Inside Acropolis

A guide to the Research & Education Space for contributors and developers

April 2014 Edition

Edited by Mo McRoberts, BBC Archive Development.

Preface

This book is *deliberately* incomplete. It's an evolving document, licensed under the terms of the *Open Government Licence, v2.0*, and to which we are welcoming contributions. You can fork the repository on GitHub, or e-mail the editor directly if you would like to contribute or have suggestions for changes.

The *Research & Education Space* (RES) is a project being jointly delivered by Jisc, the British Universities Film & Video Council (BUFVC), and the BBC. Its aim is to bring as much as possible of the UK's publicly-held archives, and more besides, to learners and teachers across the UK.

At the heart of RES is *Acropolis*, a technical platform which will collect, index and organise rich structured data about those archive collections published as *Linked Open Data* (LOD) on the Web. The collected data is organised around the people, places, events, concepts and things related to the items in the archive collections—and, if the archive assets themselves are available in digital form, that data includes the information on how to access them, all in a consistent machine-readable form.

Building on the Acropolis platform, applications can make use of this index, along with the source data itself, in order to make those collections accessible and meaningful.

This book describes how a collection-holder can publish their data in a form which can be collected and indexed by Acropolis and used by applications, and how an application developer can make use of the index and interpret the source data in order to present it to end-users in a useful fashion.

Table of contents

1 An introduction to the Acropolis platform

The Acropolis platform is made of up three main components: a specialised web crawler, *Anansi*, an *aggregator*, *Spindle*, and a public API layer, *Quilt*.

Anansi's role is to crawl the web, retrieving permissively-licensed Linked Open Data, and passing it to the aggregator for processing.

Spindle examines the data, looking for instances where the same digital, physical or conceptual entity is described in more than one place, primarily where the data explicitly states the equivalence, and aggregates and stores that information in an index.

This *subject-oriented* index is the very heart of RES: by re-arranging published data so that it's organised around the entities described by it, instead of by publisher or data-set, applications are able to rapidly locate all of the information known about a particular entity because it's collected together in one place.

Quilt is responsible for making the index available to applications, also by publishing it as Linked Open Data. Because RES maintains an *index*, rather than a complete copy of all data that it finds, applications must consume data both from the RES index and from the original data sources—and consequentially Quilt itself also conforms to the publishing recommendations in this book.

The RES project will not be directly developing end-user applications, although sample code and demonstrations will be published to assist software developers in doing so. RES only indexes and publishes data released under terms which permit re-use in both commercial and non-commercial settings.

For RES to be most useful, holders of publicly-funded archive collections across the UK need to publish Linked Open Data describing their collections (including digital assets, where they exist). Although many collections are already doing so or plan to, the RES project partners will be providing tools and advice to collection-holders in order to assist them throughout the lifetime of the project.

2 Linked Open Data: What is it, and how does it work?

Linked Open Data is a mechanism for publishing structured data on the Web about virtually anything, in a form which can be consistently retrieved and processed by software. The result is a world wide web of data which works in parallel to the web of documents our browsers usually access, transparently using the same protocols and infrastructure.

Where the ordinary web of documents is a means of publishing a page about something intended for a human being to understand, this web of data is a means of publishing data about those things.

2.1 Web addresses, URLs and URIs

Uniform Resource Locators (URLs), often known as *Web addresses*, are a way of unambiguously identifying something which is published electronically. Although there are a variety of kinds of URL, most that you day-to-day see begin with `http` or `https`: this is known as the *scheme*, and defines how the rest of the URL is structured—although most kinds of URL follow a common structure.

The scheme also indicates the communications protocol which should be used to access the resource identified by the URL: if it's `http`, then the resource is accessible using HTTP—the protocol used by web servers and browsers; if it's `https`, then it's accessible using secure HTTP (i.e., HTTP with added encryption).

> *i* The act of accessing the resource identified by a URL is known as *resolving* it.

Following the scheme in a URL is the *authority*—the domain name of the web site: it's called the authority because it identifies the entity responsible for defining the meaning and structure of the remainder of the URL. If the URL begins with `http://www.bbc.co.uk/`, you know that it's defined and managed by the BBC; if it begins with `http://www.bfi.org.uk/`, you know that it's managed by the BFI, and so on.

After the authority is an optional *path* (i.e., the location of the document within the context of the particular domain name or authority), and optional *query parameters* (beginning with a question-mark), and *fragment* (beginning with a hash-mark).

URLs serve a dual purpose: not only do they provide a name for something, but they also provide anything which understands them with the information they need to retrieve it. Provided your application is able to speak the HTTP protocol, it should in principle be able to retrieve anything using a `http` URL.

Universal Resource Indicators (URIs) are a superset of URLs, and are in effect a kind of *universal identifier*: their purpose is to name something, without *necessarily* indicating how to retrieve it. In fact, it may be that the thing named using a URI cannot possibly be retrieved using a piece of software and an Internet connection because it refers to an abstract concept or a physical object.

> *i* In other words, while URLs are used specifically to identify digital resources which can be retrieved from a Web server, URIs can be used to identify anything: the URLs we use in our browsers are all URIs, but not all URIs are URLs.

URIs follow the same structure as URLs, in that there is a scheme defining how the remainder is structured, and usually some kind of authority, but there are many different schemes, and many of them do not have any particular mechanism defined for how you might retrieve something named using that scheme.

For example, the `tag:` URI scheme provides a means for anybody to define a name for something in the form of a URI, using a domain name that they control as an authority, but without indicating any particular semantics about the thing being named.

Meanwhile, URIs which begin with `urn:` are actually part of one of a number of *sub-schemes*, many of which exist as a means of writing down some existing identifier about something in the form of a URI. For example, an ISBN can be written as a URI by prefixing it with `urn:isbn:` (for example, `urn:isbn:9781899066100`).

You might be forgiven for wondering why somebody might want to write an ISBN in the form of a URI, but in fact there are a few reasons. In most systems, ISBNs are effectively opaque alphanumeric strings: although there is usually some validation of the check digit upon data entry, once stored in a database, they are rarely interrogated for any particular meaning. Given this, ISBNs work perfectly well for identifying books for which ISBNs have been issued—but what if you want to store data about other kinds of things, too? Recognising that this was a particular need for retailers, a few years ago ISBNs were made into a subset of *Global Trade Information Numbers* (GTINs), the system used for barcoding products sold in shops.

By unifying ISBNs and GTINs, retailers were able to use the same field in their database systems for any type of product being sold, whether it was a book with an ISBN, or some other kind of product with a GTIN. All the while, the identifier remained essentially opaque: provided the string of digits and letters scanned by the bar-code reader could be matched to a row in a database, it doesn't matter precisely what those letters and numbers actually are.

Representing identifiers in the form of URIs can be thought of as another level of generalisation: it allows the development of systems where the underlying database doesn’t need to know nor care about the *kind* of identifier being stored, and so can store information about absolutely anything which can be identified by a URI. In many cases, this doesn’t represent a huge technological shift—those database systems already pay little attention to the structure of the identifier itself.

Hand-in-hand with this generalisation effect is the ability to disambiguate and harmonise without needing to coordinate a variety of different standards bodies across the world. Whereas the integration of ISBNs and GTINs took a particular concerted effort in order to achieve, the integration of ISBNs and URNs was only a matter of defining the URN scheme, because URIs are already designed to be open-ended and extensible.

Linked Open Data URIs are a subset of URIs which, again, begin with `http:` or `https:`, but do not necessarily name something which can be retrieved from a web server. Instead, they are URIs where performing resolution results in machine-readable data *about* the entity being identified.

In summary:

Term	Used for…
URLs	Identifying digital resources and specifying where they can be retrieved from
URIs	Identifying *anything*, regardless of whether it can be retrieved electronically or not
Linked Open Data URIs	Identifying anything, but in a way which means that *descriptive metadata* can be retrieved when the URI is resolved

2.2 Describing things with triples

Linked Open Data uses the *Resource Description Framework* (RDF) to convey information about things. RDF is an open-ended system for modelling information about things, which it does by breaking it down into statements (or *assertions*), each of which consists of a *subject*, a *predicate* and an *object*.

The subject is the thing being described; the predicate is the aspect or attribute of the subject being described; and the object is the description of that particular attribute.

> If you are familiar with object-oriented programming, you may find it useful to think of a subject as being an *instance*, a predicate as a *property*, and an object as a *value*. In fact, the terms are often used interchangeably.

For example, you might want to state that the book with the ISBN 978-1899066100 has the title *Acronyms and Synonyms in Medical Imaging*. You can break this assertion down into its subject, predicate, and object:

Subject	Predicate	Object
ISBN 978-1899066100	Has the title	*Acronyms and Synonyms in Medical Imaging*

Together, this statement made up of a subject, predicate and object is called a *triple* (because there are three components to it), while a collection of statements is called a *graph*.

In RDF, the subject and the predicate are expressed as URIs this helps to remove ambiguity and the risk of clashes so that the data can be published and consumed in the same way regardless of where it comes from or who's processing it. Objects *can* be expressed as URIs where you want to assert some kind of reference to something else, but can also be *literals* (such as text, numeric values, dates, and so on).

2.3 Predicates and vocabularies

RDF doesn't specify the meaning of most predicates itself: in other words, RDF doesn't tell you what URI you should use to indicate "has the title". Instead, because anybody can create a URI, it's entirely up to you whether you invent your own vocabulary when you publish your data, or adopt somebody else's. Generally, of course, if you want other people to be able to understand your data, it's probably a good idea to adopt existing vocabularies where they exist.

In essence, RDF provides the grammar, while community consensus provides the dictionary.

One of the most commonly-used general-purpose vocabularies is the DCMI Metadata Terms, managed by the Dublin Core Metadata Initiative (DCMI), and which includes a suitable title predicate:

Subject	Predicate	Object
ISBN 978-1899066100	http://purl.org/dc/terms/title	*Acronyms and Synonyms in Medical Imaging*

With this triple, a consuming application that understands the DCMI Metadata Terms vocabulary can process that data and understand the predicate to indicate that the item has the title *Acronyms and Synonyms in Medical Imaging*.

> The Dublin Core Metadata Initiative and the core of the DCMI Metadata Terms vocabulary pre-date RDF and Linked Open Data by some years: older vocabularies and classification schemes have been routinely adapted and re-purposed for RDF as it's become more widely used as an approach to representing structured data.

Because `http://purl.org/dc/terms/title` is quite long-winded, it's common to write predicate URIs in a compressed form, consisting of a namespace prefix and local name—similar to the `xmlns` mechanism used in XML documents.

Because people will often use the same prefix to refer to the same namespace URI, it is not unusual to see this short form of URIs used in books and web pages. Some common prefixes and namespace URIs are shown below:

Vocabulary	Namespace URI	Often abbreviated as
RDF Syntax	`http://www.w3.org/1999/02/22-rdf-syntax-ns#`	`rdf:`
RDF Schema	`http://www.w3.org/2000/01/rdf-schema#`	`rdfs:`
DCMI Metadata Terms	`http://purl.org/dc/terms/`	`dct:`

FOAF	http://xmlns.com/foaf/0.1/	foaf:
Vocabulary of Interlinked Datasets (VoID)	http://rdfs.org/ns/void#	void:

For example, defining the namespace prefix `dct` with a namespace URI of http://purl.org/dc/terms/, we can write our predicate as `dct:title` instead of http://purl.org/dc/terms/title. RDF systems re-compose the complete URI by concatenating the prefix URI and the local name.

i An index of all of the vocabularies referenced in this book is provided at the end of the book.

2.4 Subject URIs

In RDF, subjects are also URIs. While in RDF itself there are no particular restrictions upon the kind of URIs you can use (and there are a great many different kinds — those beginning `http:` and `https:` that you see on the Web are just two of hundreds), Linked Open Data places some restrictions on subject URIs in order to function. These are:

1. Subject URIs must begin with `http:` or `https:`.
2. They must be unique: although you can have multiple URIs for the same thing, one URI can't refer to multiple distinct things at once.
3. If a Linked Open Data consumer makes an HTTP request for the subject URI, the server should send back RDF data describing that subject.
4. As with URLs, subject URIs need to be persistent: that is, they should change as little as possible, and where they do change, you need to be able to make arrangements for requests for the old URI to be forwarded to the new one.

In practice, this means that when you decide upon a subject URI, it needs to be within a domain name that you control and can operate a web server for; you need to have a scheme for your subject URIs which distinguishes between things which are represented digitally (and so have ordinary URLs) and things which cannot; you also need to arrange for your web server to actually serve RDF when it's requested; and finally you need to decide a form for your subject URIs which minimises changes.

This may sound daunting, but it can be quite straightforward—and shares much in common with deciding upon a URL structure for a website that is intended only for ordinary browsers.

For example, if you are the *Intergalactic Alliance Library & Museum*, whose domain name is `ialm.int`, you might decide that all of your books' URIs will begin with `http://ialm.int/books/`, and use the full 13-digit ISBN, without dashes, as the key. You could pick something other than the ISBN, such as an identifier meaningful only to your own internal systems, but it makes developers' lives easier if you incorporate well-known identifiers where it's not problematic to do so.

Because this web of data co-exists with the web of documents, begin by defining the URL to the *document* about this book:

```
http://ialm.int/books/9781899066100
```

Anybody visiting that URL in their browser will be provided with information about the book in your collection. Because the URL incorporates a well-known identifier, the ISBN, if any other pieces of information about the book change or are corrected, that URL remains stable. As a bonus, incorporating the ISBN means that the URL to the document is predictable.

> Of course, the ISBN may have been entered incorrectly (or may be cancelled by the registration authority), and it would be worth planning for that eventuality—but assuming that your collection website's data is based upon information that is used operationally day-to-day, the risk of that needing to occur is kept to a minimum.

Having defined the URL for book pages, it's now time to define the rest of the structure. The Intergalactic Alliance Library & Museum web server will be configured to serve web pages to web browsers, and RDF data to RDF consumers: that is, there are multiple representations of the same data. It's useful, from time to time, to be able to refer to each of these representations with a distinct URL. Let's say, then, that we'll use the general form:

```
http://ialm.int/books/9781899066100.EXT
```

In this case, `EXT` refers to the well-known *filename extension* for the particular type of representation we're referring to.

> Media types (sometimes also called MIME types or content types) are registered with the Internet Assigned Numbers Authority (IANA). The registration document includes the preferred or commonly-used filename extensions for that type. For example, the registration document for HTML can be found on the IANA website.

Therefore, the HTML web page for our book will have the *representation-specific* URL of:

```
http://ialm.int/books/9781899066100.html
```

If you also published CSV data for your book, it could be given the representation-specific URL of:

```
http://ialm.int/books/9781899066100.csv
```

RDF can be expressed in a number of different forms, or serialisations. The most commonly-used serialisation is called *Turtle*, and typically has the filename extension of `ttl`. Therefore our Turtle serialisation would have the representation-specific URL of:

```
http://ialm.int/books/9781899066100.ttl
```

Now that we have defined the structure of our URLs, we can define the pattern used for the subject URIs themselves. Remember that the URI needs to be *dereferenceable*—that is, when a consuming application makes a request for it, the server can respond with the appropriate representation.

In order to do this, there are two options: we can use a special kind of redirect, or we can use fragments. The fragment approach works best where you have a document for each individual item, as we do here, and takes advantage of the fact that in the HTTP protocol, any part of a URL following the "#" symbol is never sent to the server.

Thus, let's say that we'll distinguish our URLs from our subject URIs by suffixing the subject URIs with `#id`. The URI for our book therefore becomes:

```
http://ialm.int/books/9781899066100#id
```

When an application requests the information about this book, by the time it arrives at our web server, it's been turned into a request for the very first URL we defined—the generic "document about this book" URL:

```
http://ialm.int/books/9781899066100
```

> The reason the "fragment" portion of the URI is stripped off the request by the time it arrives at the web server is because the HTTP protocol states that fragments are never sent *over the wire*—that is, they are not included in the protocol exchange between the client and the server. Their original use was to identify a section within a web page and allow a browser to skip straight to it even though it requested and was served the whole page. Fragments in *URLs* are regularly used for this purpose today.

When an application understands RDF and tells the server as much as part of the request, the server can send back the Turtle representation instead of an HTML web page—a part of the HTTP protocol known as *content negotiation*. Content negotiation allows a server to pick the most appropriate representation for something (where it has multiple representations), based upon the client's preferences.

With our subject URI pattern defined, we can revisit our original assertion:

Subject	Predicate	Object
`http://ialm.int/books/9781899066100#id`	`dct:title`	*Acronyms and Synonyms in Medical Imaging*

2.5 Defining what something is: classes

One of the few parts of the common vocabulary which is defined by RDF itself is the predicate `rdf:type`, which specifies the class (or classes) of a subject. Like predicates, classes are defined by vocabularies, and are also expressed as URIs. The classes of a subject are intended to convey what that subject *is*.

For example, the Bibliographic Ontology, whose namespace URI is `http://purl.org/ontology/bibo/` (commonly prefixed as `bibo:`) defines a class named `bibo:Book` (whose full URI we can deduce as being `http://purl.org/ontology/bibo/Book`).

If we write a triple which asserts that our book is a `bibo:Book`, any consumers which understand the Bibliographic Ontology can interpret our data as referring to a book:

Subject	Predicate	Object
`http://ialm.int/books/9781899066100#id`	`rdf:type`	`bibo:Book`
		Acronyms and

`http://ialm.int/books/9781899066100#id`	`dct:title`	*Synonyms in Medical Imaging*

2.6 Describing things defined by other people

There is no technical reason why your subject URIs must *only* be URIs that you control directly. In Linked Open Data, the matter of *trust* is a matter for the data consumer: one application might have a white-list of trusted sources, another might have a black-list of sources known to be problematic, another might have more complex heuristics, while another might use your social network such that assertions from your friends are considered more likely to be trustworthy than those from other people.

Describing subjects defined by other people has a practical purpose. Predicates work in a particular direction, and although sometimes vocabularies will define pairs of predicates so that you can make a statement either way around, interpreting this begins to get complicated, and so most vocabularies define predicates only in one direction.

As an example, you might wish to state that a book held in a library is about a subject that you're describing. On a web page, you'd simply write this down and link to it—perhaps as part of a "Useful resources" section. In Linked Open Data, you can make the assertion that one of the subjects of the other library's book is the one you're describing. This works exactly the same way as if you were describing something that you'd defined yourself—you simply write the statement, but somebody else's URI as the subject.

This can also be used to make life easier for developers and reduce network overhead of applications. In your "Useful resources" section, you probably wouldn't only list the URL to the page about the book: instead, you'd list the title and perhaps the author *as well* as linking to the page about the book. You can do that in Linked Open Data, too. Let's say that we're expressing the data about a subject—Roman Gaul—which we've assigned a URI of `http://ialm.int/things/2068003#id`:

Subject	Predicate	Object
`http://ialm.int/things/2068003#id`	`dct:title`	*Roman Gaul*
`http://bnb.data.bl.uk/id/resource/006889069`	`rdf:type`	`bibo:Book`
`http://bnb.data.bl.uk/id/resource/006889069`	`dct:title`	*Asterix the Gaul*
`http://bnb.data.bl.uk/id/resource/006889069`	`dct:subject`	`http://ialm.int/thing`

In this example we've defined a subject, called *Roman Gaul*, of which we've provided very little detail, except to say that it's a subject of the book *Asterix the Gaul*, whose identifier is defined by the British Library.

Note that we haven't described the book *Asterix the Gaul* in full: RDF operates on an *open world principle*, which means that sets of assertions are generally interpreted as being incomplete—or rather, only as complete as they need to be. The fact that we haven't specified an author or publisher of the book doesn't mean there isn't one, just that the data isn't present here; where in RDF you need to state explicitly that something doesn't exist, there is usually a particular way to do that.

2.7 Turtle: the terse triple language

Turtle is one of the most common languages for writing RDF in use today—although there are many others. Turtle is intended to be interpreted and generated by machines first and foremost, but also be readable and writeable by human beings (albeit usually software developers).

In its simplest form, we can just write out our statements, one by one, each separated by a full stop. URIs are written between angle-brackets (`<` and `>`), while *string literals* (such as the names of things) are written between double-quotation marks (`"`).

```
<http://ialm.int/books/9781899066100#id> <http://www.w3.org/1999/02/22-rdf-
syntax-ns#type> <http://purl.org/ontology/bibo/Book> .

<http://ialm.int/books/9781899066100#id> <http://purl.org/dc/terms/title>
"Acronyms and Synonyms in Medical Imaging" .
```

This is quite long-winded, but fortunately Turtle allows us to define and use prefixes just as we have in this book. When we write the short form of a URI, it's *not* written between angle-brackets:

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix dct: <http://purl.org/dc/terms/> .
@prefix bibo: <http://purl.org/ontology/bibo/> .

<http://ialm.int/books/9781899066100#id> rdf:type bibo:Book .

<http://ialm.int/books/9781899066100#id> dct:title "Acronyms and Synonyms in
Medical Imaging" .
```

Because Turtle is designed for RDF, and `rdf:type` is defined by RDF itself, Turtle provides a nice shorthand for the predicate: `a`. We can simply say that our book is `a bibo:Book`:

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix dct: <http://purl.org/dc/terms/> .
@prefix bibo: <http://purl.org/ontology/bibo/> .

<http://ialm.int/books/9781899066100#id> a bibo:Book .

<http://ialm.int/books/9781899066100#id> dct:title "Acronyms and Synonyms in
Medical Imaging" .
```

Writing the triples out this way quickly gets repetitive: you don't want to be writing the subject URI every time, especially not if writing Turtle by hand. If you end a statement with a semi-colon instead of a full-stop, it indicates that what follows is another predicate and object about the same subject:

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix dct: <http://purl.org/dc/terms/> .
@prefix bibo: <http://purl.org/ontology/bibo/> .

<http://ialm.int/books/9781899066100#id>
        a bibo:Book ;
        dct:title "Acronyms and Synonyms in Medical Imaging" .
```

If you end a statement with a comma instead of a semi-colon or full-stop, it means that what follows is another object with the same subject and predicate—in other words, it's a quick way of writing multiple values.

Turtle includes a number of capabilities which we haven't yet discussed here, but are important for fully understanding real-world RDF in general and Turtle documents in particular. These include:

Typed literals

> *Typed literals*: literals which aren't simply strings of text, but can be of any one of the XML Schema data types.
>
> Literal types are indicated by writing the literal value, followed by two carets, and then the datatype URI: for example, `"2013-01-26"^^xsd:date`.

Blank nodes

Blank nodes are entities for which some information is provided, but where the subject URI is not known. There are two different ways of using blank nodes in Turtle: a blank node *value* is one where in place of a URI or a literal value, an entity is partially described.

Another way of using blank nodes is to assign it a private, transient identifier (a blank node identifier), and then use that identifier where you'd normally use a URI as a subject or object. The transient identifier has no meaning outside of the context of the document: it's simply a way of referring to the same (essentially anonymous) entity in multiple places within the document.

A blank node value is expressed by writing an opening square bracket, followed by the sets of predicates and values for the blank node, followed by a closing square bracket. For example, we can state that an author of the book is a nondescript entity who we know is a person named `Nicola Strickland`, but for whom we don't have an identifier:

```
<http://ialm.int/books/9781899066100#id> dct:creator [
        a foaf:Person ;
        foaf:name "Nicola Strickland"
] .
```

Blank node identifiers are written similarly to the compressed form of URIs, except that an underscore is used as the prefix. For example, `_:johnsmith`. You don't have to do anything special to create a blank node identifier (simply use it), and the actual name you assign has no meaning outside of the context of the document—if you replace all instances of `_:johnsmith` with `_:zebra`, the actual meaning of the document is unchanged—although it may be slightly more confusing to read and write as a human.

Multi-lingual string literals

String literals in the examples given so far are written in no particular language (which may be appropriate in some cases, particularly when expressing people's names).

The language used for a string literal is indicated by writing the literal value, followed by an at-sign, and then the ISO 639-1 language code, or an ISO 639-1 language code, followed by a hyphen, and a ISO 3166-1 alpha-2 country code.

For example: `"Intergalatic Alliance Library & Museum Homepage"@en`, or `"grey"@en-gb`.

Base URIs

By default, the base URI for the terms in a Turtle document is the URI it's being served from. Occasionally, it can be useful to specify an alternative base URI. To do this, an `@base` statement can be included (in a similar fashion to `@prefix`).

For example, if a document specifies `@base <http://www.example.com/things/> .`, then the URI `<12447652#id>` within that document can be expanded to `<http://www.example.com/things/12447652#id>`, while the URI `</artefacts/47fb01>` would be expanded to `<http://www.example.com/artefacts/47fb01>`.

> i For further information on RDF's capabilities and Turtle, be sure to read through the RDF Primer and the Turtle specification.

An example of a Turtle document making use of some of these capabilities is shown below:

```
@base <http://ialm.int/> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix dct: <http://purl.org/dc/terms/> .
@prefix bibo: <http://purl.org/ontology/bibo/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .

</books/9781899066100#id>
        a bibo:Book ;
        dct:title "Acronyms and Synonyms in Medical Imaging"@en ;
        dct:issued "1997"^^xsd:gYear ;
        dct:creator _:allison, _:strickland ;
        dct:publisher [
                a foaf:Organization ;
                foaf:name "CRC Press"
        ] .

_:strickland
        a foaf:Person ;
        foaf:name "Nicola Strickland" .

_:allison
        a foaf:Person ;
        foaf:name "David J. Allison" .
```

In this example, we are still describing our book, but we specify that the title is in English (though don't indicate any particular national variant of English); we state that it was issued (published) in the year 1997, and that it's publisher—for whom we don't have an identifier—is an organisation whose name is *CRC Press*.

2.8 From three to four: relaying provenance with quads

While triples are a perfectly servicable mechanism for describing something, they don't have the ability to tell you where data is *from* (unless you impose a restriction that you only deal with data where the domain of the subject URI matches that of the server you're retrieving from). In some systems, including Acropolis, this limitation is overcome by introducing another element: a graph URI, identifying the *source* of a triple. Thus, instead of triples, RES actually stores *quads*.

When we assign an explicit URI to a graph in this way, it becomes known as a *named graph*—that is, a graph with an explicit identifier (name) assigned to it.

Turtle itself doesn't have a concept of named graphs, but there is an extension to Turtle, named TriG, which includes the capability to specify the URI of a named graph containing a particular set of triples.

> While Quilt will serve RDF/XML and Turtle when requested, it will also serve TriG: this allows applications to determine the provenance of statements stored in the RES index, allowing them to white– or black-list data sources if needed.

2.9 Why does RES use RDF?

RDF isn't necessarily the simplest way of expressing some data about something, and that means it's often not the first choice for publishers and consumers. Often, an application consuming some data is designed specifically for one particular dataset, and so its interactions are essentially bespoke and comparatively easy to define.

RES, by nature, brings together a large number of different structured datasets, describing lots of different kinds of things, with a need for a wide range of applications to be able to work with those sets in a consistent fashion.

At the time of writing (ten years after its introduction), RDF's use of URIs as identifiers, common vocabularies and data types, inherent flexibility and well-defined structure means that is the only option for achieving this.

Whether you're describing an audio clip or the year 1987, a printed book or the concept of a documentary film, RDF provides the ability to express the data you hold in intricate detail, without being beholden to a single central authority to validate the modelling work undertaken by experts in your field.

For application developers, the separation of grammar and vocabularies means that applications can interpret data in as much or as little detail as is useful for the end-users. For instance, you might develop an application which understands a small set of general-purpose metadata terms but which can be used with virtually everything surfaced through RES.

Alternatively, you might develop a specialist application which interprets rich descriptions in a particular domain in order to target specific use-cases. In either case, you don't need to know who the data comes from, only sufficient understanding of the vocabularies in use to satisfy your needs.

However, because we recognise that publishing and consuming Linked Open Data as an individual publisher or application developer may be unfamiliar territory, and so throughout the lifetime of the project we are committed to publishing documentation, developing tools and operating workshops in order to help developers and publishers work with RDF in general and RES in particular more easily.

2.10 Further reading

- *RDF 1.1 primer*
- *RDF 1.1 Turtle* (language specification)
- *RDF 1.1 TriG* (language specification)
- *Linked Data Patterns*
- *Linked Data - The Story so Far* (PDF)

3 Under the hood: the architecture of Acropolis

This section will be expanded significantly in future editions.

4 Requirements for publishers

Publishers wishing to make their data visible in the Acropolis index and useable by RES applications must conform to a small set of basic requirements. These are:

- The data must be expressed as RDF and published as Linked Open Data;
- the data must be licensed under permissive terms (in particular, it must allow re-use in both commercial and non-commercial applications);
- the licensing terms must be included in the data itself so that consumers can perform automated due diligence before using it;
- the data should use the vocabularies described in this book for best results (although you are free to use other vocabularies too).

Although RES requires that you publish Linked Open Data, that doesn't mean you can't also publish your data in other ways. While human-facing HTML pages are the obvious example, there's nothing about publishing Linked Open Data which means you can't also publish JSON with a bespoke schema, CSV, OpenOffice.org spreadsheets, or operate complex query APIs requiring registration to use.

In fact, best practice generally is that you publish in as many formats as you're generally able to, and do so in a consistent fashion. And, while your "data views" (that is, the structured machine-readable representations of your data about things) are going to be very dull and uninteresting to most human beings, that doesn't mean that you can't serve nicely-designed web pages about them as the serialisation for ordinary web browsers.

4.1 Technical requirements for data publication

4.1.1 Support the most common RDF serialisations

RDF can be serialised in a number of different ways, but there are two serialisations which RES publishers **must** provide because these are the two serialisations guaranteed to be supported by RES applications:

Name	Media type	Further information
Turtle	`text/turtle`	http://www.w3.org/TR/2014/REC-turtle-20140225/
RDF/XML	`application/rdf+xml`	http://www.w3.org/TR/2014/REC-rdf-syntax-grammar-20140225/

Turtle is increasingly the most common RDF serialisation in circulation and is very widely-supported by processing tools and libraries.

RDF/XML is an older serialisation which is slightly more well-supported than Turtle. RDF/XML is often more verbose than the equivalent Turtle expression of a graph, but as an XML-based format can be generated automatically from other kinds of XML using XSLT.

If you are considering publishing your data as JSON, you may consider publishing it as JSON-LD, a serialisation of RDF which is intended to be useful to consumers which don't understand RDF specifically. JSON-LD isn't currently supported by RES, but may be in the future.

The RES crawler will request Turtle by preference.

4.1.2 Describe the document and serialisations as well as the item

A minimal RDF serialisation intended for use by RES must include data about three distinct subjects:

Subject	Example
Document URL	`http://ialm.int/books/9781899066100`

Representation URL	`http://ialm.int/books/9781899066100.ttl`
Item URI	`http://ialm.int/books/9781899066100#id`

It is recommended that publishers describe any other serialisations which they are making available as well, but it is not currently a requirement to do so.

A description of the metadata which should be served about the document and representations is included in the Metadata about documents section.

4.1.3 Include licensing information in the data

The data about the document or representation **must** include a rights information predicate referring to the well-known URI of a supported license. See the Metadata describing rights and licensing section for further details.

> ! The RES crawler will discard data which does not include licensing data, because without it, the data cannot be used by RES applications.

4.1.4 Link to the RDF representations from the HTML variant

In your HTML representations, use the `<link>` element (within the `<head>` element) with a `rel` attribute of `"alternate"` in order to link to the other representations of the same document:

```
<link rel="alternate" type="application/rdf+xml"
href="/books/9781899066100.rdf">
<link rel="alternate" type="text/turtle" href="/books/9781899066100.ttl">
```

While it's less efficient than content negotiation (see below) for both consuming applications and for your server to access your alternative serialisations this way, linking to them from your HTML provides a useful fall-back capability in the event that content negotiation fails or has to be disabled—for example, if you need to switch your website to be served from a content delivery network which doesn't support negotiation.

It's not the preferred option because consumers must first obtain the HTML, parse it, and then request the RDF. Often, generating the HTML page will also be more expensive than generating the equivalent RDF serialisations.

4.1.5 Perform content negotiation when requests are received for item URIs

If you use fragment-based URIs, this means that your web server must be configured to perform content negotiation on requests received for the portion of the URI before the hash (`#`) sign.

For example, if your subject URIs are in the form:

```
http://ialm.int/books/9781899066100#id
```

Then when your server receives requests for the document:

```
/books/9781899066100
```

It should perform content negotiation and return an appropriate media type, including the supported RDF serialisations if requested.

When sending a response, the server **must** send an appropriate `Vary` header, and should send a `Content-Location` header referring to the representation being served. For example:

```
HTTP/1.0 OK
Server: Apache/2.2 (Unix)
Vary: Accept
Content-Type: text/turtle; charset=utf-8
Content-Location: /books/9781899066100.ttl
Content-Length: 272
…
```

The Apache web server automatically sends the correct headers when configured to perform Content Negotiation on a set of static files. See the Apache mod_negotiation module documentation for further details on its configuration.

5 Requirements for consumers

Applications built for RES must be able to understand the index assembled by Acropolis, as well as the source data it refers to. Practically, it means that they must be able to:

- retrieve and process RDF from remote web servers;
- interpret at least the “common metadata” vocabularies described in this book.

This section will be expanded significantly in future editions.

6 Common metadata

Vocabularies used in this section:

Vocabulary	Namespace URI	Prefix
RDF syntax	`http://www.w3.org/1999/02/22-rdf-syntax-ns#`	`rdf:`
RDF schema	`http://www.w3.org/2000/01/rdf-schema#`	`rdfs:`
DCMI terms	`http://purl.org/dc/terms/`	`dct:`
FOAF	`http://xmlns.com/foaf/0.1/`	`foaf:`

Dublin Core Metadata Initiative (DCMI) Terms is an extremely widely-used general-purpose metadata vocabulary which can be used in the first instance to describe both web and abstract resources.

In particular, the following predicates are recognised by Acropolis itself and may be relayed in the RES index:

Predicate	Meaning
`dct:title`	Specifies the formal title of an item
`dct:rights`	Specifies a URI for rights information (see Metadata describing rights and licensing)
`dct:license`	Alternative predicate for specifying rights information
`dct:subject`	Specifies the subject of something

The FOAF vocabulary also includes some general-purpose predicates:

Predicate	Meaning
`foaf:primaryTopic`	Specifies the primary topic of a document

`foaf:homepage`	Specifies the canonical homepage for something
`foaf:topic`	Specifies a topic of a page (may be used instead of `dct:subject`)
`foaf:depiction`	Specifies the URL of a still image which depicts the subject

6.1 Referencing alternative identifiers: expressing equivalence

Vocabularies used in this section:

Vocabulary	Namespace URI	Prefix
OWL	`http://www.w3.org/2002/07/owl#`	`owl:`

Linked Open Data in general, and RES in particular, is at its most useful when the data describing things links to other data describing the same thing.

In RDF, this is achieved using the `owl:sameAs` predicate. This predicate implies a direct equivalence relationship—in effect, it creates a synonym.

You can use `owl:sameAs` whether or not the alternative identifiers use `http:` or `https:`, although the usefulness of URIs which aren't resolveable is limited.

For example, one might wish to specify that our book has an ISBN using the `urn:isbn:` URN scheme [RFC3187]:

```
</books/9781899066100#id> owl:sameAs <urn:isbn:9781899066100> .
```

We can also indicate that the book described by our data refers to the same book at the British Library:

```
</books/9781899066100#id> owl:sameAs
<http://bnb.data.bl.uk/id/resource/011012558> .
```

Take care when using `owl:sameAs` to ensure that the subject and the object really are directly equivalent. In particular, make sure that you don't accidentally state that somebody's description of something (be it an HTML page or some other serialisation) is the same as the thing being described.

6.2 Metadata describing rights and licensing

Vocabularies used in this section:

Vocabulary	Namespace URI	Prefix
DCMI terms	`http://purl.org/dc/terms/`	`dct:`
ODRL 2.0	`http://www.w3.org/ns/odrl/2/`	`odrl:`

The data describing digital assets (including RDF representations themselves) must include explicit licensing data in order for it to be indexed by Acropolis and used by RES applications. Additionally, the RDF data must be licensed according to the terms of a supported permissive licence.

In order to express this, you can use the `dct:rights` or `dct:licence` predicates (at your option). Where the subject is an RDF representation, the object of the statement must be the well-known URI of a supported licence (see below). For other kinds of digital asset, the object of the statement can either be a well-known URI of a supported licence, or a reference to a set of terms described in RDF using the ODRL 2.0 vocabulary.

6.2.1 Well-known licences

The Acropolis crawler discards RDF data which is not explicitly licensed using one of the well-known licenses listed below. Note that the URI listed here is the URI which must be used as the object in the licensing statement.

Licence	URI
Creative Commons Public Domain (CC0)	http://creativecommons.org/publicdomain/zero/1.0/
Library of Congress Public Domain	http://id.loc.gov/about/
Creative Commons Attribution 4.0 International (CC BY 4.0)	http://creativecommons.org/licenses/by/4.0/
Open Government Licence	http://reference.data.gov.uk/id/open-government-licence
Digital Public Space Licence, version 1.0	http://bbcarchdev.github.io/licences/dps/1.0#id
Creative Commons 1.0 Generic (CC BY 1.0)	http://creativecommons.org/licenses/by/1.0/
Creative Commons 2.5 Generic (CC BY 2.5)	http://creativecommons.org/licenses/by/2.5/
Creative Commons 3.0 Unported (CC BY 3.0)	http://creativecommons.org/licenses/by/3.0/
Creative Commons 3.0 US (CC BY 3.0 US)	http://creativecommons.org/licenses/by/3.0/us/

The following example specifies that the Turtle representation of the data about our book is licensed according to the terms of the *Creative Commons Attribution 4.0 International licence*.

```
</books/9781899066100.ttl> dct:rights
<http://creativecommons.org/licenses/by/4.0/> .
```

See the Metadata describing documents section for further details on describing representations.

6.2.2 ODRL-based descriptions

This section will be expanded significantly in future editions.

7 Describing digital assets

7.1 Metadata describing documents

Vocabularies used in this section:

Vocabulary	Namespace URI	Prefix
RDF syntax	`http://www.w3.org/1999/02/22-rdf-syntax-ns#`	`rdf:`
DCMI terms	`http://purl.org/dc/terms/`	`dct:`
DCMI types	`http://purl.org/dc/dcmitype/`	`dmcit:`
FOAF	`http://xmlns.com/foaf/0.1/`	`foaf:`
W3C formats registry	`http://www.w3.org/ns/formats/`	`formats:`

7.1.1 Describing your document

If the document is actually a data-set, see also the Collections and data-sets section.

Give the document a class of `foaf:Document`:

```
</books/9781899066100> a foaf:Document .
```

Give the document a title:

```
</books/9781899066100> dct:title "'Acronyms and Synonyms in Medical Imaging'
at the Intergalatic Alliance Library & Museum"@en .
```

If the document is not a data-set, specify the primary topic (that is, the URI of the thing described by the document):

```
</books/9781899066100> foaf:primaryTopic </books/12345#id> .
```

Link to each of the serialisations:

```
</data/9781899066100> dct:hasFormat </data/9781899066100.ttl> .
</data/9781899066100> dct:hasFormat </data/9781899066100.html> .
```

7.1.2 Describe each of your serialisations

Use a member of the DCMI type vocabulary as a class:

```
</books/9781899066100.ttl> a dmcit:Text .
```

Where available, use a member of the W3C formats vocabulary as a class:

```
</books/9781899066100.ttl> a formats:Turtle .
```

Use the `dct:format` predicate, along with the MIME type beneath the `http://purl.org/NET/mediatypes/` tree:

```
</books/9781899066100.ttl> dct:format
<http://purl.org/NET/mediatypes/text/turtle> .
```

Give the serialisation a specific title:

```
</books/9781899066100.ttl> dct:title "Description of 'Acronyms and Synonyms in
Medical Imaging' as Turtle (RDF)"@en .
```

Specify the licensing terms for the serialisation, if applicable:

```
</books/9781899066100.ttl> dct:rights
<http://creativecommons.org/licenses/by/4.0/> .
```

See the Metadata describing rights and licensing section for details on the licensing statements required by RES, as well as information about supported licences.

7.1.3 Example

```
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix dct: <http://purl.org/dc/terms/> .
@prefix dmcit: <http://purl.org/dc/dcmitype/> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix formats: <http://www.w3.org/ns/formats/> .

</data/9781899066100>
        a foaf:Document ;
        dct:title "'Acronyms and Synonyms in Medical Imaging' at the
Intergalatic Alliance Library & Museum"@en .
        foaf:primaryTopic </books/12345#id> ;
        dct:hasFormat
                </data/9781899066100.ttl> ,
                </data/9781899066100.html> .

</data/9781899066100.ttl>
        a dcmit:Text, formats:Turtle ;
        dct:format <http://purl.org/NET/mediatypes/text/turtle> ;
        dct:title "Description of 'Acronyms and Synonyms in Medical Imaging'
as Turtle (RDF)"@en ;
        dct:rights <http://creativecommons.org/licenses/by/4.0/> .

</data/9781899066100.html>
        a dcmit:Text ;
        dct:format <http://purl.org/NET/mediatypes/text/html> ;
        dct:title "Description 'Acronyms and Synonyms in Medical Imaging' as a
web page"@en .
```

7.2 Collections and data-sets

Vocabularies used in this section:

Vocabulary	Namespace URI	Prefix
DCMI terms	`http://purl.org/dc/terms/`	`dct:`
VoID	`http://rdfs.org/ns/void#`	`void:`

7.2.1 Data-set auto-discovery

7.3 Images

7.4 Video

7.5 Audio

8 Describing people, projects and organisations

9 Describing places

10 Describing events

11 Describing concepts and taxonomies

Vocabularies used in this section:

Vocabulary	Namespace URI	Prefix
SKOS	`http://www.w3.org/2008/05/skos#`	`skos:`

12 Describing creative works

13 Tools and resources

Vocabulary Index

Vocabulary	Namespace URI	Prefix	Section
RDF syntax	`http://www.w3.org/1999/02/22-rdf-syntax-ns#`	`rdf:`	Common metadata
RDF schema	`http://www.w3.org/2000/01/rdf-schema#`	`rdfs:`	Common metadata
DCMI Metadata Terms	`http://purl.org/dc/terms/`	`dct:`	Common metadata, Metadata describing rights and licensing, Collections and data-sets
DCMI Types	`http://purl.org/dc/dcmitype/`	`dmcit:`	Metadata describing documents
FOAF	`http://xmlns.com/foaf/0.1/`	`foaf:`	Common metadata
ODRL 2.0	`http://www.w3.org/ns/odrl/2/`	`odrl:`	Metadata describing rights and licensing
OWL	`http://www.w3.org/2002/07/owl#`	`owl:`	Referencing alternative identifiers: expressing equivalence
SKOS	`http://www.w3.org/2008/05/skos#`	`skos:`	Describing concepts and taxonomies

VoID	`http://rdfs.org/ns/void#`	`void:`	Collections and data-sets
W3C formats registry	`http://www.w3.org/ns/formats/`	`formats:`	Metadata describing documents