDI data—within the XDI graph itself as a link contract.

The term "link contract" derives from the fact that this XDI graph describe the terms of a data sharing relationship—a link—between two or more XDI subjects just the way a real world contract expresses the terms of an agreement between two or more parties.

Link contracts are in active development at the XDI TC, however the following example illustrates the basic patterns that:

- A link contract is itself an XDI subject that defines the data sharing terms.
- This XDI subject describes the permissions granted under that contract using the same XDI statements used to actually operate on the resources.
- The authority for the contract grants the permissions by adding XDI subjects to the contract.
- If needed, those XDI subjects agree to the contract by adding an XDI signature.

Following is a simple example. **=drummond** creates a link contract, **=drummond+friend$contract**, for friends that will give them permission to access all

his **+home** information plus his **@cordance** email address. This contract itself has the separate XDI “signature block” **=drummond+friend$contract$sig**.

```
$
   $is$a
      $$xdi
   $is$$xdi
      =drummond
      =!f83.62b1.44f.2813
=drummond
   $has
      +home
      @cordance
      +friend
   $is
      =!f83.62b1.44f.2813
      =drummond.reed
      =(http://equalsdrummond.name)
   $is$a
      +person
   +person+name
      "Drummond Reed"
   +person+name+first
      "Drummond"
   +person+name+last
      "Reed"
   +tel+mobile
      "+1.206.618.8530"
=drummond+home
   $is$a
      +person+home
   +tel
      "+1.206.362.5848"
   +tel+home+office
      "+1.206.364.0992"
   +email
      "drummond@example.com"
=drummond@cordance
   $is$a
      +person+work
      +director
   +email
      "drummond.example@cordance.net"
=drummond+friend
   $has
      $contract
=drummond+friend$contract
   $is$a
      $contract
   $get
      /
         =drummond+home <-- permission to get all +home data -->
         =drummond@cordance
            +email   <-- permission to get @cordance email -->
=drummond+friend$contract$sig
   $is$a
      $contract$sig
```

Next **=drummond** adds his friend **=web*markus** to his XDI document and makes him an object of this contract.

```
...
=drummond+friend$contract
   $is$a
      $contract
   $get
      /
         =drummond+home <-- permission to get all +home data -->
         =drummond@cordance
            +email   <-- permission to get @cordance email -->
=drummond+friend$contract$sig
   $is$a
      $contract$sig
   $has
      =web*markus <-- Drummond adds Markus to this contract -->
=web*markus
   $is
      =!91f2.8153.f600.ae24
   $is$has
      =drummond+friend$contract$sig
```

The final step (which seems overly formal for personal friend relationships but can be carried out automatically by XDI clients/servers) is for **=web*markus** to accept the contract by adding his own signature.

```
...
=drummond+friend$contract
   $is$a
      $contract
   $get
      /
         =drummond+home
         =drummond@cordance
            +email
=drummond+friend$contract$sig
   $is$a
      $contract$sig
   $has
      =web*markus
=web*markus
   $is
      =!91f2.8153.f600.ae24
   $is$has
      =drummond+friend$contract$sig
=drummond+friend$contract$sig=web*markus
   $is$a
      $sig
   $d
      "2007-01-12T12:13:14Z"
   $sig$key$private$rsa$2048  <-- signature key type -->
```

```
      "...RSA-2048-signature-value..."
```

This fundamental model of using a graph to control operations on a graph enables link contracts to express permissions over any XDI operation that can be performed by any XDI subject on any XDI data.

Dictionaries

XML has schemas, RDF has ontologies, XDI has dictionaries. A dictionary is essentially an ontology written in the same language it is describing, just like a real world dictionary such as *The Oxford English Dictionary* is written in English.

XDI dictionaries are another area of active development at the XDI TC, however again their basic structure simply derives directly from the four base predicates in the XDI RDF metagraph model. Following is example of a simple XDI dictionary for personal contact and relationship data.

```
+
   $has
      person
+person
   $has
      +name
      +home
      +work
   $has$a
      $uri
      +friend
      +spouse
      +person+name
      +email
      +tel
+name
   $is$a
      $a$xsd$string
   $is$has
      +person
+home
   $is$a
      $$ <-- context -->
   $is$has
      +person
+work
   $is$a
      $$ <-- context -->
   $is$has
      +person
+friend
   $is$a
      +person
   $is$has$a
      +person
+spouse
   $is$a
      +person
```

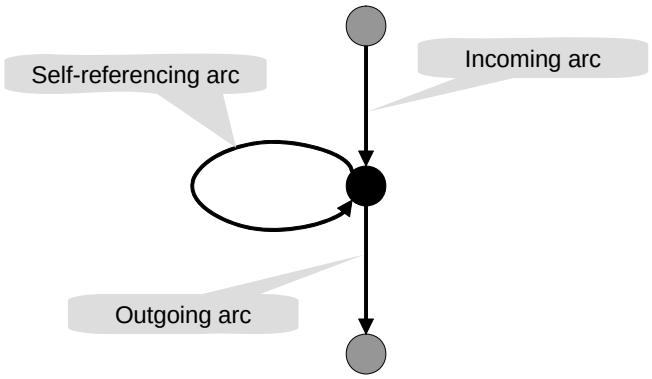
```
   $is$has$a
      +person
+person+name
   $is$a
      +name
   $is$has$a
      +person
   $has
      +first
      +last
      +legal
      +preferred
      +middle
      +nickname
+email
   $is$a
      $a$xsd$string
   $is$has$a
      +person
   $has
      +home
      +work
      +primary
      +alt
      +preferred
+tel
   $is$a
      $a$xsd$string
   $is$has$a
      +person
   $has
      +country.code
      +area.code
      +number
      +extension
   $has$a
      +home
      +work
      +primary
      +alt
      +preferred
tel+country.code
   $is$a
      $a$xsd$short
      $a$xsd$enumeration
   $1
      "+1"
   $2
      "+20"
   $3
      "+30"
   ...
tel+area.code
   $is$a
      $a$xsd$short
tel+number
      $is$a
```

```
   $a$xsd$postiveinteger
tel+extension
   $is$a
      $a$xsd$short
```

Appendix A: Overview of the XDI RDF Metagraph Model

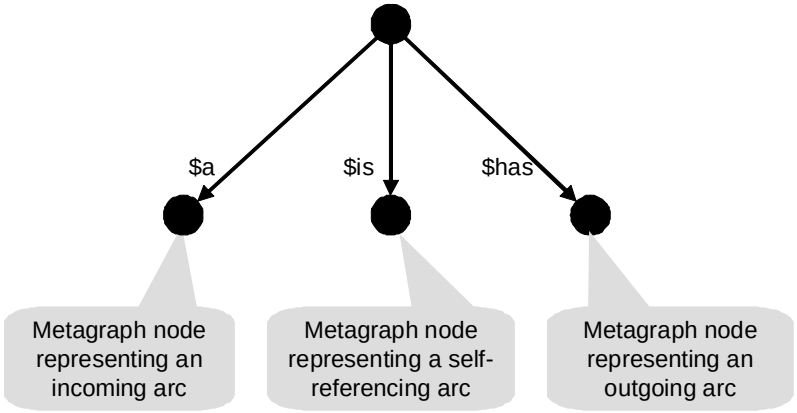
To create a self-describing graph, the XDI RDF model uses a very simple model of the structure of a directed graph itself. This *metagraph* models the three fundamental types of arcs in a directed graph:

1. Arcs originating at a node (incoming arcs).
2. Arcs terminating at a node (outgoing arcs).
3. Arcs that both originate and terminate at a node (self-referencing arcs).



To describe the arcs that relate to a specific node (the subject of the metagraph statement), the metagraph model represents each of these three types of arcs with a node identified by a specified XRI (an XRI in the **$** space):

1. **$a** identifies the node representing incoming arcs.
2. **$has** identifies the node representing outgoing arcs.
3. **$is** identifies the node representing self-referencing arcs.



Since a path through a directed graph is hierarchical (i.e., it follows outgoing arcs from one node to the next), all single-segment XDI addresses are XDI statements constructed using the **$has** predicate. The other compound metagraph predicates (**hasa** and the **$is** inverse predicates) build on these atomic metagraph predicates. More on this subject will be in the *XDI 1.0 Addressing and RDF Graph Model* specification.

Appendix B: ABNF for Proposed XRI 3.0 Syntax

This definition is currently maintained on the XRI TC wiki. See:

http://wiki.oasis-open.org/xri/XriThree/SyntaxAbnf

For convenience, following is the text as of the writing of this document:

```
xri                = xri-hier-part [ "?" iquery ] [ "#" ifragment ]
xri-reference      = xri
                   / relative-xri-ref
relative-xri-ref   = relative-xri-part [ "?" iquery ] [ "#" ifragment ]
relative-xri-part  = xri-path-abs
                   / xri-path-noscheme
                   / ipath-empty
xri-hier-part      = xri-authority xri-path-abempty
xri-authority      = global-subseg *subseg
subseg             = global-subseg
                   / local-subseg
                   / xref
global-subseg      = gcs-char [ local-subseg / xref / literal ]
local-subseg       = lcs-char [ xref / literal ]
gcs-char           = "=" / "@" / "+" / "$"
lcs-char           = "*" / "!"
literal            = 1*xri-pchar
literal-nc         = 1*xri-pchar-nc
xref               = "(" [ xref-value ] ")"
xref-value         = xri-reference
                   / iri
xri-path           = xri-path-abempty
                   / xri-path-abs
                   / xri-path-noscheme
                   / ipath-empty
xri-path-abempty   = *( "/" xri-segment )
xri-path-abs       = "/" [ xri-segment-nz *( "/" xri-segment ) ]
xri-path-noscheme  = xri-segment-nc *( "/" xri-segment )
xri-segment        = [ literal ] *subseg
xri-segment-nz     = ( literal / subseg ) *subseg
xri-segment-nc     = ( literal-nc / subseg ) *subseg
xri-pchar          = iunreserved / pct-encoded / xri-sub-delims / ":"
xri-pchar-nc       = iunreserved / pct-encoded / xri-sub-delims
xri-reserved       = xri-gen-delims / xri-sub-delims
xri-gen-delims     = ":" / "/" / "?" / "#" / "[" / "]" / "(" / ")"
                   / gcs-char / lcs-char
xri-sub-delims     = "&" / ";" / "," / "'"
```

Appendix C: ABNF for X3 Standard

This definition is currently maintained on the X3 Format page of the XDI TC wiki. See:

http://wiki.oasis-open.org/xdi/X3Format#head-96008a639cc086583ca0ef3eaec2283df214a6cd

For convenience, following is the text as of the writing of this document:

Base

Using RFC 4234 syntax, the base ABNF for X3 is just six lines:

```
X3          = *( "[" sub *( "[" pred *( "[" obj "]" ) "]" ) "]" )
sub         = [ comment ] xri-reference [ comment ]
pred        = [ comment ] xri [ comment ]
obj         = [ comment ] ( xri-reference / literal / X3 ) [ comment ]
literal     = """ *char """
comment     = "<--" *c-char "-->"
```

Characters

The ABNF rule for **char** is based on the same ABNF rule in JSON. This approach uses the backslash character for escaping. The only difference between the JSON and X3 escaped character sets is the addition of opening and closing square brackets.

```
char        = unescaped / escaped
unescaped   = %x20-21 / %x23-5A / %x5E-10FFFF
escaped     = escape (
            %x22 /          ; "    quotation mark  U+0022
            %x5B /          ; [    open sq bracket U+005B
            %x5C /          ; \    reverse solidus U+005C
            %x5D /          ; ]    clos sq bracket U+005D
            %x2F /          ; /    solidus         U+002F
            %x62 /          ; b    backspace       U+0008
            %x66 /          ; f    form feed       U+000C
            %x6E /          ; n    line feed       U+000A
            %x72 /          ; r    carriage return U+000D
            %x74 /          ; t    tab             U+0009
            %x75 4HEXDIG )  ; uXXXX                U+XXXX
escape      = %x5C          ; \
```

Comment Characters

The ABNF for **c-char** rule simply adds opening and closing angle brackets to the escaped character set:

```
c-char      = c-unescaped / c-escaped / escaped
c-unescaped = %x20-21 / %x23-3B / %x3D / %x3F-5A / %x5E-10FFFF
c-escaped   = escape (
            %x3C /          ; <    open an bracket U+003C
            %x3E )          ; <    clos an bracket U+003E
```

Typed Data Formats

X3 is designed to be able to carry data encoded in any popular text-based interchange format, while still retaining the ability for the data object to be identified, described, and linked using X3.

To do this, X3 uses typed data formats – a way of tagging X3 data with a data format identifier that tells the X3 processor how to handle that form of data.

X3 data format tags use the proposed XDI **$a** identifier. The **$a** identifier, when used at the start of an XDI predicate, identifies that the datatype belongs to the XDI type ontology. The **$a** identifier, when used as a modifier of other XDI predicates, indicates the data format type.

The first two proposed data format types are **$json** (ABNF from RFC 4627) and **$xml** (ABNF from XML 1.1).

Appendix D: X3 Whitespace Formatting Rules

This definition is currently maintained on the X3 Format page of the XDI TC wiki. See:

http://wiki.oasis-open.org/xdi/X3Format#head-40fb71a81051db413e8d7a5de4b8c5c838f2ff99-3

For convenience, following is the text as of the writing of this document:

4. Place each X3 subject, predicate, object on a new line.
5. Indent each X3 predicate one tab from the parent subject.
6. Indent each X3 object one tab from the parent predicate.
7. Put each closing bracket that immediately follows another closing bracket on a new line, indented to match its paired opening bracket.
8. If an object is a multi-line literal, put the opening square bracket and opening quote on the first line, put the closing quote and closing square bracket on the last line, and put the literal on its own set of lines in between, maintain the same indent for all lines.
9. If an object is a literal in a typed data format, display it as a multi-line literal (above), and observe all newlines and indents relative to the starting line.
10. If an object is X3, place the opening square bracket alone on the object line, begin the new X3 block on a new line, indent the entire X3 block one additional tab, and iterate the rules above.
11. Put each comment that appears *after an opening square bracket* immediately after the opening square bracket, then place the subject/predicate/object on a new line with the same indent as the line containing the comment.
12. Put each comment that appears *before a closing square bracket* on the same line as the subject/predicate/object it describes, indented one tab.

Appendix E: X3 Simple Formatting Rules

This definition is currently maintained on the X3 Format page of the XDI TC wiki. See:

http://wiki.oasis-open.org/xdi/X3Format#head-b49c0509b5cd5aa063340905a78989554e4320da-3

For convenience, following is the text as of the writing of this document:

1. First convert the X3 to X3 Whitespace.
2. For any multi-line literal, replace the opening square bracket and opening quote with a backslash "\", and replace the closing quote and closing square bracket with a backslash.
3. For each X3 object (at any level of nesting) that contains nested X3, replaced the opening square bracket "[" with a forward slash "/".
4. Delete all lines that contain only closing square brackets.
5. Delete all remaining opening and closing square brackets. (Note that escaped square brackets within XRIs or data should not be affected by this operation as they must be escaped.)

Appendix F: XML Schema for XDI RDF

This definition is currently maintained on the XDI RDF XML Schema page of the XDI TC wiki. See:

http://wiki.oasis-open.org/xdi/XdiRdfXmlSchema

Appendix G: Example XML Instance Document

This definition is currently maintained on the XDI RDF XML Schema page of the XDI TC wiki. See:

http://wiki.oasis-open.org/xdi/XdiRdfXmlSchema#head-1844ea58ed06950ed0b2c8f094ee54d56b281ed3

Appendix E: Revision History

V12 – 2009-01-28

- Revised metagraph inversion predicate to **$is** and updated all examples.
- Replaced **$context$...** with **$$...** per list discussion.
- Replaced the term *global cross-references* with the term *compound XRI.*
- Replaced the table in the RDF Graph Structure and Addressing section with the address forms discussed on the TC mailing list.
- Updated the RDF/OWL analogies in the *Base Grammer* section.
- Revised the end of Appendix A to simplify the metagraph description.
- Updated the ABNF in Appendix B to include xref subsegments.
- Updated the XDI Dictionary example to include examples of contexts.

V11 – 2008-10-21

- Replaced “direct concatenation” with “global cross-reference” in XRI 3.0 Syntax section.
- Added Variables and Typed Operations sections.
- Replaced !x with $x version identifiers in versioning examples.
- Updated the XDI link contract example to reflect the most recent work (see http://wiki.oasis-open.org/xdi/XdiOneIssues/LinkContractPattern).
- Added an appendix with the proposed ABNF for XRI 3.0 and snapshots of the wiki content referenced in several other appendicies to improve readability as a standalone document.

V10 – 2008-04-23

- Replaced **$type** with **$a** to further align with the metagraph model.
- Added a link to Markus Sabadello's new XDI Validator utility.
- Revised several headings.
- Made one correction in the examples.

V9 – 2008-02-24

- Removed the n-segment addressing syntax proposed (but not actually used) in V8 due to >3 segment parsing issues raised by Bill Barnhill.
- Made **$is** predicate reflexive, i.e., eliminated **$a$is** for inverse.
- Simplified inverse of **isa** to **$a** (formerly **$aisa**).
- Reintroduced support for XDI cross-references using XRI cross-reference syntax and explained how this was logically equivalent to the long form of traversing the XDI contexts explicitly.
- Renamed X3 Display to X3 Simple (per suggestion from Bill Barnhill to avoid confusion with another technology named X3D).
- Added first draft text for Appendix A.

V8 – 2008-01-30

- Restructured the document into a standalone introduction to XDI RDF.
- Updated the inverse relation for base predicates to use **$a**.
- All examples rewritten in X3.
- Added extensive examples of XDI RDF topics that were only lightly touched on in earlier versions.
- Removed XDI graph example, as X3 format makes it relatively easy to visualize the graph. (It may be returned in a future version.)
- Changed all appendicies into references to XDI wiki pages so they will track current work.

V7 – 2007-10-03

- Added inverse relations for **$is**, **$is$a**, **$has**, **hasa**.
- Removed the placeholder cardinality syntax (this will become part of the XDI dictionary).
- Updated **$type** children to use the **$** space.
- Corrected XDI RDF table examples for "group membership" (added $has predicate).

V6 – 2007-09-06

- General rewrite of introductory sections based on feedback from XDI RDF presentations and implementations.
- Added new section on XRI 3.0 syntax.
- Added new section on nested XDI documents.
- Revision to XDI RDF v4 schema in Appendix A to rename **<xdi:o>** to **<xdi:ref>** and change the type of **<xdi:data>** to *anyType*.
- Revised example XDI document in Appendix B to conform XDI RDF v4 schema.

V5 – 2007-05-21

- Clarified references to Higgins and Higgins contexts in the XDI Cards section.
- Updated references to XRI Syntax 2.1 to XRI Syntax 3.0.
- Miscellaneous editing throughout for readability.
- Moved revision history to Appendix.

V4 – 2007-03-26

- Updated terminology to use "ATI" instead of "3L".
- Simplified table listing options for XDI RDF model.
- Adjusted text and examples to use proposed XRI 3.0 syntax.
- Updated changed terminology in graph examples.

V3 – 2007-02-14

- New terminology.

- Updated the XDI RDF model section to specify XDI RDF statement forms that can be expressed with XRIs.
- Replaced the base XDI RDF predicate names with their simpler English equivalents (suggested by Bill Barnhill, Paul Trevithick, and Laurie Rae, among others).
- Changed **$data** to **$type** (per suggestion from Bill Barnhill) and removed it from the base predicate table (because it is only used as an object).
- Added information about the RDF/RDFS/OWL equivalents for the four base XDI RDF predicates.
- Updated the link contract examples to show a simpler way to express group membership.
- Added section show X3 syntax.

V2 – 2007-02-06

- Shortened the <subject>, <predicate>, and <object> element names in the XDI RDF schema to make instance documents easier to read.
- Revised the XDI RDF diagram to illustrate link contracts instead of ontologies.
- Provided a more typical example of an XDI instance document including examples of link contracts.
- Added a phone number example to the XDI RDF tables in the XDI Dictionary Definitions section to show a complex object definition.