Parsing RDF documents using PHP

"An introduction to the RDF XML syntax and how to parse RDF documents using the PHP version of Repat."

(c) 2002 by Luis Argerich
lrargerich@yahoo.com

Introduction

RDF is a model to represent metadata defined by the W3C, it is a key resource in the "semantic web" activities. RDF can be used to describe almost anything, web sites, pages, documents, books, collections, soccer matches, etc. The XML Syntax of RDF defines a vendor-neutral syntax to represent RDF information and exchange information between systems. While RDF is great to express metadata parsing RDF is complicated since there're many syntaxes allowed and covering all the valid syntactic representations requiere sometimes a lot of code/time. In this tutorial we present the Rdf_class, a port of the RDF repat parser to PHP. With this class you'll be able to parse any kind of RDF file from PHP scripts and then you will be able to write all sort of RDF-enabled applications from repositories to querying engines or data exchanging applications.

What is RDF?

RDF is a model for describing metadata.

TheW3C defines RDF:

"Resource Description Framework (RDF) is a foundation for processing metadata; it provides interoperability between applications that exchange machine-understandable information on the Web."

The RDF model is based on statements, an statement is used to say "something" about resources. The "something" is referred as a "property" of the resource, so an statement indicates the value of a resource's property. Let's see how the W3C defines Resource, Properties and Statements.

Resources:

All things being described by RDF expressions are called *resources*. A resource may be an entire Web page; such as the HTML document "http://www.w3.org/Overview.html" for example. A resource may be a part of a Web page; e.g. a specific HTML or XML element within the document source. A resource may also be a whole collection of pages; e.g. an entire Web site. A resource may also be an object that is not directly accessible via the Web; e.g. a printed book. Resources are always named by URIs plus optional anchor ids (see [URI]). Anything can have a URI; the extensibility of URIs allows the introduction of identifiers for any entity imaginable.

Properties:

A *property* is a specific aspect, characteristic, attribute, or relation used to describe a resource. Each property has a specific meaning, defines its permitted values, the types of resources it can describe, and its relationship with other properties. This document does not address how the characteristics of properties are expressed; for such information, refer to the RDF Schema specification).

Statements:

A specific resource together with a named property plus the value of that property for that resource is an RDF *statement*. These three individual parts of a statement are called, respectively, the *subject*, the *predicate*, and the *object*. The object of a statement (i.e., the property value) can be another resource or it can be a literal; i.e., a resource (specified by a URI) or a simple string or other primitive datatype defined by XML. In RDF terms, a *literal* may have content that is XML markup but is not further evaluated by the RDF processor. There are some syntactic restrictions on how markup in literals may be expressed

Every statement has 3 parts:

- Subject
- Predicate
- Object

Staements, an example.

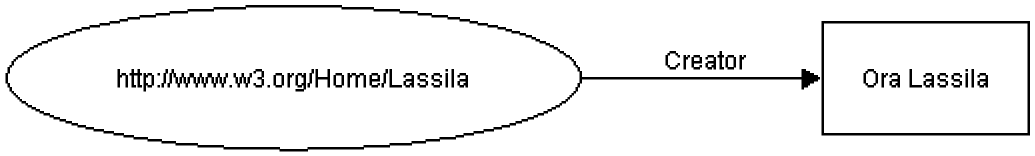
Example:

Ora Lassila is the creator of the resource http://www.w3.org/Home/Lassila.

This sentence has the following parts:

Subject (Resource)	http://www.w3.org/Home/Lassila
Predicate (Property)	Creator
Object (literal)	"Ora Lassila"

We can also see that RDF is also a model for describing graphs, where subjects and objects are graph nodes and the predicates define directed arcs from a subject to an object, we could have represented the above statement using the following graph:



The XML syntax for RDF

RDF is a model for describing metadata and can be represented using several syntaxes, the most used syntax for RDF is the XML syntax that we are going to describe below.

When RDF is represented using XML all the statements are enclosed in an RDF element, the RDF element should have a namespace prefix pointing to the RDF syntax specification:

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
</rdf:RDF>
```

In the XML syntax for RDF you "describe" resources indicating resource properties and resource values, a "Description" element is used to describe a resource. Properties are represented as XML elements and objects are the content of the properties, our sample statement can be represented in XML as follows:

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:s="http://description.org/schema/">
  <rdf:Description about="http://www.w3.org/Home/Lassila">
    <s:Creator>Ora Lassila</s:Creator>
  </rdf:Description>
</rdf:RDF>
```

We can see that we describe the resource " http://www.w3.org/Home/Lassila" indicating the name of the Creator. The property has the namespace prefix "s" and the namespace URI may describe what the property says about the resource.

Parsing RDF documents using class_rdf_parser.php

The RDF parser class is a PHP por of "repat" an RDF parser based on Expat by Jason Williams, the "C" version of repat uses the "C" version of expat while the PHP version of repat uses, you guess, the expat parser bundled in PHP. (AKA XML extension).

Repat is an event-driven RDF parser, basically you define a handler for "RDF" statements and then just parse the document, whenever the parser identifies an RDF statement yur callback function will be called.

This is an example on how to use the class:

```
/* Put the URI of the document to parse here, you can use a file name if
you want too */
$base="http://luigi.melpomenia.com.ar/example5.rdf";

$statements=0;
$input = fopen($base,"r");
$rdf=new Rdf_parser();
$rdf->rdf_parser_create( NULL );
$rdf->rdf_set_user_data( $statements );
$rdf->rdf_set_statement_handler( "my_statement_handler" );
$rdf-
$rdf->rdf_set_base($base );
$done=false;
while(!$done)
{
  $buf = fread( $input, 512 );
  $done = feof($input);
  if ( ! $rdf->rdf_parse( $buf, strlen($buf), feof($input) ) )
  {
    die("An error was detected while parsing may be the document is not
well-formed");
  }
}
/* close file. */
fclose( $input );
$rdf->rdf_parser_free();
printf( "Total statements: ". $statements );
```

If you are familiar with the XML parsing functions of PHP you'll find the above code easy to understand.

Note that we parse the RDF document using 512 byte chunks, we can define the buffer size as we want (we control how we read the RDF document). Note that this allows us to parse really huge RDF documents cosuming a constant amount of memory that is a good advantage to prevent a server crash if our script parses huge files.

So we have to define an statement handler, this is an example:

```
function my_statement_handler(
      &$user_data,
      $subject_type,
      $subject,
      $predicate,
      $ordinal,
      $object_type,
      $object,
      $xml_lang )
```

```
{
      //$statements = $user_data;
      ++$user_data;
      printf( "ordinal($ordinal) triple(" );
      switch( $subject_type )
      {
      case RDF_SUBJECT_TYPE_URI:
            printf( "\"%s\"", $subject );
            break;
      case RDF_SUBJECT_TYPE_DISTRIBUTED:
            printf( "distributed(\"%s\")", $subject );
            break;
      case RDF_SUBJECT_TYPE_PREFIX:
            printf( "prefix(\"%s\")", $subject );
            break;
      case RDF_SUBJECT_TYPE_ANONYMOUS:
            printf( "anonymous(\"%s\")", $subject );
            break;
      }

      printf( ", \"%s\", ", $predicate );

      switch( $object_type )
      {
      case RDF_OBJECT_TYPE_RESOURCE:
            printf( "\"%s\"", $object );
            break;
      case RDF_OBJECT_TYPE_LITERAL:
            printf( "literal(\"%s\")", $object );
            break;      case RDF_OBJECT_TYPE_XML:
            printf( "XML" );
            break;
      }

      printf( ")<br/>\n" );
}
```

The handler receives several paramenters, let's see each one:

- user_data: This is a PHP variable set with rdf_set_user_data, all the rdf parser handlers will receive this variable and you can do whatever you want with it. You can ignore for example if you don't need it. In our example we passed a $statements variable and we use it to count the number of statements in the RDF document.

- subject_type: The subject type of the RDF statement, can be one of the following values:
    - RDF_SUBJECT_TYPE_URI
    - RDF_SUBJECT_TYPE_DISTRIBUTED
    - RDF_SUBJECT_TYPE_PREFIX
    - RDF_SUBJECT_TYPE_ANONYMOUS

  We are going to see what each subject_type means later in this tutorial.

- subject: The subject of the RDF statement

- predicate: The predicate of the RDF statement

- object_type: The object type of the RDF statement, can be one of the following values:
    - RDF_OBJECT_TYPE_RESOURCE
    - RDF_OBJECT_TYPE_LITERAL
    - RDF_OBJECT_TYPE_XML

- object: The object value of the RDF statement.

- ordinal: If the statement is a member of a collection this argument has the ordinal position of the member in the collection

- xml_lang: The value of the xml_lang attribute if set.

There're other handlers that can be set but they are optional, we can parse rdf files using just the statement handler, we'll describe the other handlers later in this tutorial.

Simple statements

The rdfdump.php script let's you parse a RDF document outputting the statements as they are found. We'll use the script to parse and see how to process several examples as we explain the RDF syntax.

If we use our rdfdump.php script to parse the presented document (example1.rdf) we get the folllowing output:

```
ordinal(0) triple("http://www.w3.org/Home/Lassila", "http://description.org/schema/Creator", literal("Ora Lassila"))
Total statements: 1
```

So we can see that the parser returns "http://www.w3.org/Home/Lassila" as the subject and that it is a SUBJECT_TYPE_URI, the predicate is "http://description.org/schema/Creator" and the object is "Ora Lassila", being the object_type: OBJECT_TYPE_LITERAL

Statement 1	
subject	http://www.w3.org/Home/Lassila
subject_type	SUBJECT_TYPE_URI
predicate	http://description.org/schema/Creator
object	Ora Lassila
object_type	OBJECT_TYPE_LITERAL

Using the default namespace

We can also write the same using the default namespace for RDF elements:

```
<?xml version="1.0"?>
<RDF
  xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:s="http://description.org/schema/">
  <Description about="http://www.w3.org/Home/Lassila">
    <s:Creator>Ora Lassila</s:Creator>
  </Description>
</RDF>
```

This is example1b.rdf and the parser produces the same result.

Properties as attributes

RDF also allows an abbreviated syntax where properties can be written as XML attributes:

```
<?xml version="1.0"?>
<RDF
  xmlns="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:s="http://description.org/schema/">
  <Description about="http://www.w3.org/Home/Lassila"
                   s:Creator="Ora Lassila" />

</RDF>
```

This is example1c.rdf and the parser again produces exactly the same result.

This is an example with more than one property:

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:s="http://description.org/schema/">
  <rdf:Description about="http://www.w3.org">
    <s:Publisher>World Wide Web Consortium</s:Publisher>
    <s:Title>W3C Home Page</s:Title>
    <s:Date>1998-10-03T02:27</s:Date>
  </rdf:Description>
</rdf:RDF>
```

The parser will produce 3 statements. The baove document is equivalent to example2b.rdf:

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:s="http://description.org/schema/">
  <rdf:Description about="http://www.w3.org"
       s:Publisher="World Wide Web Consortium"
       s:Title="W3C Home Page"
       s:Date="1998-10-03T02:27"/>
</rdf:RDF>
```

Resources as property values

In the examples we've seen so far property values (objects) were represented with string literals. But we can also use a "resource" as the value of a resource property, example:

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:s="http://description.org/schema/">
  <rdf:Description about="http://www.w3.org/Home/Lassila">
    <s:Creator rdf:resource="http://www.w3.org/staffId/85740"/>
  </rdf:Description>

  <rdf:Description about="http://www.w3.org/staffId/85740">
    <s:Name>Ora Lassila</s:Name>
    <s:Email>lassila@w3.org</s:Email>
  </rdf:Description>
</rdf:RDF>
```

This is example3.rdf, you can see that the "creator" of the "http://www.w3.org/Home/Lassila" resource is another resource, the attribute resource is used in the XML RDF notation to indicate that a property value is a resource. Then we describe some properties about that resource.

The RDF Dump produces the following output:

```
ordinal(0) triple("http://www.w3.org/Home/Lassila", "http://description.org/schema/Creator",
"http://www.w3.org/staffId/85740")
ordinal(0) triple("http://www.w3.org/staffId/85740", "http://description.org/schema/Name", literal("Ora Lassila"))
ordinal(0) triple("http://www.w3.org/staffId/85740", "http://description.org/schema/Email", literal("lassila@w3.org"))
Total statements: 3
```

The object_type of the first statement is RDF_OBJECT_TYPE_RESOURCE indicatin g that the object is a resource that may be described in the same RDF document or not.

This is what the parser produces

Statement 1	
subject	http://www.w3.org/Home/Lassila
subject_type	RDF_SUBJECT_TYPE_URI
predicate	http://description.org/schema/Creator
object	Ora Lassila
object_type	RDF_OBJECT_TYPE_RESOURCE

Statement 2	
subject	http://www.w3.org/staffId/85740
subject_type	RDF_SUBJECT_TYPE_URI

predicate	http://description.org/schema/Name
object	Ora Lassila
object_type	RDF_OBJECT_TYPE_LITERAL

Statement 3	
subject	http://www.w3.org/staffId/85740
subject_type	RDF_SUBJECT_TYPE_URI
predicate	http://description.org/schema/Email
object	lassila@w3.org
object_type	RDF_OBJECT_TYPE_LITERAL

Inner description elements

example3b.rdf shows the same document using another syntax form:

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:s="http://description.org/schema/">
  <rdf:Description about="http://www.w3.org/Home/Lassila">
    <s:Creator>
      <rdf:Description about="http://www.w3.org/staffId/85740">
        <s:Name>Ora Lassila</s:Name>
        <s:Email>lassila@w3.org</s:Email>
      </rdf:Description>
    </s:Creator>
  </rdf:Description>
</rdf:RDF>
```

In this format there's a inner Description element for the "Creator" property so the value of the creator property is the resource indicated by the "about" attribute in the inner Description element. The parser as you can guess produces the same output.

Properties as attributes

Using attributes to describe properties we can write the same as:

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:s="http://description.org/schema/">
<rdf:Description about="http://www.w3.org/Home/Lassila">
    <s:Creator rdf:resource="http://www.w3.org/staffId/85740"
       s:Name="Ora Lassila"
       s:Email="lassila@w3.org" />
  </rdf:Description>
```

```
</rdf:RDF>
```

The above example is example3c.rdf

Containers

Frequently you have to refer to a collection of resources, for example you may want to say that a course is attended by "n" students. The RDF model defines "containers" to group resources. There're 3 different containers:

Bag:

An unordered list of resources or literals. *Bag*s are used to declare that a property has multiple values and that there is no significance to the order in which the values are given. *Bag* might be used to give a list of part numbers where the order of processing the parts does not matter. Duplicate values are permitted.

Sequence:

An ordered list of resources or literals. *Sequence* is used to declare that a property has multiple values and that the order of the values is significant. *Sequence* might be used, for example, to preserve an alphabetical ordering of values. Duplicate values are permitted.

Alternative:

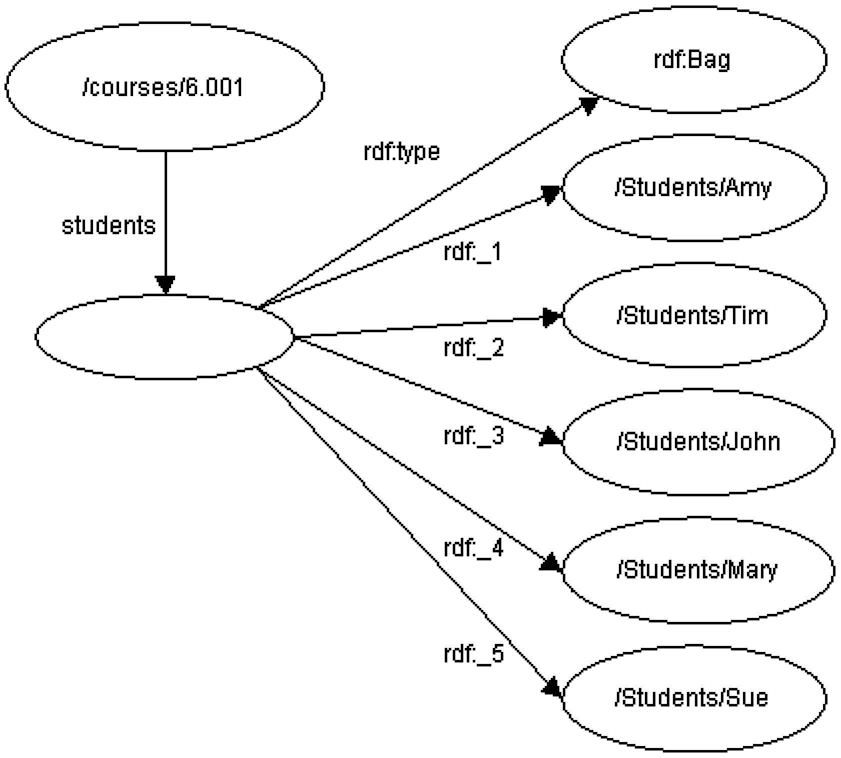
A list of resources or literals that represent alternatives for the (single) value of a property. *Alternative* might be used to provide alternative language translations for the title of a work, or to provide a list of Internet mirror sites at which a resource might be found. An application using a property whose value is an *Alternative* collection is aware that it can choose any one of the items in the list as appropriate.

A collection is represented by a resource where the "type" property indicates that the resource is a collection, type may be "Bag", "Seq" or "Alt". Collection members are represented as properties named "_1", "_2", etc.

For example the following statement defines a collection:

The students in course 6.001 are Amy, Tim, John, Mary, and Sue.

And the statement can be modeled as a graph in the form:



And this is the XML RDF document representing the above graph:

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:s="http://description.org/schema/">
<rdf:Description about="http://mycollege.edu/courses/6.001">
    <s:students>
      <rdf:Bag>
        <rdf:li resource="http://mycollege.edu/students/Amy"/>
        <rdf:li resource="http://mycollege.edu/students/Tim"/>
        <rdf:li resource="http://mycollege.edu/students/John"/>
        <rdf:li resource="http://mycollege.edu/students/Mary"/>
        <rdf:li resource="http://mycollege.edu/students/Sue"/>
      </rdf:Bag>
    </s:students>
  </rdf:Description>
</rdf:RDF>
```

Note that there're some "anonymous" resources such as the rdf:Bag and the rdf:li elements.An anonymous resource has subject_type = SUBJECT_TYPE_ANONYMOUS The document presented is example4.rdf and the rdfdump.php script dumps the following:

```
ordinal(0) triple(anonymous("http://luigi.melpomenia.com.ar/example4.rdf#genid1)", "http://www.w3.org/1999/02/22-rdf-
syntax-ns#type", "http://www.w3.org/1999/02/22-rdf-syntax-ns#Bag")
ordinal(0) triple("http://mycollege.edu/courses/6.001", "http://description.org/schema/students",
"http://luigi.melpomenia.com.ar/example4.rdf#genid1")
ordinal(1) triple(anonymous("http://luigi.melpomenia.com.ar/example4.rdf#genid1)", "http://www.w3.org/1999/02/22-rdf-
syntax-ns#_1", "http://mycollege.edu/students/Amy")
ordinal(2) triple(anonymous("http://luigi.melpomenia.com.ar/example4.rdf#genid1)", "http://www.w3.org/1999/02/22-rdf-
syntax-ns#_2", "http://mycollege.edu/students/Tim")
ordinal(3) triple(anonymous("http://luigi.melpomenia.com.ar/example4.rdf#genid1)", "http://www.w3.org/1999/02/22-rdf-
syntax-ns#_3", "http://mycollege.edu/students/John")
ordinal(4) triple(anonymous("http://luigi.melpomenia.com.ar/example4.rdf#genid1)", "http://www.w3.org/1999/02/22-rdf-
syntax-ns#_4", "http://mycollege.edu/students/Mary")
ordinal(5) triple(anonymous("http://luigi.melpomenia.com.ar/example4.rdf#genid1)", "http://www.w3.org/1999/02/22-rdf-
syntax-ns#_5", "http://mycollege.edu/students/Sue")
Total statements: 7
```

Note how the ordinal argument can be used to obtain the number of each element in a collection. Let's see each statement:

The first statement says that the anonymous resource http://luigi.melpomenia.com.ar/example4.rdf#genid1 is of type "Bag"

Then we say that the resource http://mycollege.edu/courses/6.001 has "students" defined by the resource http://luigi.melpomenia.com.ar/example4.rdf#genid1 (The anonymous resource introduced before)

Then we present properties _1,_2,_3,_4 and _5 for the anonymous resource (The Bag) indicating the students that form the course.

So we have a "course" resource that has a "students" property pointing to an anonymous resource with type "Bag", and then we define _1.._5 properties for that anonymous resource.

RDF_SUBJECT_TYPE_ANONYMOUS will be the "subject_type" for a statement where the subject is anonymous. Note that when a subject is anonymous the parser creates an URI to identy it using the base URI and a "#genidX" locator.

example5.rdf shows a collection of type "Alt":

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:s="http://description.org/schema/">
  <rdf:Description about="http://x.org/packages/X11">
    <s:DistributionSite>
      <rdf:Alt>
        <rdf:li resource="ftp://ftp.x.org"/>
        <rdf:li resource="ftp://ftp.cs.purdue.edu"/>
        <rdf:li resource="ftp://ftp.eu.net"/>
      </rdf:Alt>
    </s:DistributionSite>
  </rdf:Description>
</rdf:RDF>
```

As you can see the document describes 3 alternate sites that can be used to obtain the distribution of a package.

This is the output of the rdfdump.php script using our parser:

```
ordinal(0) triple(anonymous("http://luigi.melpomenia.com.ar/example5.rdf#genid1"), "http://www.w3.org/1999/02/22-rdf-
syntax-ns#type", "http://www.w3.org/1999/02/22-rdf-syntax-ns#Alt")
ordinal(0) triple("http://x.org/packages/X11", "http://description.org/schema/DistributionSite",
"http://luigi.melpomenia.com.ar/example5.rdf#genid1")
ordinal(1) triple(anonymous("http://luigi.melpomenia.com.ar/example5.rdf#genid1"), "http://www.w3.org/1999/02/22-rdf-
syntax-ns#_1", "ftp://ftp.x.org")
ordinal(2) triple(anonymous("http://luigi.melpomenia.com.ar/example5.rdf#genid1"), "http://www.w3.org/1999/02/22-rdf-
syntax-ns#_2", "ftp://ftp.cs.purdue.edu")
ordinal(3) triple(anonymous("http://luigi.melpomenia.com.ar/example5.rdf#genid1"), "http://www.w3.org/1999/02/22-rdf-
syntax-ns#_3", "ftp://ftp.eu.net")
Total statements: 5
```

Statements about members of a container

When we define a collection of type "Bag", "Seq" or "Alt" we create a new resource, if we use that resource in an RDf "Description" element we'll be describing properties that affect the whole collection. Some times we want to say something about each member of the collection instead of referring to the collection.

For example:

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:s="http://description.org/schema/">
<rdf:Bag ID="pages">
  <rdf:li resource="http://foo.org/foo.html" />
  <rdf:li resource="http://bar.org/bar.html" />
</rdf:Bag>

<rdf:Description about="#pages">
  <s:Creator>Ora Lassila</s:Creator>
</rdf:Description>
</rdf:RDF>
```

Says that "Ora Lassila" is the creator of the Bag "pages" but nothing is said about the creator of each page in the collection.

Just in case you want to check the document is example6.rdf and the output of the rdfdump program is:

```
ordinal(0) triple("http://luigi.melpomenia.com.ar/example6.rdf#pages", "http://www.w3.org/1999/02/22-rdf-syntax-
ns#type", "http://www.w3.org/1999/02/22-rdf-syntax-ns#Bag")
ordinal(1) triple("http://luigi.melpomenia.com.ar/example6.rdf#pages", "http://www.w3.org/1999/02/22-rdf-syntax-ns#_1",
"http://foo.org/foo.html")
ordinal(2) triple("http://luigi.melpomenia.com.ar/example6.rdf#pages", "http://www.w3.org/1999/02/22-rdf-syntax-ns#_2",
"http://bar.org/bar.html")
ordinal(0) triple("http://luigi.melpomenia.com.ar/example6.rdf#pages", "http://description.org/schema/Creator", literal("Ora
Lassila"))
Total statements: 4
```

RDF defines the "aboutEach" attribute to say something about each member of a collection.

Example:

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:s="http://description.org/schema/">
<rdf:Bag ID="pages">
  <rdf:li resource="http://foo.org/foo.html" />
  <rdf:li resource="http://bar.org/bar.html" />
</rdf:Bag>

<rdf:Description aboutEach="#pages">
  <s:Creator>Ora Lassila</s:Creator>
</rdf:Description>
</rdf:RDF>
```

This is example7.rdf now the parser produces the following:

```
ordinal(0) triple("http://luigi.melpomenia.com.ar/example7.rdf#pages", "http://www.w3.org/1999/02/22-rdf-syntax-
ns#type", "http://www.w3.org/1999/02/22-rdf-syntax-ns#Bag")
ordinal(1) triple("http://luigi.melpomenia.com.ar/example7.rdf#pages", "http://www.w3.org/1999/02/22-rdf-syntax-ns#_1",
"http://foo.org/foo.html")
ordinal(2) triple("http://luigi.melpomenia.com.ar/example7.rdf#pages", "http://www.w3.org/1999/02/22-rdf-syntax-ns#_2",
"http://bar.org/bar.html")
ordinal(0) triple(distributed("#pages"), "http://description.org/schema/Creator", literal("Ora Lassila"))
Total statements: 4
```

Note that the subject_type of the fourth statement is RDF_SUBJECT_TYPE_DISTRIBUTED indicating that the subject is each memeber of the resource referred by the "#pages" ID, you can the process what that means in your application.

Containers defined by an URI pattern:

There's another distributive attribute defined by RDF, let's take a look at the following RDF document:

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:s="http://description.org/schema/">
<rdf:Description about="http://foo.org/doc/page1">
  <s:Copyright>1998, The Foo Organization</s:Copyright>
</rdf:Description>
<rdf:Description about="http://foo.org/doc/page2">
  <s:Copyright>1998, The Foo Organization</s:Copyright>
</rdf:Description>
</rdf:RDF>
```

We know that the document describes the "copyright" property of two resources: http://foo.org/doc/page2 and http://foo.org/doc/page1, we may want for example to define a property for all the resources under http://foo.org/doc/ and we may find that writing a lot of statements is hard. That's why RDF defines the "aboutEachPrefix" attribute that let's you define properties for all resources descendant of a URI pattern. For example:

```
<rdf:Description aboutEachPrefix="http://foo.org/doc">
  <s:Copyright>1998, The Foo Organization</s:Copyright>
</rdf:Description>
```

This defines that all resources descendant of http://foo.org/doc will have a Copyright property with value "1998, The Foo Organization".

An example is example8.rdf:

```
<?xml version="1.0"?>
<rdf:RDF
  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:s="http://description.org/schema/">
<rdf:Description aboutEachPrefix="http://foo.org/doc">
  <s:Copyright>1998, The Foo Organization</s:Copyright>
</rdf:Description>

</rdf:RDF>
```

And the parser produces:

```
ordinal(0) triple(prefix("http://foo.org/doc"), "http://description.org/schema/Copyright", literal("1998, The Foo
Organization"))
Total statements: 1
```

Note that the subject_type of the statement is RDF_SUBJECT_TYPE_PREFIX, thus indicating that this property should be applied to all the resources under the presented prefix.

Statements about statements.

When we say something about a resource we use a statement, but we may want to say something about a statement, for example:

> *Ralph Swick says that Ora Lassila is the creator of the resource http://www.w3.org/Home/Lassila.*

In order to express this in RDF we have to model the statement as a resource, we can do it using what is called a "reified statement"

```
  <rdf:Description>
    <rdf:subject resource="http://www.w3.org/Home/Lassila" />
    <rdf:predicate resource="http://description.org/schema/Creator" />
    <rdf:object>Ora Lassila</rdf:object>
    <rdf:type resource="http://www.w3.org/1999/02/22-rdf-syntax-
ns#Statement" />
    <a:attributedTo>Ralph Swick</a:attributedTo>
  </rdf:Description>
```

This is a "reified" statement modeled as a resource note that we define the subject, prediate and object of the statement, then we define the "type"as a resource. and we add a property indicating that the statement was attributed to Ralph Swick.

This is the result of the parser:

```
ordinal() triple(anonymous("http://luigi.melpomenia.com.ar/example9.rdf#genid1"), "http://www.w3.org/1999/02/22-rdf-
syntax-ns#subject", "http://www.w3.org/Home/Lassila")
ordinal() triple(anonymous("http://luigi.melpomenia.com.ar/example9.rdf#genid1"), "http://www.w3.org/1999/02/22-rdf-
syntax-ns#predicate", "http://description.org/schema/Creator")
ordinal() triple(anonymous("http://luigi.melpomenia.com.ar/example9.rdf#genid1"), "http://www.w3.org/1999/02/22-rdf-
syntax-ns#object", literal("Ora Lassila"))
ordinal() triple(anonymous("http://luigi.melpomenia.com.ar/example9.rdf#genid1"), "http://www.w3.org/1999/02/22-rdf-
syntax-ns#type", "http://www.w3.org/1999/02/22-rdf-syntax-ns#Statement")
ordinal(0) triple(anonymous("http://luigi.melpomenia.com.ar/example9.rdf#genid1"), "a:attributedTo", literal("Ralph
Swick"))
Total statements: 5
```

Again there's an anonymous resource with properties "subject", "predicate","object" and "type" as well as a statement about that resource. When you see a statement where the " http://www.w3.org/1999/02/22-rdf-syntax-ns#type " property is " http://www.w3.org/1999/02/22-rdf-syntax-ns#type" you should know that the resource is in fact, a statement and process it accordingly.

More handlers

The rdf parser class allows other handlers to be set besides the statement_handler.

The Warning Handler

Use `$rdf->rdf_set_warning_handler($handler_name);` to set it.
The handler receives just one parameter: $message ej:

```
$rdf->rdf_set_warning_handler("my_warning_handler");
function my_warning_handler($message) {
  ...
}
```

This handler will be called whenever the parser finds an error in an RDF document, teh document may be well formed but have some form of invalid RDF syntax that the parser reports calling this handler. The message describes the error detected by the parser.

The element and character_data handlers.

RDF can be "embedded" inside other XML vocabularies, in such a case you may want to parse not only RDF elements but the whole XML document. The rdf parser class let's you define handlers for non-RDF elements as well as non-RDF data inside elements:

```
$rdf->set_element_handler($start_handler_name,$end_handler_name);
```

The start handler name receives &$user_data, $name and $attributes. $user_data is the user_data if set, name is the name of the non-rdf element and attributes is an asociative array with the XML attributes.

The end handler receives &$user_data and $name, being $name the name of the ending non-RDF element.

Finally the character_data_handler is set using:

```
$rdf->rdf_set_character_data_handler($handler_name);
```

And the handler receives &$user_data and $data being $data the characters found inside a non-RDF element.

These handlers are almost identical to the handlers that the XML parser of PHP uses for generic XML files.

Summary

We've seen a broad overview of the XML RDF syntax and how to process RDF documents using the Rdf_parse class, a port of the repat parser in PHP. As we've seen the class can parse RDF documents following all the syntactic variations that RDF allows producing events that can be processed in your application. This class allows you to independize your application from the different RDF syntaxes that can be used to convey RDF information, using this class you can write all sort of applications using RDF using PHP.
Since the class is based on repat the code is compact and the class is fast and quite solid.

Resources

- The RDF home page at the W3C : http://www.w3.org/RDF/
- The XML syntax for RDF specification: http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/
- The "C" version of Repat by Jason diammond: http://injektilo.org/rdf/repat.html
- The "PHP" version of repat: class_rdf_parse.php: http://phpxmlclasses.sourceforge.net/