APPENDIX TO
Semantic Web
Tutorial Using N3

Tim Berners-Lee
Dan Connolly
Sandro Hawke

For Presentaton
May 20, 2003

http://www.w3.org/2000/10/swap/doc/

Table of Contents

1 Comparing Formats

There are a many languages in use today for exchanging RDF-structured information. Here's a sample rendered in many different ways. Converters exist between some of these formats, but not others (yet).

1.1 English (Very Informal)

There is person, Pat, known as "Pat Smith" and "Patrick Smith". Pat has a pet dog named "Rover".

1.2 English Hypertext (Informal)

Here the ambiguity of terms is addressed by making the words be hypertext links. The links may or may not work, depending on the servers involved.

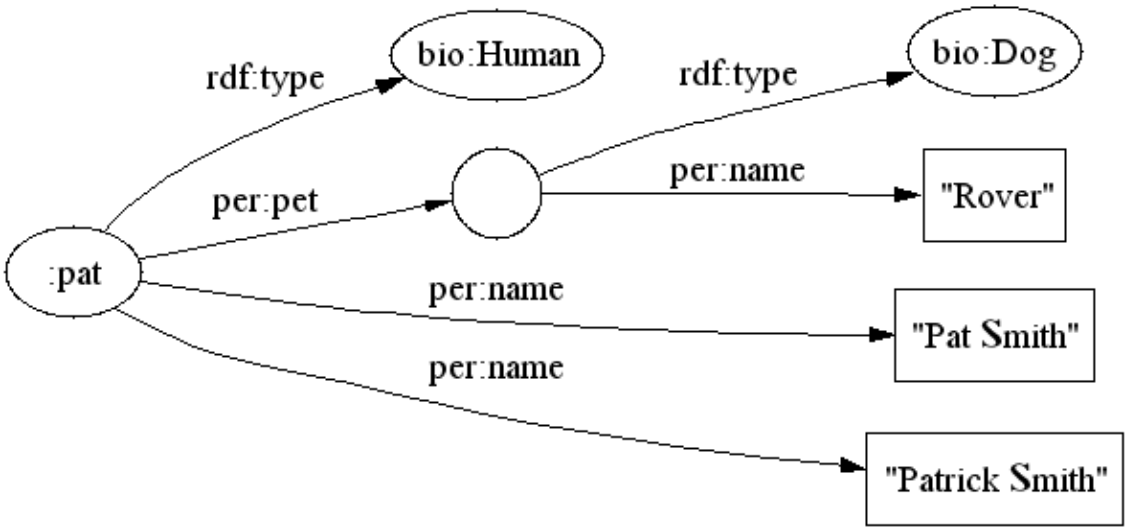
Pat (http://www.w3.org/2000/10/swap/test/demo1/about-pat#pat) is a
(http://www.w3.org/1999/02/22-rdf-syntax-ns#type) human
(http://www.w3.org/2000/10/swap/test/demo1/biology#Human) with the names
(http://www.w3.org/2000/10/swap/test/demo1/friends-vocab#name) "Pat Smith" and "Patrick Smith". Pat
(http://www.w3.org/2000/10/swap/test/demo1/about-pat#pat) has a pet
(http://www.w3.org/2000/10/swap/test/demo1/friends-vocab#pet), a
(http://www.w3.org/1999/02/22-rdf-syntax-ns#type) dog
(http://www.w3.org/2000/10/swap/test/demo1/biology#Dog), with the name
(http://www.w3.org/2000/10/swap/test/demo1/friends-vocab#name) "Rover".

1.3 N3

```
@prefix : <http://www.w3.org/2000/10/swap/test/demo1/about-pat#> .
@prefix bio: <http://www.w3.org/2000/10/swap/test/demo1/biology#> .
@prefix per: <http://www.w3.org/2000/10/swap/test/demo1/friends-vocab#> .

:pat     a bio:Human;
     per:name "Pat Smith",
              "Patrick Smith";
     per:pet  [
         a bio:Dog;
         per:name "Rover" ] .
```

1.4 Directed Labeled Graph



1.5 RDF/XML

```
<rdf:RDF xmlns="http://www.w3.org/2000/10/swap/test/demo1/about-pat#"
    xmlns:bio="http://www.w3.org/2000/10/swap/test/demo1/biology#"
    xmlns:per="http://www.w3.org/2000/10/swap/test/demo1/friends-vocab#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">

    <bio:Human rdf:about="#pat">
        <per:name>Pat Smith</per:name>
        <per:name>Patrick Smith</per:name>
        <per:pet>
            <bio:Dog>
                <per:name>Rover</per:name>
            </bio:Dog>
        </per:pet>
    </bio:Human>
</rdf:RDF>
```

1.6 N-Triples

With @prefix:

```
@prefix : <http://www.w3.org/2000/10/swap/test/demo1/about-pat#> .
@prefix bio: <http://www.w3.org/2000/10/swap/test/demo1/biology#> .
@prefix per: <http://www.w3.org/2000/10/swap/test/demo1/friends-vocab#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.

:pat rdf:type bio:Human.
:pat per:name "Pat Smith".
:pat per:name "Patrick Smith".
:pat per:pat _:genid1.
_:genid1 rdf:type bio:Dog.
_:genid1 per:name "Rover".
```

In standard form:

```
<http://www.w3.org/2000/10/swap/test/demo1/about-pat#pat> <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.w3.org/2000/10/swap/test/demo1/biology#Human> .
<http://www.w3.org/2000/10/swap/test/demo1/about-pat#pat> <http://www.w3.org/2000/10/swap/test/demo1/friends-vocab#name> "Pat Smith" .
<http://www.w3.org/2000/10/swap/test/demo1/about-pat#pat> <http://www.w3.org/2000/10/swap/test/demo1/friends-vocab#name> "Patrick Smith" .
<http://www.w3.org/2000/10/swap/test/demo1/about-pat#pat> <http://www.w3.org/2000/10/swap/test/demo1/friends-vocab#pet> _:genid1 .
_:genid1 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.w3.org/2000/10/swap/test/demo1/biology#Dog> .
_:genid1 <http://www.w3.org/2000/10/swap/test/demo1/friends-vocab#name> "Rover" .
```

1.7 Prolog

Without namespaces, this is very terse:

```
human(pat).
dog(rover).            % we have to assign a name
name(pat, "Pat Smith").
name(pat, "Patrick Smith").
name(rover, "Rover").
pet(pat, rover).
```

One approach to namespaces:

```
ns(ns1_, "http://www.w3.org/2000/10/swap/test/demo1/about-pat").
ns(bio_, "http://www.w3.org/2000/10/swap/test/demo1/biology#").
ns(per_, "http://www.w3.org/2000/10/swap/test/demo1/friends-vocab#").
bio_Human(ns1_pat).
bio_Dog(rover).      # unprefix could be be NodeIDs...
per_name(ns1_pat, "Pat Smith").
per_name(ns1_pat, "Patrick Smith").
per_name(rover, "Rover").
per_pet(ns1_pat, rover).
```

1.8 SQL

1.8.1 Table Relating URIs to Internal IDs

URIs are big, variable-length keys; it’s much more efficient to translate them into an internal id. If the "uri" is NULL, this resource is anonymous (a bNode, like a NodeID in RDF/XML).

```
CREATE TABLE uri (
  id INT AUTO_INCREMENT PRIMARY KEY,    # PRIMARY = UNIQUE and NOT NULL
  uri BLOB,   # BLOB is also called LONGVARBINARY
  UNIQUE KEY uri (uri(64))  # length is just a tuning knob
);
INSERT INTO uri (uri) VALUES (’http://www.w3.org/1999/02/22-rdf-syntax-ns#type’);
INSERT INTO uri (uri) VALUES (’http://www.w3.org/2000/10/swap/test/demo1/biology#Human’);
INSERT INTO uri (uri) VALUES (’http://www.w3.org/2000/10/swap/test/demo1/biology#Dog’);
INSERT INTO uri (uri) VALUES (’http://www.w3.org/2000/10/swap/test/demo1/friends-vocab#name’);
INSERT INTO uri (uri) VALUES (’http://www.w3.org/2000/10/swap/test/demo1/friends-vocab#pet’);
INSERT INTO uri (uri) VALUES (’http://www.w3.org/2000/10/swap/test/demo1/about-pat#pat’);
INSERT INTO uri (uri) VALUES (NULL);   # this is rover, who has no URI

mysql> select * from uri;
+----+-------------------------------------------------------------+
| id | uri                                                         |
+----+-------------------------------------------------------------+
|  1 | http://www.w3.org/1999/02/22-rdf-syntax-ns#type             |
|  2 | http://www.w3.org/2000/10/swap/test/demo1/biology#Human     |
|  3 | http://www.w3.org/2000/10/swap/test/demo1/biology#Dog       |
|  4 | http://www.w3.org/2000/10/swap/test/demo1/friends-vocab#name |
|  5 | http://www.w3.org/2000/10/swap/test/demo1/friends-vocab#pet  |
|  6 | http://www.w3.org/2000/10/swap/test/demo1/about-pat#pat     |
|  7 | NULL                                                        |
+----+-------------------------------------------------------------+
```

1.8.2 Option 1: One Table, One Column Per Predicate

This is a simple, intuitive approach, but...

- You need to add a new column with each new predicate you use
- Predicates must be known to be individual-valued or data-valued; and if data-valued, then what type?
- You cannot have multivalued (cardinality > 1) properties

```
CREATE TABLE resource (
  id INT PRIMARY KEY,
  type INT,           # http://www.w3.org/1999/02/22-rdf-syntax-ns#type
  name varchar(255),  # http://www.w3.org/2000/10/swap/test/demo1/friends-vocab#name
  pet INT             # http://www.w3.org/2000/10/swap/test/demo1/friends-vocab#pet
);
INSERT INTO resource (id, type, name, pet) VALUES (6, 2, 'Pat Smith', 7);
INSERT INTO resource (id, type, name) VALUES (7, 3, 'Rover');

mysql> select * from resource;
+----+------+-----------+------+
| id | type | name      | pet  |
+----+------+-----------+------+
|  6 |    2 | Pat Smith |    7 |
|  7 |    3 | Rover     | NULL |
+----+------+-----------+------+
```

1.8.3 Option 2: One Table Per Class, One Column Per Predicate

Here we avoid having such wide tables, and our modeling better matches the normal database modeling. On the other hand, we have some conceptual redundancy between human.name and dog.name, because there is no support for inheritance.

```
CREATE TABLE human (  # http://www.w3.org/2000/10/swap/test/demo1/biology#Human
  id INT PRIMARY KEY,
  name varchar(255),  # http://www.w3.org/2000/10/swap/test/demo1/friends-vocab#name
  pet INT             # http://www.w3.org/2000/10/swap/test/demo1/friends-vocab#pet
);
CREATE TABLE dog (  # http://www.w3.org/2000/10/swap/test/demo1/biology#Dog
  id INT PRIMARY KEY,
  name varchar(255)   # http://www.w3.org/2000/10/swap/test/demo1/friends-vocab#name
);
INSERT INTO human VALUES (6, 'Pat Smith', 7);
INSERT INTO dog   VALUES (7, 'Rover');

mysql> select * from human, dog where human.pet=dog.id;
+----+-----------+------+----+-------+
| id | name      | pet  | id | name  |
+----+-----------+------+----+-------+
|  6 | Pat Smith |    7 |  7 | Rover |
+----+-----------+------+----+-------+
```

1.8.4 Option 3: One Table Per Predicate

Here we can support duplicate values.

```
CREATE TABLE name (  # http://www.w3.org/2000/10/swap/test/demo1/friends-vocab#name
  subject INT NOT NULL,
  object varchar(255),
  INDEX(subject),
  UNIQUE INDEX(subject, object)
);
CREATE TABLE pet (   # http://www.w3.org/2000/10/swap/test/demo1/friends-vocab#pet
  subject INT NOT NULL,
  object INT,
  INDEX(subject),
```

```
  UNIQUE INDEX(subject, object)
);
CREATE TABLE type (   # http://www.w3.org/1999/02/22-rdf-syntax-ns#type
  subject INT NOT NULL,
  object INT,
  INDEX(subject),
  UNIQUE INDEX(subject, object)
);
INSERT INTO name VALUES (6, 'Pat Smith');
INSERT INTO name VALUES (6, 'Patrick Smith');
INSERT INTO name VALUES (7, 'Rover');
INSERT INTO pet VALUES (6, 7);
INSERT INTO type VALUES (6, 2);
INSERT INTO type VALUES (7, 3);

mysql> select * from name;
+---------+---------------+
| subject | object        |
+---------+---------------+
|       6 | Pat Smith     |
|       6 | Patrick Smith |
|       7 | Rover         |
+---------+---------------+

mysql> select * from pet;
+---------+--------+
| subject | object |
+---------+--------+
|       6 |      7 |
+---------+--------+

mysql> select * from type;
+---------+--------+
| subject | object |
+---------+--------+
|       6 |      2 |
|       7 |      3 |
+---------+--------+
```

1.8.5 Option 4: One Table of Triples, One Table Of Resources

If we redo our URIs table as a "resources" table, with literals as well as URIs, we have some more options.

```
CREATE TABLE resources (
  id INT AUTO_INCREMENT PRIMARY KEY,    # PRIMARY = UNIQUE and NOT NULL
  # either provide a uri
  uri BLOB,
  # or a literal_value, which might have a datatype and language
  literal_value BLOB,
  datatype INT,
  language VARCHAR(5),
  UNIQUE KEY (uri(64))  # length is just a tuning knob
);
INSERT INTO resources (uri) VALUES ('http://www.w3.org/1999/02/22-rdf-syntax-ns#type');
INSERT INTO resources (uri) VALUES ('http://www.w3.org/2000/10/swap/test/demo1/biology#Human');
INSERT INTO resources (uri) VALUES ('http://www.w3.org/2000/10/swap/test/demo1/biology#Dog');
INSERT INTO resources (uri) VALUES ('http://www.w3.org/2000/10/swap/test/demo1/friends-vocab#name');
INSERT INTO resources (uri) VALUES ('http://www.w3.org/2000/10/swap/test/demo1/friends-vocab#pet');
INSERT INTO resources (uri) VALUES ('http://www.w3.org/2000/10/swap/test/demo1/about-pat#pat');
INSERT INTO resources (uri) VALUES (NULL);   # this is rover, who has no URI
INSERT INTO resources (literal_value) VALUES ('Pat Smith');
INSERT INTO resources (literal_value) VALUES ('Patrick Smith');
INSERT INTO resources (literal_value) VALUES ('Rover');

mysql> select * from resources;
+----+-------------------------------------------------------------+--------------+----------+----------+
```

```
| id | uri                                                                | literal_value | datatype | language |
+----+--------------------------------------------------------------------+---------------+----------+----------+
|  1 | http://www.w3.org/1999/02/22-rdf-syntax-ns#type                    | NULL          |     NULL | NULL     |
|  2 | http://www.w3.org/2000/10/swap/test/demo1/biology#Human            | NULL          |     NULL | NULL     |
|  3 | http://www.w3.org/2000/10/swap/test/demo1/biology#Dog              | NULL          |     NULL | NULL     |
|  4 | http://www.w3.org/2000/10/swap/test/demo1/friends-vocab#name       | NULL          |     NULL | NULL     |
|  5 | http://www.w3.org/2000/10/swap/test/demo1/friends-vocab#pet        | NULL          |     NULL | NULL     |
|  6 | http://www.w3.org/2000/10/swap/test/demo1/about-pat#pat            | NULL          |     NULL | NULL     |
|  7 | NULL                                                               | NULL          |     NULL | NULL     |
|  8 | NULL                                                               | Pat Smith     |     NULL | NULL     |
|  9 | NULL                                                               | Patrick Smith |     NULL | NULL     |
| 10 | NULL                                                               | Rover         |     NULL | NULL     |
+----+--------------------------------------------------------------------+---------------+----------+----------+
```

Then we can have a simple table of triples:

```
CREATE TABLE triples (
  subject INT NOT NULL,
  predicate INT NOT NULL,
  object INT NOT NULL,
  UNIQUE INDEX(subject, predicate, object),
  INDEX(predicate, object),
  INDEX(object, predicate)
);
INSERT INTO triples VALUES (6, 4, 8);
INSERT INTO triples VALUES (6, 4, 9);
INSERT INTO triples VALUES (7, 4, 10);
INSERT INTO triples VALUES (6, 1, 2);
INSERT INTO triples VALUES (7, 1, 3);
INSERT INTO triples VALUES (6, 5, 7);

mysql> select * from triples;
+---------+-----------+--------+
| subject | predicate | object |
+---------+-----------+--------+
|       6 |         1 |      2 |
|       6 |         4 |      8 |
|       6 |         4 |      9 |
|       6 |         5 |      7 |
|       7 |         1 |      3 |
|       7 |         4 |     10 |
+---------+-----------+--------+

mysql> select s.id, s.uri, p.uri as "predicate",
       o.id, o.uri, o.literal_value as "lit"
       from triples, resources as s, resources as p, resources as o
       where s.id=triples.subject AND
             p.id=triples.predicate AND
             o.id=triples.object;
+----+-----------------------------------------------------+-----------------------------------------------------+----+-----------------------------------------------------+--------------+
| id | uri                                                 | predicate                                           | id | uri                                                 | lit          |
+----+-----------------------------------------------------+-----------------------------------------------------+----+-----------------------------------------------------+--------------+
|  6 | http://www.w3.org/2000/10/swap/test/demo1/about-pat#pat | http://www.w3.org/1999/02/22-rdf-syntax-ns#type             |  2 | http://www.w3.org/2000/10/swap/test/demo1/biology#Human | NULL         |
|  6 | http://www.w3.org/2000/10/swap/test/demo1/about-pat#pat | http://www.w3.org/2000/10/swap/test/demo1/friends-vocab#name |  8 | NULL                                                | Pat Smith    |
|  6 | http://www.w3.org/2000/10/swap/test/demo1/about-pat#pat | http://www.w3.org/2000/10/swap/test/demo1/friends-vocab#name |  9 | NULL                                                | Patrick Smith |
|  6 | http://www.w3.org/2000/10/swap/test/demo1/about-pat#pat | http://www.w3.org/2000/10/swap/test/demo1/friends-vocab#pet  |  7 | NULL                                                | NULL         |
|  7 | NULL                                                | http://www.w3.org/1999/02/22-rdf-syntax-ns#type             |  3 | http://www.w3.org/2000/10/swap/test/demo1/biology#Dog   | NULL         |
|  7 | NULL                                                | http://www.w3.org/2000/10/swap/test/demo1/friends-vocab#name | 10 | NULL                                                | Rover        |
+----+-----------------------------------------------------+-----------------------------------------------------+----+-----------------------------------------------------+--------------+
```

1.9 XML (but not RDF/XML)

The **striped** or **alternating-normal form** approach uses a markup language designed for the application domain. Conversion to and from triples requires specialized software

```
<Human>
  <uri>http://www.w3.org/2000/10/swap/test/demo1/about-pat#pat</uri>
  <name>Pat Smith</name>
  <pet>
    <Dog>
     <name>Rover</name>
    </Dog>
  </pet>
</Human>
```

An alternative is to use **XML Triples**. This does not involve domain-specific markup. Several candidate DTDs/Schemes have been proposed; this is just one strawman. For some applications, this kind of syntax may be easier than RDF/XML or a striped syntax.

```
<!DOCTYPE Graph [
 <!ENTITY rdf    "http://www.w3.org/1999/02/22-rdf-syntax-ns#">
 <!ENTITY bio    "http://www.w3.org/2000/10/swap/test/demo1/biology#">
 <!ENTITY ns1    "http://www.w3.org/2000/10/swap/test/demo1/about-pat#">
 <!ENTITY per    "http://www.w3.org/2000/10/swap/test/demo1/friends-vocab#">
]>
<Graph>
  <Triple>
    <subject><uri>&ns1;pat</uri></subject>
    <predicate><uri>&rdf;type</uri></predicate>
```

```
    <object><uri>&bio;Human</uri></object>
  </Triple>
  <Triple>
    <subject><uri>&ns1;pat</uri></subject>
    <predicate><uri>&per;name</uri></predicate>
    <object><literal>Pat Smith</literal></object>
  </Triple>
  <Triple>
    <subject><uri>&ns1;pat</uri></subject>
    <predicate><uri>&per;pet</uri></predicate>
    <object><nodeID>genid1</nodeID></object>
  </Triple>
  <Triple>
    <subject><nodeID>genid1</nodeID></subject>
    <predicate><uri>&rdf;type</uri></predicate>
    <object><uri>&bio;Dog</uri></object>
  </Triple>
  <Triple>
    <subject><nodeID>genid1</nodeID></subject>
    <predicate><uri>&per;name</uri></predicate>
    <object><literal>Rover</literal></object>
  </Triple>
</Graph>
```

1.10 Javascript and Python

The RDF model maps fairly well to several common programming constructs, especially those found in interpreted object-oriented languages, like Javascript and Python. Still, this mapping is not perfect.

This simple approach ignores URI and cardinality issues; it only allows pat to have one name and one pet:

```
pat = Human()
rover = Dog()
pat.name = "Pat Smith"
rover.name = "Rover"
pat.pet = rover
```

We can begin to consider cardinality:

```
pat = Human()
rover = Dog()
pat.name.append("Pat Smith")
rover.name.append("Rover")
pat.pet.append(rover)
```

But to be more complete, we need something like this which loses the simplicity of the built-in model:

```
pat = Resource()
rover = Resource()
pat.addProperty(  ns.rdf.type, ns.bio.Human)
rover.addProperty(ns.rdf.type, ns.bio.Dog)
pat.addProperty(  ns.per.name, "Pat Smith")
rover.addProperty(ns.per.name, "Rover")
pat.addProperty(  ns.perpet, rover)
```

1.11 But remember...

Beware of thinking of RDF as a format for serailizing objects. The semantic web is different - it is weblike.

- Any document can (potentially) say anything about anything. There is no set of "slots" or "attributes" for a class. The properties defined in a schema are not the only properties which one can use to describe something which is in that class.

  An object can be in many classes. When you create a semantic web document about something, others can deduce more things about it, in vocabularies you have never heard of.

  Entity-Relationship and UML diagrams are useful for describing RDF -- so long as you remember the above.

One of the interesting challenges is how to blend languges such as n3 seamlessly into object-oriented procedural languages.

2 Installing Cwm

2.1 Prerequisites

- **Python (http://www.python.org/)**. Currently (2003/2) cwm needs Python2.2 or later. Python comes with many systems. If you don't have it, on debian type apt-get install python. Mac OSX comes with it after 10.2. On earlier, get fink then type `fink` (http://fink.sourceforge.net/) `install python`. On cygwin (http://www.cygwin.com/), select python in the setup.exe installer..
- If you are reading things in RDF/XML format, you will need the **pyxml** python extension. Under debian, type `apt-get install pyxml.` Otherwise, untar it and in the right directory type python setup.py install. Under cygwin, select it in the the cygwin setup.exe.
- If you are using the public key cryptography builtins, you will need the **amkCrypto** python extension. Hopefully the necessary functionality will be in pycrypto soon (next release later than 2002/05). One version had a print statement in number.py#inverse --the print statement needs to be removed. Edit directory names appropriately in setup.py then run `python setup.py install`. Necessary changes for OSX:

You can do all this under MSWindows but the easiest thing is to install cygwin (a unix-like environment for Windows) and run everything in there.

2.2 Download

The python source files are available as a compressed tar file

- Download cwm.tgz now

Get that, unwrap it in some suitable directory.

Set up an alias (.bat file, etc) to make the cwm command. In bash this is:

```
cwm="python /whereverYouPutThem/cwm.py"
```

You should be all set.

2.3 Source

This swap code (http://dev.w3.org/cvsweb/2000/10/swap/) is not guaranteed but is open source and available if you want to play.

Using CVS
: from the public w3c CVS repository (http://dev.w3.org/cvsweb/). Check out the whole tree to develop. This includes the test data - if you don't need that, delete the test subdirectory. Make a fresh directory where you want to put stuff from dev.w3.org.

```
$ cvs -d:pserver:anonymous@dev.w3.org:/sources/public login
password? anonymous
$ cvs -d:pserver:anonymous@dev.w3.org:/sources/public get 2000/10/swap
```

From the web
: Get the files one by one. cwm.py is the main application file. You can browse the source files on the web, but this is **not** a practical way to install the system.

3 Cwm Command Line arguments

You can get a list of these by typing cwm --help

```
Command line RDF/N3 tool

 <command> <options> <inputURIs>

--pipe        Don't store, just pipe out *

--rdf         Input & Output ** in RDF M&S 1.0 insead of n3 from now on
--n3          Input & Output in N3 from now on
--rdf=flags   Input & Output ** in RDF and set given RDF flags
--n3=flags    Input & Output in N3 and set N3 flags
--ntriples    Input & Output in NTriples (equiv --n3=spartan -bySubject -quiet)
--language=x  Input & Output in "x" (rdf, n3, etc)  --rdf same as: --language=rdf
--languageOptions=y     --n3=sp same as:  --language=n3 --languageOptions=sp
--ugly        Store input and regurgitate *
--bySubject   Store input and regurgitate in subject order *
--no          No output *
              (default is to store and pretty print with anonymous nodes) *
--strings     Dump :s to stdout ordered by :k whereever { :k log:outputString :s }

--apply=foo   Read rules from foo, apply to store, adding conclusions to store
--filter=foo  Read rules from foo, apply to store, REPLACING store with conclusions
--rules       Apply rules in store to store, adding conclusions to store
--think       as -rules but continue until no more rule matches (or forever!)
--engine=otter use otter (in your $PATH) instead of llyn for linking, etc
--why         Replace the store with an explanation of its contents
--mode=flags  Set modus operandi for inference (see below)
--flatten     turn formulas into triples using LX vocabulary
--unflatten   turn described-as-true LX sentences into formulas
--think=foo   as -apply=foo but continue until no more rule matches (or forever!)
--purge       Remove from store any triple involving anything in class log:Chaff
--purge-rules Remove from store any triple involving log:implies
--crypto      Enable processing of crypto builtin functions. Requires python crypto.
--help        print this message
--revision    print CVS revision numbers of major modules
--chatty=50   Verbose output of questionable use, range 0-99
--with        Pass any further arguments to the N3 store as os:argv values


            * mutually exclusive
            ** doesn't work for complex cases :-/
Examples:
  cwm --rdf foo.rdf --n3 --pipe     Convert from rdf/xml to rdf/n3
  cwm foo.n3 bar.n3 --think         Combine data and find all deductions
  cwm foo.n3 --flat --n3=spart

See http://www.w3.org/2000/10/swap/doc/cwm  for more documentation.

Mode flags affect inference extedning to the web:
 e   Errors loading schemas of definitive documents are fatal
 m   Schemas and definitive documents laoded are merged into the meta knowledge
     (otherwise they are consulted independently)
 r   Needed to enable any remote stuff.
 s   Read the schema for any predicate in a query.

 u   Generate unique ids using a run-specific


Flags for N3 output are as follows:-

a   Anonymous nodes should be output using the _: convention (p flag or not).
d   Don't use default namespace (empty prefix)
```

```
i   Use identifiers from store - don't regen on output
l   List syntax suppression. Don't use (..)
n   No numeric syntax - use strings typed with ^^ syntax
p   Prefix suppression - don't use them, always URIs in <> instead of qnames.
q   Quiet - don't make comments about the environment in which processing was done.
r   Relative URI suppression. Always use absolute URIs.
s   Subject must be explicit for every statement. Don't use ";" shorthand.
t   "this" and "()" special syntax should be suppresed.


        Flags to control RDF/XML output (after --rdf=) areas follows:

        c  - Don't use elements as class names
        d  - Don't use default namespace



    Flags to control RDF/XML INPUT (after --rdf=) follow:
        S  - Strict spec. Unknown parse type treated as Literal instead of error.
        T  - take foreign XML as transparent and parse any RDF in it
             (default it is to ignore unless rdf:RDF at top level)
        L  - If non-rdf attributes have no namespace prefix, assume in local <#> namespace
        D  - Assume default namespace decalred as local document is assume xmlns=""
```

4 Comparing Rule-Based Systems

Cwm acts as a rules processor, using information written in N3 rules to guide it in manipulating the RDF/N3 information it has stored. While rules processors are not exactly commonplace, and understanding them is not manditory for the working programmer, they do have a long and solid history. Where does cwm fit into that history?

The field has sometimes been called Knowledge-Based Systems (http://wombat.doc.ic.ac.uk/foldoc/foldoc.cgi?knowledge-based+system) or Expert Systems (http://wombat.doc.ic.ac.uk/foldoc/foldoc.cgi?expert+system), but now people often just says something "uses rules."

4.1 Four Kinds of Rules

There seem to be four different ways people think about and want to use rules:

1. **Derivation or Deduction Rules.** These are cwm's normal rules. Each rule expresses the knowledge that if one set of statements happens to be true, then some other set of statements must also be true. In most cases this is the same as what is sometimes called a *logical implication*, a *material conditional*, or a *Horn clause*.
2. **Transformation Rules.** These are what cwm uses with --filter. Each rule relates truth in one knowledge base to truth in another. Transformation rules on n-tuples can be rewritten as derivation rules on (n+1)-tuples, where the additional element specifies the knowledge base.
3. **Integrity Constraints.** These are rules of the form "it must be true that ....". Cwm does not have these rules, but they can be emulated with derivation rules like "if it is not true that .... then we-have-an-error" and a check for "we-have-and-error".
4. **Reaction or Event-Condition-Action (ECA) Rules.** These involve a notion of action, not just inference, when a rule applies. Reaction rules may be emulated by wrapping a reaction rule system in a procedure which queries for actions to perform.

4.2 A Varierty of Engines

Along with the variety in types of rules, there is also a wide variety in types of rules engines and general logic processors or "reasoners".

4.2.1 Automated Theorem Provers

Automated reasoning using first-order became generally feasible in 1965 with Robinson's resolution and hyperresolution algorithms. Today a raft of automated theorem provers continue this tradition, but they see little use in general computing. The strength here is general expressive power: the machine does perform classical logic operations; the weakness is that such systems generally become too practical, real-world application problems.

A focal point for this research is Thousands of Problems for Theorem-Provers (TPTP) (http://www.tptp.org/), which includes links to provers and a conversion utility between logic languages. You can play a little with its web form (http://www.cs.jcu.edu.au/cgi-bin/tptp/tptp2X) (try problem ALG001-1).

In the RDF/Semantic Web community, people have used at least OTTER (http://www-unix.mcs.anl.gov/AR/otter/) and SNARK (http://belo.stanford.edu:8080/iwregistrar/Lookup?type=1&value=SNARK&mode=full).

4.2.2 Logic Programming

In 1970-1972, Prolog (http://wombat.doc.ic.ac.uk/foldoc/foldoc.cgi?Prolog) introduced Logic Programming (http://wombat.doc.ic.ac.uk/foldoc/foldoc.cgi?logic+programming), which took a restricted form of first-order logic (Horn clauses) and offered to prove things with them in a deterministic order, very much like running a

program. Prolog always chains backward from a query.

(1970-1975 also saw (http://www.byte.com/art/9509/sec7/art19.htm) the introduction of C, Pascal, Scheme, Smalltalk, and Microsoft BASIC.)

Tabled Prolog (as in XSB (http://xsb.sourceforge.net/)) allows rules to be written without worrying about looping, much like with cwm.

4.2.3 Production Systems

A different approach became popular with OPS5 (http://wombat.doc.ic.ac.uk/foldoc/foldoc.cgi?OPS5) (which is since continued into CLIPS (http://www.ghg.net/clips/CLIPS.html) and JESS (http://herzberg.ca.sandia.gov/jess/)), presenting rules as being "triggered" and causing actions.

Unlike Prolog, these are usually (but not always) chaining forward from the givens, like cwm.

4.2.4 Modern Reasoners

Modern reasoners and rule systems often use a complex hybrid of strategies, or are simply developed for a focussed application domain.

Some worth mentioning include:

1. Euler handles N3 and many of cwm’s operations, using a backward chaining approach with loop-avoidance techniques.
2. JTP (http://belo.stanford.edu:8080/iwregistrar/Lookup?type=1&value=JTP&mode=full) is used for web-reasoning in the KSL InferenceWeb project (http://www.ksl.stanford.edu/software/IW/)
3. RACER (http://www.fh-wedel.de/~mo/racer/) and FaCT (http://www.cs.man.ac.uk/~horrocks/FaCT/) are leading description-logic (ontology) reasoners
4. OpenCyc (http://www.opencyc.org/)