The XDI RDF Model

V8, 30 January 2008

This document is a work-in-progress from the OASIS XDI Technical Committee. Contributors include:

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

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

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

Table of Contents

Introduction

XDI (XRI Data Interchange) is an open standard data interchange 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 second 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 now that the OASIS XRI Technical Committee has completed the final specification in the XRI 2.0 cycle (XRI Resolution 2.0 Committee Draft 02).

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

Motivations

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

- *Simplicity*. While the XDI ATI model was relatively simple, consisting of only 8 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 core 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. With the XDI RDF model, 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 aggregation. The XDI RDF model adds two more fundamental relation types for representing inheritance and composition.
- *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 each ref and link has its own explicit XDI address, and all XDI operations can be performed on refs and links 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.* With the XDI RDF model, the XDI graph can be represented using the same (or simpler) notational formats as other RDF graphs, including a very simple notation called X3 inspired by RDF N3 notation.

About the Proposed XRI 3.0 Syntax

The XRI addressing used in the XDI RDF model is based on the proposed ABNF for XRI Syntax 3.0. The key features of this syntax and their differences from XRI 2.0 syntax (specified in the OASIS XRI Syntax 2.0 Committee Specification) are described in this section.

Global Context Symbols

The proposed ABNF for XRI Syntax 3.0 reduces the set of XRI global context symbols—single characters expressing the global context of an identifiers—from the 5 specified in XRI 2.0 to the four shown below. (In XRI Syntax 2.0, the "!" symbol was used as both a global context symbol and a local context symbol. The XRI TC has reached concensus to deprecate it as a global context symbol beginning in XRI 3.0.)

GCS Char	Applies To	Description
$	Classes (dictionary metagraph)	The self-context. $ is the root of the XRI dictionary specified by the OASIS XRI Technical Committee and XDI dictionary specified by the OASIS XDI Technical Committee.
+	Classes (dictionary graph)	The generic context.
=	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 groups or organization).

Direct Concatenation

The second key feature of XRI 3.0 needed by XDI RDF is called *direct concatenation*. It is the ability for any two valid XRI 3.0 absolute XRIs (excluding fragments) to be directly concatenated to create another valid XRI 3.0 XRI. This feature is core to the XDI RDF addressing algorithm.

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

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

Using direct concatenation, these three XRIs can be joined into a single XRI that functions as a single 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 standard hierarchical or opaque identifiers.

Cross-References

A third key XRI feature used by XDI 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. This feature, called *cross-references*, uses parentheses to syntactically encapsulate the cross-referenced identifier. It is vital to XDI RDF because it enables all any URI to be included in an XRI expressing an XDI RDF statement. This 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/doc,html>
```

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

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

The XDI RDF Model

Basic Structure and Addressing

In the XDI RDF model, all data is structured and addressed using RDF subject/predicate/object statements encoded as XRIs. Each XRI representing an XDI RDF statement consists of up to three segments, each of which is itself either an absolute or relative XRI.

First segment (RDF Subject)	Second segment (RDF Predicate)	Third segment (RDF Object)	Addresses & Describes
XRI	Absent	Absent	The XDI graph (set of all XDI RDF statements) rooted in an XDI subject.
XRI	XRI	Absent	The XDI graph rooted in the target(s) of the predicate, which may be either: b) a data literal, b) a set of XDI subjects, or c) another XDI context.
XRI	XRI	XRI	Same as the first row above, except expressed as the object of an XDI RDF statement.

The address of any node in the XDI RDF graph consists of the XRI necessary to identify the starting subject node plus a sequence of zero or more XRI segments necessary to identify the path from this subject node to the target node(s). This produces a pattern of non-empty XRI segments that looks like the following (whitespace added for readability):

```
subject-xri / predicate-xri / subject-xri / predicate-xri / …
```

In this pattern:

- A single segment identifies a starting XDI RDF subject node. In an XDI RDF graph context, all XRIs must be unique, so this first XRI identifies a unique XDI RDF subject in the context in which this XRI is to be resolved.
- The second segment represents an XDI RDF predicate that uniquely identifies an XDI RDF object, which may be either: a) a data literal, b) a set of zero or more XDI subject nodes, or c) a new XDI RDF context (see below).
- The third segment represents the XDI RDF object and completes the XDI RDF statement. *Note that if the third segment is another absolute XRI representing another XDI subject, then this XDI RDF statement includes the address of that subject both in the relative context of this particular XDI RDF statement and in its own absolute context. If the XRI in the third segment is relative, then this XDI subject is only addressable in the context of this statement.*
- A fourth non-null segment begins a second XDI RDF statement, whose subject is the object of the first statement.
- A fifth non-null segment would complete the expression of two complete XDI RDF statements.
- And so on—this pattern can be repeated to any length.

Contexts, Context References, and Context Descriptors

The XDI RDF addressing model also permits addressing of *contexts*—different instances of XDI RDF graphs, each with its own unique 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) URL) accessible via at least concrete interaction protocol (such as HTTPS or HTTPS). It is a core principle of XDI design 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.

- To *enable* correlation, one or more absolute XRIs for the subject may be shared across any subset of these contexts.
- To *prevent* correlation, one or more relative XRIs may be assigned to the subject in that one context.

Any combination of the two approaches may be used in any combination of contexts to fufull applicable security and privacy policies.

In an XDI address, the transition from one context into another is indicated by an empty segment after an XDI predicate. The empty segment indicates to an XDI RDF processor that further addressing begins in the new context. This pattern looks like the following (whitespace added for readability):

```
subject-xri-in-context-1 / predicate-xri // subject-xri-in-context-2 / predicate-xri // …
```

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 context. XDI context descriptors are the XDI equivalent of the XRDS (Extensible

Resource Descriptor Sequence) documents used in XRI resolution. XDI context descriptors use predicates rooted in the **$context** dictionary that function like XRDS service endpoints. See examples in the *Examples* section.

The Four Core Predicates

The XDI RDF model uses four predicates defined in the XDI $ dictionary to express fundamental relations between XDI subjects (an XDI dictionary is an 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 XDI RDF metagraph model described briefly in Appendix A.

Predicate	Inverse Predicate	Relation Type	RDF/RDFS/OWL Analogs
$is	$a$is	Equivalence	`owl:sameAs` for XDI RDF subject equivalence. `owl:equivalentClass` for XDI RDF object equivalence.
isa	$a$is$a	Inheritance	`rdfs:subClassOf` for for XDI RDF subjects. `rdf:type` for XDI RDF objects.
$has	$a$has	Aggregation	`exns:hasXXX`, where `exns` is the specified namespace and `XXX` is the property name.
hasa	$a$has$a	Composition	`exns:hasxxx`, where `exns` is the specified namespace and `xxx` is the property name.

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

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

Note: the **$has** relation plays a special role in XDI addressing. Any two XDI subjects with a **$has** relationship can be concatenated into a single XRI representing this relationship. If an XDI client has the appropriate access rights, it can perform XDI discovery by traversing **$has** relations using context descriptors. See examples in the *Examples* section.

The Four Atomic REST 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. The atomic operations all operate on rows in the logical "table" of XDI RDF statements represented in any XDI RDF document (see the following sections for examples).

These four atomic REST operations are summarized in the following table:

XDI RDF Graph Operation	CRUD Equivalent	Description
$get	read	Read one or more rows from the XDI RDF table.
$add	create	Write one or more new rows to the XDI RDF table.
$mod	update	Modify a data element value in a row of the XDI RDF table.
$del	delete	Delete one or more rows from the XDI RDF table.

Declaring XRIs for these explicit XDI protocol operations establishes the basis for permissioning in XDI link contracts (see the *Examples* section). Note that another predicate, **$set**, aggregates the **$add**, **$mod**, and **$del** operations for purposes of permissioning in XDI link contracts.

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 formal 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 an XRI: **$type.** A limited **$type** dictionary will be specified by the XDI TC, however the **$type** dictionary is infinitely extensible by all XDI users.

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

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

Examples:

```
$type$mime$(text/html)
$type$mime$(application/atom+xml)
$type$xsd$string
$type$xsd$boolean
```

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

Link Contracts

One of the core design goals of XDI is that controls over XDI data sharing—everything from authorization and access control to data usage and termination—be expressable as XDI documents so they are fully contained in the XDI graph. This not only permits XDI rights management to be portable across XDI service providers, but it enables standard XDI operations to be used to create, manage, delete, and share XDI permissions.

An XDI document used to express such data sharing controls is called a *link contract* and is rooted in the **$contract** dictionary. See examples in the *Examples* section.

X3 Serialization Format

XDI documents can 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 expressed in a much more compact text serialization 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.

Standard X3

The ABNF for standard X3 format—the format actually transmitted across the wire—uses an extremely 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 (**=drummond** is a personal XRI registered by Drummond Reed, co-chair of the XDI TC).

```
[=drummond[$is[=!F83.62B1.44F.2813]][$is$a[+person]][+person+name
["Drummond Reed"]][+email["drummond.example@cordance.net"]]]
```

X3 Whitespace (X3W)

The lack of whitespace in standard X3 makes 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 (X3W) that includes whitespace for readability. X3W is still technicall valid X3 because all whitespace is ignored. The rules for X3W are included as Appendix C.

Following is the example X3 document from above displayed in X3W format.

```
[=drummond
   [$is
      [=!F83.62B1.44F.2813]
   ]
   [$is$a
      [+person]
   ]
   [+person+name
      ["Drummond Reed"]
   ]
   [+email
      ["drummond.example@cordance.net"]
   ]
]
```

X3 Display (X3D)

While X3W is relatively easy to read, 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 X3W into X3 display format (X3D), which eliminates square brackets altogether. This version is very simple to read *and* write.

Following is the same example X3 document displayed in X3D.

```
=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. Though they are not normatively part of the XDI RDF graph, they are are programmatically accessible in the graph model because exactly one comment can either prepend or postpend an XDI subject, predicate, or object.

In X3D 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 that:

- Human-readable documents use X3D.
- On-the-wire transmissions use standard X3.
- X3W be used only for debugging and technical documentation.

Validation and Conversion

An online tool called *X3 Converter* for validating and converting between XDI RDF documents in all formats (X3, X3W, X3D, XDI/XML, XDI/JSON) has been developed by XDI TC member Markus Sabadello. It is publicly available at:

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 X3D. All examples can be cut-and-pasted into the 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 typically included on a standard business card (plus some identifiers and metadata that 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 predicates, **$is**, asserts that **=drummond** is the same resource identified by two synonymous XRIs, **=!F83.62B1.44F.2813** (a persistent i-number) and **=drummond.reed** (a reassignable i-name), plus one URL "cast" as an XRI, **=(http://equalsdrummond.name)**. The second predicate, **isa**, asserts that the subject is a subtype of the generic **+person** class.

All the other example predicates express attributes of **=drummond** that have literal values. 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. A project called Community Dictionary Service with the Identity Schemas Working Group at Identity Commons has begun work on implementing this service.

Any specific attribute value can be requested from this context using the XDI address of the attribute. 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

So far our examples have shown XDI documents containing only one subject. However an XDI document can contain any number of subjects, with or without relations between them. The next example shows **=drummond** creating multiple "personas" containing sets of attributes appropriate to specific contexts.

```
=drummond
   $has  <-- expresses aggregation relations -->
      +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 **$has** relation to the subject **=drummond**. This relation can be queried just like any other attribute. For example, if an XDI **$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)
```

This response informs the XDI client that more information is contained in the aggregate 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).

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$last**), or its HTTP(S) URI (**urihttp** or **urihttps**). It can also describe its context type (**isa**) using one or more XRIs rooted in the **$context** dictionary. And it can have backwards references to other XDI contexts that reference it, called its *supercontexts*.

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

```
$
   $is$a
      $context$xdi
```

```
   $a$context$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—in fact 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
      $context$xdi
   $a$context$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
   $context$xdi
      /
         $
            $is$a
               $context$xdi
            $a$context$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"
```

For readers familiar with the XRDS discovery architecture in XRI Resolution 2.0, the **$context$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/$context$xdi
```

…and receive the following response:

```
=drummond
   $context$xdi
      /
         $
            $is$a
               $context$xdi
            $a$context$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 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
   $context$xdi
      /
         $
            $is$a
               $context$xdi
            $a$context$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"
   $context$openid
      /
         $
            $is$a
               $context$openid$v!2.0
               $context$(http://specs.openid.net/auth/2.0/signon)
            $a$context$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/$context$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 two attributes 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 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 still 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 versioning of any portion of XDI 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 server 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 using version logging can provide "virtual" version snapshots, as 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

The 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: 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 can 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 not need 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
      !3
   $is
      =!F83.62B1.44F.2813
=drummond$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 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 include the caveat "*if the requester has permission*". The 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 they 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 pattern that:

- A link contract is itself an XDI subject that defines the data sharing terms.
- This XDI subject is then used as an XDI predicate to describe the permissions granted under that contract.

In this example, **=drummond** has a link contract with **@cordance** that gives **@cordance** permission to get his email address.

```
$
   $is$a
      $context$xdi
   $a$context$xdi
      =drummond
      =!F83.62B1.44F.2813
=drummond
   $has
      +home
      @cordance
   $has$a
      @cordance$contract
   @cordance$contract
      /
         @cordance
            $get
               /
                  =drummond+home
                     +email
   $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"
@cordance$contract
   $is$a
      $contract
   $a$has
      @cordance
   $contract
      /
         @cordance
            $get
               +email
```

Following is an example of the XDI document for **@cordance** showing the link contract it originally created, the link to **=drummond** who accepted this contract, and the data **=drummond** is sharing under the contract.

```
$
   $is$a
      $context$xdi
   $a$context$xdi
      @cordance
      @!8FB6.9DFC.D1FD.E073
@cordance
   $has
      $contract
@cordance$contract
   $is$a
      $contract
   $a$has
      @cordance
   @a$has$a
      =drummond
   $contract
      /
         @cordance
            $get
               +email
=drummond
   @cordance$contract
      /
         @cordance
            $get
               /
                  =drummond+home
                     +email
```

```
=drummond+home
   +email
      "drummond@example.com"
```

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*.

XDI dictionaries are another area in active development at the XDI TC, however again their basic structure simply derives directly from the four core 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
   $has$a
      $uri
      +home
      +work
      +friend
      +spouse
      +person+name
      +email
      +tel
+home
   $a$has$a
      +person
+work
   $a$has$a
      +person
+friend
   $is$a
      +person
   $a$has$a
      +person
+name
   $is$a
      $type$xsd$string
+person+name
   $is$a
      +name
   $a$has$a
      +person
   $has
      +first
```

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

Appendix A: The XDI RDF Metagraph Model

[Coming in V9]

Appendix B: ABNF for Standard X3

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

Appendix C: X3W 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-6fa1d6b753a6f5617e8669ed2246fb7793a6003d-2

Appendix D: X3D 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-06494da03d3023d1478776c359f7246dd6f4609f-2

Appendix E: 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 F: 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

V8 – 2008-01-30

- Restructured the document into a standalone introduction to XDI RDF.
- Updated the inverse relation for core 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 core 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 core predicate table (because it is only used as an object).
- Added information about the RDF/RDFS/OWL equivalents for the four core 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.