Instance Record and Object Notation (irON) Specification

Specification Document - 18 February 2010

Latest version:
http://openstructs.org/iron/iron-specification

Last update:
$Date: 2010/02/18 12:32:43 $

Revision:
Revision: 0.9

Editors:
Frédérick Giasson - Structured Dynamics
Michael Bergman - Structured Dynamics

Authors:
Michael Bergman - Structured Dynamics
Frédérick Giasson - Structured Dynamics

Abstract

irON (instance record and Object Notation) is a abstract notation and associated vocabulary for specifying RDF triples and schema in non-RDF forms. Its purpose is to allow users and tools in non-RDF formats to stage interoperable datasets using RDF. The notation supports writing RDF and schema in JSON (irJSON), XML (irXML) and comma-delimited (CSV) formats (commON). The notation specification includes guidance for creating instance records (including in bulk), linkages to existing ontologies and schema, and schema definitions. Profiles and examples are also provided for each of the irXML, irJSON and commON serializations.

Status of This Document

NOTE: *This section describes the status of this document at the time of its publication. Other documents may supersede this document.*

This specification is an evolving document. Via its code and vocabulary release site, the authors welcome suggestions on the irON notation or its various serializations, including irXML, irJSON and commON. Users and developers are also welcomed to participate in the Google discussion group for the irON notation. The current specification is also available in download as a PDF .

This document may be updated or added to based on implementation experience, but no commitment is made by the authors regarding future updates.

Table of Contents

Abstract

Status of This Document

BACKGROUND AND OVERVIEW
- Purpose
- Terminology of this Document
- irON Concepts and Vocabulary
- The Attribute Concept
- The Type or Value Format Concepts
- The Record Concept
- The Dataset Concept
- The Schema Concept
- The Linkage Concept
- Relation to RDF
- Role and Choice of the Three Profiles

THE INSTANCE RECORD AND OBJECT NOTATION
- Introduction
- Profiles, Modules and Files
  - The Three Profiles of irXML, irJSON and commON
  - Modules or Sections
  - Files and MIME Types
  - The Special metaFile Attribute
- Vocabulary and Reserved Keywords
  - Standard, Reserved Vocabulary
  - Primitives

        Types
        General Object Types
        Other Reserved Terms
        A Note on User Interface Attributes
    Local or Global References
        Reference Resolution
    Dataset Object
        Core Dataset Attributes
        Abstract Dataset Specification Example
    Adding Metadata
        Suggested Metadata Attributes
    Instance Record Object
        Instance Record Attributes
        Abstract Instance Record Specification Example
    Structure Schema Object
        Structure Schema Attributes
        Abstract Structure Schema Specification Example
    Schema Linkage Object
        Schema Linkage Attributes
        Linking a Dataset to a Structure Schema
        Abstract Linkage Specification Example
    Processing Options
    Augmenting Attributes with Metadata
        Description of Use Case
        Metadata Example
        Limitations to Reified Metadata
    Guidelines for Dataset Scoping

SUB-PART 1: irXML PROFILE
    Role and Use
    No Current Processor
    Differences from Generic irON
        Augmenting Attributes with Metadata
        Altered Keyword Set
    Summary of Conventions
    XML File Structure
    Dataset Object
        Dataset Examples
            Using the metaFile Attribute
            Embedding the Metadata
    Instance Record Object
        Instance Record Example
    Structure Schema Object
        Structure Schema Example
    Linkage Object
        Linkage Example
        Linking a Dataset to a Structure or Schema Linkage
            Case #1
            Case #2
    Specific irXML Examples
        Example #1: First irXML Example
        Converting irXML into RDF

SUB-PART 2: irJSON PROFILE
    Role and Use
    Differences from Generic JSON
    Differences from Generic irON
    Summary of Conventions
    JSON File Structure
    Dataset Object
        Dataset Examples
            Using the metaFile Attribute
            Embedding the Metadata
    Instance Record Object
        Instance Record Example
    Structure Schema Object
        Structure Schema Example
    Linkage Object
        Linkage Example
        Linking a Dataset to a Structure or Schema Linkage
            Case #1
            Case #2
    Specific irJSON Examples
        Example #1: Bibliographic Record
        Example #2: Best Buy Mp3 Player
        Example #3: A irJSON Bibliographic Vocabulary to BibTeX Schema Linkage
        Example #4: A irJSON Bibliographic Vocabulary to RDF Schema Linkage
        Converting irJSON into RDF

SUB-PART 3: commON PROFILE
    Role and Use
    Differences from Generic CSV
    Differences from Generic irON
        No Schema Module, Other Changes
        Augmenting Attributes with Metadata

Specific Processing Options
Instance Record Presentation Styles
Reduced Keyword Set
General commON Design
Summary of Conventions
Dataset Object
Dataset Example
Instance Record Object
Instance Record Table Examples
Instance Record Row Style
Instance Records Stacked Style
Schema Linkage
Linkage Example
Downloadable Examples

ACKNOWLEDGEMENTS

REFERENCES

BACKGROUND AND OVERVIEW

This section provides background information on the specification and this document.

Purpose

irON (instance record and Object Notation) is a abstract notation and associated vocabulary for specifying RDF triples and schema in non-RDF forms. Its purpose is to allow users and tools in non-RDF formats to stage interoperable datasets using RDF. The notation supports writing RDF and schema in JSON (irJSON), XML (irXML) and comma-delimited (CSV) formats (commON). The notation specification includes guidance for creating instance records (including in bulk), linkages to existing ontologies and schema, and schema definitions. Profiles and examples are also provided for each of the irXML, irJSON and commON serializations.

irON is premised on these considerations and observations:

- RDF (Resource Description Framework) is a powerful canonical data model for data interoperability [1]
- However, most existing data is not written in RDF and many authors and publishers prefer other formats for various reasons
- Many formats that are easier to author and read than RDF are variants of the attribute-value pair construct [2], which can readily be expressed as RDF, and
- A common abstract notation for converting to RDF would also enable non-RDF formats to become somewhat interchangeable, thus allowing the strengths of each to be combined.

The irON notation and vocabulary is designed to allow the conceptual structure ("schema") of datasets to be described, to facilitate easy description of the instance records that populate those datasets, and to link different structures for different schema to one another. In these manners, more-or-less complete RDF data structures and instances can be described in alternate formats and be made interoperable. irON provides a simple and naïve information exchange notation expressive enough to describe most any data entity.

The notation also provides a framework for extending existing schema. This means that irON and its various serializations can represent many existing, common data formats and standards, while also providing a vehicle for extending them.

For different reasons and for different audiences, the formats of XML, JSON and CSV (spreadsheets) were chosen as the representative formats across which to formulate the abstract irON notation. Further rationale for these choices is discussed under their respective profiles below.

The abstract irON notation is written in a pseudo-XML syntax. Specific syntax examples for each of the three irON serializations are also provided in the code example sections for each format profile.

Terminology of this Document

The keywords "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC 2119].

Namespace URIs of the general form "http://www.example.com/." represents some application-dependent or context-dependent URI as defined in RFC 2396 [RFC 2396].

As used herein, the name and concept of *attribute* is used interchangeable with *property*. Both of these are equivalent to a *predicate* in an RDF triple. It is recommended that parsers for the various irON serializations recognize both terms (and their variants) interchangeably [3].

irON Concepts and Vocabulary

irON is based on a set of concepts that provide the common language of this specification:

- *Attribute* - each record (and other instances) is characterized by one or more attributes, which provide descriptive characteristics for that record. Every attribute is matched with a value, which can range from descriptive text strings to lists or numeric values. These values optionally have specified *formats*
- *Type* - "type" is a frequent term in many different language specifications. Here, *type* denotes a class of "things" that is a frequent container (or set) for classifying and describing things in relation to other things. Multiple entities or individuals may be members of a given *type*, such as *Person* or *Book*
- *Record* - a structured format for describing the attribute(s) and associated metadata and identifiers for a single bibliographic entity; a record is often defined with a standard structure or scope for conveying the description of multiple records
- *Dataset* - a set of one or more similar records that describe similar "things". The dataset is the normal unit ("container") of data exchange between different publishers and different consumers
- *Metadata* - dataset-level or record-level metadata that describes broad characteristics of the data itself, such as creation date or author, version number, etc.

- *Schema (structure)* - the conceptual relationships between types and for relating attributes, and
- *Linkage* (schema) - the specified mapping to relate the nature of class types and attributes in a given dataset (set of records) to other schema or structures.

This section describes these concepts in some detail.

In addition, irON has a set of reserved keywords that comprise its standard irON vocabulary. This vocabulary is composed of the minimal set of attributes used to describe “any” instance record in the world. There is a core set of attributes shared amongst all instance records. Some of these core attributes have been introduced for some specific purposes such as: user interface display, instance record identification and accessibility, instance record data maintenance, etc.

The Attribute Concept

Recall that **irON** has an ***attribute:value*** pair orientation. Any argument that appears in the first part of this pair is an *attribute*. In other various systems and languages, an "attribute" may also be known as a *property*, *predicate*, *field, feature*, *parameter*, *dimension*, *characteristic* or *independent variable*. As used herein, the name and concept of *attribute* is used interchangeable with *property* and *field*. These are equivalent to a *predicate* in an RDF triple [4].

An *attribute* in **irON** is the human-readable description for what its following value *means* and how it is to be interpreted. Some attribute keywords are classificatory in nature (in which case they are called *types*), some attributes are processing in nature (signalling a new module and specification to the parser), and some are designed to describe the nature of the instance, either for *records* or other objects.

Each attribute may be declared with an *allowedType*. An *allowedType* must be an Object or its various sub-types. An Object or its sub-types may be: 1) the general designation of *Object* itself (separate from a literal or string; see below); 2) a reference to a given class or set separately provided via the *type* declaration; or, 3) a keyword of *dataset*, *schema* or *Linkage* that signals a particular processing module to the parser.

The use of *allowedType* tells the **irON** validator whether an incoming record is valid or not according to its schema specification. (Note the declaration is optional for **irON** schema extensions; not providing the declaration simply means the parser is unable to check the proper domain of the instance.) It is the way that only properly specified data and records are accepted by the system.

Thus, *attributes* can be used in many areas of **irON**. However, their principle use is to provide the descriptive fields for characterizing records.

The Type or Value Format Concepts

The second part of the ***attribute:value*** pair orientation is the *value*. Depending on the kind of attribute that is declared in a **irON** schema, only certain kinds of values may be accepted for the value argument.

For most record descriptors, the *value format* for the value is a string (which can be further qualified as to the acceptable *format* of that string). In other cases, only specified values may be allowed.

When the attribute is *type* it denotes a class of "things"; that is, a container (or set) for classifying and describing things in relation to other things. *Types* can be assigned as descriptors for records. Multiple entities or individuals may be members of a given *type*, such as *Person* or *Book*.

By convention, *types* in **irON** are initial capitalized, such as *Person* or *Book*.

irON provides the *allowedValue* attribute to enforce proper type assignments or to ensure correct string formatting. As with *allowedType* (see above), these declarations are provided for the standard **irON** vocabulary, but are optional for extensions.

The Record Concept

The aim of **irON** is to describe *records*. A record is the main concept in the **irON** notation. A record is simply a means to represent and convey the information (”attributes”) describing a given instance. An instance is the thing at hand. These instances may be individuals (such as a given person or book) or may represent groupings of things (such as the entire holdings or collection of books in a given library).

A record may convey information about multiple instances, but each block of information ("record") for each instance should only pertain to that instance. Thus, for example, if the instance is a paper citation, the instance is the paper. If that paper instance asserts multiple authors, each with different institutional affiliations, those are attributes of the authors, not of the paper.

When you find you are attempting to describe multiple entities in a given record, one option is to create a separate record for each entity and assert their relationships. But this best practice may not always be possible or desired. As alternatives, **irON** also provides facilities to assign metadata in separate files (see tge soecuak *metaFile* attribute below) or to annotate primary attributes with their own explanatory metadata (see *Augmenting Attributes with Metadata below*).

When multiple records are being conveyed, they are treated as an array object and designated with the *recordList* keyword in any serialization profile.array.

The Dataset Concept

Records do not exist in a vacuum. The *dataset* provides a container object for records that allows additional information (the *metadata*) about records in the aggregate. Dataset information is not limited to the descriptions or attributes within the records themselves. A dataset can also be used to describe information about the creation of instances records, and to link external resources to them (like the *schema* structures and *linkages* discussed below).

A dataset can be seen as an aggregation of records used to keep a reference between the records and their source (provenance). A dataset can be split into multiple dataset *segments*. Each segment is written to a file serialized in some way. Each segment of a dataset shares the same `<id>` of the dataset.

A dataset is not a database, though a database can be a dataset. As a conveyance of similar records (that is, describing the same basic "things"), datasets have a consistency of scope. A circumstance where there are heterogeneous records for quite disparate "things" would suggest creating multiple datasets to homogenize those representations.

The Schema Concept

The *schema* structure is used to describe the structural relationships amongst the class *types* and *attributes* used to describe records. This schema aims to create basic taxonomies of *types* and *attributes* that can be used as a simple graph or network structure to perform simple reasoning over the record instances. The schema structure is also used to define any structural features of a dataset: the class types and formats of the attributes used to describe the records of the dataset.

The Linkage Concept

The schema *linkage* object is a specification that links the *types* and *attributes* used to describe records to types and attributes of other formats and languages used to describe data. The linkage schema leads to transformation rules to convert records in other formats. A set of special attributes has been created to define a schema linkage. Another keyword used for this linkages is the *mapTo* attribute.

Relation to RDF

The pivotal premise of irON is the desirability of using the RDF data model as the canonical basis for interoperable data. RDF provides a data model capable of representing any extant data structure and any extant data format. This flexibility makes RDF a perfect data model for federating across disparate data sources.

RDF is a data model that is expressed as simple *subject-predicate-object* “triples”. A *triple* is also known as a “statement” and is the basic “fact” or asserted unit of knowledge in RDF. Multiple statements get combined together by matching the *subjects* or *objects* as “nodes” to one another, with the *predicate*s acting as connectors or “edges” between those nodes. As these node-edge-node triple statements get aggregated, a network structure emerges, known as the RDF *graph* [1]. When these connections are coherent, the graph becomes a conceptual and schematic representation of the domain at hand and its relationships, and can be reasoned over and have other useful analysis done to it.

In irON, basic instance data is represented as simple attribute-value pairs where the *subject* is the instance itself, the *predicate* is the attribute, and the *object* is the value. Such instance records are also known as the *ABox*. The structural relationships within RDF are defined in *ontologies*, also known as the *TBox*, which are basically equivalent to a schema [4]. RDF vocabularies and schema guide how participating data can be represented and built up into more complex structures and conceptual world views.

The simple design of irON is in keeping with the limited roles and work associated with an ABox. Only attributes and metadata for an instance are being asserted. Conceptual relationships are dealt with separately via the Schema object (see the *Structure Schema Object* below). Specialized work, such as checking data validity, can be applied against these instance records, but is external to this specification.

The focus of irON, then, is the conveyance of these instance records (ABox) (though there are some limited provisions for communicating the TBox conceptual relationships and linkages to them; see below). The ability of irON to act as it does as an abstract notation across multiple, non-RDF data forms is based on this clean understanding of the roles of the ABox and TBox.

Role and Choice of the Three Profiles

RDF is not yet a common data model. And, in any case, RDF can be serialized with a number of formats such as XML, N3, N-triples, Turtle, or RDFa. However, despite these serialization options, and no matter the format, these RDF variants still are presented and organized around the "triples" construct of *subject* - *predicate* - *object*.

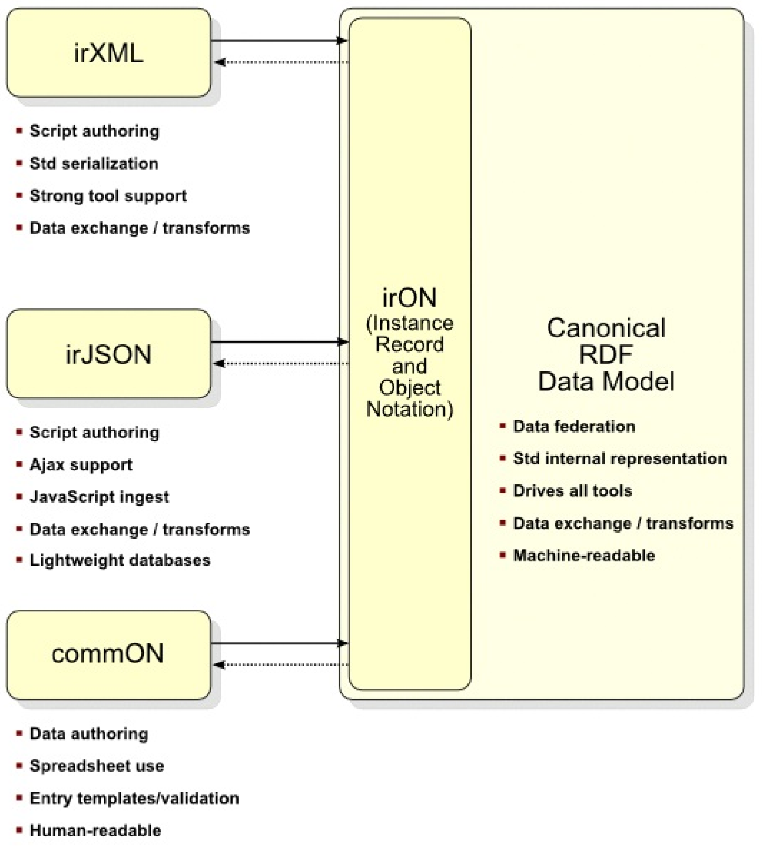
There are much more common data formats in the wild. In order to derive a properly inclusive abstract notation, then, it is important to select a number of these leading formats and to generalize around them. The derivation of the irON abstract notation and vocabulary is thus based on three leading data formats with a diversity of purposes, applications and user bases.

The first serialization selected is XML, or eXtensible Markup Language. XML has become the leading data exchange format and syntax for modern applications. It is frequently adopted by industry groups for standards and standard exchange formats. There is a rich diversity of tools that support the language, importantly including capable parsers and query languages. There is also a serialization of RDF in XML. As implemented in the irON notation, we call this serialization ***irXML***.

The second serialization selected is JSON, JavaScript Object Notation. JSON has become very popular as a Web 2.0 data exchange format and is often the format of choice to drive JavaScript applications. There is a growing richness of tools that support JSON, including support from leading Web and general scripting languages such as JavaScript, Python, Perl, Ruby and PHP. JSON is relatively easy to read, and is also now growing in popularity with lightweight databases, such as CouchDB. As implemented in the irON notation, we call this serialization ***irJSON***.

The third serialization is CSV, or comma-separated values. In existence for decades, but made famous by Microsoft as a spreadsheet exchange format, CSV is very useful since spreadsheets can be used as authoring front-ends and applications to the creation of datasets. CSV is less expressive and capable as a date format than the other irON serializations, yet still has a key-value pair orientation. And, via spreadsheets, datasets can be easily authored and inspected, while also providing a rich tools environment including sorting, formatting, data validation, calculations, etc. As implemented in the irON notation, we call this CSV serialization ***commON***.

The following diagram shows how these three formats relate to irON and then the canonical RDF target data model:



We have used the unique differences amongst XML, JSON and CSV to guide the embracing abstract notation within irON. This design makes RDF the canonical choice for driving all internal tools and services. Via transforms from external forms (and vice versa) RDF becomes the data *lingua franca* at the core of data interoperability systems.

Once all external data is converted into RDF, this internal representation can then be used for reverse transforms into the original form, a process known as "round-tripping". However, because RDF is the more capable data model, some internal RDF capabilities can not be transformed into these external formats. However, it is possible to transform the exact external input data back to its original form.

After the definition of the abstract irON notation itself, each of these three serializations — irXML, irJSON, and commON — is discussed under its own profile with examples below.

THE INSTANCE RECORD AND OBJECT NOTATION

irON (instance record and object notation) is an abstract notation and vocabulary for specifying datasets and instance records that can be converted to RDF under various serializations. The serializations themselves contain the syntax and necessary conventions for that specific format.

In general, each irON serialization — currently available as XML (irXML), JSON (irJSON) and comma-delimited CSV form (commON) — includes most of the abstract notations and vocabulary within irON. However, because of format-specific differences, there are portions of the notation that are not available to a specific serialization profile. These are noted in the individual profile descriptions.

Introduction

There are a number of "objects" or sections possible within an irON specification. These include: datasets; records; attributes; classes or types (*types*); a (structure) schema; and a schema linkage. Each of these is described below with abstract examples.

Profiles, Modules and Files

The Three Profiles of irXML, irJSON and commON

The abstract irON notation is expressed in three different serializations — irXML, irJSON and commON. Each has a different audience and often slightly different purposes.

Because of these differences, not all generic capabilities of irON nor all of its vocabulary may be applied to each serialization. This section and the following one on ***Vocabulary and Reserved Keywords*** describe these differences.

Further, after completion of the discussion of the abstract irON notation, major sub-parts follow that describe each of the three serializations in detail and present code and syntax examples.

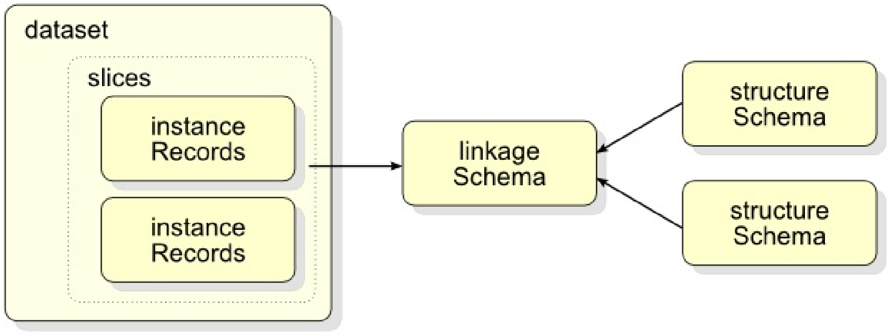
Modules or Sections

irON or its serialization specifications may occur in a number of modules, or sections. All are optional. Depending on the serialization, these modules or sections may also be provided or not in separate files.

The module components in irON are:

- datasets — these are the controlling structures. They have some core attributes that define their linkages to the governing schema and may optionally include metadata or links to a metadata file (via the *metaFile* attribute) that more generally describes the dataset. Most importantly, the dataset is the wrapper for instance records. When datasets are provided with an already used identifier (*id* attribute), the dataset acts as a "slice", which could represent new incremental record additions to a previously defined dataset
- records — instance records are the main vehicle for transmitting actual data. A *recordList* contains one or more records, each of which is described by a few or many attributes. Some of these attributes are reserved, but most can be freeform and define any useful data characteristic
- schema — this is the most structured of the modules, and has the capability to describe both the relationships amongst major concepts (classes, called *types* herein) in the structure and the attributes that help describe those classes and the instances that populate them. The schema provides separate means to describe the overall schema structure (including outline or hierarchical or taxonomic relationships and equivalences and linkages) and defining the types and formats for attributes [4]
- linkage — this "bridging" module provides the vocabulary for mapping dataset attributes and class types to one or more (internal or external) structural schema. Such structural schemas could be OWL ontologies, a relational database schema, or any other vocabularies used by other systems to describe and exchange data
- options — this module (not shown in the diagram) is for converter or parser instructions, and is not directly related to the actual data or values of a dataset or instance records.

The relationship between these modules is as follows:



In its current version, the irXML and irJSON serializations support all of these modules. The commON serialization does not at present support the schema module. In commON, one of the other serializations or RDF is necessary at this time to provide a structural schema definition.

Thus, here is the module coverage for the three irON formats:

	irXML	irJSON	commON
dataset	X	X	X
linkage	X	X	X
record	X	X	X
schema	X	X	
options			X
attributeList	X	X	X
typeList	X	X	X
prefixList	X	X	X
recordList	X	X	X
metaFile	X	X	X

Also note the *xxxList* entries in the table above. In irJSON, these entries are actually an array object. In irXML and commON, they are unordered lists processed in sequence until the listing ends.

Files and MIME Types

The MIME types for these serializations are `application/iron+xml`, `application/iron+json`, or `application/iron+csv`, respectively, for irXML, irJSON or commON.

Both irXML and irJSON can be packaged into single or multiple files, with keywords and conventions (see below) signaling the various modules. In commON, at present, all specifications must occur in a single file. For all three serializations, the file type should match the standard extension for that serialization. Namely, that is:

- `*.xml` - irXML
- `*.js` - irJSON
- `*.csv` - commON.

The Special metaFile Attribute

In addition to these modules, there is also a special *metaFile* attribute that enables descriptions such as creation specifics, author, etc., to be provided in a separate file. This facility can be helpful when multiple datasets or records re-use identical metadata descriptions. The value of the *metaFile* attribute must be a fully specified file reference.

Of course, such metadata may also be embedded directly in a dataset or instance record, avoiding the requirement for a separate file.

Vocabulary and Reserved Keywords

irON has a limited vocabulary. Each of the terms in this vocabulary is reserved from general use.

Standard, Reserved Vocabulary

The standard irON vocabulary consists of the following terms. Each term and its use is explained in later sections.

Avoid using any of these vocabulary or attribute names except for the purposes outlined herein.

Primitives

irON has two basic constructs for its assigned values: *primitives* and *types*. This section provides the primitive vocabulary.

Primitives are the most basic data structures of *irON*. In JSON Schema, we are referring to them as *formats* and in XML we are referring to them as *DataTypes*. These are the strings and integers of this world. At their cores, primitives are represented as a literal: a sequence of characters composing a whole. Each primitive is a value is a rule, or a set of rules patterns that tells how the primitive value should be formatted, presented or described. The only unrestricted literal is called the String primitive.

Primitives are different things depending on the **irON** serialization profile. For **irJSON**with its JSON serialization, primitives are referred to as *format*. (In the XML serialization of irON the similar concept is referred to as *datatypes*.) In JSON, primitives can be be validated by using JSON Schemas, and in XML value formats can be validated using XML Schemas. Primitives are thus a general concept applicable to most common data serializations.

Note that the list of primitives can be extended by specialized BibJSON parsers.

Value Format		Description
String		A String is a series of characters. A format can be defined for a string; such a format could be a date with a particular form, etc.
	Id	The Id basic type is a special case of a String. This String has to represent the ID (partial with @ or full with @@) of a record
	Uri	A URI identifier conforming to the specification document: RFC3986 - Uniform Resource Identifier (URI): Generic Syntax
	Url	A URL identifier conforming to the specification document: RFC1738 - Uniform Resource Locators (URL)
Integer		An integer value
Boolean		A boolean value. A boolean value can be described as being "0" and "1" or "true" and "false"
Float		A float value
Datetime		A datetime value conforming to the ISO 8601 standard specification
Date		A date part of a datetime value conforming to the ISO 8601 standard specification
Time		A time part of a datetime value conforming to the ISO 8601 standard specification
Mime		A mime type listed on the IANA mime types registry

Types

irON also has *types* of records. Each record has at least one type.

General Object Types

The more general type is the Object type. All other record types are sub-types of the Object type. The type of a record is asserted using the *type* attribute, or by the reserved processing keywords of *dataset*, *schema* or *linkage*.

Type		Description
Object		An Object is a record. The Object type is the more general record type that exists. All records of a dataset are Objects, but they can be more specific if the *type* attribute specify another type for a record
	Dataset	The Dataset object is a specialized Object and reserved keyword that signals the basic Dataset record
	Schema	The Schema (structure) object is a specialized Object and reserved keyword that signals the basic Schema (structure definition) record
	Linkage	The Linkage object is a specialized Object and reserved keyword that signals the basic Linkage record

Except for the three reserved sub-types, the predominate use of type is for classifying entity records via the *type* declaration. Inference can be performed on the hierarchy of *types*, which are themselves declared in the *schema* via the *subTypeOf* attribute (among other structural properties). This means that if we have a *Magazine* sub-type of *Periodical* that is a sub-type of *Collection*, then we can infer that a record of type *Magazine* is also a record of types *Periodical* or *Collection*.

Even if not defined in any *schema*, by convention all types are *subTypeOf* the *Object* root type.

Note the matrix also indicates whether the term is required or suggested (if not recommended, it is optional), what major module the term may belong to (shaded section), and whether the term applies to one of the three serializations:

	Require	Suggest	dataset	schema	linkage	record	irXML	irJSON	commON
addMapping					X		X	X	
allowedType				X			X	X	
allowedValue				X			X	X	

altLabel		X				X	X	X	X
attributeName					X	X	X	X	X
attribute					X		X		
attributeList				X	X	X	X	X	X
createDate			X				X	X	X
creator			X				X	X	X
curator			X				X	X	X
dataset			X				X	X	X
description		X	X			X	X	X	X
encoding				X			X	X	
equivalentPropertyTo				X			X	X	
equivalentTypeTo				X			X	X	
format				X			X	X	
href		X				X	X	X	X
id	X		X			X	X	X	X
language			X				X	X	
linkage			X		X		X	X	X
linkedType					X		X	X	X
listSeparator									X
listSeparatorEscape									X
maintainer			X				X	X	X
mapTo					X		X	X	X
maxValues			X				X	X	
metaData						X	X		
metaFile			X	X		X	X	X	
minValues			X				X	X	
orderedValues			X				X	X	
prefixList					X		X	X	X
prefLabel		X	X			X	X	X	X
prefURL		X	X			X	X	X	X
record						X	X		
recordList						X	X	X	X
ref						X	X	X	
schema			X	X			X	X	
seqNum									X
source			X				X	X	X
subPropertyOf				X			X	X	
subTypeOf				X			X	X	
typeName					X		X	X	X
type					X	X	X	X	X
typeList				X	X		X	X	X
updateDate			X				X	X	X
uri		X				X	X	X	X
version	X		X	X	X		X	X	X

Most all vocabulary applies to the irXML and irJSON serializations. The commON specification does not include vocabulary related to the (structure) schema and the *metaFile* attribute.

Another intent of the specification is to be sparse in terms of requirements. For instance, this reserved vocabulary is fairly minimal and optional in most all cases. The irON specification supports skeletal submissions.

Other Reserved Terms

Though not strictly prevented, it is best practice to avoid vocabulary in standard use for a given serialization. It is best to avoid target RDF or related standard vocabulary such as ***sameAs***, ***seeAlso*** or ***equivalentClass***, for example.

While the irON and related processors will accept these terms, they can prove problematic after ingest and conversion.

A Note on User Interface Attributes

As a general aid to user interfaces, we recommend some standard (shared amongst all datasets and instance records) attributes to describe datasets and instance records. These attributes have primary been introduced to help user interface systems to display, search, etc., these instance records.

UI-	

influenced Attribute	**Discussion**
prefLabel	The *prefLabel* attribute is used to describe human readable labels for datasets and instance records. It is a complement to the separate, alternative labels (*altLabel*) attribute*. prefLabel* by definition can not be a list or array. The *prefLabel* is a reserved attribute for the specific name or label you wish to be used and have appear in user interfaces about the thing at hand. It has no more importance than being your preferred designated label for that thing. Thus, as the main human readable label for that thing, format it according to what you would like to see in user interfaces. A *prefLabel* can be provided separately from other labels, such as *name*. Instead of creating big lists of attributes used to refer to different kinds of instance records (name, title, short title, given name, family name, etc...), the *prefLabel* of an instance record tells the system to display the preferred label within the user interface. So, when you describe an instance record of type *person*, you can describe this instance record with the *prefLabel* "Jim Pitman", along with the attribute *name* "James W. Pitman". You may have as many label attributes as you wish, but only prefLabel is the preferred one chosen for rendering in user interfaces.
description	Exactly the same mindset applies to the use of the *description* attribute. Often systems want to have short descriptions of the instance records they manage. However, instead of wanting to use a label to refer to an instance record, they want a description of this instance record. If an instance record is of type *person*, for example, its *description* could be the short biography of that person. If the type of the instance record is a *document*, then the description of this instance record could be its abstract. The same logic behind the *prefLabel* attribute is applied to the *description* attribute.
prefURL	Often systems can benefit from a reference to a Web page with additional information about an instance record. The use of the *prefURL* attribute is also recommended for user interface generation purposes. *prefURL* by definition can not be a list or array. *prefURL* is a URL reference to a Web page. It is the single URL chosen by the system for rendering a URL for a given thing within user interfaces. For whatever instance or object you are considering, think of the assignment to *prefURL* as representing the "best" human-viewable Web page available for it. If an instance record is of type *person*, its *prefURL* can be his personal Web page, a Web page with his CV, biography, etc. If the instance record is of type *organization*, its *prefURL* can be its enterprise Web page, etc. Like the *prefLabel*, the notion of "preferred" and "alternative" Web pages (*href*) also applies.

Local or Global References

To make irON specifications easier to read, this specification allows either local or global references to the locations of the cross-referenced objects. This section explains these conventions and how, when used, the references are resolved to their full address. The local ID of a record is local to its dataset. The global ID of a record is a URI. Each global ID should be resolvable on the specified network.

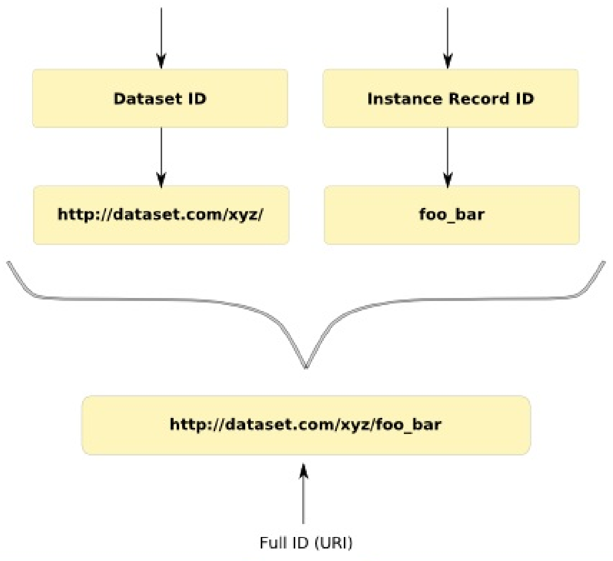
Reference Resolution

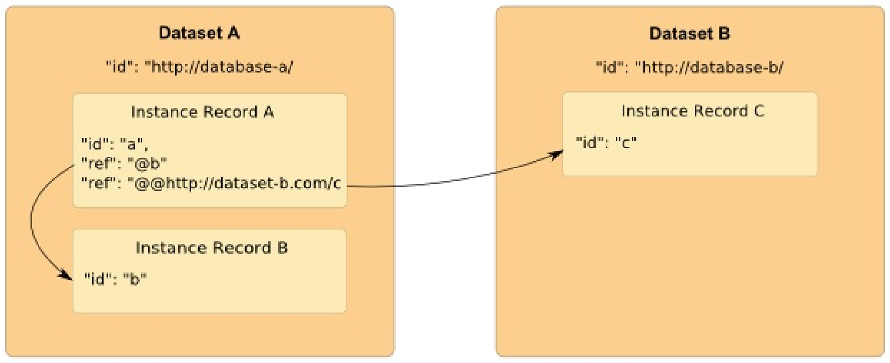
We have two kind of ids:

1. ID of a *Dataset*
2. ID of an *instance record*

The ID of an instance record is a partial ID local to the dataset. The ID of the dataset is the *base ID* used to create a complete reference ID of the instance records of the dataset. A full ID is created by concatenating the (base) ID of a dataset with the ID of an instance record. The full ID created has to be a valid URI.

If the value of an attribute starts with "@", it means that the value is a reference to a local ID. If the value os an attribute starts with "@@", it means that the value of the attribute refers to a global ID (URI). If the value refers to a local ID, it means that the record it references is in the same dataset. If the value refers to a global ID, it means that this instance record can be local, or remote. If it is remote, its representation should be resolvable on the network specified by the URI. Here is the figure describing this resolution mechanism:





Dataset Object

A Dataset is used to document information about the creation of instances records, and to link external resources to them (like the linkage and structure schemas; more about this below).

A Dataset can be seen as an aggregation of instance records used to keep a reference between the instance records and their source (provenance). A dataset can be split into multiple dataset slices. Each slice can be written in a separate file. Each slice of a dataset shares the same `<id>` of the dataset.

The *dataset* attribute introduces the Dataset object. This object is composed of multiple "string": "value" references. Each *string* refers to an attribute. Each *value* can be a string, an array of strings, or an object. The meaning and usage of each attribute is described below.

Core Dataset Attributes

These are the core attributes recommended to be included with any dataset or dataset slice specification.

Attribute/Keyword	**Requirement Note**	**Allowed Type(s)**	**Allowed Value(s)**	**Definition**
id	Required	Object	String	Identifier string used to uniquely identify the dataset
prefLabel	Recommended If the *prefLabel* of a dataset is not specified, systems displaying information about datasets will have a hard time figuring out how to present information about the dataset to users. If such a case happen, the system will have to fallback by displaying the ID of the dataset, or some generic label.	Object	String	Human readable label used to refer to this dataset. The *prefLabel* is the preferred label to use to refer to the dataset and is the preferred string used in the user interface. If not specified, the id is used as the label.
metaFile	Optional Not used if the actual metadata attributes are embedded in the dataset specification	Object	String *[format: uri (as a file reference)]*	This is a reference to an external record object; see next table for suggested dataset metadata. The reference to the file has to be a URI. If the file is local to a file system, the "file:" schema should be used. If the file is on the Web, the "http:" schema has to be used, etc.
schema	Optional	Dataset	▪ String *[format: url]*	URL reference where the structure schema can be

	The structure schema can be embedded in a dataset file or linked from the dataset description. If the structure schema is embedded and if a URL is specified, the schema with the biggest version will be used by the system. If both versions are the same, the system may use either one. If no structure schema is accessible, the system will ignore the specification.		▪ embedded schema linkage definition [object] ▪ Array(String *[format: url]*) ▪ Array(embedded schema linkage definition [object])	retrieved from the Web. Otherwise, a user can put the description of the schema in an object as the value of this attribute. More about this below.
linkage	Optional The schema linkage can be embedded in a dataset file or linked from the dataset description. If the schema linkage is embedded and if a URL is specified, the schema with the largest version number (most recent) will be used by the system. If both versions are the same, the system may use either one. If no schema linkage is accessible, the system ignores the extended capabilities it gives.	Dataset	▪ String *[format: url]* ▪ embedded schema linkage definition [object] ▪ Array(String *[format: url]*) ▪ Array(embedded schema linkage definition [object])	URL reference where the structure schema can be retrieved from the Web. Otherwise, a user can put the description of the schema in an object as the value of this attribute. More about this below.

Abstract Dataset Specification Example

Here is an example of an abstract dataset specification, with additional attributes beyond the core.

```
<dataset>
    <id />
    <prefLabel />
    <description />
    <source />
    <createDate />
    <creator />
    <curator />
    <maintainer />
        <prefLabel />
        <prefURL />
        <ref />
    <linkage />
    <schema />
</dataset>
```

Adding Metadata

The *metaFile* attribute refers to a separate instance record file. (Alternatively, the same attributes used for metadata be embedded in the dataset specification itself.) This dataset and slice design allows:

1. Reuse of the metadata (and its file) if desired
2. *de minimis* specification for a dataset slice as opposed to a fully specified dataset
3. A means for separately tracking dataset slice metadata and provenance from the metadata of the dataset itself
4. A simple dataset "wrapper" for dataset slices for adding or updating records to a dataset, and
5. A flexible and expandable metadata framework that piggybacks on the structure of an instance record.

Suggested Metadata Attributes

Note these attributes follow the general instance record object specification (see below), and may contain any arbitrary *attributeName* attributes as desired. Alternatively, as noted, these same attributes and values may be embedded within the dataset specification above.

Attribute	Requirement Note	Allowed Type(s)	Cardinality	Allowed Value(s)	Definition
id	Required	Object	[1]	primitive: Id	Identifier string used to uniquely identify the dataset
prefLabel	Recommended If the *prefLabel* of a dataset is not specified, systems displaying information about datasets will have a hard time figuring out how to present information about the dataset to users. If such a case happens, the system will has to fallback by displaying the ID of the dataset, or some generic label.	Object	[0-1]	primitive: String	Human readable label used to refer to this dataset. The *prefLabel* is the preferred label to use to refer to the dataset and is the preferred string used in the user interface. If not specified, the id is used as the label.
description	Recommended	Object	[0-1]	primitive: String	Human readable description of the dataset.
source	Optional	Dataset	[0-*]	primitive: Id	An attribute describing the source of the dataset. The description of the record is available within the array of records of the dataset.
createDate	Optional	Dataset	[0-1]	primitive: Datetime	Date of the creation of the dataset
creator	Optional	Dataset	[0-*]	primitive: Id	An attribute describing the creator of the dataset.

					The description of the record is available within the array of records of the dataset.
curator	Optional	Dataset	[0-*]	primitive: Id	An attribute describing the curator of the dataset. The description of the record is available within the array of records of the dataset.
maintainer	Optional	Dataset	[0-*]	primitive: Id	An attribute describing the maintainer of the dataset. The description of the record is available within the array of records of the dataset.

Instance Record Object

The *recordList* attribute refers to an array (irJSON) or unordered listing (irXML and commON) of instance records. An individual record is denoted by the attribute *record*.

Note: *The names of the attributes to be used in the instance records specification* ***must*** *be equivalent to the keywords shown unless otherwise indicated.*

Instance Record Attributes

Attribute	Allowed Type(s)	Cardinality	Allowed Value(s)	Basis	Description
id	Object	[1]	primitive: Id	irON	Identifier string used to uniquely identify the record.
type	Object	[1-*]	type: Object	irON	Class (type) of the record being described.
text	Object	[0-*]	primitive: String	BKN	literal string value, typically part of a longer string of text which may be taken as the literal string value. The value of "text" is typically intended to used as anchor text within html <a></a> tags in a typical html rendering of the field. Plain text without any intended hyperlink may be provided as an object in an array with a "text" property only. Text intended to be hyperlinked is provided as an object with a "text" property as well as usually a "ref" or "href" property, and optionally any number of additional properties.
prefLabel	Object	[0-1]	primitive: String	irON	Human readable label used to refer to this instance. The *prefLabel* is the preferred label to use to refer to the given instance and is the preferred string used in the user interface. If not specified, the id is used as the label.
altLabel	Object	[0-*]	primitive: String	irON	Human readable strings that are either synonyms, lingo, jargon, acronyms or other alternatives to refer to the instance and its preferred label. *altLabel* may also be used to help map or disambiguate instances.
metaFile	Object	[0-1]	primitive: Uri	irON	This is a reference to an external record object; see next table for suggested dataset metadata. The reference to the file has to be a URI. If the file is local to a file system, the "file:" schema should be used. If the file is on the Web, the "http:" schema has to be used, etc.
description	Object	[0-1]	primitive: String	irON	Human readable description of the record.
prefURL	Object	[0-1]	primitive: Url	irON	Often systems can benefit from a reference to a Web page with additional information about an record. The use of the *prefURL* attribute is also recommended for user interface generation purposes. *prefURL* is a URL reference to a Web page. For whatever instance or object you are considering, think of the assignment to *prefURL* as representing the "best" human-viewable Web page available for it. If an record is of type *person*, its *prefURL* can be his personal Web page, a Web page with his CV, biography, etc. If the record is of type *organization*, its *prefURL* can be its enterprise Web page, etc. Like the *prefLabel*, the notion of "preferred" and "alternative" Web pages (*href*) also applies.
href	Object	[0-*]	primitive: Url	irON	*href* is a single or list of valid URIs that refers to or describes the current instance (or attribute, if it is metadata being reified). In the absence of a *prefURL*, the first *href* encountered may be substituted as the user interface URL. When there are multiple values provided for *href* in combination with a *prefURL* attribute, they act as "see also" link references for the object.
uri	Object	[0-1]	primitive: Url	irON	For some instances or objects there may be a "canonical" address (an URI, or if viewable in a Webbrowser, an URL) that is the "official" reference to the thing. Sometimes this Web reference may only be machine readable; sometimes that is lacking and a standard Web page is appropriate, in which case *uri* is likely equivalent to the value for the *prefURL* attribute.
ref	Object	[0-1]	Id	irON	A *ref* attribute refers to the local ID if the first character of the id is "@". A *ref* attribute refers to a global ID (URI) if the first two characters of the id is "@@". If the *ref* attribute refer to the global ID of an record, this record can be local, or remote (this means that the record referred by the *ref*

					attribute is defined in another dataset).

Abstract Instance Record Specification Example

In pseudo-form, here is the set of core instance record attributes:

```
<record>
    <id />
    <type />
    <prefLabel />
    <description />
      <attributeList />
        <attribute1 />
        <attribute2 />
        <attribute3 />
        <attributeX />
            <prefLabel />
            <prefURL />
            <ref />
    </attributeList>
</record>
```

Multiple `<record>` objects can be wrapped in the `<recordList>` attribute.

Structure Schema Object

The structure schema is used to describe the structural relationships amongst the class types and attributes used to describe instance records. This schema aims to create basic taxonomies of *types* and *attributes* that can be used as a simple TBox to perform simple reasoning over the instance record instances [4]. The structure schema is also used to define any structural features of a dataset: the class types and formats of the attributes used to describe the instance records of the dataset.

Note 1: *The names of the attributes to be used in the schema specification* ***must*** *be equivalent to the keywords shown unless otherwise indicated.*

Note 2: *To enable the use of more complex TBoxes [4] for the instances records that have been described, the schema linkage has to be used to link the types and attributes of the instance records to the types and attributes of the more complex TBox format/language.*

Note 3: *At this time, the structure schema object is* ***NOT*** *available for the commON serialization. commON records can still be linked to schema (see next section), but the schema specification must occur in a non-commON manner.*

Structure Schema Attributes

Attribute/Keyword	Requirement	Allowed Type(s)	Cardinality	Allowed Value(s)	Description
schema	Required	Dataset	[1]	type: Object	This keyword introduces the schema object; that is, the structure schema specification. In its entirety, this object defines the structure schema.
version	Required	Schema / Linkage	[1]	primitive: String	This is a simple string defining the version of the schema. The version of a schema is defined by a simple decimal number " *X* . *Y* " where *X* and *Y* are greater than 0. The version number is used to resolve potential reference ambiguities.
metaFile	Optional	Object	[0-1]	primitive: Uri	This is a reference to an external record object; see next table for suggested dataset metadata. The reference to the file has to be a URI. If the file is local to a file system, the "file:" schema should be used. If the file is on the Web, the "http:" schema has to be used, etc.
typeList	Optional	Schema / Linkage	[0-1]	type: Object	Each element of the object is a class type with the name of the type to describe. The value of that attribute is an object. This attribute introduces an array or listing of these *types*. The object is a key-value pair with the attributes described below.
subTypeOf	Optional	Object	[0-*]	primitive: String	This attribute states that the parent "parent type string" is a sub-type-of the "value type string". With the example above, this means: "article" is a sub type of "book". If an array of types is specified, it means that the parent "parent type string" is a sub-type-of the union of all the "value type string".
equivalentTypeTo	Optional	Object	[0-*]	primitive: String	This attribute states that the parent "parent type string" is an equivalent-type-to the "value type string". If an array of types is specified, it means that the parent "parent type string" is a equivalent-type-to the union of all the "value type string".
attributeList	Optional	Schema / Linkage	[0-1]	type: Object	Each element of the object is an attribute with the name of the attribute to describe. The value of that attribute is an object. This attribute introduces an array or listing of these *attributes*. The object is a key-value pair with the attributes described below.

subPropertyOf	Optional	Object	[0-1]	primitive: String	This attribute states that a the parent "parent attribute string" is a sub-property-of the "value attribute string". With the example above, this means: "name" is a sub property of "label". If an array of attributes is specified, it means that the parent "parent attribute string" is a sub-property-of the union of all the "value attribute string".
equivalentPropertyTo	Optional	Object	[0-1]	primitive: String	This attribute states that the parent "parent attribute string" is an equivalent-property-to the "value attribute string". If an array of attributes is specified, it means that the parent "parent attribute string" is a equivalent-property-of the union of all the "value attribute string".
minValues	Optional	Object	[0-1]	primitive: Integer	Specifies the minimum number of values that an attribute can refer to. If the minValues is not specified, no minimal number of values is required.
maxValues	Optional	Object	[0-1]	primitive: Integer	Specifies the maximum number of values that an attribute can refer to. In irJSON, if a maxValues greater than 1 is specified, it means that the values of that attribute will be introduced by a JSON array. If the maxValues is not specified, no maximum number of values is required.
orderedValues	Optional	Object	[0-1]	primitive: String	Specifies if the values of an attribute are ordered or not. Possible values: (1) **ordered**, (2) **unordered**. If they are, the software that manage the dataset as to keep the order of the values for that attribute in order. **By default**, the value of the orderedValues attribute is **unordered**.
allowedType	Optional	Object	[0-1]	primitive: String	The *allowedType* attribute is used to specify how an attribute can be used used to describe a certain type of record. If the *allowedType* for an attribute is "Person", then this means that this attribute can only be used to describe records with a type "Person". If the *allowedType* is not specified for an attribute of the structure schema, we consider that it can be used to describe any type of record.
allowedValue	Optional	Object	[0-1]	primitive: String	The *allowedValue* attribute is used to specify what type of value an attribute can have. If the *allowedValue* for an attribute is "String", then this means that the value of the property can only be a string literal. If the *allowedValue* for an attribute is "Document", then this means that the value of this property can only be a reference to a record of type "Document". If the *allowedValue* is not specified for an attribute of the structure schema, we consider that this attribute can have any value. Cardinality of values can be introduced with the *Array(...)* processing keyword. Array(String) means that the value of that attribute can be an array of a specified *value format* or *type*
encoding	Optional	Object	[0-1]	primitive: String	Name of the encoding used to encode the characters of a string primitive of an allowed value of an attribute. By default, the encoding of a string primitive is UTF-8.
language	Optional	Object	[0-1]	primitive: String	Language identifier of a string primitive of an allowed value of an attribute. It has to be available on the IANA language subtags registry. This recommandation come from the W3C internalionalization task force. If the language attribute is not specified, any language can be associated to the string primitive.

Note 1: *If a type is not specified the type is assumed as "any" and no validation can be performed against its values.*

Note 2: *The structure Schema is also used by the system to list all the class types and attributes used to describe instance records from a particular dataset.*

Abstract Structure Schema Specification Example

```
<schema>
    <version />
    <metaFile />
    <typeList />
        <someTypeName />
               <subTypeOf />
               <equivalentTypeTo />
    <attributeList />
        <someAttributeName />
               <subAttributeOf />
               <equivalentAttributeTo />
</schema>
```

Schema Linkage Object

The schema linkage object is a new kind of specification that links the *types* and *attributes* used to describe instance records to types and attributes of other formats and languages used to describe data. The linkage schema leads to transformation rules to convert instance records in other formats. A set of special attributes has been created to define a schema linkage as described in the table below.

Note: *The names of the attributes to be used in the schema specification* ***must*** *be equivalent to the keywords shown unless otherwise indicated.*

Schema Linkage Attributes

Attribute/Keyword	Requirement	Allowed Type(s)	Cardinality	Allowed Value(s)	Description
linkage	Required	Dataset	[1]	type: Object	This keyword introduces the linkage object; the linkage specification to a schema. In its entirety, this object defines the schema linkage.
version	Required	Schema / Linkage	[1]	primitive: String	This is a simple string defining the version of the schema. The version of a schema is defined by a simple decimal number " *X* . *Y* " where *X* and *Y* are greater than 0. The version number is used to resolve potential reference ambiguities.
metaFile	Optional	Object	[0-1]	primitive: Uri	This is a reference to an external record object; see next table for suggested dataset metadata. The reference to the file has to be a URI. If the file is local to a file system, the "file:" schema should be used. If the file is on the Web, the "http:" schema has to be used, etc.
linkedType	Required	Linkage	[1]	primitive: Mime	Each schema linkage describe transformation rules between a vocabulary of a certain format to another vocabulary of another format. The *linkedType* attribute is used to specify the format of the target vocabulary. In the examples below, notice the use of the MIME types "application/rdf+xml" and "application/x-bibtex" as a value of the *linkedType* attribute. Parsers that will process these schema linkages have to know how to transform the schema linkage transformation rules, according to the *linkedType* MIME, to be able to properly serialize the rules in the destination format.
prefixList	Optional	Linkage	[0-1]	type: Object	Each element of the object is a "prefix:": "full-reference" key-value pair. In the schema, if the "foo:" prefix is encountered at the beginning of the value of the attribute *mapTo* , then the "foo:" prefix is substituted by its "full-reference". This is a way to make references smaller and easier for humans to read.
attributeList	Optional	Schema / Linkage	[0-*]	type: Object	A list of attributes used to describe the records of a dataset. They can be defined such that they link to external format/language attributes.
attributeName	Optional (any attribute identifier string allowed, except for the reserved keywords)	Object	[0-*]	type: Object	Name of the attribute used to describe records. This attribute is defined by a *type* attribute and an optional *mapTo* attribute.
mapTo	Optional If the *mapTo* attribute is omitted, this means that the attribute is only specified for listing (listing of attributes and types) purposes.	Object	[0-*]	primitive: String	A reference to the external format/language property that is identical as the one used to describe the records of this dataset. If the "foo:" prefix is used in the value of the mapTo attribute, it will be replaced by the full identifier of the attribute. If an array of values is specified, it means that the *attributeName* equally maps to the properties referenced by each of the value.
typeList	Optional	Schema / Linkage	[0-1]	type: Object	A list of *types* used to describe the records of a dataset. They can be defined such that they link to external format/language types.
typeName	Optional (any attribute identifier string allowed, except for the reserved keywords)	Object	[0-*]	type: Object	Name of the *type* used to describe records. This type is defined by an optional *mapTo* attribute.
mapTo	Optional	Object	[0-*]	primitive: String	A reference to the external format/language type that is identical as the one used to describe the records of this dataset. If the "foo:" prefix is used in the value of the mapTo attribute, it will be replaced by the full identifier of the type. If an array of values is specified, it means that the *attributeName* equally maps to the properties referenced by each of the value.
addMapping	Optional	Object	[0-1]	type: Object	Sometimes the mapping between two class types or between two different formats is not straightforward. Sometimes we have to add more key-value pairs to the linkage. This attribute is used to add "attribute": "value" tuple to the class

					types linkage transformation. For example, let's say that we have a "phdthesis" class type in our current schema, and that we only have a "bibo:Thesis" in the linked schema. The fact that the "thesis" is a "phd" thesis is described as an "attribute": "value" pair in the linked schema. The additional description that we have to add to the transformation is the tuple: "bibo:degree": "bibo_degrees:phd". Check the example #4 under the **BibJSON** profile below for a complete example of the usage of the *addMapping* attribute.

Linking a Dataset to a Structure Schema

There are two ways to link a dataset to a Linkage or structure Schema:

1. By using the *linkage* or *schema* to link a *dataset* to the description of these schemas.
2. By embedding the *linkage* or *schema* in specification.

***Note:** the list of "linkage" means that more than one linkage can be defined for a dataset. The goal is to enable data descriptors to be able to define schema linkages for multiple formats and languages if needed.*

Abstract Linkage Specification Example

```
<linkage>
    <version />
    <metaFile />
    <linkedType />
    <prefixList />
        <somePrefix />
    <attributeList />
        <someAttributeName />
               <mapTo />
    <typeList />
        <someTypeName />
               <mapTo />
</linkage>
```

Processing Options

For the commON serialization ***only***, some may prefer to present and manipulate their information with slightly different options for conventions. These processing choices are invoked via the *options* keyword. The current ones availalble in irON are the following:

Attribute	Description	Default
listSeparator	a single character used to separate items in a list entry	|
listSeparatorEscape	how the character gets escaped if it is present in a value without separating a list	%7C
seqNum	yes, no	yes

The *listSeparator* attribute sets what the delimiter is for a list of values for a single attribute. The default value is the pipe characters ('|'), though these characters are also possible: comma (','), semi-colon (';'), and squiggle ('~'). The data publisher has to specify a value for the *listSeparatorEscape* attribute to tell the commON processor engine how to escape/unescape the list separator character.

The *seqNum* attribute is a Boolean (yes, no) value. If set to yes, a value is added to the instance record listings that enables all values to be sorted in offline applications (especially spreadsheets for the commON serialization).

There are also a couple of instance record styles that are acceptable for commON, as described under its *Profile* below.

Augmenting Attributes with Metadata

The irON notation describes instance records using attribute/value key pairs. As we noted above, these can be mapped to *subject-predicate-object* triples in RDF since the subject is implied by the instance record itself. This means that all instance records are described using attribute/value *statements*. However, there are some cases where we could want to *state* something about these *statements*. These *statements* about *statements* are a form of metadata (which is called "reification" in the RDF realm [5].)

A *reification statement* is a statement about a statement. Generally, you can view a *reification statement* as being some kind of "meta" information about *statements*. However, since the "reification" terminology is a bit unusual for most people, we use "metadata" as a more understandable substitute.

Metadata can be used in multiple use-cases. It can be used to annotate information (information described using *statements*). It can be used to add specific information about a *statement* such as the date when it has been stated, the creator of the statement, etc. It can be used to describe information about how a specific statement should be rendered in some user interface. And, so forth.

One main metadata usecase for irON is its usage to specify how specific *statements* should be rendered in some user interfaces as described in the *A Note on User Interface Attributes* section.

Description of Use Case

Another use case can be shown where the instance, for example, is a paper citation. This paper citation has multiple authors. Though the instance is about the paper and not its authors, we may also want to state the institutional affiliation of each of the paper's multiple authors. We do this via the metadata ("reification") convention in irON.

Metadata Example

```
<record>
    <id />
    <prefLabel />
    <affiliated />
        <ref />
        <prefLabel />
        <prefURL />
</record>
```

For the example above, we have the abstract irON notation of a triple *id-affiliated-ref* where *id* is the ID of the subject record, and *ref* is a reference to the object record. Both records are related by the *affiliated* attribute (or relation). The *prefLabel* and *prefURL* attributes are *reification statements* or metadata about this triple statement (that is, they follow immediately after the *ref* reference).

If we take as the example that *Bob-affiliated-SomeUniversity* then the *prefLabel* could be the name of this affiliation to display in some user interface (Web page), and the *prefURL* could be a reference to a Web page that talks about this affiliation between Bob and http://someUniversity.edu.

This metadata that further describes the primary attributes (*affiliated*, in this case) with additional information can be an easy shorthand and provide immediately useful information to user interfaces (for example). This irON convention provides a mechanism, in essence, for expanding the depth and richness of the instance characterizations.

Limitations to Reified Metadata

Unfortunately, you are not always assured that systems ingesting such *reification statements* will actually recognize them or, more often, store them persistently. Further, once one begins describing metadata about primary attributes, the temptation is to nest those characterizations even further. If we can describe the author's institution, why not the city that institution is located in or whether it is public or private?

For these reasons, it may be safer (though more cumbersome) to devote a separate instance record to each author and more fully describe the author there. Still, when there is confidence in the processing application, it can be quite efficient to use the irON metadata shorthand.

The syntax for the irON attribute metadata shorthand differs by serialization.

Guidelines for Dataset Scoping

The irON notation is not applicable to all datasets nor all circumstances. However, there are a couple of guidelines that can extend the applicability and usefulness of this notation.

First, try to limit your datasets to relatively similar "things". If your problem domain at hand involves much data and relationships, try to segregate or cluster multiple contributing datasets according to the similarity of the instances (thus, people *v* products *v* organizations *v* localities *v* events, etc.).

Second, and related, try to scope each record within a dataset to the instance itself. References to external things or entities are fine and work great, but try to define the attributes of those external things into their own datasets.

In these manners you can keep attributes listings bounded and manageable. You will also keep datasets more understandable and without requiring massively dimensioned tables or structure. These considerations will also help bound the ability to create input templates with validation and controlled vocabularies based on existing applications.

SUB-PART 1: irXML PROFILE

This sub-part of the irON specification describes the eXtensible Markup Language (XML) serialization, ***irXML***.

Role and Use

The purpose of irXML is to provide a standard syntax and interchange format for the irON notation. Based on the eXtensible Markup Language (XML), irXML provides a syntax and serialization well understood by most enterprise developers.

Via the shared irON notation, irXML also offers a pathway for moving appropriate XML data structures into either RDF, JSON or CVS. As such, irXML is likely less an authoring environment as a notation for cross-format conversions.

The specifications provided herein can be used in separate, modular ways in multiple files, or combined. The linkage and structure schema provide useful flexibility and extensibility to the basic XML notation.

The MIME type for irXML is `application/iron+xml`; files written in it should have the `*.xml` extension.

No Current Processor

Unlike irJSON and commON, which have available parsers and processors, irXML has not yet been committed to code. As a result, its specification, while useful from an understanding and educational viewpoint, has not yet been tested with applications. This testing will likely result in changes.

We invite knowledgeable XML developers to tackle a conversion. The editors would be pleased to provide assistance to any group that wishes to incorporate this option.

Differences from Generic irON

The entire vocabulary and set of modules and objects in irON are available and used by irXML. For all *xxxList* attributes, irXML treats them as unordered lists, rather than arrays as in irJSON.

As a result, attributes for individual items must be used in irXML. These two specific irON attributes are *attribute* and *record*.

The *options* keyword is not used in irXML, nor its three attributes of *listSeparator*, *listSeparatorEscape*, and *seqNum* (it is advisable to avoid use of these terms so that conversions to commON are not confused).

Augmenting Attributes with Metadata

The method for adding metadata to primary instance attributes is in line with the standard irON notation:

```
<record>
    <id />
    <prefLabel />
    <affiliated />
        <ref />
        <prefLabel />
        <prefURL />
</record>
```

Altered Keyword Set

irXML has these vocabulary differences from the standard irON vocabulary:

Added	Not Used
attribute	options
record	listSeparator
metaData	listSeparatorEscape
	seqNum

It is recommended not to use the **Not Used** terms in a irXML specification as they might pose conflicts with other irON serializations. In addition, the meaning of format in irXML differs from irJSON in that the reference structure is the data types in XML Schema (XSD).

Summary of Conventions

1. irXML strictly conforms to the XML syntax
2. irON vocabulary keywords are reserved
3. All standard irON conventions and usages are followed
4. Listed items are provided as unordered lists, not arrays.
5. irXML files can be validated using XML schemas or DTDs
6. Any XML technologies can be used over irXML files such as XSL Transformations (XSLT)

XML File Structure

Each XML document is composed of a root dataset object. This root object wraps the standard dataset definition objects. The recordList element introduces a list of record elements which are records belonging to the dataset.

```
<dataset>
    ...
</dataset>

<recordList >
    <record />
    ...
</recordList>
```

Dataset Object

The *dataset* keyword introduces the Dataset XML object. This element is composed of multiple sub-elements describing the attributes of a dataset. Each sub-element refers to an attribute. Each value of each attribute element can be a literal (or something else if defined in a XML schema or a DTD), or a local or global reference to another instance record. The meaning and usage of each attribute is as described for the generic irON notation.

Dataset Examples

Using the metaFile Attribute

Here is an irXML dataset example using the *metaFile* attribute.

```
<dataset>
    <id>http://dataset.com/xyz/</id>

    <metaFile>http://dataset.com/abc/</metaFile>

    <linkage>http://dataset.com/schema/linkage.js</linkage>
    <schema>http://dataset.com/schema/structure.js</schema>
</dataset>
```

Embedding the Metadata

Alternatively, rather than invoke a separate instance record file with the metadata information, it can also be included with the dataset specification:

```
dataset>
    <id>http://dataset.com/xyz/</id>
    <linkage>http://dataset.com/schema/linkage.js</linkage>
    <schema>http://dataset.com/schema/structure.js</schema>
</dataset>

<recordList>
    <record>
       <id>http://dataset.com/xyz/</id>
       <prefLabel>Author Data</prefLabel>
       <description>Dataset bibliographic publications</description>
       <source>
          <ref>@ustanford</ref>
          <metaData>
             <prefURL>http://www.stanford.edu/</prefURL>
             <prefLabel>Stanford University</prefLabel>
          </metaData>
       </source>
       ...
    </record>
</recordList>
```

Instance Record Object

The *record* attribute wraps the specific attributes for each instance object. The *recordList* attribute wraps one or more records.

Instance Record Example

***Note:** The names of the attributes to be used in the instance records specification **must** be equivalent to the keywords shown unless otherwise indicated*:

```
<recordList>
        <record>
                <id>MR2463406</id>
                <type>Article</type>
                <prefLabel>Coloured loop-erased random walk on the complete graph</prefLabel>
                <year>2008</year>
                <number>6</number>
                <volume>17</volume>
                <mrclass>60G60 (05C80 60G50)</mrclass>
                <pages>727--740</pages>
                <href>http://dx.doi.org/10.1017/S0963548308009115</href>
                <author>
                        <ref>@MRauthor:855220</ref>
                        <metaData>
                                <prefLabel>Jim Pitman</prefLabel>
                                <altLabel>Pitman, Jim</altLabel>
                                <prefURL>http://www.stat.berkeley.edu/~pitman/</prefURL>
                        </metaData>
                </author>
                <author>
                        <ref>@MRauthor:855220</ref>
                        <metaData>
                                <prefLabel>Alappattu, Jomy</prefLabel>
                                <href>http://www.informatik.uni-trier.de/~ley/db/indices/a-tree/a/Alappattu:Jomy.html</href>
                        </metaData>
                </author>
                <journal>
                        <ref>@some_journal_id_1</ref>
                        <metaData>
                                <prefLabel>Combin. Probab. Comput.</prefLabel>
                                <prefURL>http://journals.cambridge.org/action/displayJournal?jid=CPC</prefURL>
                        </metaData>
                </journal>
        </record>

        <record>
                <id>some_journal_id_1</id>
                <type>Journal</type>
                <prefLabel>Combin. Probab. Comput.</prefLabel>
                <issn>0963-5483</issn>
        </record>
        <record>
                <id>MRauthor:855220</id>
                <type>Person</type>
                <prefLabel>Alappattu, Jomy</prefLabel>
        </record>
        <record>
                <id>MRauthor:855220</id>
                <type>Person</type>
                <prefLabel>Jim Pitman</prefLabel>
                <altLabel>Jim W. Pitman</altLabel>
        </record>
</recordList>
```

Structure Schema Object

XML Schema, DTDs and any other XML technologies such as XML Transformations (XSLT) can be used to define schemas (and processing tools) for describing instance records. They can be used to specify how attributes and type of objects should be described, using what attributes and what values.

The structure schema is used to describe the structural relationships amongst the types and attributes used to describe instances records, and is introduced via the *schema* keyword element. This schema aims to create basic taxonomies of types and attributes that can be used as a simple TBox [4] to perform simple reasoning over the instance record instances. The structure schema is also used to define any structural features of a dataset: the types and formats of the attributes used to describe the instance records of the dataset. The *typeList* element refers to a list of class types. Each class type is a XML element.

Structure Schema Example

Note 1: *The names of the attributes to be used in the schema specification must be equivalent to the keywords shown unless otherwise indicated.*

Note 2: *To enable the use of more complex TBoxes for the instances records that have been described, the schema linkage (see next) has to be used to link the types and attributes of the instance records to the types and attributes of the more complex TBox format/language.*

```
<schema>
        <version>0.1</version>

        <typeList>
                <Article>
                        <subTypeOf>Book</subTypeOf>
                </Article>
                <Book>
                        <subTypeOf>Document</subTypeOf>
                </Book>
                <Document>
                        <subTypeOf>Thing</subTypeOf>
                </Document>
        </typeList>

        <attributeList>
                <name>
                        <subPropertyOf>label</subPropertyOf>
                        <allowedValue>String</allowedValue>
                        <allowedType>Thing</allowedType>
                </name>
                <title>
                        <subPropertyOf>label</subPropertyOf>
                        <allowedValue>String</allowedValue>
                        <allowedType>Document</allowedType>
                </title>

        </attributeList>

</schema>
```

Linkage Object

The schema linkage is a new kind of specification that aims to link the class types and attributes used to describe instances records to class types and attributes of other formats and languages; it is introduced via the *linkage* keyword element. The schema linkage leads to transformation rules to convert instance records in other formats. A set of special attributes has been created to define this linkage as described in the table below.

Linkage Example

The names of the attributes to be used in the schema specification ***must*** *be equivalent to the keywords shown unless otherwise indicated.*

```
<linkage>

        <version>0.1</version>
        <linkedType>application/rdf+xml</linkedType>

        <prefixList>
                <bibo>http://purl.org/ontology/bibo/</bibo>
                <dcterms>http://purl.org/dc/elements/1.1/</dcterms>
        </prefixList>

        <attributeList>
                <year>
                        <mapTo>dcterms:created</mapTo>
                </year>
                <author>
                        <mapTo>bibo:authorList</mapTo>
                </author>
                <isPartOf>
                        <mapTo>dcterms:isPartOf</mapTo>
                </isPartOf>
        </attributeList>

        <typeList>
                <Article>
                        <mapTo>bibo:Article</mapTo>
                </Article>
        </typeList>

</linkage>
```

Linking a Dataset to a Structure or Schema Linkage

There are two ways to link a dataset to a Linkage or structure Schema:

1. By using the *linkage* or *schema* to link a *dataset* to the description of these schemas.
2. By embedding the *linkage* or *schema* in an XML object.

Here are two examples demonstrating each possibility:

Case #1

```
<dataset>
        <...> ... </...>

        <linkage>http://dataset.com/schema/linkage.js</linkage>
        <schema>http://dataset.com/schema/structure.js</schema>
<dataset>
```

Case #2

```
<dataset>
        <...> ... </...>

        <linkage>
                <...> ... </ ...>

                <attributeList>
                        <...> ... </ ...>
                </attributeList>

                <typeList>
                        <...> ... </ ...>
                </typeList>

        </linkage>
<dataset>
```

Note: *The list of "linkage" means that more than one linkage can be defined for a dataset. The goal is to enable data descriptors to be able to define schema linkages for multiple formats and languages if needed.*

Specific irXML Examples

Here is a set of examples that show you the irXML notation and vocabulary in action.

Example #1: First irXML Example

Here is an example of the description of a bibliographic record using irXML. This example demonstrates the publication of an article, in a book which is part of a series.

Note: *The irXML schema related to this example could be changed so that the information about the series, the book and the publisher are part of the description of the article. This has to be decided by the data publisher (the one that creates the dataset).*

```
<dataset>
        <id>http://bibserver.berkeley.edu/datasets/</id>
        <prefLabel>Publications of James Pitman</prefLabel>
        <description>Publications of James Pitman</description>
        <createDate>01/01/2008</createDate>

        <source>
                <ref>@ustanford</ref>
                <metaData>
                        <prefLabel>Stanford University</prefLabel>
                        <prefURL>http://www.stanford.edu/</prefURL>
                </metaData>
        </source>

        <creator>
                <ref>@jpitman</ref>
                <metaData>
                        <prefLabel>Jim Pitman</prefLabel>
                        <prefURL>http://www.stat.berkeley.edu/~pitman/</prefURL>
                </metaData>
        </creator>

        <linkage>http://dataset.com/schema/linkage.js</linkage>
        <schema>http://dataset.com/schema/structure.js</schema>
</dataset>

<recordList>
        <record>
                <id>MR2276901</id>
                <type>Article</type>
                <prefLabel>Two recursive decompositions of Brownian bridge related to the asymptotics of random
mappings</prefLabel>
                <description>Aldous and Pitman (1994) studied asymptotic distributions, as n tends to infinity, of various
functionals of a uniform random mapping of a set of n elements, by constructing a mapping-walk and showing these mapping-
walks converge weakly to a reflecting Brownian bridge. Two different ways to encode a mapping as a walk lead to two
different decompositions of the Brownian bridge, each defined by cutting the path of the bridge at an increasing sequence of
recursively defined random times in the zero set of the bridge. The random mapping asymptotics entail some remarkable
identities involving the random occupation measures of the bridge fragments defined by these decompositions. We derive
various extensions of these identities for Brownian and Bessel bridges, and characterize the distributions of various path
fragments involved, using the theory of Poisson processes of excursions for a self-similar Markov process whose zero set is
the range of a stable subordinator of index between 0 and 1.</description>
                <year>2006</year
                <pages>269--303</pages>
                <arxiv>math.PR/0402399</arxiv>
                <keywords>Path decomposition</keywords>
                <keywords>Path rearrangement</keywords>
                <keywords>Random mapping</keywords>
                <keywords>Combinatorial stochastic process</keywords>
                <bibnumer>117</bibnumer>
                <volume>1874</volume>
                <address>Berlin</address>
                <mrclass>60C05 (60J65)</mrclass>
                <author>
                        <metaData>
                                <prefLabel>Aldous, David</prefLabel>
                        </metaData>
                </author>
                <author>
                        <ref>@jpitman</ref>
                        <metaData>
                                <prefLabel>Jim Pitman</prefLabel>
                                <prefURL>http://www.stat.berkeley.edu/~pitman/</prefURL>
                        </metaData>
                </author>
                <isPartOf>
                        <ref>@book_id</ref>
                        <metaData>
                                <prefLabel>In memoriam Paul-André Meyer: Séeminaire de Probabilités</prefLabel>

                        </metaData>
                </isPartOf>
        </record>

        <record>
                <id>jpitman</id>
                <type>Person</type>
                <prefLabel>Jim Pitman</prefLabel>
                <prefURL>http://www.stat.berkeley.edu/~pitman/</prefURL>

                <name>Jim Pitman</name>
                <homepage>http://www.stat.berkeley.edu/~pitman/</homepage>
        </record>

        <record>
                <id>ustanford</id>
                <type>Organization</type>
                <prefLabel>Stanford University</prefLabel>
                <prefURL>http://www.stanford.edu/</prefURL>
        </record>

        <record>
                <id>book_id</id>
                <type>Book</type>
                <prefLabel>In memoriam Paul-Andrée Meyer: Séminaire de Probabilités</prefLabel>

                <title>In memoriam Paul-Andrée Meyer: Séminaire de Probabilités</title>

                <editor>
                        <metaData>
                                <prefLabel>Michel Émery</prefLabel>
                        <metaData>
                </editor>
                <editor>
                        <metaData>
                                <prefLabel>Marc Yor</prefLabel>
                        <metaData>
                </editor>

                <isPartOf>
                        <ref>@series_id</ref>
                        <metaData>
                                <prefLabel>Lecture Notes in Math.</prefLabel>
                        <metaData>
                </isPartOf>
        </record>

        <record>
                <id>series_id</id>
                <type>Series</type>
                <prefLabel>Lecture Notes in Math.</prefLabel>
                <volume>1874</volume>
                <publisher>
                        <metaData>
                                <prefLabel>Springer</prefLabel>
                        <metaData>
                </publisher>
        </record>

</recordList>

```

Converting irXML into RDF

This section is not yet drafted. It will explain how a irXML-to-RDF converter can be written, and how it is expected to behave. We will explain how a linkage schema can be used to create transformation rules that will take irXML statements and then create RDF triples by applying the rules defined in the linkage schema.

SUB-PART 2: irJSON PROFILE

This sub-part of the irON specification describes the JavaScript Object Notation (JSON) serialization, ***irJSON***. Its use and genesis was spurred by development of the BibJSON specification for the Bibliographic Knowledge Network project [6]. Technically speaking, BibJSON is a specific instantiation of the irJSON specification.

Role and Use

The purpose of irJSON is to enable datasets, instance records, data structures and the linkages between them to be specified using the JavaScript Object Notation (JSON). JSON is the native data input form for JavaScript Web applications and widgets and has become a common data exchange format and framework in its own right. JSON is a little cumbersome to write by hand, but is readily supported by all leading scripting languages with many libraries, validators, converters and editors extant for reading and ingesting JSON data.

Even the simplest key-value pair representation of an instance record needs some syntactic grammar and some interpretation conventions. In irJSON, the full range of the JSON syntax is used to serialize instance records. Some conventions are added along with the vocabulary to properly interpret the JSON syntax in the context of an irON instance record.

The specifications provided herein can be used in separate, modular ways in multiple files, or combined. The linkage and structure schema provide unlimited flexibility and extensibility to the basic JSON notation.

The MIME type for irJSON is `application/iron+json`; files written in it should have the `*.js` extension.

Differences from Generic JSON

The full range of the JSON syntax is used to serialize instance records. Some conventions are added along with the vocabulary to properly interpret the JSON syntax. This means that all the JSON rules and conventions are applied here: the data structures available, the encoding practices, etc. However, there are some terminology differences between the two notations. Here is a table that reifies the meaning of each concept:

irJSON Vocabulary Term	JSON term
string	string
array	array
object	object
record (instance record)	object
attribute	name

Though not an official part of the JSON notation, the schema logic within irJSON also builds and is consistent with the emerging JSON Schema effort. JSON Schema is a specification for a JSON-based format for defining the structure of JSON data. Furthermore, the formats supported by irJSON under the format attribute are the same as those specified for JSON Schema.

Properly formatted irJSON files will validate with the standard JSON validator, JSLint. Though there are JSON Schema validators also available, none of those are yet at a sufficient state of maturity to validate irJSON schema.

Differences from Generic irON

The entire vocabulary and set of modules and objects in irON are available and used by irJSON. For all *xxxList* attributes, irJSON treats them as objects with arrays, the same as JSON.

The *options* keyword is not used in irJSON, nor its three attributes of *listSeparator*, *listSeparatorEscape*, and *seqNum* (it is advisable to avoid use of these terms so that conversions to commON are not confused).

Summary of Conventions

1. irJSON strictly conforms to the JSON syntax
2. irON vocabulary keywords are reserved
3. All standard irON conventions and usages are followed.

JSON File Structure

Each JSON document is composed of a root JSON object. This root object is composed of a “dataset” string (attribute) and a “recordList” string (attribute). The "dataset" attribute introduces the Dataset object. The "recordList" attribute introduce an array of instance record objects.

```
{
    "dataset": {
        "...": "..."
    },
    "recordList": [
        {
            "...": "..."
        }
    ]
}
```

Dataset Object

The "dataset" attribute introduces the Dataset JSON object. This object is composed of multiple "string": "value" references. Each *string* refers to an attribute. Each *value* can be a string an array or an object. The meaning and usage of each attribute is as described for the generic irON notation.

Dataset Examples

Using the metaFile Attribute

Here is an irJSON dataset example using the *meta* attribute:

```
{
    "dataset": {
        "id": "http://dataset.com/xyz/",

        "metaFile":"http://dataset.com/abc/",

        "linkage": "http://dataset.com/schema/linkage.js",
        "schema": "http://dataset.com/schema/structure.js"
    }
}
```

Embedding the Metadata

Alternatively, rather than invoke a separate instance record file with the metadata information, it can also be included with the dataset specification directly:

```
{
    "dataset": {
        "id": "http://dataset.com/xyz/",

        "linkage": "http://dataset.com/schema/linkage.js",
        "schema": "http://dataset.com/schema/structure.js"
    },
    "recordList": [
        {
            "id": "http://dataset.com/xyz/",

            "prefLabel": "Author Data",
            "description": "Dataset bibliographic publications",
            "source": {
                "prefLabel": "Stanford University",
                "prefURL": "http://www.stanford.edu/",
                "ref": "@ustanford"
            },
            "createDate": "01/01/2009",
            "creator": {
                    "prefLabel": "Jim Pitman",
                    "prefURL": "http://www.stat.berkeley.edu/~pitman/",
                    "ref": "@jpitman"
                },
            "curator": {
                    "...": "..."
                },
            "maintainer": {
                    "...": "..."
                },

        }
    ]
}
```

Instance Record Object

The *recordList* attribute refers to an array of instance record(s). Each instance record is a JSON object.

Instance Record Example

Note: *The names of the attributes to be used in the instance records specification* ***must*** *be equivalent to the keywords shown unless otherwise indicated.*

```
{
        "recordList": [
            {
                "id": "MR2463406",
                "type": "Article",
                "prefLabel": "Coloured loop-erased random walk on the complete graph",
                "year": "2008",
                "number": "6",
                "volume": "17",
                "mrclass": "60G60 (05C80 60G50)",
                "pages": "727--740",
                "href": "http://dx.doi.org/10.1017/S0963548308009115",
                "author": [
                    {
                        "prefLabel": ["Pitman, Jim", "Jim Pitman"],
                        "prefURL": "http://www.stat.berkeley.edu/~pitman/",
                        "ref": "@MRauthor:855220"
                    },
                    {
                        "prefLabel": "Alappattu, Jomy",
                        "href": "http://www.informatik.uni-trier.de/~ley/db/indices/a-tree/a/Alappattu:Jomy.html",
                        "ref": "@MRauthor:855220"
                    }
                ],
                "journal": [
                    {
                        "prefLabel": "Combin. Probab. Comput.",
                        "prefURL": "http://journals.cambridge.org/action/displayJournal?jid=CPC",
                        "ref": "@some_journal_id_1"
                    }
                ]
            },
            {
                "id": "some_journal_id_1",
                "type": "Journal",
                "prefLabel": "Combin. Probab. Comput.",
                "issn": "0963-5483"
            },
            {
                "id": "MRauthor:855220",
                "type": "Person",
                "prefLabel": "Alappattu, Jomy"
            },
            {
                "id": "MRauthor:855220",
                "type": "Person",
                "prefLabel":"Jim Pitman",
                "altLabel":"Jim W. Pitman"
            }
        ]
}
```

Structure Schema Object

JSON schema can be used to define schemas for describing instance records (they also have a reference to common data type formats).They can be used to specify how attributes and type of objects should be described, using what attributes and what values.

The structure schema is used to describe the structural relationships amongst the *types* and *attributes* used to describe instances records. This schema aims to create basic taxonomies of types and attributes that can be used as a simple TBox [4] to perform simple reasoning over the instance record instances. The structure schema is also used to define any structural features of a dataset: the types and formats of the attributes used to describe the instance records of the dataset.

The *typeList* attribute refers to an array of class types. Each class type is a JSON object.

Structure Schema Example

Note 1: *The names of the attributes to be used in the schema specification* ***must*** *be equivalent to the keywords shown unless otherwise indicated.*

Note 2: *To enable the use of more complex TBoxes for the instances records that have been described, the schema linkage (see next) has to be used to link the types and attributes of the instance records to the types and attributes of the more complex TBox format/language.*

```
{
    "schema": {
        "version": "0.1",

        "typeList": {
                "Article": {
                        "subTypeOf": "Book"
                },
                "Book": {
                        "subTypeOf": "Document"
                },
                "Document": {
                        "subTypeOf": "Thing"
                }
        },

        "attributeList": {
                "name": {
                        "subPropertyOf": "label",
                        "allowedValue": "String"
                        "allowedType": "Thing"
                },

                "title": {
                        "subPropertyOf": "label",
                        "AlowedValue": "String"
                        "allowedType": "Document"
                }
        }
}
```

Linkage Object

The schema linkage is a new kind of specification that aims to link the class types and attributes used to describe instances records to class types and attributes of other formats and languages. The schema linkage leads to transformation rules to convert instance records in other formats. A set of special attributes has been created to define this linkage as described in the table below. In the example 3 and 4 below, we are demonstrating some linkage between a dataset and external formats such as BibTeX and RDFS/OWL ontologies properties and types.

Linkage Example

Note: *The names of the attributes to be used in the schema specification* ***must*** *be equivalent to the keywords shown unless otherwise indicated.*

```
{
    "linkage": {
        "version": "0.1",
        "linkedType": "application/rdf+xml",

        "prefixList": {
            "bibo": "http://purl.org/ontology/bibo/",
            "dcterms": "http://purl.org/dc/elements/1.1/"
        },

        "attributeList": {
            "year": {
                "mapTo": "dcterms:created"
            },

            "author": {
                "mapTo": "bibo:authorList"
            },

            "isPartOf": {
                "mapTo": "dcterms:isPartOf"
            }
        },

        "typeList": {
            "Article": {
                "mapTo": "bibo:Article"
            }
        }
    }
}
```

Linking a Dataset to a Structure or Schema Linkage

There are two ways to link a dataset to a Linkage or structure Schema:

1. By using the *linkage* or *schema* to link a *dataset* to the description of these schemas.
2. By embedding the *linkage* or *schema* in a JSON object

Here are two examples demonstrating each possibility:

Case #1

```
{
    "dataset": {
        "...": "...",
        "linkage": "http://dataset.com/schema/linkage.js",
        "schema": "http://dataset.com/schema/structure.js"
    }
}
```

Case #2

```
{
    "dataset": {
        "...": "...",
        "linkage":
         {
              "...": "..."
         }
    }
}
```

Note: *the list of "linkage" means that more than one linkage can be defined for a dataset. The goal is to enable data descriptors to be able to define schema linkages for multiple formats and languages if needed.*

Specific irJSON Examples

Here is a set of examples that show you the irJSON notation and vocabulary in action.

Example #1: Bibliographic Record

Here is an example of the description of a bibliographic record using irJSON. This example demonstrates the publication of an article, in a book which is part of a series.

Note: *the irJSON schema related to this example could be changed so that the information about the series, the book and the publisher are part of the description of the article. This has to be decided by the data publisher (the one that creates the dataset).*

```
{
    "dataset": {
        "id": "http://bibserver.berkeley.edu/datasets/",
        "prefLabel": "Publications of James Pitman",
        "description": "Publications of James Pitman",
        "source": {
            "prefLabel": "Stanford University",
            "prefURL": "http://www.stanford.edu/",
            "ref": "@ustanford"
        },

        "createDate": "01/01/2008",
        "creator": {
            "prefLabel": "Jim Pitman",
            "prefURL": "http://www.stat.berkeley.edu/~pitman/",
            "ref": "@jpitman"
        },

            "linkage": "http://dataset.com/schema/linkage.js",
            "schema": "http://dataset.com/schema/structure.js"
        },

        "recordList": [
            {
                "id": "MR2276901",
                "type": "Article",
                "prefLabel": "Two recursive decompositions of Brownian bridge related to the asymptotics of random
mappings",
                "description": "Aldous and Pitman (1994) studied asymptotic distributions, as n tends to infinity, of
various functionals of a uniform random mapping of a set of n elements, by constructing a mapping-walk and showing these
mapping-walks converge weakly to a reflecting Brownian bridge. Two different ways to encode a mapping as a walk lead to two
different decompositions of the Brownian bridge, each defined by cutting the path of the bridge at an increasing sequence of
recursively defined random times in the zero set of the bridge. The random mapping asymptotics entail some remarkable
identities involving the random occupation measures of the bridge fragments defined by these decompositions. We derive
various extensions of these identities for Brownian and Bessel bridges, and characterize the distributions of various path
fragments involved, using the theory of Poisson processes of excursions for a self-similar Markov process whose zero set is
the range of a stable subordinator of index between 0 and 1.",
                "year": "2006",
                "pages": "269--303",
                "arxiv": "math.PR/0402399",
                "keywords": [
                                "Path decomposition",
                                "Path rearrangement",
                                "Random mapping",
                                "Combinatorial stochastic process"
                        ],
                "bibnumer": "117",
                "volume": "1874",
                "address": "Berlin",
                "mrclass": "60C05 (60J65)",
                "author": [
                    {
                        "prefLabel": "Aldous, David"
                    },
                    {
                        "prefLabel": "Pitman, Jim",
                        "prefURL": "http://www.stat.berkeley.edu/~pitman/",
                        "ref": "@jpitman"
                    }
                ],
                "isPartOf": [
                {
                    "prefLabel": "In memoriam Paul-André Meyer: Séeminaire de Probabilités",
                    "ref": "@book_id"
                }
               ]
            },
            {
               "id": "jpitman",
               "type": "Person",
               "prefLabel":"Jim Pitman",
               "homepage": "http://www.stat.berkeley.edu/~pitman/"
            },
            {
                "id": "ustanford",
                "type": "Organization",
                "prefLabel": "Stanford University",
                "homepage": "http://www.stanford.edu/"
            },
            {
                "id": "book_id",
                "type": "Book",
                "title": "In memoriam Paul-Andrée Meyer: Séminaire de Probabilités",
                "editors": [
                    {
                        "prefLabel": "Michel Émery"
                    },
                    {
                        "prefLabel": "Marc Yor"
                    }
                ],
                "isPartOf": [
                    {
                        "prefLabel": "Lecture Notes in Math.",
                        "ref": "@series_id"
                    }
                ]
            },
            {
                "id": "series_id",
                "type": "Series",
                "title": "Lecture Notes in Math.",
                "volume": "1874",
                "publisher": [
                  {
                    "prefLabel": "Springer"
                  }
                ]
           }
        ]
}
```

Example #2: Best Buy Mp3 Player

```
{
   "dataset": {
       "id": "http://stores.bestbuy.com/",
       "prefLabel": "BestBuy dataset of goods",
       "source": {
           "prefLabel": "Best Buy",
           "prefURL": "http://bestbuy.com"
       },

       "createDate": "01/01/2009",
       "creator": {
           "prefLabel": "Best Buy",
           "prefURL": "http://bestbuy.com"
       },

       "linkage": "http://dataset.com/schema/linkage.js",
       "schema": "http://dataset.com/schema/structure.js"
    },

   "recordList": [
       {
           "id": "sku8773062",
           "type": "mp3-player",
           "prefLabel": "Apple® - iPod nano® 16GB* MP3 Player",
           "price": "199.99",
           "color": "black",
           "sku": "8773062",
           "model": "MB918LL/A",
           "catalogWebPage": "http://www.bestbuy.com/site/olspage.jsp?skuId=8773062&type=product&id=1204332007749",
           "availableInRetailStore": [
               {
                   "prefLabel": "Colorado Sprg Ii CO (Store 298)",
                   "prefURL": "http://stores.bestbuy.com/298/",
                   "ref": "@298"
               }
           ]
       },
       {
           "id": "298",
           "type": "retail-store",
           "prefLabel": "Colorado Sprg Ii CO (Store 298)",
           "address": "7675 N Academy Blvd Market At Chapel Hill",
           "state": "CO",
           "postal-code": "CO 80920",
           "phone": "719-593-0414"
       }
   ]
}
```

Example #3: A irJSON Bibliographic Vocabulary to BibTeX Schema Linkage

Here is an example of a schema linkage. This schema describes the relationships between a irJSON bibliographic vocabulary and BibTeX.

```
{
        "linkage": {
                "version": "0.1",
                "linkedType": "application/x-bibtex",

                "attributeList": {
                        "address": {
                                    "mapTo": "address"
                        },
                        "author": {
                                "mapTo": "author"
                        },
                        "title": [
                                {
                                    "mapTo": "booktitle"
                                },
                                {
                                    "mapTo": "title"
                                }
                        ],
                        "chapter": {
                                "mapTo": "chapter"
                        },
                        "ref": {
                                "mapTo": "crossref"
                        },
                        "edition": {
                                "mapTo": "edition"
                        },
                        "editor": {
                                "mapTo": "editor"
                        },
                        "eprint": {
                                "mapTo": "eprint"
                        },
                        "howpublished": {
                                "mapTo": "howpublished"
                        },
                        "institution": {
                                "mapTo": "institution"
                        },
                        "journal": {
                                "mapTo": "journal"
                        },
                        "key": {
                                "mapTo": "key"
                        },
                        "month": {
                                "mapTo": "month"
                        },
                        "note": {
                                "mapTo": "note"
                        },
                        "number": {
                                "mapTo": "number"
                        },
                        "organization": {
```

```
                "mapTo": "organization"
            },
            "pages": {
                "mapTo": "pages"
            },
            "publisher": {
                "mapTo": "publisher"
            },
            "school": {
                "mapTo": "school"
            },
            "series": {
                "mapTo": "series"
            },
            "type": {
                "mapTo": "type"
            },
            "href": {
                "type": "string",
                "mapTo": "href"
            },
            "volume": {
                "mapTo": "volume"
            },
            "year": {
                "mapTo": "year"
            }
        },

        "typeList": {
            "article": {
                "mapTo": "article"
            },
            "book": {
                "mapTo": "book"
            },
            "booklet": {
                "mapTo": "booklet"
            },
            "conference": {
                "mapTo": "conference"
            },
            "inbook": {
                "mapTo": "inbook"
            },
            "incollection": {
                "mapTo": "incollection"
            },
            "inproceedings": {
                "mapTo": "inproceedings"
            },
            "manual": {
                "mapTo": "manual"
            },
            "mastersthesis": {
                "mapTo": "mastersthesis"
            },
            "misc": {
                "mapTo": "misc"
            },
            "phdthesis": {
                "mapTo": "phdthesis"
            },
            "proceedings": {
                "mapTo": "proceedings"
            },
            "techreport": {
                "mapTo": "techreport"
            },
            "unpublished": {
                "mapTo": "unpublished"
            }
        }
    }
}
```

Example #4: A irJSON Bibliographic Vocabulary to RDF Schema Linkage

This other example demonstrates how we can create another linkage file to link this irJSON Bibliographic Vocabulary attributes and types to RDFS/OWL ontologies properties and classes. This example shows the flexibility of a irJSON linkage file and how it can be used to link the same simple instance record vocabulary to different format/languages.

```
{
    "linkage": {
        "version": "0.1",
        "linkedType": "application/rdf+xml",

        "prefixList": {
            "bibo": "http://purl.org/ontology/bibo/",
            "dcterms": "http://purl.org/dc/elements/1.1/",
            "rdfs": "http://www.w3.org/2000/01/rdf-schema#",
            "rdf": "http://www.w3.org/1999/02/22-rdf-syntax-ns#",
            "bibo_degrees": "http://purl.org/ontology/bibo/degrees/",
            "address": "http://schemas.talis.com/2005/address/schema#",
            "bkn": "http://purl.org/ontology/bkn#"
        },

        "attributeList": {
            "address": {
                "mapTo": "address:localityName"
            },
            "author": {
                "mapTo": "dcterms:creator"
            },
            "title": {
                "mapTo": "dcterms:title"
            },

            "chapter": {
                "mapTo": "bibo:chapter"
            },
            "ref": {
                "mapTo": "rdf:resource"
            },
            "edition": {
                "mapTo": "bibo:edition"
            },
            "editor": {
                "mapTo": "bibo:editor"
            },
            "eprint": {
                "mapTo": "rdfs:seeAlso"
            },
            "howpublished": {
                "mapTo": "dcterms:publisher"
            },
            "institution": {
                "mapTo": "dcterms:contributor"
            },
            "journal": {
                "mapTo": "dcterms:isPartOf"
            },
            "key": {
                "mapTo": "foo:bar"
            },
            "month": {
                "mapTo": "dcterms:date"
            },
            "note": {
                "mapTo": "skos:note"
            },
            "number": {
                "mapTo": "bibo:number"
            },
            "organization": {
                "mapTo": "bibo:organizer"
            },
            "pages": {
                "mapTo": "bibo:pages"
            },
            "publisher": {
                "mapTo": "dcterms:publisher"
            },
            "school": {
                "mapTo": "rdfs:seeAlso"
            },
            "series": {
                "mapTo": "dcterms:isPartOf"
            },
            "type": {
                "mapTo": "rdf:type"
            },
            "href": {
                "mapTo": "bkn:url"
            },
            "volume": {
                "mapTo": "bibo:volume"
            },
            "year": {
                "mapTo": "dcterms:date"
            }
        },

        "typeList": {
            "article": {
                "mapTo": "bibo:Article"
            },
            "book": {
                "mapTo": "bibo:Book"
            },
            "booklet": {
                "mapTo": "bibo:Booklet"
            },
            "conference": {
                "mapTo": "bibo:Conference"
            },
            "inbook": {
                "mapTo": "bibo:Chapter"
            },
            "incollection": {
                "mapTo": "bibo:BookSection"
            },
            "inproceedings": {
                "mapTo": "bibo:Article"
```

```
                        },
                        "manual": {
                                "mapTo": "bibo:Manual"
                        },
                        "mastersthesis": {
                                "mapTo": "bibo:Thesis",
                                "addMapping": {
                                        "bibo:degree": "bibo_degrees:ma"
                                }
                        },
                        "misc": {
                                "mapTo": "bibo:Document"
                        },
                        "phdthesis": {
                                "mapTo": "bibo:Thesis",
                                "addMapping": {
                                        "bibo:degree": "bibo_degrees:phd"
                                }
                        },
                        "proceedings": {
                                "mapTo": "bibo:Proceedings"
                        },
                        "techreport": {
                                "mapTo": "bibo:Report"
                        },
                        "unpublished": {
                                "mapTo": "bibo:Document",
                                "addMapping": {
                                        "bibo:status": "bibo:unpublished"
                                }
                        }
                }
        }
}
```

Converting irJSON into RDF

This section is not yet drafted. It will explain how a irJSON-to-RDF converter can be written, and how it is expected to behave. We will explain how a linkage schema can be used to create transformation rules that will take irJSON statements and then create RDF triples by applying the rules defined in the linkage schema.

SUB-PART 3: commON PROFILE

This sub-part of the irON specification describes the comma-delimited or comma-separated values (CSV) serialization, ***commON***.

Role and Use

The most common data authoring environment in the world is the spreadsheet. Spreadsheets are a ubiquitous tool for knowledge workers. And, CSV (comma-separated values), a very old file format that predates personal computers but was embraced by Microsoft as a spreadsheet representation, is a nearly ubiquitous data exchange format that is also easily read by humans.

Most simple data can be developed and provided as text datasets based on attribute-value pairs (also known as key-value pairs and many other variants) [2]. In spreadsheets, a tabular view of similar "things" (instances) can be readily presented where the records of those instances represent the rows in the table or spreadsheet, the attributes or properties or "fields" describing those things are listed in the columns. Indeed, this basic framework is also what is used in relational data tables.

When exported, CSV only contains the cell data values from a spreadsheet. While this has the disadvantage of losing formatting, formulas and other niceties of spreadsheets in their native form, it also makes the data exchanged clean and relatively uniform. During data development and preparation the spreadsheet can be used in all of its native capabilities to provide data validation, sorting, formatting, cell referencing, calculations of (say) totals and subtotals, etc. This means that templates with useful prompts and controlled vocabularies and rapid editing and entry functions can be quickly developed for a spreadsheet, then followed by clean data export using CSV for use by other tools and for data federation.

Moreover, with just a little bit of extra specification, the staging of this data and then its export using CSV can also achieve broader operability. These are the rationales for commON.

The purpose of commON it to provide an easy data authoring and dataset creation environment for knowledge workers. By following a few conventions and using common spreadsheet tools, knowledge workers with domain knowledge but little or no programming and scripting language skills can rapidly and effectively create small datasets (databases) or can extract information from existing spreadsheets for integration and interoperability.

In keeping with the irON notation, commON consists of a number of modules that define the various aspects of a full specification. These modules may be specified in a single CSV file or in multiple separate files. The start of a module is signaled by a process-based reserved keyword (see below) that the parser recognizes.

CSV as generally used is "schema-less". In order to embrace the irON notation, vocabulary and structure, commON introduces a number of conventions that must be followed in order to achieve the irON objective of staging data for RDF interoperability.

The MIME type for irJSON is `application/iron+csv`.

Differences from Generic CSV

Though in use for decades, CSV only received a formal MIME-type specification in 2005 [7]. CSV has relatively few conventions and limited syntax, as explained in [7].

commON is fully compliant with RFC 4180 and the parser and ingesters used by the structWSF framework [8]. commON has been validated for Microsoft Excel CSV and Open Office CSV.

Of course, in addition to the core CSV specification, commON adds many conventions and constructs in keeping with the irON notation.

Differences from Generic irON

As a relatively "schema-less" framework, CSV presents a number of challenges to embrace the full slate of capabilities within the irON notation. While it is possible with many conventions and restrictions to achieve the full slate of irON capabilities, some have been dropped from commON to promote simplicity and ease-of-use. These capabilities may be added back in over time, particularly if better conventions can be discovered.

This section outlines these differences between irON and commON.

No Schema Module, Other Changes

As the most complex specification within irON, the schema module has been dropped from commON (at least for the present). This decision still allows useful datasets to be authored and linked to existing schema, but the actual schema specification must either occur through one of the other irON serializations (irXML or irJSON), or directly via RDF or OWL.

In commON, the ingest of attributes, class types and records has been standardized as a list process, with the operation signaled by keyword and convention. As a result, the irON attributes of *attribute*, *type* and *record* are not used in commON.

The irON attributes of *format* type and *addMapping* have also been dropped from commON to streamline the specification.

Augmenting Attributes with Metadata

The section within the main irON notation described the approach for *Augmenting Attributes with Metadata*. For the commON serialization, once a primary attribute is stated for an instance, metadata can be added for that attribute by:

1. Listing the metadata in the next column to the right of the subject attribute, and
2. Using the nested notation of *&primaryAttribute&attributeMetadata* to designate the new metadata.

The key to the syntax is appending of the metadata attribute with the ampersand ('&') designator to the primary attribute (which has already been designated with the standard & designator), leading to the linked ampersand convention.

Here is an example of this syntax with *paper* as the primary attribute, and the *source* attribute as the metadata about the paper:

```
&paper                  &paper&source
Lecture Notes in Math.  Journal of Irreproducible Results
```

Specific Processing Options

For preference and readability reasons, some may prefer to present and manipulate their information with slightly different options for conventions. These processing choices are invoked via the *&&options* keyword. The current ones availalble in commON are:

Attribute	Description	Default
listSeparator	a single character used to separate items in a list entry	|
listSeparatorEscape	how the character gets escaped if it is present in a value without separating a list	%7C
seqNum	yes, no	yes

The *listSeparator* attribute sets what the delimiter is for a list of values for a single attribute. The default value is the pipe characters ('|'), though these characters are also possible: comma (','), semi-colon (';'), and squiggle ('~'). The data publisher has to specify a value for the *listSeparatorEscape* attribute to tell the commON processor engine how to escape/unescape the list separator character.

The *seqNum* attribute is a Boolean (yes, no) value. If set to yes, a value is added to the instance record listings that enables all values to be sorted in offline applications (especially spreadsheets for the commON serialization).

Note: It is recommended to set the *seqNum* value to 'yes' if you are using the *Stacked* style (see below) for instance records.

Instance Record Presentation Styles

As discussed for the *Instance Record Object* below, there are two entry presentation styles available for instance records: *Row* and *Stacked*. See further that section.

Reduced Keyword Set

As a result, these existing irON attributes and keywords are not presently available for use within commON:

addMapping	equivalentTypeTo	schema
attribute	format	subPropertyOf
equivalentPropertyTo	record	subTypeOf
	ref	

It is recommended not to use these terms in a commON specification as they might pose conflicts with other irON serializations.

General commON Design

The "schema-less" nature of CSV necessitates introducing some broad design considerations to commON. These are:

- All keywords and modules are introduced by a standard character, with the ampersand ('&') chosen for this purpose
- Module or processing sections require a further convention, with the double ampersand ('&&') prefix chosen for this purpose
- Minor, but set structural conventions in relation to specified rows and columns, requirements help instruct the commON ingesters and parsers, and
- A design objective has been to limit the number of conventions and requirements to as small a set as possible.

Summary of Conventions

Thus, here are the specific commON conventions:

1. irON vocabulary keywords are reserved
2. All section (object) types begin with && (which may also signal slight differences to the parser depending on mode keywords)
3. All attribute names begin with a single ampersand (&)
4. In the instance record layout (*&&recordList*):
   - the first row is restricted to the listing of attribute names (using the single ampersand prefix)
   - *&id* is required and must be placed in the first column
   - *&type*, if provided (and is highly recommended), must be placed in the second column
   - attributes with a list of values must have the values separated by the pipe ('|') character (or the alternative character per the *&listSeparator* attribute)
5. Metadata about primary attributes for the instance are denoted by the linked ampersand notation of *&primary&primaryMetadata*, so long as this metadata immediately follows the first separate listing of the primary *attributeName*
6. The *&attributeList*, *&prefixList* and *&typeList* sections list each entry sequentially by row with `key:value` in Cols 1 and 2
7. Blank rows may be inserted anywhere for readability
8. Comment rows begin with # (it might also appear as "# in CSV file) and are ignored during processing.

Dataset Object

The dataset object is signaled by the *&&dataset* keyword. The following example includes some metadata about the dataset, as well.

Dataset Example

&&dataset							
&id	&prefLabel	&description	&createDate	&updateDate	&linkage	&source	
http://mkbergman.com/swt/	Sweet Tools	Sweet Tools is a comprehensive listing of about 800 semantic Web and -related tools, with most open source	8/12/2006	8/1/2009	http://example.com/link	@mkbergman	
&&recordList							
&id	&type	&name	&prefLabel	&superType	&role	&description	&prefURL
mkbergman	person	Michael Bergman	Michael Bergman	person	Contractor	a project consultant	http://www.mkbergman.com

Instance Record Object

The instance record object is signaled by the *&&recordList* keyword. By convention, the next line must include the attribute names (& prefix) by column, followed by the instance records row-by-row until the file ends or a another processing keyword ('&&') is encountered.

Also, as noted, the first two columns in this tabular layout are for *&id* and *&type*, respectively.

There is no practical limit on the number of attributes (columns) that can be specified. For readability it is advisable to break such instance tables into digestible chunks. Also, attributes can be repeated when the schema allows multiple entries.

Instance Record Table Examples

As noted, there are two possible instance record table styles using commON. The parser recognizes both styles without further notation or instruction. Which style you use is a matter of your own preference.

Instance Record Row Style

Here is an example of the *Row* style, where all attributes are listed in columns in a single row. If there are multiple, same attributes, and you are not using the list separator convention, each duplicate attribute would be listed in its own column:

&&recordList							
&id	&type	&name	&prefLabel	&superType	&role	&description	&prefURL
fgiasson	person	Giasson, Frederick	Frederick Giasson	person	Contractor	a project consultant	http://fgiasson.com/blog
michael_bergman	person	Michael Bergman	Michael Bergman	person	Contractor	a project consultant	http://www.mkbergman.com
structured_dynamics	organization	Structured	Structured	organization	Major	project	http://structureddynamics.com

```
                                 Dynamics LLC   Dynamics LLC                Contractor     contractor
Berkeley          university     University     University     organization NSF Funded     sponsoring     http://www.berkeley.edu/
                                 of             of                          Partner        organization
                                 California,    California,
                                 Berkeley       Berkeley
BibServer         service        BibServer      BibServer      project      Participating  a BKN node     http://bibserver.berkeley.edu/
                                                                            Service        and service
pitman            person         Pitman, Jim    Pitman, Jim    person       PI             BKN project    http://www.stat.berkeley.edu/~pitman/
                                                                                           director
```

Instance Records Stacked Style

Here is an example of the *Stacked* style, where duplicate values for a single attribute are listed in a stacked manner row-by-row until the listing is completed. This style is useful when you want to see long attribute names, such as URIs, for example:

Note: The ID of the first column can be reitirated until it reachs the end of the description of the record if it is what the data publisher prefer.

```
&&recordList
&id                               &type     &prefLabel            &prefURL          &isAuthorOfTitle
info:lib:am:2009-02-18:           Person    Maria Francisca Abad-                   Acceso Abierto y revistas médicas españolas
maria_francisca_abad_garcia                 Garcia



                                                                                    Una base de datos de recursos web médicos: una solución a medida para
                                                                                    una recuperación más eficaz de información de Internet

                                                                                    [Open access and the Spanish medical journals]

                                                                                    Produccion científica de la Comunitat Valenciana en materias de
                                                                                    biomedicina y ciencias de la salud a través de las bases de datos del
                                                                                    Institute of Scientific Information (ISI) 2000-2004.

                                                                                    La base de datos de recursos web de la biblioteca médica virtual del
                                                                                    COMV
                                                                                    Uso de internet por los médicos colegiados de Valencia: un estudio de
                                                                                    viabilidad de la Biblioteca Médica Virtual del Colegio Oficial de
                                                                                    Médicos de Valencia

                                                                                    Ampliando el horizonte de MEDLINE : 100 bases de datos bibliográficas
                                                                                    médicas gratuitas en la red

                                                                                    Information needs and uses: an analysis of the literature published
                                                                                    in Spain, 1990-2004
                                  Person                                            Personal Data in a Large Digital Library
info:lib:am:1971-02-01:                     Jose Manuel Barrueco  http://www.uv.e
jose_manuel_barrueco                                              /=barrueco

                                                                                    Cataloging Economics preprints: an introduction to the RePEc project
                                                                                    Personal Data in a Large Digital Library
                                                                                    WoPEc usage in 1999AD
                                                                                    Distributed Cataloging on the Internet: the RePEc project
                                                                                    Automated Extraction of Citation Data in a Distributed Digital
                                                                                    Library.
                                                                                    ReLIS: una biblioteca digital distribuida para Documentación.
                                                                                    Personal Data in a Large Digital Library.
```

Schema Linkage

The linkage to schema is signaled by the *&&linkage* keyword. Within that, there can be a number of sub-sections, such as *&version* and *&linkedType* (both required), and then *&prefixList*, *&attributeList* or *&typeList*. Each of the *&xxxList* sub-sections conform to the name:value pair where the first (Col 1) entry is the name of the listed attribute and the next column is its referenced *&mapTo* value.

Linkage Example

Note in this example the prefix list is not used.

```
&&linkage


&version          &linkedType
0.1               rdf


&attributeList    &mapTo
name              http://xmlns.com/foaf/0.1/name
prefLabel         http://www.w3.org/2008/05/skos#prefLabel
role              http://purl.org/ontology/bkn/central#role
node              http://purl.org/ontology/bkn/central#node
homepage          http://xmlns.com/foaf/0.1/homepage
about             http://purl.org/ontology/bkn/central#about
```

```
contribution      http://purl.org/ontology/bkn/central#contribution
affiliation       http://purl.org/ontology/bkn/central#affiliation
parent_org        http://purl.org/ontology/bkn/central#parent_org
givenname         http://xmlns.com/foaf/0.1/givenname
family_name       http://xmlns.com/foaf/0.1/family_name
workplaceHomepage http://xmlns.com/foaf/0.1/workplaceHomepage
page              http://xmlns.com/foaf/0.1/page
logo              http://xmlns.com/foaf/0.1/logo
tags              http://purl.org/ontology/bkn/central#tags
olb               http://purl.org/ontology/bkn/central#olb
gdocs_id          http://purl.org/ontology/bkn/central#gdocs_id

&typeList         &mapTo
person            http://xmlns.com/foaf/0.1/Person
software          http://purl.org/ontology/bkn#Software
standard          http://purl.org/ontology/bkn#Standard
organization      http://xmlns.com/foaf/0.1/Organization
service           http://purl.org/ontology/bkn#Service
license           http://purl.org/ontology/bkn#License
journal           http://purl.org/ontology/bibo/Journal
encyclopedia      http://purl.org/ontology/bkn#Encyclopedia
university        http://purl.org/ontology/bkn#University
department        http://purl.org/ontology/bkn#Department
archive           http://purl.org/ontology/bkn#Archive
dataset           http://purl.org/ontology/bkn#Dataset
directory         http://purl.org/ontology/bkn#directory
project           http://xmlns.com/foaf/0.1/Project
```

To invoke a prefix list, here is an example for the last entry above:

```
&prefixList &mapTo
foaf        http://xmlns.com/foaf/0.1/
```

which, when referenced in a *&typeList*, would appear as follows:

```
&typeList &mapTo
project   foaf:Project
```

Downloadable Examples

Because they are difficult to reproduce in this format, here are two examples of complete commON files that are available for inspection:

- The BKN project dataset, which is also used to populate the templates on the BibKN.org Web site, and
- Sweet Tools, Mike Bergman's listing of 800+ semantic Web and -related tools.

Additionally, to demonstrate how controlled vocabularies, validation tables and the like may be linked in with a commON CSV output, we also provide the entry spreadsheet for Sweet Tools that shows these options in action. Saving as a CSV creates the very same commON CSV input file noted directly above.

These files and uses are explained in the accompanying document to this specification, Annex: A commON Case Study using *Sweet Tools*.

ACKNOWLEDGEMENTS

Work towards this specification was supported in part by the Bibliographic Knowledge Network Project (NSF Award 0835851). The support was specifically for the BibJSON version of irJSON and the parser and converter designs [6].

REFERENCES

[1] For a detailed discussion of RDF, see Michael K. Bergman, 2009. "Advantages and Myths of RDF," in *AI3 blog*, April 8, 2009. See http://www.mkbergman.com/483/advantages-and-myths-of-rdf/.

[2] An attribute-value system is a basic knowledge representation framework comprising a table with columns designating "attributes" (also known as *properties*, *predicates*, *features*, *parameters*, *dimensions*, *characteristics* or *independent variables*) and rows designating "objects" (also known as *entities*,

instances, *exemplars*, *elements* or *dependent variables*). Each table cell therefore designates the value (also known as *state*) of a particular attribute of a particular object. This is the basic table presentation of a spreadsheet or relational data table.

Attribute-values can also be presented as pairs in a form of an associative array, where the first item listed is the attribute, often followed by a separator such as the colon, and then the value. JSON and many simple data struct notations follow this format. This format may also be called *attribute-value pairs*, *key-value pairs*, *name-value pairs*, *alists* or others. In these cases the "object" is implied, or is introduced as the name of the array.

[3] As used herein, the name and concept of *attribute* is used interchangeable with *property*. Both of these are equivalent to a *predicate* in an RDF triple. It is recommended that parsers for the various irON serializations recognize both terms (and their variants) interchangeably.

[4] We use the reference to the "ABox" and "TBox" in accordance with this working definition for description logics:

> "Description logics and their semantics traditionally split *concepts* and their relationships from the different treatment of *instances* and their attributes and roles, expressed as fact assertions. The concept split is known as the TBox (for *terminological* knowledge, the basis for *T* in *TBox*) and represents the schema or taxonomy of the domain at hand. The TBox is the structural and intensional component of conceptual relationships. The second split of instances is known as the ABox (for *assertions*, the basis for *A* in *ABox*) and describes the attributes of instances (and individuals), the roles between instances, and other assertions about instances regarding their class membership with the TBox concepts."

[5] Frank Manola and Eric Miller, eds., 2004. *RDF Primer*, W3C Recommendation 10 February 2004. See especially http://www.w3.org/TR/2004/REC-rdf-primer-20040210/#reification.

[6] BibJSON is a JSON-based format designed for representation of bibliographic data, meaning data about documents of various kinds, and about the people, organizations and subjects connected to those documents. BibJSON enhances and extends the data model of BibTeX in a number of ways. Its formal specification is currently nearing public release. BibJSON is an initiative of the Bibliographic Knowledge Network. BKN is a project to develop a suite of tools and services to encourage formation of virtual organizations in scientific communities of various types. BKN is a project started in September 2008 with funding by the NSF Cyber-enabled Discovery and Innovation (CDI) Program (Award # 0835851). The major participating organizations are the American Institute of Mathematics (AIM), Harvard University, Stanford University and the University of California, Berkeley.

[7] The CSV mime type is defined in *Common Format and MIME Type for Comma-Separated Values (CSV) Files* [RFC 4180]. A useful overview of the CSV format is provided by The Comma Separated Value (CSV) File Format. Also, see that author's related CTX reference for a discussion of how schema and structure can be added to the basic CSV framework; see http://www.creativyst.com/Doc/Std/ctx/ctx.htm, especially the section on the comma-delimited version (http://www.creativyst.com/Doc/Std/ctx/ctx.htm#CTC).

[8] structWSF is a platform-independent Web services framework for accessing and exposing structured RDF data, with generic tools driven by underlying data structures. Its central perspective is that of the dataset. Access and user rights are granted around these datasets, making the framework enterprise-ready and designed for collaboration. Since a structWSF layer may be placed over virtually any existing datastore with Web access -- including large instance record stores in existing relational databases -- it is also a framework for Web-wide deployments and interoperability.