# The XDI Graph Model

To meet the design goals in the preceding section, the XDI TC developed the semantic graph model defined in this and the following sections.

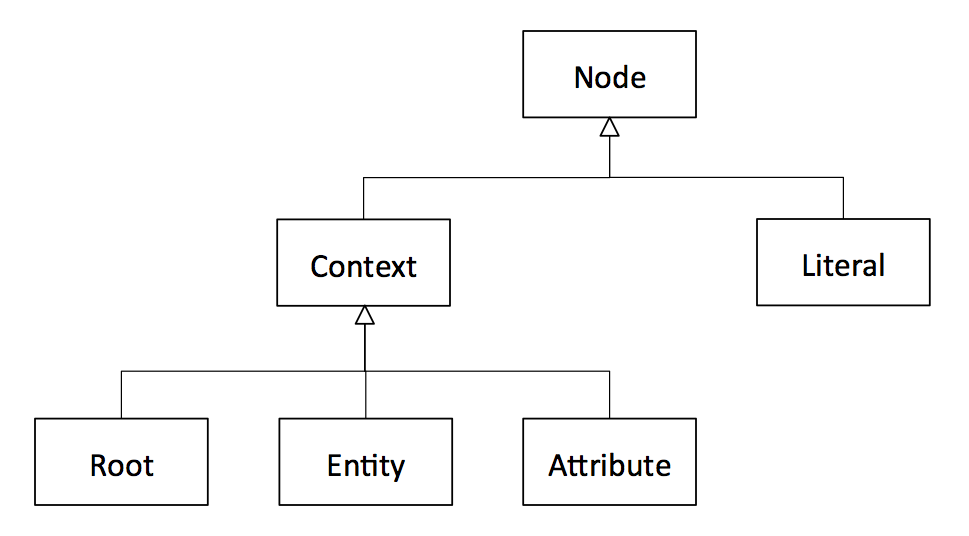
## Overview

The XDI graph model builds on the RDF *subject-predicate-object* triples model [ref]. This model in turn builds on the Entity-Attribute-Value (EAV) data model that dates back over 40 years. Note that in RDF, a graph node containing a data value is called a *literal*. So the RDF data model could also be termed an Entity-Attribute-Literal (EAL) model.

With RDF 1.1 datasets [ref], the model was expanded to *context-subject-predicate-object* quads. The fourth component—context—represents a named RDF graph. The XDI graph model also has a fourth component representing the root of an XDI graph. Thus it is called a Root-Entity-Attribute-Literal (REAL) model.

## Node Types

Figure \_\_\_ shows a simple UML class diagram (not an XDI graph) of the highest level node types in the XDI REAL graph model.



All graph nodes are one of two fundamental types: *literal nodes* or *context nodes*.

### Literal Nodes

As in RDF, XDI literal nodes are the terminal leaf nodes of the graph. They contain the raw data values described by all the other metadata in the graph. XDI natively supports the six data types defined by JSON [ref]:

1. Number (double-precision floating-point format in JavaScript)
2. String (double-quoted Unicode, with backslash escaping)
3. Boolean (true or false)
4. Array (an ordered, comma-separated sequence of values enclosed in square brackets; the values do not need to be of the same type)
5. Object (an unordered, comma-separated collection of key:value pairs enclosed in curly braces, with the ':' character separating the key and the value)
6. null (empty—note that this is not the equivalent of *undefined*, which is when an XDI attribute has no literal node at all)

In addition to the basic data type semantics provided by JSON, the type of a literal MAY be further described using one or more XDI type statements (see *Arc Types and Statement Types*).

### Context Nodes

All non-literal nodes in the XDI graph model are called *context nodes*. In RDF the term “context” is only used to describe the top level of semantic context available in the RDF 1.1 graph model, i.e., a named RDF graph. In addition, RDF blank nodes can be used to add a type of context to the relationship between other nodes. However, RDF does not use the term “context” for this purpose.

In XDI the term “context” is used uniformly across all levels of the REAL model to describe all forms of semantic context, including when:

* A graph root node provides context for another graph root node, an entity node, or an attribute node.
* An entity node provides context for another entity node or an attribute node.
* An attribute node provides context for another attribute node.

See *Contextual Arcs and Contextual Statements*, below.

All context nodes MUST have:

1. Exactly one *context type* identified in XDI syntax by a single *context symbol*.
2. One or more *context roles* identified in XDI syntax by zero or more pairs of *context brackets*.

### Context Types and Symbols

The XDI REAL model defines six global context types in three groups:

1. **Classes** represent entity and attribute types.
2. **Instances** represent entity and attribute individuals.
3. **Authorities** represent the two specific types of entities that are ultimately responsible for control of XDI graphs.

The context symbols for each type are shown in the table below:

|  |  |  |  |
| --- | --- | --- | --- |
| **Group** | **Context Type** | **Symbol** | **Also Known As** |
| Classes | Reserved | **$** | keywords, dollar words, |
| Unreserved | **#** | hashtags, dictionary words |
| Instances | Ordered | **@** | at numbers |
| Unordered | **\*** | thing names, thing numbers |
| Authorities | Personal | **=** | equals names, equals numbers |
| Legal | **+** | plus names, plus numbers |

A definition of each context type is provided in the *Entity* section below.

Note that XDI syntax also uses three other single-character symbols:

1. **&** (ampersand) to identify a literal arc (see *Literal Arcs and Literal Statements*, below).
2. **!** (exclamation mark) to indicate an immutable XDI identifier (see *Mutable and Immutable Identifiers*).
3. **/** (forward slash) to separate XDI subjects, predicates, and objects in XDI statement format (see *Statement Format*).

### Context Roles and Brackets

The XDI REAL model defines six context roles in two groups:

1. **Primary roles:** every context node MUST have exactly one primary role.
2. **Secondary roles:** depending on the context, a context node MAY have one or more secondary roles.

The context brackets for each role are shown in the table below:

|  |  |  |  |
| --- | --- | --- | --- |
| **Group** | **Context Role** | **Brackets** | **Also Known As** |
| Primary | Entity | none | plain, naked |
| Attribute | **< >** | chevrons |
| Root | **( )** | parentheses |
| Secondary | Collection | **[ ]** | square brackets |
| Definition | **| |** | pipes |
| Variable | **{ }** | curly brackets |

Each context role is defined in its own section below.

## Arc Types and Statement Types

An RDF graph is a labeled directed graph in which every predicate represents a directed arc from a subject node to an object node. Each RDF subject/predicate/object *statement* represents exactly one such arc.

The same is true of the XDI graph model, however in XDI, an arc MUST be one of three types:

1. **Literal arcs** describe the relationship between a context node and a literal node.
2. **Contextual arcs** define the identity, type, and role of one context node in the context of another context node.
3. **Relational arcs** describe any other relationship between two nodes.

Each type of arc is expressed using a specific type of XDI statement as defined in this section.

### Literal Arcs and Literal Statements

In the XDI REAL model, a literal node MUST be the object of exactly one *literal arc* expressed by exactly one *literal statement*. The subject of a literal arc MUST be an XDI attribute node. An XDI attribute node MUST have no more than one literal arc.

There are two key differences between XDI literal arcs and RDF predicates whose object is a literal node:

1. **In RDF, the semantic meaning of a literal is expressed by its predicate arc.** In XDI, the semantic meaning of a literal is expressed by the sequence of XDI attribute node(s) that precede the literal arc.
2. **In RDF, a literal may have its own datatype and language attributes.** In XDI, a literal node is always an atomic leaf node. Any other semantic description of a literal node MUST be expressed using one or more XDI type statements (see *Relational Arcs and Relational Statements*, below).

Because of the first difference above, an XDI literal arc is the semantic equivalent of the rdf:value property in RDF [ref]. Thus in XDI, all literal arcs MUST have the same XDI identifier: the ampersand character &. This is called the *literal symbol*. All XDI literal statements MUST use the literal symbol as the predicate. Examples:

|  |  |  |
| --- | --- | --- |
| **Subject** | **Predicate** | **Object** |
| =example<#email> | & | "foo@example.com" |
| +example<#main><#tel> | & | "+44-2222-888888" |
| \*!1234[<#event>]<@78><$t> | & | "2010-09-20T10:11:12Z" |

In XDI JSON serialization format:

{

"=example<#email>": {

"&": "foo@example.com"

},

"+example<#main><#tel>": {

"&": "+44-2222-888888"

},

"\*!1234[<#event>]<@78><$t>": {

"&": "2010-09-20T10:11:12Z"

}

}

Because an XDI attribute node may only contain one literal node, that literal node may be uniquely addressed by appending the literal symbol & to the XDI address of the attribute node. Examples:

|  |  |
| --- | --- |
| **XDI Address of Attribute Node** | **XDI Address of Literal Node** |
| =example<#email> | =example<#email>& |
| +example<#main><#tel> | +example<#main><#tel>& |
| \*!1234[<#event>]<@78><$t> | \*!1234[<#event>]<@78><$t>& |

### Contextual Arcs and Contextual Statements

In the RDF graph model, a blank node exists to provide context for other nodes, however a blank node does not have a URI. It can only be identified relative to the RDF graph in which it exists. [ref]

In the XDI graph model, all context nodes can provide context for other context nodes, and all context nodes are uniquely addressable. With the exception of the common root node, a context node MUST be the object of exactly one *contextual arc* expressed by a *contextual statement*. The subject of a contextual statement MUST be another context node, called the *parent node*. Only the common root node has no parent. The predicate of a contextual statement MUST be empty. The object of a contextual statement MUST have an XDI identifier that is unique in that context.

The result of these requirements is that XDI context nodes form a rooted directed acyclic graph, called a *semantic tree*, in which every node is uniquely addressable and every node has a semantic meaning. The absolute XDI address of a context node is the sequence of XDI identifiers for each contextual arc that must be traversed to go from the common root node to the target context node.

If the common root node of an XDI graph is itself assigned a URI, all nodes in the graph become globally addressable in the universal URI addressing space as recommended by [WebArch]. See the *XDI Addressing* section for details.

Following is an example of three contextual statements (each with the empty predicate) that establish the context for the final literal statement. In this example, =example and #car are XDI entities; <#interior> and <#color> are XDI attributes.

|  |  |  |
| --- | --- | --- |
| **Subject** | **Predicate** | **Object** |
| =example |  | #car |
| =example#car |  | <#interior> |
| =example#car<#interior> |  | <#color> |
| =example#car<#interior><#color> | & | “black” |

Figure \_\_\_ shows the same set of statements in XDI JSON serialization format. Note that when serialized the empty predicate in a contextual statement is represented by two forward slashes:

{

"=example": {

"//": [

"#car"

]

},

"=example#car": {

"//": [

"<#interior>"

],

"<#interior>": {

"//": [

"<#color>"

]

},

"<#interior><#color>": {

"&": "black"

}

}

}

Contextual statements are inherent in the XDI addresses of the subjects and objects of literal or relational statements. Therefore contextual statements are not included in the JSON serialization by default and are only added if they are explicitly requested using the \_\_\_\_\_\_\_\_\_\_\_\_\_ parameter (see the *Serialization* section for details). Figure \_\_\_ shows the same example graph without the contextual statements:

{

"=example#car": {

"<#interior><#color>": {

"&": "black"

}

}

}

### Relational Arcs and Relational Statements

Any relationship between two XDI graph nodes that is not described by a literal or contextual arc is described by a *relational arc* expressed by a *relational statement*. The predicate of a relational statement MUST be a sequence of one or more XDI entities.

XDI relational arcs are the equivalent of RDF properties that describe the relationship between two RDF resources. Examples:

|  |  |  |
| --- | --- | --- |
| **Subject** | **Predicate** | **Object** |
| =person-1 | #friend | =person-2 |
| =person-1 | #friend | =person-3 |
| =person-1 | #best#friend | =person-3 |
| =person-1 | #employer | +example.company |
| [#device]\*!:uuid:1234 | #owner | =person-1 |

In the XDI JSON serialization, a predicate of a relational arc is prefixed with a forward slash character:

{

"=person-1": {

"/#friend": [

"=person-2",

"=person-3"

],

"/#best#friend": [

"=person-3"

],

"/#employer": [

"+example.company"

]

},

"[#device]\*!:uuid:1234": {

"/#owner": [

"=person-1"

]

}

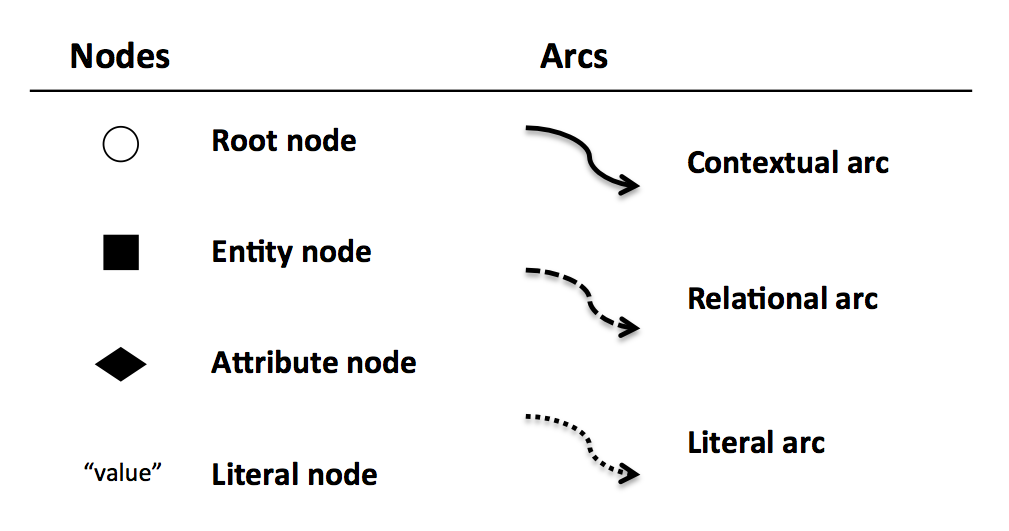
}

Relational statements may also be used to assert type or subclass relationships. In an XDI *type* or *subclass statement*, the subject is the context node being described, the predicate is $is#, and the object is the class of which the subject is a member (corresponding to rdf:type), or the class of which the subject is a subclass (corresponding to rdfs:subClassOf). Examples:

|  |  |  |
| --- | --- | --- |
| **Subject** | **Predicate** | **Object** |
| #sedan | $is# | #car |
| #car | $is# | #vehicle |
| \*!:uuid:1234 | $is# | #car |
| =person-1 | $is# | #carpenter |
| <#email> | $is# | $string |

## Visual Graph Diagramming Notation

For consistency across implementations, the XDI Technical Committee RECOMMENDS the notation shown in Figure \_\_\_ for visual diagramming of XDI graphs.



The root node symbol (a circle) is suggestive of the parentheses ( ) used in XDI syntax, and the attribute node symbol (a diamond) is suggestive of the chevron brackets < >. The root node symbol is open to represent that an XDI graph is only a container of XDI statements. The entity and attribute node symbols are solids to represent concrete identities and properties.

For diagrams that support color, it is RECOMMENDED to use:

1. A red outlined circle for the common root node.
2. A blue outlined circle for a peer root node.
3. A green outlined circle for an inner root node.

Literal nodes are a direct representation of the JSON value. If the value is truncated to save space, it is RECOMMENDED that the portion shown end in ellipses.

All contextual and relational arcs MUST be labeled. A literal arc MAY be labeled with the ampersand symbol, but it is not recommended. For a contextual arc, the label MUST be the unique XDI identifier of the object context node. For a relational arc, the label MUST be the predicate of the relational statement it represents.

Since there are many ways to organize an XDI graph diagram that uses this notation, the following two forms are RECOMMENDED:

1. **Free form.** In this organization, the common root appears roughly in the center of the diagram, and arcs are arranged radiating outward from it so as to best communicate the semantic information in the graph.
2. **Tree form.** This organization mimics a typical file or directory tree layout. The common root node appears in the upper-left-hand corner, contextual and literal arcs follow a grid, and only relational arcs are curved.

The choice of form depends on the particular XDI graph being shown. It is RECOMMENDED that viewing/editing tools support both forms and enable viewers to switch between them dynamically.

Figure \_\_\_ shows the example XDI graph from the Introduction section in free form.

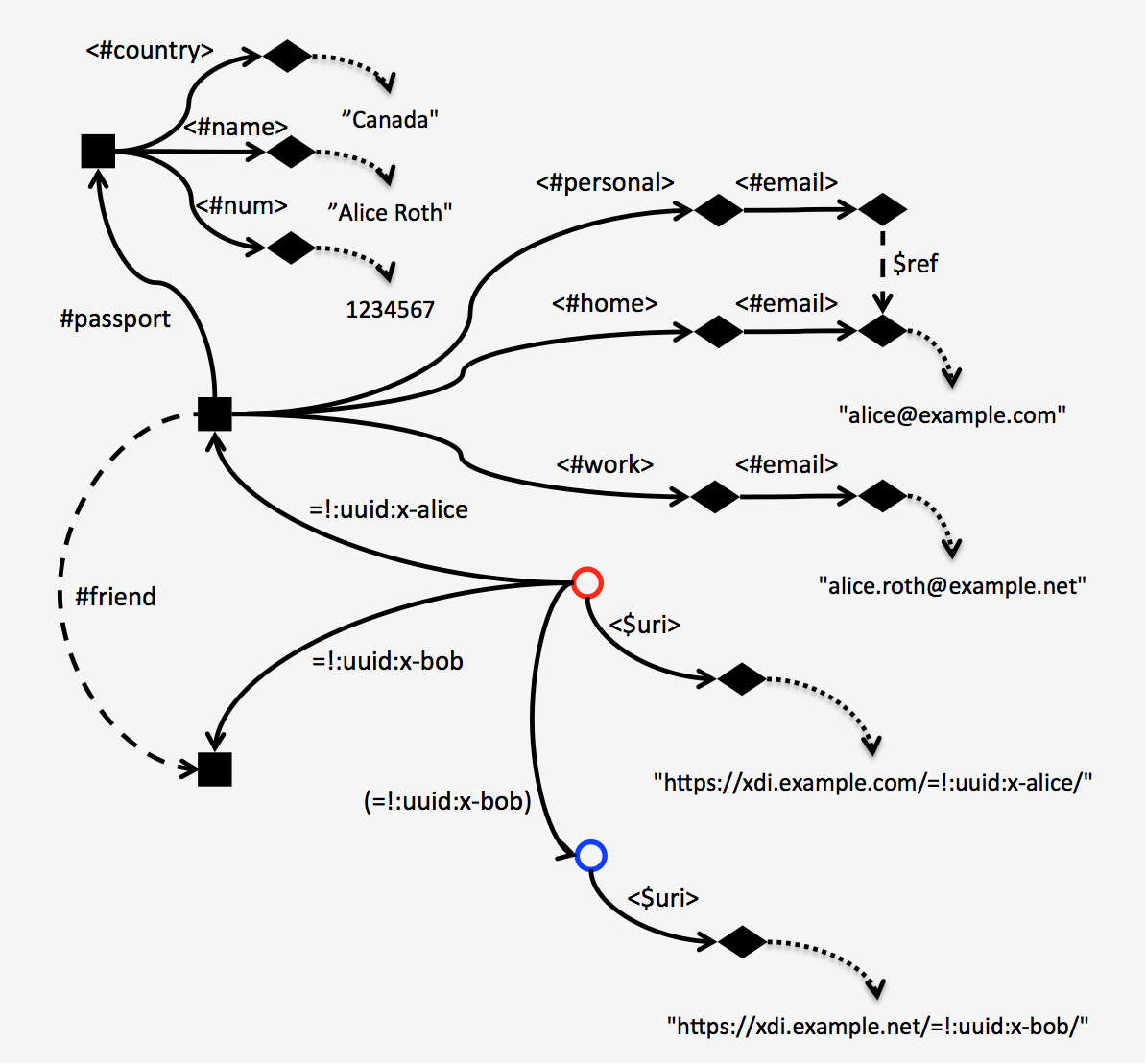
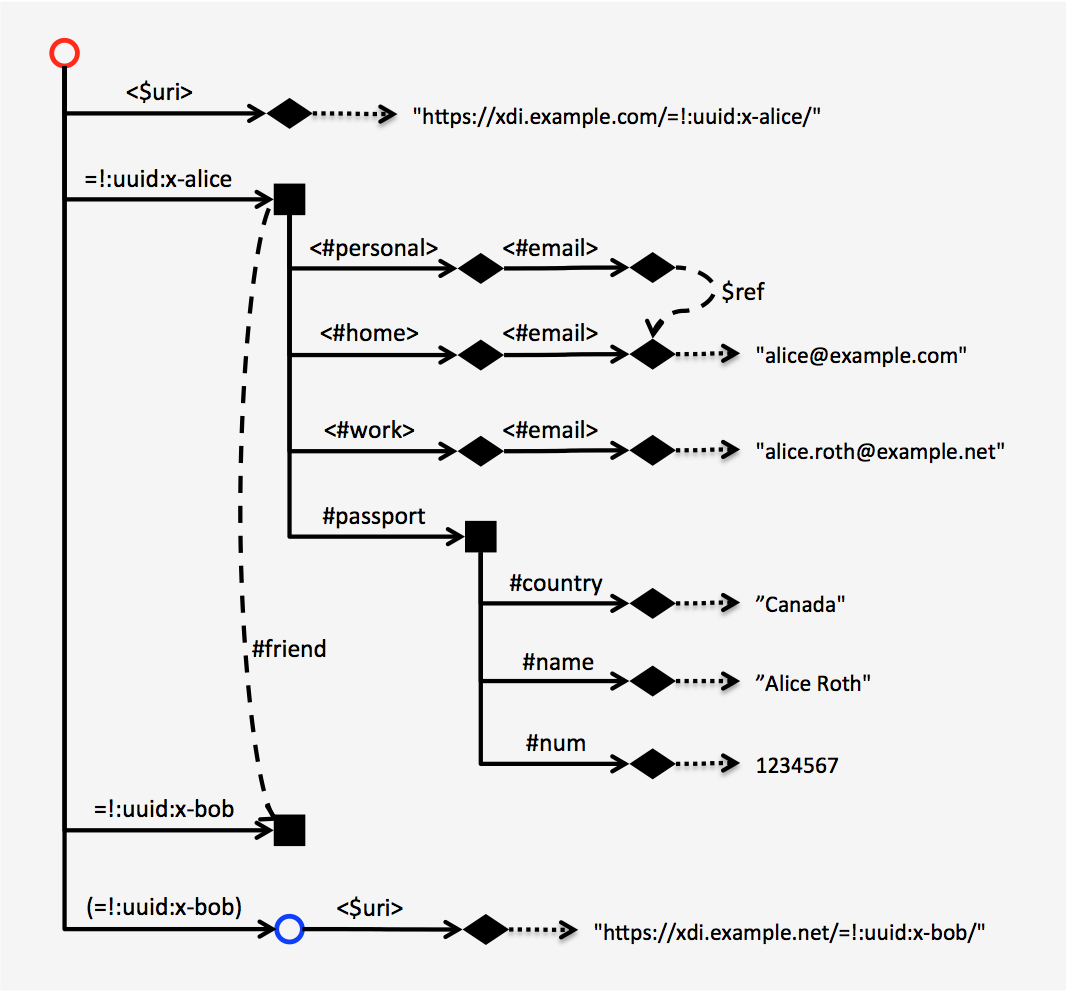


Figure \_\_\_ shows the same graph in tree form.



# Entities

In the XDI REAL model (and the Entity-Attribute-Value model upon which it is based), an *entity* is anything that can be identified and described independently, whether tangible or intangible (except an XDI graph itself). An entity may represent a person, group, organization, physical or digital object, concept, definition, or even a variable that may itself represent any set of these things.

From a linguistic perspective, entities are the “nouns” of XDI. However, this does not mean an entity is the only type of node that can serve as the subject of an XDI statement. In the XDI REAL model, either a root node (representing an entire XDI graph) or an attribute node may also serve as an XDI subject (and both are disjoint from entities). Thus an XDI entity is not exactly the same thing as an RDF resource—the latter may be anything with a URI (which would include XDI root and attribute nodes).

XDI entities fall into three groups: classes, instances, and authorities.

## Classes

A *class*, also known as a *concept* in description logic, is a set of entities that have some attribute(s) or propert(ies) in common. The set of entities belonging to the class are its *members*. In XDI, the instances of a class share the same *definition*.

XDI classes fall into two groups: reserved and unreserved.

### Reserved ($ Symbol)

A *reserved class* is a class defined by the XDI Technical Committee to establish the universal grammar of XDI. The goal of the XDI TC is to define the smallest set of reserved classes that produce the greatest degree of semantic interoperability across XDI graphs.

The XDI identifier of a reserved class MUST begin with the $ context symbol. The $ context symbol by itself represents the class of all reserved classes. Reserved class names are also known as *dollar words* or *keywords*. Examples:

$uri

$do

$and

$or

$not

$public

A reserved class name MUST be immutable and MUST NOT use the XDI immutability symbol. A reserved class name MUST be defined in a specification from either: 1) the OASIS XDI Technical Committee (including this specification), 2) another OASIS Technical Committee specified by the OASIS XDI Technical Committee, or 3) another standards body specified by the OASIS XDI Technical Committee.

The set of reserved class names of all non-negative integers (e.g., $0, $1, $2, $3…), also called *dollar numbers*, is reserved for named variables and MUST NOT be defined for any other purpose. See *Named Variables*.

### Unreserved (# Symbol)

An unreserved class is a class defined by any XDI authority other than the OASIS XDI Technical Committee or its specified delegate. The XDI identifier of an unreserved class MUST begin with the # context symbol. The # context symbol by itself represents the class of all unreserved classes. Unreserved class names SHOULD be defined in XDI dictionaries using XDI dictionary definitions (see *Definitions*, below). Unreserved class names are also known as *tags*, *hashtags*, or *dictionary words*. Examples:

#email

#passport

#home

#work

#friend

#enemy

An unreserved class name MUST be immutable and MUST NOT use the XDI immutability symbol.

Unreserved dictionary words MAY be defined by any XDI authority in any XDI context. Unreserved dictionary words whose semantics are intended to be confined to a specific set of XDI contexts SHOULD be defined by an XDI authority (a person or other legal entity) for those contexts and SHOULD be defined in that authority’s own XDI dictionary context. Dictionary words that are intended to be generic, i.e., to share the same semantics in all XDI graphs, SHOULD be defined directly in the common root context. See *Roots*, below.

This begs the question of authority for generic XDI dictionary words. Like the nouns in a human language, such words represent a community consensus about shared semantics. Thus it is RECOMMENDED that generic XDI dictionary words be specified in an XDI community dictionary cooperatively maintained by the XDI authorities contributing to that community.

This is the model popularized (and proven to scale) by Wikipedia for human-readable concept definitions, and also being followed by machine-readable community ontologies such as schema.org.

## Instances

An instance, also known as an *individual* in description logic, is a member of a class. In the XDI graph model, the XDI identifier for an instance node does not by itself convey any semantics about the class. It only conveys whether the instance is ordered or unordered in relationship to other instances in the same context. The relationship of an instance node to a class MAY be asserted in three ways:

1. **By putting an instance node in the context of a collection node.** By definition that instance is a member of the collection. See *Collections*, below.
2. **By describing an instance node with one or more XDI type statements.** See *Relational Arcs and Relational Statements*, above.

### Ordered (@ Symbol)

The XDI identifier of an ordered instance MUST begin with the @ context symbol. The @ context symbol by itself represents the class of all ordered instances. The identifier following the @ context symbol MUST be a non-negative integer, called the *order number*. The logical order of a set of ordered instances in a context MUST be by order number, beginning with the number zero if present. The document order of the ordered instances in a serialized XDI JSON document MUST be ignored. An example of ordered entity instances (in this case inside a collection):

{

"=example#favorite[#car]@0": {

"$ref": [

=example[#car]\*!:uuid:5dbb329c-2f26-4ef3-9297-dc95f42b2a40

]

},

"=example#favorite[#car]@1": {

"$ref": [

=example[#car]\*!:uuid:505dc1a6-5e65-417f-9e74-7fd218fabf6b

]

},

"=example#favorite[#car]@2": {

"$ref": [

=example[#car]\*!:uuid:36b894f5-2158-46cb-8492-934825d71985

]

}

}

Ordering also applies to attribute instances as shown in this example:

{

"=example": {

"<#pref>[<#email>]<@0>": {

"&": "alice#example.com"

},

"<#pref>[<#email>]<@1>": {

"&": "alice.roth@example.net"

},

"<#pref>[<#email>]<@2>": {

"&": "stillalice@example.org"

}

}

}

### Unordered (\* Symbol)

The XDI identifier of an unordered instance MUST begin with the \* context symbol. The \* context symbol by itself represents the class of all unordered instances. Unordered instance identifiers are also known as *star names* (mutable) or *star numbers* (immutable).

A set of unordered instance nodes in a context MUST NOT be interpreted as having any logical order regardless of their XDI identifiers or their document order in a serialized XDI JSON document.

An unordered instance may represent any “thing”, so the \* context is the XDI namespace for the Internet of Things. An unordered instance within a collection conveys the type of thing; an unordered instance outside a collection does not convey type information unless described by an XDI type statement.

Following is an example of unordered entity instances (in this case star numbers using the ! immutability symbol):

{

"+example[#item]\*!:uuid:f81d4fae-7dec-11d0-a765-00a0c91e58d1": {

"<#price>": {

"&": "24,995"

}

},

"+example[#item]\*!:uuid:9ce739f0-7665-11e2-bcfd-0800200cb439": {

"<#price>": {

"&": "36,995"

}

},

"+example[#item]\*!:uuid:3a96e460-7be9-f7e4-b92a-83d3c45e0ea4": {

"<#price>": {

"&": "18,495"

}

}

}

Following is an example of unordered attribute instances. This example also uses UUIDs for immutable identifiers. Note that these instance identifiers will not change even if the literal value changes.

{

"=example": {

"[<#email>]<\*!:uuid:35bcc3c0-da48-df9b-a16b-0002a5d557c4>": {

"&": "alice#example.com"

},

"[<#email>]<\*!:uuid:fbc71e40-da47-47a6-a00e-0002a5d577b5>": {

"&": "alice.roth@example.net"

},

"[<#email>]<\*!:uuid:62079220-da48-21cc-aca9-0002a5d51fe6>": {

"&": "stillalice@example.org"

}

}

}

## Authorities

Two specific classes are important enough to be distinguished by their own context symbol because they represent the entities ultimately responsible for control of XDI graphs:

1. Natural persons.
2. All other legal entities.

These classes are referred to as *XDI authorities*.

### Personal (= Symbol)

The XDI identifier of a natural person is called a *personal authority*. A personal authority MUST begin with the = context symbol (selected to suggest equality among peers). The = context symbol by itself represents the class of all personal authorities. Personal authority identifiers are also known as *equal names* (mutable) or *equal numbers* (immutable).

As explained in *XDI Addressing* (below), the XDI identifier for a personal authority may be either a *native reference* or an *external reference*. Examples of native references:

=example

=example-name

=example.name

=!:uuid:f81d4fae-7dec-11d0-a765-00a0c91e93c1

External references may be either to globally unique URIs or to locally unique identifiers. In either case they are encapsulated in parentheses. Examples:

=(https://example.name/)

=(mailto:foo@example.com)

+example.company=(local.name)

+(https://example.com/)=(local-name)

An XDI personal authority represents a new form of digital identity for individuals. This is referred to as *sovereign identity* because XDI’s heterarchical and contextual graph model enables an individual to interact with other XDI authorities (both personal and legal) as an independent autonomous peer. [ref to <http://blogs.law.harvard.edu/doc/2013/10/14/iiw-challenge-1-sovereign-identity-in-the-great-silo-forest/>].

### Legal (+ Symbol)

Administrative identity.

The XDI identifier of any legal entity other than a natural person is called a *legal authority*. A legal authority MUST begin with the + context symbol. The + context symbol by itself represents the class of all legal authorities. Legal authority identifiers are also known as *plus names* (mutable) or *plus numbers* (immutable).

Examples:

+example

+example-company

+example.org

+!: uuid:9ce739f0-7665-11e2-bcfd-0800200c18f2

+(https://example.com/)

+(mailto:division@example.com)

An XDI legal authority may represent any type of “legal person” that is not a natural person, including a group, association, sole proprietorship, partnership, corporation, or any type of governing, political, or social body. It may also represent any form of non-personal identifier to which legal rights may apply, including trademarks, trade names, service marks, trust marks, etc.

# Attributes < >

In the Entity-Attribute-Value (EAV) model, an *attribute* is a property of an entity that does not exist independently of the entity it describes. An attribute (and only an attribute) can have a literal value; an entity by itself cannot have a value.

In RDF subject-predicate-object graph model, an attribute of a resource node is described by a predicate whose object is a literal node. In RDF, an attribute is not required to be unique; a resource may have multiple predicates with the same URI describing multiple literal values for the same attribute (e.g., multiple email addresses for a person).

In the XDI REAL model, an attribute is represented by a context node in an attribute role. The XDI identifier of an attribute context node MUST be an entity class or entity instance enclosed in chevron brackets < >. Examples:

=example<#email>

+example-company<#support><#tel>

+(https://example.com/)#shipping#address<#city>

\*!: uuid:9ce739f0-7665-11e2-bcfd-0800200c18f2<#price>

Any type of XDI context node in any role (root, entity, attribute, collection, definition, variable) MAY have an attribute node. Note that attributes of a root node are attributes of that XDI graph as a whole and not attributes of any entity within that graph.

There are three reasons to model attributes as context nodes. First, it means all XDI attributes, like all XDI contexts, are uniquely addressable. This applies even if an entity has multiple values of the same attribute—this can be modeled as a collection where each instance is uniquely identified.

{

"=example": {

"[<#email>]<\*!:uuid:35bcc3c0-da48-df9b-a16b-0002a5d557c4>": {

"&": "alice#example.com"

},

"[<#email>]<\*!:uuid:fbc71e40-da47-47a6-a00e-0002a5d577b5>": {

"&": "alice.roth@example.net"

},

"[<#email>]<\*!:uuid:62079220-da48-21cc-aca9-0002a5d51fe6>": {

"&": "stillalice@example.org"

}

}

}

Secondly, attributes can specialize other attributes (see *Specialization and Generalization*). For example, <#home> and <#work> can be used to to specialize <#email>.

=example<#home><#email>

=example<#work><#email>

Thirdly, an attribute may itself have attributes. For example, to express the timestamp when the literal value of an attribute was assigned, add the <$t> attribute.

=example<#work><#email><$t>

{

"=example": {

"<#work><#email>": {

"&": "alice.roth@example.net"

},

"<#work><#email><$t>": {

"&": "2010-09-20T10:11:12Z"

}

}

}

As defined in *Literal Arcs and Literal Statements*, only an attribute node can have a literal node, and it can have exactly zero or one literal node. The semantics of the relationship between an attribute node, its literal node, and the value of that literal node are very precise:

1. **If an attribute node does not have a literal node**, then the value of that attribute is *undefined*.
2. **If an attribute node has a literal node and its value is the JSON null value,** then the value of that attribute is *null*.
3. **If an attribute node has a literal node and its value is an empty JSON string,** then the value of that attribute is *empty*.
4. **If an attribute node has a literal node and its value is any other JSON value,** then the value of that attribute is the literal JSON value.

The following example in JSON illustrates each of these four rules.

1. =example<#home><#email> is undefined. Note that in this case, the attribute node must be defined with an explicit contextual statement because no statement with the literal symbol will exist.
2. =example<#work><#email> is null.
3. =example<#work><#email><$t> is empty.
4. =example<#work><#email> has the literal JSON value false.

{

"=example": {

"<#home>": {

"//": [

"<#email>"

]

},

"<#work><#email>": {

"&": null

},

"<#work><#email><$t>": {

"&": ""

},

"<#employed>": {

"&": false

}

}

}

# Roots ( )

After entities and attributes, the third primary role for a context node in the XDI REAL model is to represent the root of an XDI graph. While there is exactly one ultimate root node for all XDI graphs—the *common root node*—due to the heterarchical design of XDI, it also has two other types of root nodes: *peer roots* and *inner roots*.

## The Common Root

Every XDI graph MUST have exactly one *common root node*. It is so named because it the one logical node shared by all XDI graphs. To use the analogy of trees in a forest, if every tree represents an XDI graph, the common root node is the earth.

The XDI address of the common root node is the *empty address*. Thus any XDI statement that does not begin with a peer root address or an inner root address is by definition relative to the common root node. The set of all XDI statements that are relative only to the common root node and not to a peer root node is called the *common graph*.

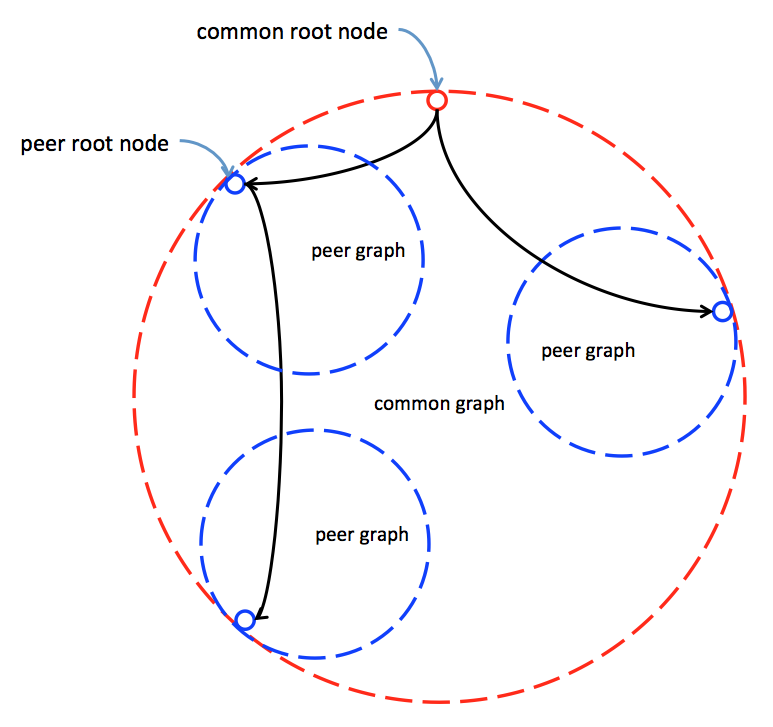
The common root node MUST NOT be the object of a direct contextual statement. It MAY be the object of an inverse contextual statement. See *Inverse Relations*.

The common root node of any XDI graph MAY describe the location of its own XDI endpoint using the <$uri> attribute as defined in the *XDI Discovery* specification [ref]. Note that attributes of a root node are attributes of an XDI graph as a whole and not attributes of any entity within that graph.

## Peer Roots

A *peer root node* is a context node in one XDI graph that represents the common root node of another separate XDI graph. This concept is fundamental to XDI architecture—peer root nodes are how the XDI graph model can represent peer-to-peer relationships between independent XDI graphs when each graph itself is a rooted tree. A node that serves as a peer root node in one XDI graph MUST serve as the common root node of its own XDI graph.

Peer root nodes may be nested to any depth within a single XDI graph. The set of contextual arcs describing these peer root nodes forms a hierarchical rooted tree. However each peer root node can be envisioned as a point on a global circle representing the logical common root node of all XDI graphs. Pick any specific starting point on this circle, and the references to the other starting points (peer roots) may be arranged hierarchically. However if you move to a different starting point, you will discover a different hierarchy. Each hierarchy represents the set of XDI peer root nodes known to a particular peer.



The graph contained by a peer root node is called a *peer graph*. A peer graph MUST be a subset of the independent XDI graph which the peer root node represents. Every peer graph is a subset of the logical XDI common graph. Thus the XDI statements in every peer graph must be logically consistent. (The same is not true for *Inner Graphs*, below).

The XDI identifier of a peer root node MUST be enclosed in parentheses ( ). The identifier contained within the parentheses MUST be either an XDI entity identifier or an absolute URI. Examples:

(=example)

(+example-company)

(\*!: uuid:9ce739f0-7665-11e2-bcfd-0800200c18f2)

(https://example.com/)

Like any other context node, peer root nodes MAY be nested to any depth. This enables XDI authorities to separate different XDI graphs at different XDI endpoints for different purposes and link them for the purpose of discovery. Examples:

(=example)(#household)

(+example-company)(#legal)(#mexico)

In keeping with the XDI REAL model, peer root and inner root nodes MUST precede entity or attribute nodes in the context tree. Like the common root node, a peer root node MAY use the <$uri> attribute to describe the network location of its XDI endpoint. Using the XDI protocol to discover the URI for XDI peer root nodes is defined in the *XDI Discovery* specification [ref]. Following is an example XDI graph from which the URI of two peer roots can be discovered:

{

"(=example)": {

"<#uri>": {

"&": "https://xdi.example.com/"

}

},

"(=example)(#household)": {

"<#uri>": {

"&": "https://xdi.example.com/household/"

}

}

}

The common root node of an XDI graph may also describe its own XDI address by virtue of an XDI equivalence statement to a peer root node. See *Equivalence Relations*.

{

"(=example)": {

"/$ref": [

""

]

},

"/$is$ref": [

"(=example)"

]

}In the XDI Messaging specification, the term “peer” is used for any an agent or actor that sends and receives XDI messages between XDI endpoints.

## Inner Roots

The third type of root node plays a very special role in XDI architecture. An *inner root node* represents the root of an XDI graph that is the object of an XDI relational statement. The graph contained by an inner root node is called an *inner graph*.

The XDI identifier of an inner root node MUST be enclosed in parentheses ( ) and MUST NOT be preceded by an XDI context symbol. The identifier contained within the parentheses MUST include the subject and predicate of the XDI relational statement whose object is the inner root node. The subject and predicate MUST be separated by a forward slash. Examples:

(=example/#nominated)

(+example-company/#hired)

(\*!: uuid:9ce739f0-7665-11e2-bcfd-0800200c18f2/#buyer)

In RDF terms, each context node in an inner graph represents a *reification* of an XDI statement. [ref] The subject and predicate of the statement are expressed by the XDI identifier of the inner root node. The object of the statement is the context node in the inner graph. Examples:

|  |  |
| --- | --- |
| **XDI Statement** | **Reified Statement** |
| =a/#b/=c | (=a/#b)=c |
| =alice/#buddy/=charlie | (=alice/#buddy)=charlie |

As in RDF, once a statement has been reified, other XDI statements can now be made about that statement. This pattern is very common in XDI graphs since reification can be used to describe any relationship between two entities. Examples:

(=a/#b)=c/#d/=e

(=alice/#buddy)=charlie/#dentist/=edith

(=example/#hired)=abc/#employer/+example-company

(+example-company/+acquired)+other-co<$year>/&/"2014"

An inner root node or a context node within it can also serve as the object of an XDI statement.

=a/#b/(=c/#d)=e

=alice/#buddy/(=charlie/#dentist)=edith

Like other context nodes, peer root nodes MAY be nested to any depth. This enables “statements about statements about statements”. Examples:

(=a/#b)(=c/#d)=e/#f/=g

(=alice/#buddy)(=charlie/#dentist)=edith/#friend/=greg

In the JSON serialization, when a sequence of peer roots and/or inner roots is an XDI subject, they are serialized as first level nested JSON objects. See *Serialization*.

{

"(=alice/#buddy)": {

"=charlie": {

"/#dentist": [

"=edith"

]

}

},

"(=example/#hired)": {

"=abc": {

"/#employer": [

"+example-company"

]

}

},

"(+example-company/+acquired)": {

"+other-co": {

"<$year>": {

"&": "2014"

}

}

},

"=alice": {

"/#buddy": [

"(=charlie/#dentist)=edith"

]

}

}

There is a critical distinction between peer graphs and inner graphs. Peer graphs are independent graphs that each contain a subset of the logical XDI common graph. By contrast every inner graph can only be understood in the context of the unique XDI subject/predicate relationship that defines it. Therefore XDI statements in inner graphs are not required to be logically consistent with statements in the logical XDI common graph. XDI statements contained by an inner graph are relative to its specific inner root node and can only be merged with another XDI graph by also merging that inner root node.

Therefore inner graphs are part of the XDI common graph and can be visualized as wholly contained “graphs within graphs”.

