The XDI Graph Model

2012-02-06

Editor: Drummond Reed, XDI TC Co-Chair

This document is a work-in-progress from the OASIS XDI Technical Committee reflecting contributions from many members of the TC. Its purpose is to provide an overview and examples of the XDI graph model that has been developed over the past several years by the TC and is now in the process of being formally defined in the XDI 1.0 specifications.

Note that it is only an informative document only and is not normative for any specifications from the TC.

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

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

Earlier versions of this document were called The XDI RDF Model *and were maintained on the http://wiki.oasis-open.org/xdi/XdiRdfModel wiki page.*

Table of Contents

Introduction

XDI (XRI Data Interchange) is an open standard semantic graph model, 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. This model (originally called the XDI RDF Model and now just the XDI Graph Model) provides the foundation for the XDI 1.0 specifications.

This document provides an overview of the XDI graph model and examples of how it can be applied to many well-known challenges in distributed data sharing.

About XRI 3.0 Syntax

XDI structured data sharing is rooted in the capabilities of XRI structured identifiers. Addressing of the XDI graph is based on the ABNF specified in XRI Syntax 3.0. Key features of this syntax are described in this section.

Global and Local Context Symbols

The first key feature of XRI syntax is single character symbols that represent abstract global contexts—shared logical root nodes for XRI identifier graphs. In XRI 2.0 there were five such symbols. The ABNF for XRI 3.0 reduces that set to the four shown below.[1]

GCS Char	Description
$	The self-context. $ is also the root of the XDI grammar dictionary specified by the OASIS XDI Technical Committee.
=	The personal context, i.e., the ultimate authority for an XRI in this context is an individual person.
@	The organizational context, i.e., the ultimate authority for an XRI in this context is a group, organization, or institution.
+	The generic context—the root of XRIs that have no specified identifier authority but evolve by shared consensus (e.g., Wikipedia).

In addition to the global context symbols, XRI 3.0 has the same two local context symbols as XRI 2.0.

[1] In XRI 2.0, the "!" symbol was used as both a global context symbol and a local context symbol. In XRI 3.0 its use as a global context symbol was deprecated.

LCS Char	Description
!	The context for immutable identifiers, i.e., identifiers that are permanently assigned to a resource and will not change. Also used in XDI to represent instances of a class.
*	The context for mutable identifiers, i.e., identifiers that may be reassigned to different resources over time. Also used in XDI to express ordering of instances.

Subsegments and Composite XRIs

Generic URI syntax as defined in IETF RFC 3986 supports hierarchical structure within the path component of a URI using forward slashes to delimit path segments. XRI syntax adds a secondary level of structure: the ability to *subsegments* within a path segment. Subsegments are delimited by any of the global or local context symbols described above, or by cross-references (see below).

A property of XRI subsegment syntax is that single-segment XRIs (XRIs that consist of only one or more subsegments) may be concatenated to form a third valid single-segment XRI. This is called a *composite XRI*. For example, following are three absolute single-segment XRIs representing an organization, a tag, and a person, respectively:

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

These three XRIs can be concatenated into a single composite XRI.

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

Within this identifier, each component XRI appears in the context of its predecessor. This structure, similar to nested elements in XML, enables construction of identifiers whose semantics are both machine- and human-understandable. Such *semantic identifiers* support introspection, federated discovery, algorithmic mapping, and other benefits not readily available from opaque identifiers.

Cross-References

A third key feature of XRI syntax is called *cross-references*. This syntax enables an XRI to: a) group other XRIs into a single syntactic component, and b) encapsulate identifiers from other identifier syntaxes and namespaces the same way XML can encapsulate and "tag" data from other native data sources.

XRI syntax uses parentheses for this purpose. This cross-reference syntax is vital to XDI because it enables any URI to be "cast" into an XRI and thus included in an XDI statement. It also enables algorithmic transformation of conventional RDF documents into XDI documents.

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

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

Each of these URIs can be expressed as a relative XRI cross-reference by enclosing it in parentheses:[2]

```
(http://example.name)
(http://dc.org/tag/author)
(http://example.com/example.html)
```

Now we have three single-subsegment XRIs that can be composed into a single XRI representing an XDI statement for the same triple expressed in N3 above. The first segment is the XDI subject, the second is the XDI predicate, and the third the XDI object.

```
(http://example.name)/(http://dc.org/tag/author)/(http://example.com/example.html)
```

[2] Note that escaping of parentheses characters within the URI, plus other standard URI delimiters like # and ?, is necessary during this step (and not shown in this example). These transformation rules are defined in the XRI 3.0 Syntax spec.

Part 1: Key Concepts of the XDI Graph Model

Comparison with RDF Graph Model

The XDI graph model is a close cousin to the RDF graph model, i.e., both are based on subject-predicate-object triples. However, due to the problem space for which XDI was developed—global data sharing across contexts—there are subtle but important differences between the two models as explained the following table:

RDF graph model	XDI graph model	Explanation
Not addressable	100% addressable	RDF graphs do not require unique addressability of all nodes within the graph. With XDI, all nodes in all contexts must be uniquely addressable with an XRI. XDI grammar also includes *synonyms*: a means for expressing that two XRIs identify the same logical XDI resource.
Opaque identifiers	Semantic identifiers	In RDF, the URIs used to identify nodes and arcs are opaque values. In XDI, each component XRI within an XDI address itself represents an XDI statement, so an XDI processor can semantically "read" these XRIs to understand XDI graphs.
Blank nodes	Context nodes	In RDF, blank nodes are not addressable or portable across contexts. In XDI, all subjects are *context nodes* that are functionally similar to RDF blank nodes in that they can be both the subject and object of XDI statements. However at the same time they are addressable, both absolutely and relatively, and this address can be ported across contexts.
Named graphs	Nested graphs	RDF named graphs (as supported in SPARQL) are a solution to providing context (graph addressability) to RDF graphs. However they require the use of quads instead of triples. With the use of context nodes and semantic identifiers, XDI support nested graphs to any depth using only triples.
No shared context	Global and local contexts	In RDF, every RDF graph is an independent set of RDF statements. In XDI, there is a shared global context – the abstract root of the logical XDI graph. Every local XDI graph located at a concrete network endpoint shares this logical root node. This enables attributes of a local XDI graph – such as its URI(s) – to be discovered across the logical XDI graph using *XDI discovery*.

Contexts: The Fundamental Building Block of XDI Graphs

In the RDF graph model of nodes and arcs, there are three types of nodes:

1. *Standard nodes* represent the URI-identifiable resources in the graph.
2. *Blank nodes* are the special type of RDF graph node that may be either a subject or an object, but which are not identifiable outside the context of the RDF graph in which they appear.
3. *Literal nodes* contain the actual data being described.

In the XDI graph model, there are also three types of nodes:

1. *Root nodes* represent the URI-identifiable locations of local XDI graphs, which may be nested to support discovery.
2. *Context nodes* represent all XRI-identifiable resources in the graph.
3. *Terminal nodes* contain the actual data being described.

From an RDF standpoint, a *XDI context* is an RDF graph that may contain or be linked to other RDF graphs. Each XDI context, starting with the logical root context, is the root of its own unique XRI addressing space. This is what makes the entire logical graph addressable.

Since each XDI context is the root of its own RDF graph, XDI contexts offer similar functionality to RDF *named graphs*. However the universe of RDF named graphs is a single flat addressing space. In the XDI graph model, one XDI context may be nested within another XDI context, to any depth. This enables XDI to address shared logically shared resources and data across multiple contexts. It also enables relative graph addresses to be portable when subgraphs are copied or moved across contexts.

Local Graphs, Root Nodes, and XDI Discovery

Although the XDI global addressing space is one giant *logical graph*, no single network location stores the entire graph. Any subset of the graph addressable at a URI-addressable network endpoint (called an *XDI endpoint*) is called a *local graph*.

Every local graph is a subset of the XDI logical graph located at a *root node*. Since a critical feature of XDI is trusted navigation of local graphs to discover new resources and data in the logical graph, the root node properties necessary to support this will be defined in the *XDI Discovery* specification.

For example, following is an XDI statement expressing the first instance of a URI for the local graph of **=example**:

```
=example$uri!1/!/(data:,http://xdi.example.com/)
```

Note that the XDI endpoint identified by an XRI may also be discoverable using other discovery protocols including:

- XRDS discovery defined by the OASIS XRI Resolution 2.0 Specification.
- XRD discovery defined by the OASIS XRD 1.0 Specification.

- Simple Web Discovery as defined by the IETF Simple Web Discovery draft specification.

See the XDI TC wiki page on XDI discovery for more information.

XDI Addressing

In the XDI graph model, the structure of the graph is expressed using XDI statements encoded as composite XRIs. There is exactly one XDI statement for every arc in the graph. These XRIs form paths in the XDI directed graph, making the entire graph addressable.

The structure of the XDI graph can be expressed in the following set of ABNF statements that define an XDI address (these build on the ABNF from XRI 3.0):

```
xdi-address     = xdi-subject [ "/" xdi-predicate "/"  xdi-object ]
xdi-subject     = xdi-segment
xdi-predicate   = xdi-segment
xdi-object      = xdi-segment
xdi-segment     = [ literal ] *xdi-subseg
xdi-subseg      = global-subseg
                  / local-subseg
                  / xref
global-subseg   = gcs-char [ local-subseg / xdi-ref / literal ]
local-subseg    = lcs-char [ xdi-ref / literal ]
xdi-ref         = "(" [ xdi-ref-value ] ")"
xdi-ref-value   = xdi-address
                  / iri
```

Following this ABNF, there are three basic types of addresses within the graph:

Address	Examples
Root node (Always appears as a set of cross-references)	(=example1) (=example1)(!1234) (=example1)(!1234)(+passport) (http://xdi.example.com/root/)
Context	=example1 =example1!1234 =example1!1234$d =example1+passport
XDI statement (Always appears as a cross-reference when used as an XDI object)	(=example1+age/!/($)) (=example1+tel/+home*2/($)) (@example1/+employees/=example2)

Encoding XDI Literals as XRIs Using the Data URI Scheme

An XDI literal may be expressed as part of the XDI graph by encoding it as an XRI cross-reference using the Data URI Scheme defined RFC 2397.[3] Following is an example of a phone number expressed in this fashion:

```
=example+tel!3/!/(data:,+1-206-555-1212)
```

Public and Private Addresses

Like RDF, it is a core design principle that an XDI subject may be identified and described in any number of XDI contexts by any number of XDI authorities. In each context the XDI subject may be addressable either: a) absolutely, b) relatively within that context, or c) both. This is important from a privacy perspective:

- To *enable* correlation of an XDI subject across contexts, one or more absolute XRIs for that subject may be shared across these contexts. These are typically *public* or *omnidirectional* XDI addresses, although they may also be pseudonyms if the scope of sharing is limited.
- To *prevent* correlation of an XDI subject across contexts, one or more relative XRIs may be assigned to that subject in within each context and not shared across contexts. These are called *private* or *unidirectional* XDI addresses.[4]

Any combination of these two approaches may be used to fulfill the security and privacy requirements of a particular context.

XDI Messages

All interactions with an XDI endpoint using the XDI protocol take place by sending and receiving XDI graphs called *XDI messages*. The basic requirements of XDI messages will be defined in the *XDI Protocol* specification.

Following is the basic template for unsigned XDI messages (squiggly brackets represent template variables):

```
{from-graph}/$add/{from}$msg{id}
{from}$msg{id}/$is()/{to-graph}
{from}$msg{id}/$d!/(data:,{datestamp})
{from}$msg{id}/$do/{link-contract}
{from}$msg{id}$do/{operation}/{target}
```

In this template:

- **{from-graph}** is the XRI of the local graph originating the message (always expressed as a cross-reference).
- **{from}** is the XRI of the message sender (the XDI authority originating the message) which may or may not be the same as the {from-graph}.
- **{id}** is the persistent i-number of the message.

[3] http://tools.ietf.org/html/rfc2397

[4] Note that a private XDI address may be absolute, but in this case it must not be shared across contexts.

- **{to-graph}** is the XRI of the local graph for receiver. Always expressed as a cross-reference. This is the XRI on which XDI discovery is performed to discover the concrete XDI endpoint URI. There may be multiple **{to-graph}** statements - one for each recipient.
- **{datestamp}**is a literal datestamp of the message in XML datetime format.
- **{link-contract}** is the XRI of the root node ($do context) of the link contract authorizing the requested operation on the requested target.
- **{operation}** is the requested XDI operation ($get, $add, $mod, $del, $copy, $move), plus extension (if any).
- **{target}** is the XDI address or statement that is the target of the operation. There may be multiple **{target}** statements - one for requested subgraph or statement.

Notes:

- **{from-graph}** and **{to-graph}** are always expressed as XRI cross-references because by definition XDI endpoints are physical graphs.
- Including the **{link-contract}** reference makes it very efficient for XDI servers to perform authorization, because they know exactly which link contract to evaluate. For a public link contract, this value is '"$public"'.

Signed XDI messages include one additional statement containing the message signature. More information an examples of both unsigned and signed XDI messages are given on the XDI TC wiki page on XDI message patterns.

Part 2: XDI Semantics

XDI Grammar: The XDI Metagraph

In RDF, the semantics expressed in an RDF graph are defined by the ontolog(ies) it uses. Ontology languages such as OWL have been developed to enable shared RDF semantics across the Web.

XDI documents use a very simple upper ontology known as the *XDI metagraph model* (or less formally as *XDI grammar*). This model is based on the concept of a metagraph ("graph describing a graph"). To understand this model, start with the four basic concepts of an RDF graph.

Concept	Represents
Subject	A node that is the source of an arc
Predicate	An arc
Object	A node that is the target of an arc
Context	The graph itself

Second, assign XRIs to represent each of these concepts.

Concept	XRI
Subject	$is
Predicate	$has
Object	$a
Context	()

Third, define the semantics for each of these XRIs used as a metagraph predicate.

Concept	XRI	Statement	Semantics
Subject	$is	x/$is/y	X is subject Y
Predicate	$has	x/$has/y	X has predicate Y
Object	$a	x/$a/y	X has subtype Y
Context	()	x/()/y	X has subcontext Y

Semantics of $is as a Predicate: Equivalence

In **x/$is/y**, the **$is** predicate asserts that the subject identified by XRI X is the subject identified by XRI Y. This means X and Y are logically equivalent, i.e., that they both identify the same XDI resource. Such XRIs are called *synonyms*, and they are a common design pattern in XDI graphs.

Since equivalence is reflexive, the following two XDI statements are semantically equivalent:

```
x/$is/y
y/$is/x
```

In English this closely matches the use of the verb “is” to assert the equivalence of two nouns. For example:

```
X is Y.
Y is X.
```

The ability to substitute English “is” statements for XDI **$** statements works with both classes and individuals.

```
+car/$is/+auto
A car is an auto.

+auto/$is/+car
An auto is a car.

bob/$is/bob.jones
Bob is Bob Jones.

bob.jones/$is/bob
Bob Jones is Bob.
```

Note that the semantics of **$is** are not identical to that of the RDF **owl:sameAs** predicate. To support efficient navigation of the XDI graph, a **$is** statement is unidirectional and canonical—it points in only one direction between two XDI nodes, and it requires the source node to redirect to the target. Thus when an XDI processor encounters a **$is** statement, it moves to the target node and continue processing from that node. (It has been suggested that the XDI TC define a different XRI such as **$same** to express the same semantics as the RDF **owl:sameAs** predicate. Alternately the TC could simply use a cross-reference to the OWL URI for **owl:sameAs**.)

Like all XRIs, the XRI **$is** may also be used as a context itself. When used as the root context of an XDI predicate, **$is** expresses the inverse of the arc expressed by the balance of the predicate. For example:

```
abraham/+son/cain
cain/$is+son/abraham
alice/+friend/bob
bob/$is+friend/alice
```

In English:

```
Abraham has a son Cain.
Cain is the son of Abraham.
Alice has a friend Bob.
Bob is the friend of Alice.
```

This universal XDI semantics of **$is** for predicate inversion applies to the XDI metagraph predicates themselves as summarized in the following table.

Concept	XRI	Statement	Semantics	Inverse Statement	Semantics
Subject	$is	x/$is/y	X is subject Y	y/$is$is/x	Y is subject X
Predicate	$has	x/$has/y	X has predicate Y	y/ishas/x	Y is a predicate of X
Object	$a	x/$a/y	X has subtype Y	y/isa/x	Y is a type of X
Context	()	x/()/y	X has subcontext Y	y/$is()/x	Y is a subcontext of X

Although **$is$is** may appear redundant because **$is** is already reflexive, it in fact expresses the precise semantics of a non-canonical relationship. For example, in the following statements, the XRI **+x** is canonical for the XDI resource it identifies, while the synonyms **+y** and **+z** are non-canonical:

```
+y/$is/+x
+z/$is/+x
+x/$is$is/+y
+x/$is$is/+z
```

Semantics of $has as a Predicate: Relation

In **x/$has/y**, the **$has** predicate asserts that the subject identified by XRI X has the predicate identified by XRI Y. This means Y is a relation of X. The inverse, **ishas**, means that X is a relation of Y.

```
+person/$has/+friend
+friend/$is$has/+person
+father/$has/+son
+son/$is$has/+father
```

In English:

```
A person has a friend.
A friend is something a person has.
A father has a son.
A son is something a father has.
```

$has statements are used in XDI dictionaries to define the relationship of classes. However they may also be used to describe the relations of an individual.

```
+person/$has/+friend
alice/$has/+friend
+father/$has/+son
abraham/$has/+son
```

In English:

```
A person has a friend.
Alice has a friend.
```

```
A father has a son.
Abraham has a son.
```

Semantics of $a as a Predicate: Subtype and Supertype

In **x/$a/y**, the **$a** predicate asserts that the subject identified by XRI X has the object identified by XRI Y. In the metagraph model this means X is an incoming arc to node Y, i.e., X is a predicate that describes the type of object Y. So the statement **x/$a/y** asserts that Y is a subtype of X, and **x/isa/y**, asserts that Y is a supertype of X.

```
+vehicle/$a/+car
+car/$is$a/+vehicle
+shape/$a/+circle
+circle/$is$a/+shape
```

In English:

```
A type of vehicle is a car.
A car is a type of vehicle.
A type of shape is a circle.
A circle is a type of shape.
```

Like **$has** statements, **$a** statements can be used both in XDI dictionaries to define classes and in XDI instances to describe individuals.

```
+person/$a/+son
cain/$is$a/+son
+place/$a/+city
seattle/$is$a/+city
```

In English:

```
A type of person is a son.
Cain is a son.
A type of place is a city.
Seattle is a city.
```

Semantics of () as a Predicate: Subcontext and Supercontext

In **x/()/y**, the **()** predicate asserts that the context identified by XRI X has the subcontext identified by XRI Y. In the metagraph model this means Y is a subject in the RDF graph identified by X. The inverse, **$is()**, asserts that the subject X is a member of graph Y.

In an XDI dictionary, this predicate is used to express the properties of classes.

```
+car/$()/+year
+year/$is$()/+car
+circle/$()/+diameter
+diameter/$is()/+circle
```

In English:

```
A car has a property of year.
A year is a property of a car.
A circle has a property of diameter.
A diameter is a property of a circle.
```

Once again, **()** statements can be used both to define classes and to describe individuals.

```
+work/()/+person
+work/()/=example1
+home/()/+address
+home/()/=example1
```

In English:

```
Work is a context of person.
Work is a context of Example1.
Home is a context of address.
Home is a context of Example1.
```

XDI addresses linking contexts are formed by concatenating each context address in the order of the context containment hierarchy.

```
+work/()/+person        ==>      +work+person
+work/()/=example1      ==>      +work=example1
+home/()/+address       ==>      +home+address
+home/()/=example1      ==>      +home=example1
```

```
+person/$is()/+work        ==>   +work+person
=example1/$is()/+work      ==>   +work=example1
+address/$is()/+home    ==>      +home+address
+example1/$is()/+home      ==>   +home=example1
```

XDI Protocol Operations

Following the REST model, the XDI protocol supports four atomic operations on the XDI graph itself.

XDI 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 (may only be applied to XDI literals).
$del	delete	Delete one or more existing statements from the graph.

Three additional XDI operations are defined for other standard graph operations.

XDI Graph Operation	Description
$copy	Push synchronize a portion of the graph from one context to another (in essence, mirror a set of operations performed in one context to another context).
$move	Move a portion of the graph from one context to another (copy it then delete the original).
$do	The abstract root context for all XDI operations. $do is used both for link contracts and for defining RPC-style operations using the XDI protocol. This topic will be covered in more detail in the XDI 1.0 specifications

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

XDI Dictionaries

XML has schemas, RDF has ontologies, XDI has *dictionaries*. An XDI dictionary is an XDI graph that provides semantic definitions of a set of XRIs using XDI metagraph statements. For example, an XDI dictionary for contact data (e.g., the XDI equivalent of vCard) would define XRIs for contact data types (e.g., name, telephone number, postal address, email address, home context, work context, etc.) together with the relationships between them.

An XDI dictionary may be a static XDI file, or it may be available for interaction at an XDI endpoint. The latter is called an *XDI dictionary service*, and it will play a key role in establishing interoperable XDI semantics across communities of use. The XDI TC anticipates that some XDI dictionaries for public semantic sharing may operate in open community models similar to Wikipedia and DBpedia.

The XDI Type Dictionary

To enable the XDI graph to be fully self-describing, the XDI TC defines a special XDI dictionary to describe specializations of the **$a** abstract root type described above. Like all XDI dictionaries, this *XDI type dictionary* is extensible by all XDI users.

For XDI literals, three main branches of the XDI type dictionary have been proposed:

- **$mime** will encompass the IANA-specified MIME media types.
- **$xsd** will encompass the W3C-specified XML Schema datatypes.
- **$json** will encompass the IETF-specified native JSON datatypes.

Each of these will be further specialized using simple conventions for mapping http: URI fragments to XRIs. Following are some examples:

```
$a$mime$text$html!
$a$mime$application$atom+xml!
$a$xsd$string!
$a$xsd$boolean!
$a$json$array!
$a$json$object!
```

XDI Variables

Operations on the XDI graph often need to refer to nodes in the graph for which the client does not yet know the XRI, or to a set of nodes in the graph meeting a certain criteria. Such operations need a special XRI that the server can recognize represents a variable and not a literal XRI.

The XDI variable identifier **($)** is defined for this purpose. Variables in any XDI document follow the same rule as all other XRIs in XDI documents: they must be unique within their context. Thus if more than one variable is needed in the same context, an XDI authority may assign unique variable identifiers as needed. A convention is to use digits, e.g., **($1)**, **($2)**, **($3)**.

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 composite XRIs to subtype standard XDI operations. Examples:

Operation XRI	Description
geta$mime$text$html!	Return the requested XDI resource as an HTML document.
geta$boolean!	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 an XRI (such as a persistent XRI i-number) to the new resource.

Link Contracts and XDI Authorization

One of the core design goals of XDI is to permit controls over XDI data sharing—*XDI authorization*—to be expressed within the XDI graph itself. This enables XDI permissions to be viewed, shared, moved, and managed just like any other part of the XDI graph. It also enables XDI authorizations to be fully portable across XDI service providers.

The portion of an XDI graph used to express authorization is called a *link contract*. The structure of a link contracts is based on using the XDI **$do** operator as a context. This **$do** context may be placed within any other context to create a link contract governing the sharing of data described by the link contract graph.

This pattern is the same for all XDI data sharing relationships, no matter who the XDI authority is, what data is being shared, what permissions are being granted, or what other policies are asserted as part of the link contract.

Link contracts are one of the most sophisticated structures in XDI graphs; they are most easily understood by viewing the XDI graph pattern examples explained in Part 4.

Part 3: XDI Serialization Formats

Transmitting XDI documents requires serializing the XDI graph. Like RDF, XDI may be serialized in multiple formats that will be specified in the *XDI 1.0 Serialization Specification*. Two serialization formats have been developed by the XDI TC:

- **XML** was the first format developed. However, because XDI data is already fully structured, the overhead of XML adds little value.
- **JSON** is the preferred serialization format for on-the-wire transmission due to its simplicity, compactness, efficiency, and popularity.

The example in this document will use JSON.

JSON Serialization Rules

The rules for serializing the XDI graph in JSON serialization format (specified using IETF RFC 3885 requirements keywords[5]) are.

1. An XDI JSON graph serialization MUST be valid JSON according to RFC 4627.[6]
2. The graph MUST be serialized as a single top-level JSON object ("root context object").
3. Every XDI statement in the graph MUST be serialized as a JSON object ("context object") contained within the root context object.
4. For every context object, the string representing the JSON object key MUST include of the first two segments of the XDI statement, i.e., the XRIs representing the XDI subject and XDI predicate. The key MUST include the forward slash separating the XDI subject and XDI predicate, and MUST NOT include a trailing slash after the XDI predicate.
5. For every context object, the value MUST be a JSON array.
6. If the XDI predicate is "!", then the predicate is a literal arc and the value of the JSON array MUST be interpreted as an XDI literal.
7. If the XDI predicate is "()", then the predicate is a contextual arc and the value of the JSON array MUST be an array of strings representing XRIs.
8. If the XDI predicate has any other value, then the XDI predicate is a relational arc, and:
    a. If a value in the array is a string, it MUST be interpreted as an XRI.
    b. If a value in the array is a JSON object, it MUST be interpreted as a nested XDI graph, in which all the XRIs MUST be cross-references.

Note that these serialization rules mean that the entire root context graph is effectively indexed by subject/predicate keys, and so are any nested graphs.

[5] http://www.ietf.org/rfc/rfc3885.txt
[6] http://www.ietf.org/rfc/rfc4627.txt

JSON Example

Following is an example XDI JSON document.

```
{
    "()/$is$is": [
        "(=!1111.2222.3333.4444)"
    ],
    "$d/!": [
        "2010-11-12T10:11:12Z"
    ],
    "=example/$is": [
        "=!1111.2222.3333.4444"
    ],
    "=!1111.2222.3333.4444/$is$a": [
        "+person"
    ],
    "=!1111.2222.3333.4444/+friend": [
        "=example2",
        "(mailto:friend@example.com)",
        "(http://example.com/friend)"
    ],
    "=!1111.2222.3333.4444+age/!": [
        33
    ],
    "=!1111.2222.3333.4444+vegetarian/!": [
        true
    ],
    "=!1111.2222.3333.4444+favorite+colors/!": [
        "red",
        "blue",
        "green"
    ],
    "=!1111.2222.3333.4444+address+street!1/!": [
        "123 Corliss Ave N"
    ],
    "=!1111.2222.3333.4444+address/+street!2/!": [
        "Apt 44"
    ],
    "=!1111.2222.3333.4444+address+city/!": [
        "Seattle"
    ],
    "=!1111.2222.3333.4444+address+state/!": [
        "WA"
    ],
    "=!1111.2222.3333.4444+address+postal.code/!": [
        "98133"
    ]
}
```

Part 4: Example XDI Graph Patterns

This portion of the document is currently maintained as two standalone files: *XDI Graph Patterns* and *XDI Statements for XDI Graph Patterns*. The latest versions of both documents (as well as of this document) are available from the following XDI TC wiki page:

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