Towards an eIO based RDF Dump Pipeline, and Beyond?

An attempt to use the unified IO system

Alessandro Vullo

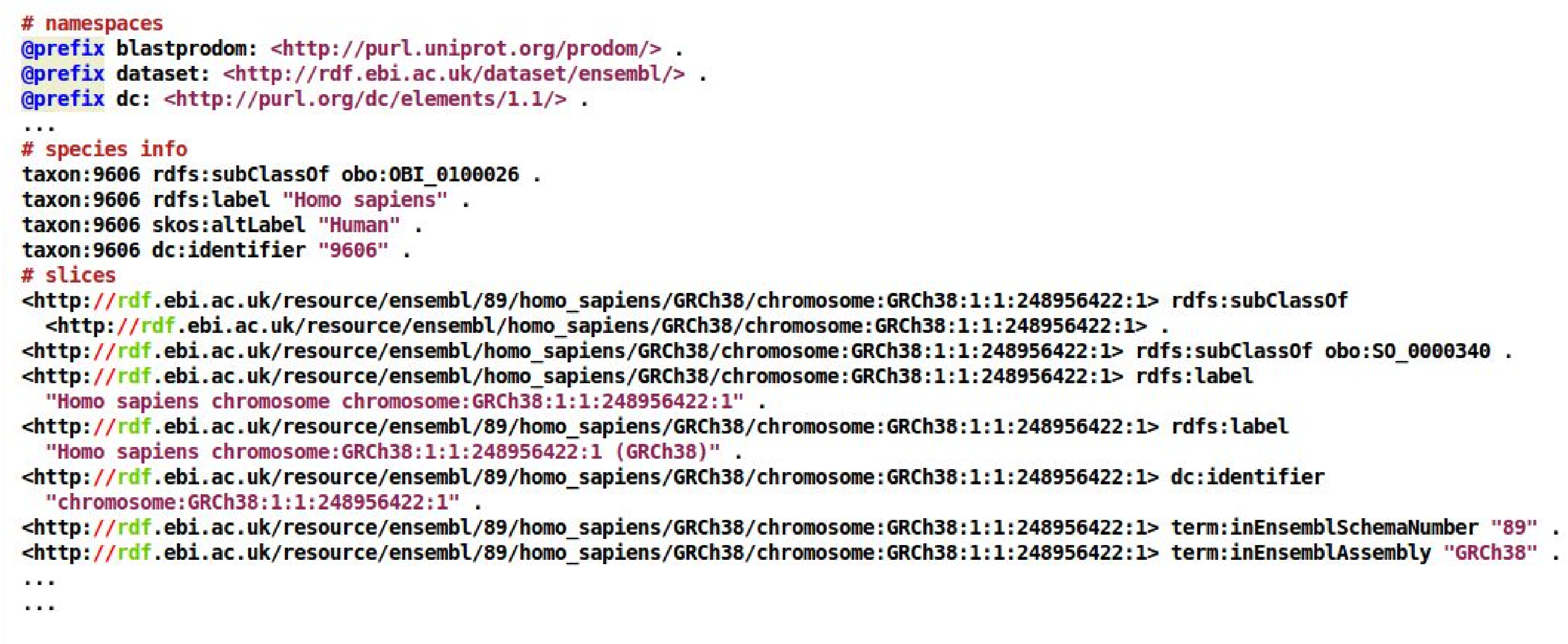
e! Core team meeting - June 2017

RDF

- Resource Description Framework
- Originally described as metadata data model
- General method for description/modeling of information implemented in web resources
- Basic idea:
  - make statements (triples) about (web) resources
  - ***subject-predicate-object***
- Uses a variety of syntax notations/data serialisation formats
  - e.g. RDF/XML, SPARQL, **Turtle**, N-Triples etc.

Current RDF Dump System

- eHive pipeline (production)
  - e! features and xrefs for each species in Turtle
- fetch-all-the-things-and-dump model
  - gene-transcript-exon-translation(-protein features)
  - Bio::EnsEMBL::Production::DBSQL::BulkFetcher
- Mongoose libs
  - Bio::EnsEMBL::RDF::EnsemblToTripleConverter
  - a couple of other ones



```
# namespaces
@prefix blastprodom: <http://purl.uniprot.org/prodom/> .
@prefix dataset: <http://rdf.ebi.ac.uk/dataset/ensembl/> .
@prefix dc: <http://purl.org/dc/elements/1.1/> .
...
# species info
taxon:9606 rdfs:subClassOf obo:OBI_0100026 .
taxon:9606 rdfs:label "Homo sapiens" .
taxon:9606 skos:altLabel "Human" .
taxon:9606 dc:identifier "9606" .
# slices
<http://rdf.ebi.ac.uk/resource/ensembl/89/homo_sapiens/GRCh38/chromosome:GRCh38:1:1:248956422:1> rdfs:subClassOf
  <http://rdf.ebi.ac.uk/resource/ensembl/homo_sapiens/GRCh38/chromosome:GRCh38:1:1:248956422:1> .
<http://rdf.ebi.ac.uk/resource/ensembl/homo_sapiens/GRCh38/chromosome:GRCh38:1:1:248956422:1> rdfs:subClassOf obo:SO_0000340 .
<http://rdf.ebi.ac.uk/resource/ensembl/homo_sapiens/GRCh38/chromosome:GRCh38:1:1:248956422:1> rdfs:label
  "Homo sapiens chromosome chromosome:GRCh38:1:1:248956422:1" .
<http://rdf.ebi.ac.uk/resource/ensembl/89/homo_sapiens/GRCh38/chromosome:GRCh38:1:1:248956422:1> rdfs:label
  "Homo sapiens chromosome:GRCh38:1:1:248956422:1 (GRCh38)" .
<http://rdf.ebi.ac.uk/resource/ensembl/89/homo_sapiens/GRCh38/chromosome:GRCh38:1:1:248956422:1> dc:identifier
  "chromosome:GRCh38:1:1:248956422:1" .
<http://rdf.ebi.ac.uk/resource/ensembl/89/homo_sapiens/GRCh38/chromosome:GRCh38:1:1:248956422:1> term:inEnsemblSchemaNumber "89" .
<http://rdf.ebi.ac.uk/resource/ensembl/89/homo_sapiens/GRCh38/chromosome:GRCh38:1:1:248956422:1> term:inEnsemblAssembly "GRCh38" .
...
...
```

```
...
...
<http://rdf.ebi.ac.uk/resource/ensembl/ENSG00000127720> rdf:type obo:SO_0001217 .
<http://rdf.ebi.ac.uk/resource/ensembl/ENSG00000127720> rdf:type term:protein_coding .
<http://rdf.ebi.ac.uk/resource/ensembl/ENSG00000127720> rdfs:label "METTL25" .
<http://rdf.ebi.ac.uk/resource/ensembl/ENSG00000127720> dc:description
  "methyltransferase like 25 [Source:HGNC Symbol;Acc:HGNC:26228]" .
<http://rdf.ebi.ac.uk/resource/ensembl/ENSG00000127720> obo:RO_0002162 taxon:9606 .
<http://rdf.ebi.ac.uk/resource/ensembl/ENSG00000127720> dc:identifier "ENSG00000127720" .
<http://rdf.ebi.ac.uk/resource/ensembl/ENSG00000127720> rdfs:seeAlso <http://identifiers.org/ensembl/ENSG00000127720> .
<http://identifiers.org/ensembl/ENSG00000127720> rdf:type identifiers:ensembl .
<http://identifiers.org/ensembl/ENSG00000127720> sio:SIO_000671 [a ident_type:ensembl; sio:SIO_000300 "ENSG00000127720"] .
...
<http://rdf.ebi.ac.uk/resource/ensembl.transcript/ENST00000248306#Exon_1> rdf:type sio:SIO_001261 .
<http://rdf.ebi.ac.uk/resource/ensembl.transcript/ENST00000248306#Exon_1> sio:SIO_000628
  <http://rdf.ebi.ac.uk/resource/ensembl.exon/ENSE00000910621> .
<http://rdf.ebi.ac.uk/resource/ensembl.transcript/ENST00000248306#Exon_1> sio:SIO_000300 1 .
<http://rdf.ebi.ac.uk/resource/ensembl.protein/ENSP00000248306> obo:RO_0002162 taxon:9606 .
<http://rdf.ebi.ac.uk/resource/ensembl.protein/ENSP00000248306> dc:identifier "ENSP00000248306" .
<http://rdf.ebi.ac.uk/resource/ensembl.protein/ENSP00000248306> rdfs:seeAlso <http://identifiers.org/ensembl/ENSP00000248306> .
<http://identifiers.org/ensembl/ENSP00000248306> rdf:type identifiers:ensembl .
<http://identifiers.org/ensembl/ENSP00000248306> sio:SIO_000671 [a ident_type:ensembl; sio:SIO_000300 "ENSP00000248306"] .
<http://rdf.ebi.ac.uk/resource/ensembl.transcript/ENST00000248306> obo:SO_translates_to
  <http://rdf.ebi.ac.uk/resource/ensembl.protein/ENSP00000248306> .
<http://rdf.ebi.ac.uk/resource/ensembl.protein/ENSP00000248306> rdf:type term:protein .
<http://rdf.ebi.ac.uk/resource/ensembl.protein/ENSP00000248306> rdfs:seeAlso gene3d:3.40.50.150 .
...
...
```

Fetch-all-the-things

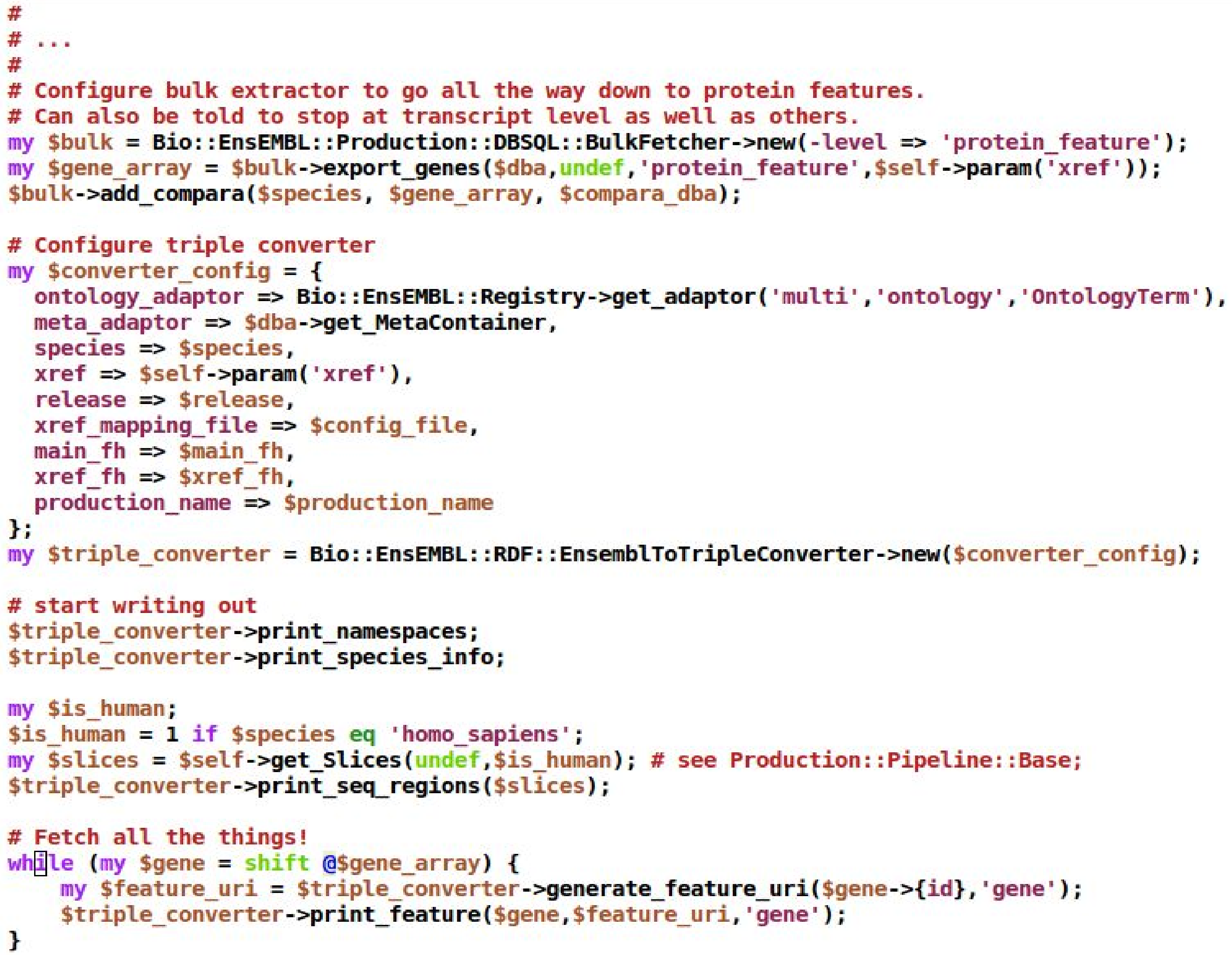
BulkFetcher

- ‘quickly’ fetches all gene-transcript-exon-translation(-protein features)
- no API, direct SQL
- returns unorthodox “features”
  - i.e. array of recursive hashes
- not the focus, but has to be taken into account

Dump-all-the-things

Bio::EnsEMBL::RDF::EnsemblToTripleConverter

- specialised methods to dump different things:
  - name spaces (prefixes)
  - species info
  - slices
  - BulkFetcher-derived “features”



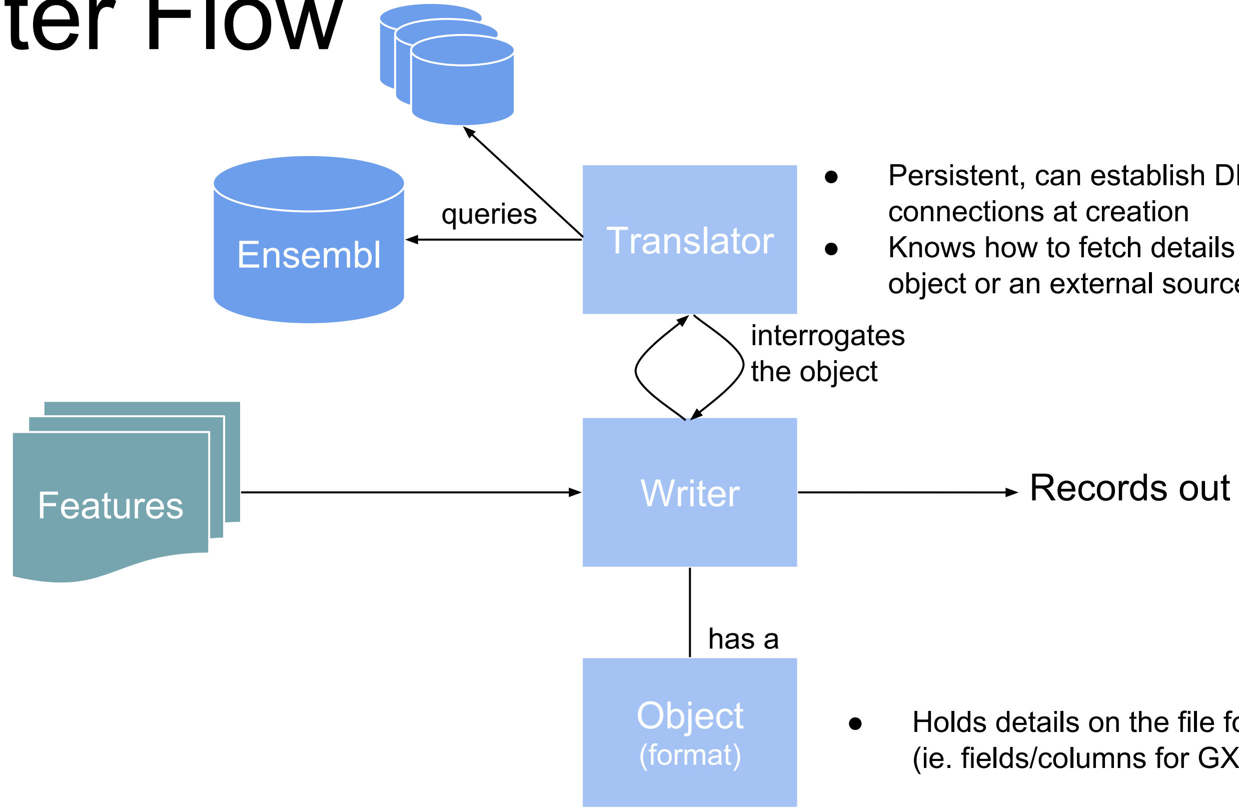
```perl
#
# ...
#
# Configure bulk extractor to go all the way down to protein features.
# Can also be told to stop at transcript level as well as others.
my $bulk = Bio::EnsEMBL::Production::DBSQL::BulkFetcher->new(-level => 'protein_feature');
my $gene_array = $bulk->export_genes($dba,undef,'protein_feature',$self->param('xref'));
$bulk->add_compara($species, $gene_array, $compara_dba);

# Configure triple converter
my $converter_config = {
  ontology_adaptor => Bio::EnsEMBL::Registry->get_adaptor('multi','ontology','OntologyTerm'),
  meta_adaptor => $dba->get_MetaContainer,
  species => $species,
  xref => $self->param('xref'),
  release => $release,
  xref_mapping_file => $config_file,
  main_fh => $main_fh,
  xref_fh => $xref_fh,
  production_name => $production_name
};
my $triple_converter = Bio::EnsEMBL::RDF::EnsemblToTripleConverter->new($converter_config);

# start writing out
$triple_converter->print_namespaces;
$triple_converter->print_species_info;

my $is_human;
$is_human = 1 if $species eq 'homo_sapiens';
my $slices = $self->get_Slices(undef,$is_human); # see Production::Pipeline::Base;
$triple_converter->print_seq_regions($slices);

# Fetch all the things!
while (my $gene = shift @$gene_array) {
    my $feature_uri = $triple_converter->generate_feature_uri($gene->{id},'gene');
    $triple_converter->print_feature($gene,$feature_uri,'gene');
}
```

The Task

RDF pipeline to use ensembl-io writer model

- retain fetch-all-the-things approach
- writer object consumes things and emits records
    - namespaces
    - seq regions
    - “features”

eIO Writer Flow



- Persistent, can establish DB connections at creation
- Knows how to fetch details from object or an external source

- Holds details on the file format (ie. fields/columns for GXF formats)

Motivation

- Fragmented readers, serialized across code bases
- Presentation layer properties in writers
- Tightly tied to Ensembl objects

Writer

- Writer is subclassed by format (GTF, GFF3, Fasta)
- Writer knows needed fields, how to format a record
- Writer sets up any format specific needs in Translator
    - ie GFF3 Writer tells Feature translator how to convert a strand

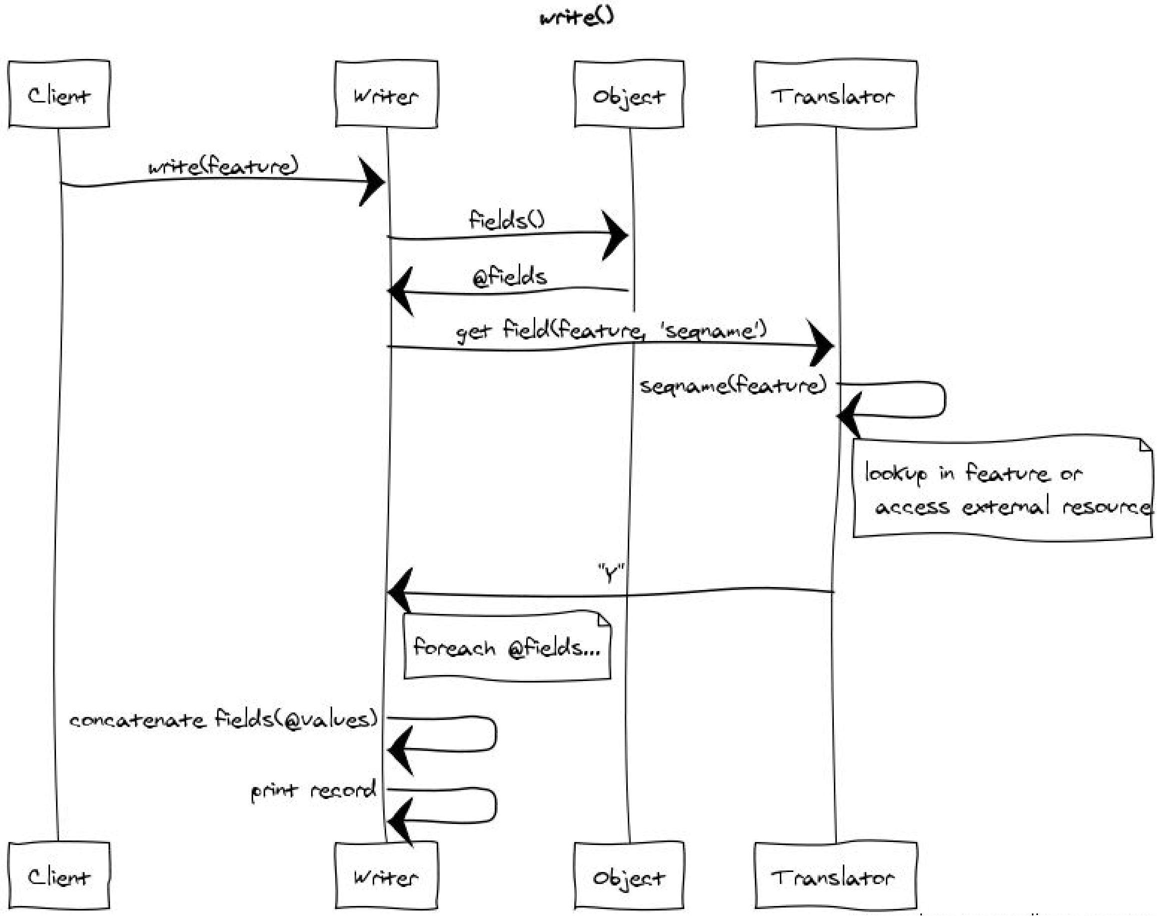
Writer

```
new()
open(filename)
close()
write(object[, translator_ref])
translator(translator_ref)  # if not in new()
```

Writer

```
my $translator = Bio::EnsEMBL::IO::Translator::Feature->new();
my $serializer = Bio::EnsEMBL::IO::Writer::GFF3->new($translator);
$serializer->open('/tmp/outfile.gff');

$serializer->write($feature);
```

Example (GXF formats)



Translator

```
get_field(object, field)
   Args[1]    : ref, object to query
   Args[2]    : scalar, field name
   Returntype: string, hashref, or undef
-  Fetch a single field from the object or external source(s)

batch_fields(object, fields)
   Args[1]    : ref, object to query
   Args[2]    : listref, field values to return, in this order
   Returntype: array
-  Fetch one or more fields from the object or external source(s)
```

Translator (callbacks)

```
my %ens_field_callbacks = (seqname   => 'seqname',
                           source    => 'source',
                           type      => 'type',
                           start     => 'start',
                           attribute => sub { $other_translator->get_attribute(@_) }
                           … );
sub start {
    my $self = shift;
    my $object = shift;

    return $object->start();
}
```

Translator (callbacks)

```
add_callbacks(callbacks)
   Args[1]    : hashref, list of fields and callbacks
```

- Sub-classing isn’t always necessary
- IO users can override callbacks to add custom functionality

```
fetch_callback(field)
   Args[1]    : string, field to retrieve callback for
   Returntype: string
```

Your task (serializer)

For each format and data source, derive classes from:

- Bio::EnsEMBL::IO::Writer
- Bio::EnsEMBL::IO::Object *(if needed)*
- Bio::EnsEMBL::IO::Translator

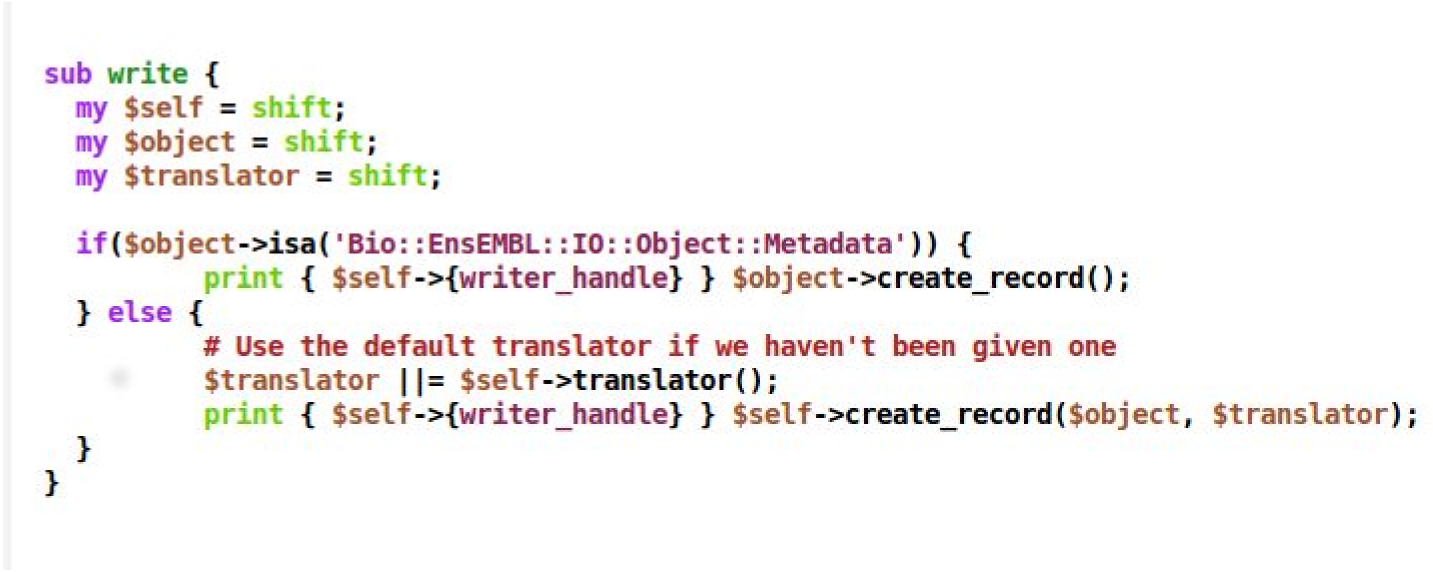
Eg. GFF3 chromosome dumps:

- Bio::EnsEMBL::IO::Writer::GFF3 (via Bio::EnsEMBL::IO::Writer::ColumnBasedGeneric)
- Bio::EnsEMBL::IO::Object::GFF3
- Bio::EnsEMBL::IO::Translator::Feature

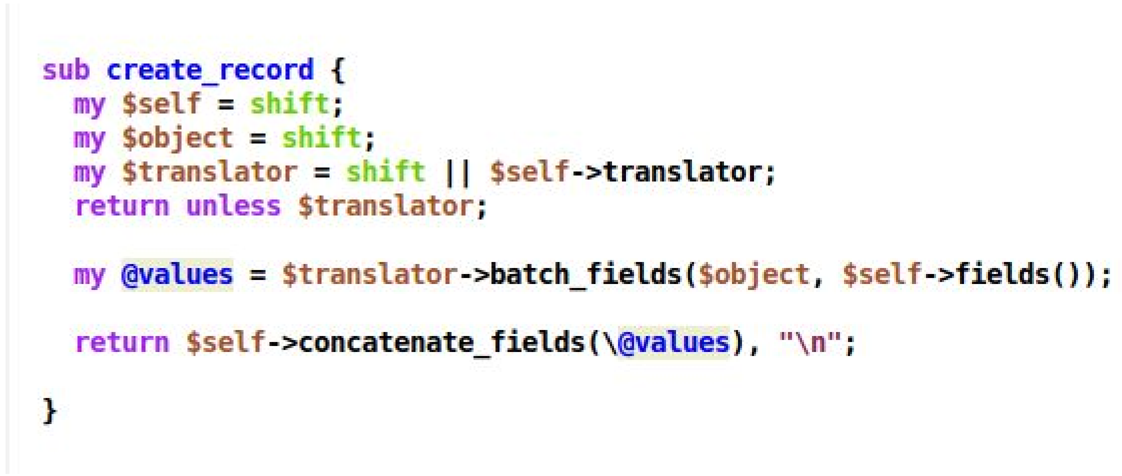
Case Study: Column Based Formats

- GTF, GFF3, VCF4 etc.
- Writers inherit from `Bio::EnsEMBL::IO::Writer::ColumnBasedGeneric`
  - define common write method
  - Object::Metadata/Writer subclasses implement method to create format specific header/record

ColumnBasedGeneric Writer

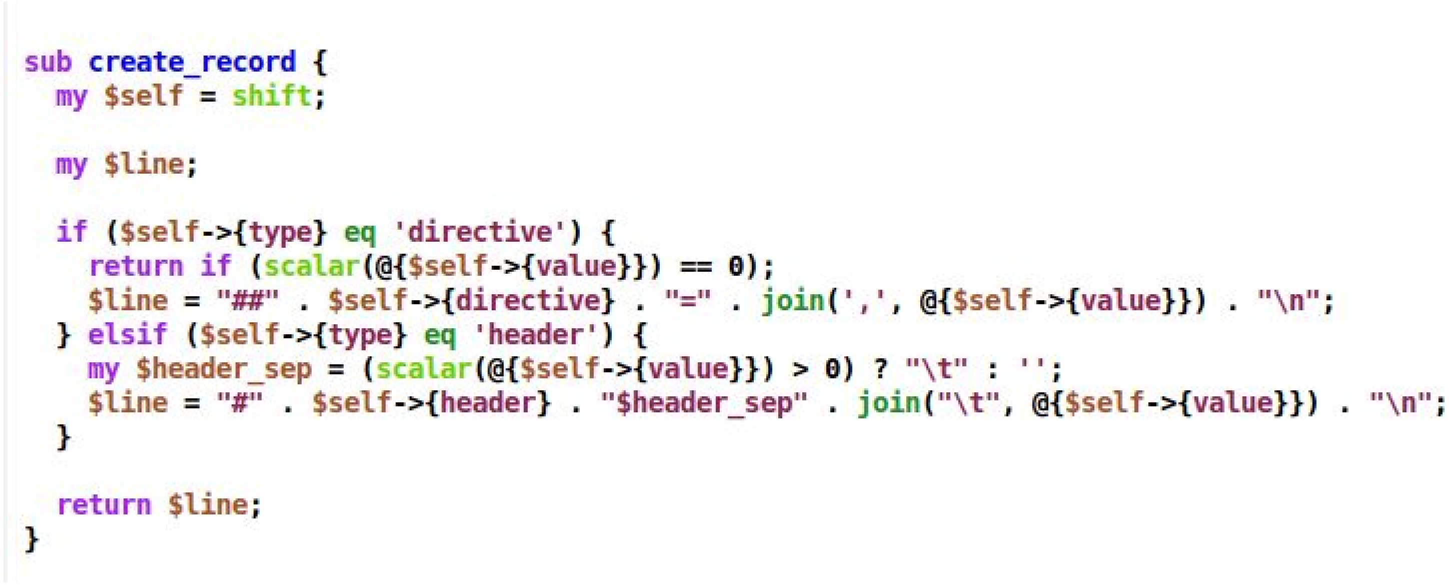


```
sub write {
  my $self = shift;
  my $object = shift;
  my $translator = shift;

  if($object->isa('Bio::EnsEMBL::IO::Object::Metadata')) {
          print { $self->{writer_handle} } $object->create_record();
  } else {
          # Use the default translator if we haven't been given one
          $translator ||= $self->translator();
          print { $self->{writer_handle} } $self->create_record($object, $translator);
  }
}
```



```perl
sub create_record {
  my $self = shift;
  my $object = shift;
  my $translator = shift || $self->translator;
  return unless $translator;

  my @values = $translator->batch_fields($object, $self->fields());

  return $self->concatenate_fields(\@values), "\n";

}
```

Object::VCF4Metadata



```perl
sub create_record {
  my $self = shift;

  my $line;

  if ($self->{type} eq 'directive') {
    return if (scalar(@{$self->{value}}) == 0);
    $line = "##" . $self->{directive} . "=" . join(',', @{$self->{value}}) . "\n";
  } elsif ($self->{type} eq 'header') {
    my $header_sep = (scalar(@{$self->{value}}) > 0) ? "\t" : '';
    $line = "#" . $self->{header} . "$header_sep" . join("\t", @{$self->{value}}) . "\n";
  }

  return $line;
}
```

Dumping RDF: Considerations

- pipeline dumps features/xrefs separately
  - two writers?
- namespaces/species info: file prelude, metadata header?
- different things need different translators
  - seq regions
  - BulkFetcher items

Issues

- don’t deal with classic e! features, not even objects!
- what is a record?
  - definitively not just a triple (though is the atomic unit of information)
  - set of predicates associated to the same subject?
- no fixed format record, dynamic set of triples
  - different things translates to different sets of triples/predicates, even if of the same ‘type’
  - no collate-format-dump beautiful simplicity

Interface Design

```
my $strans = Bio::EnsEMBL::IO::Translator::Slice->new(%sargs);
my $ftrans =
  Bio::EnsEMBL::IO::Translator::BulkFetcherFeature->new(%fargs);
my $writer = Bio::EnsEMBL::IO::Writer::RDF->new();
$writer->open('outfile.ttl');

$writer->write($namespaces);
$writer->write($species);
for $slice in (@slices) { $writer->write($slice, $strans); }
for $feature in (@features) { $writer->write($feature, $ftrans); }
```

Borrowing from Mongoose

`Bio::EnsEMBL::Utils::RDF`

- utility functions, common shortcuts for formatting RDF
- common namespace definitions

`Bio::EnsEMBL::Utils::RDF::Mapper`

- convert Ensembl internal names for things into:
    - identifiers.org URIs
    - specific host org namespace for the data type (if known)

Seq Regions

```
my %field_callbacks = (version         => 'version',
                       production_name => 'production_name',
                       taxon_id        => 'taxon_id',
                       scientific_name => 'scientific_name',
                       name            => 'name',
                       cs_name         => 'cs_name',
                       cs_version      => 'cs_version',
                       uri             => 'uri');

my $translator = Bio::EnsEMBL::IO::Translator::Slice->new($meta_adaptor);
```

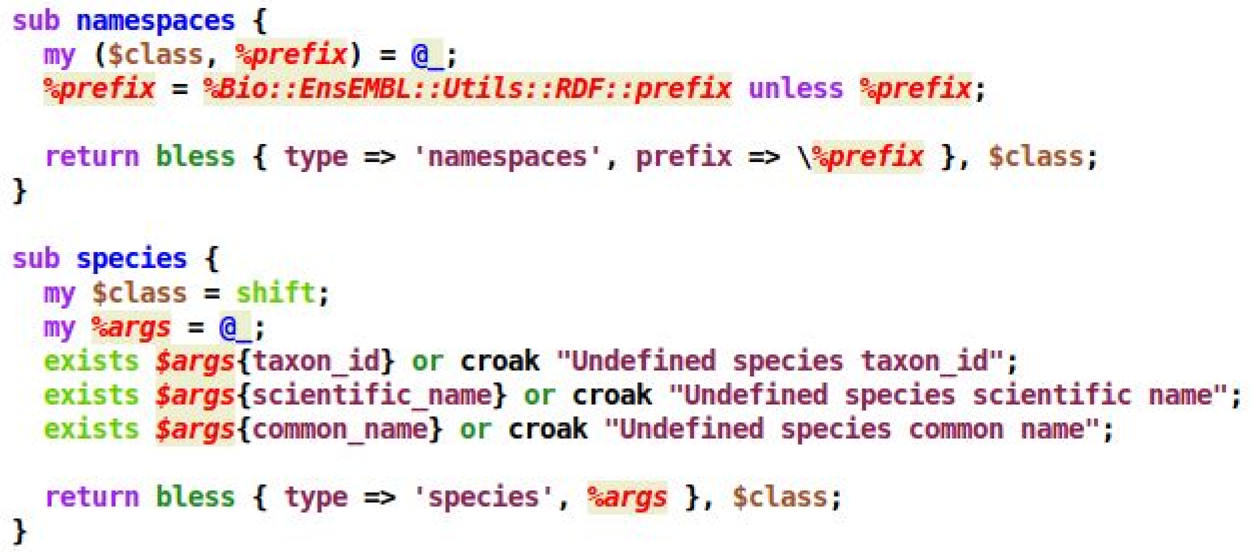
BulkFetcher Features

```
my %field_callbacks = (type => 'type',
                       id    => 'id',
                       name  => 'name',
                       ...
                       start => 'start',
                       ...
                       transcripts    => 'transcripts',
                       ...
                       uri       => 'uri');
my $translator =
     Bio::EnsEMBL::IO::Translator::BulkFetcherFeature->new($xref_mapping_file,
                                                           $ontology_adaptor
                                                           $meta_adaptor);
```

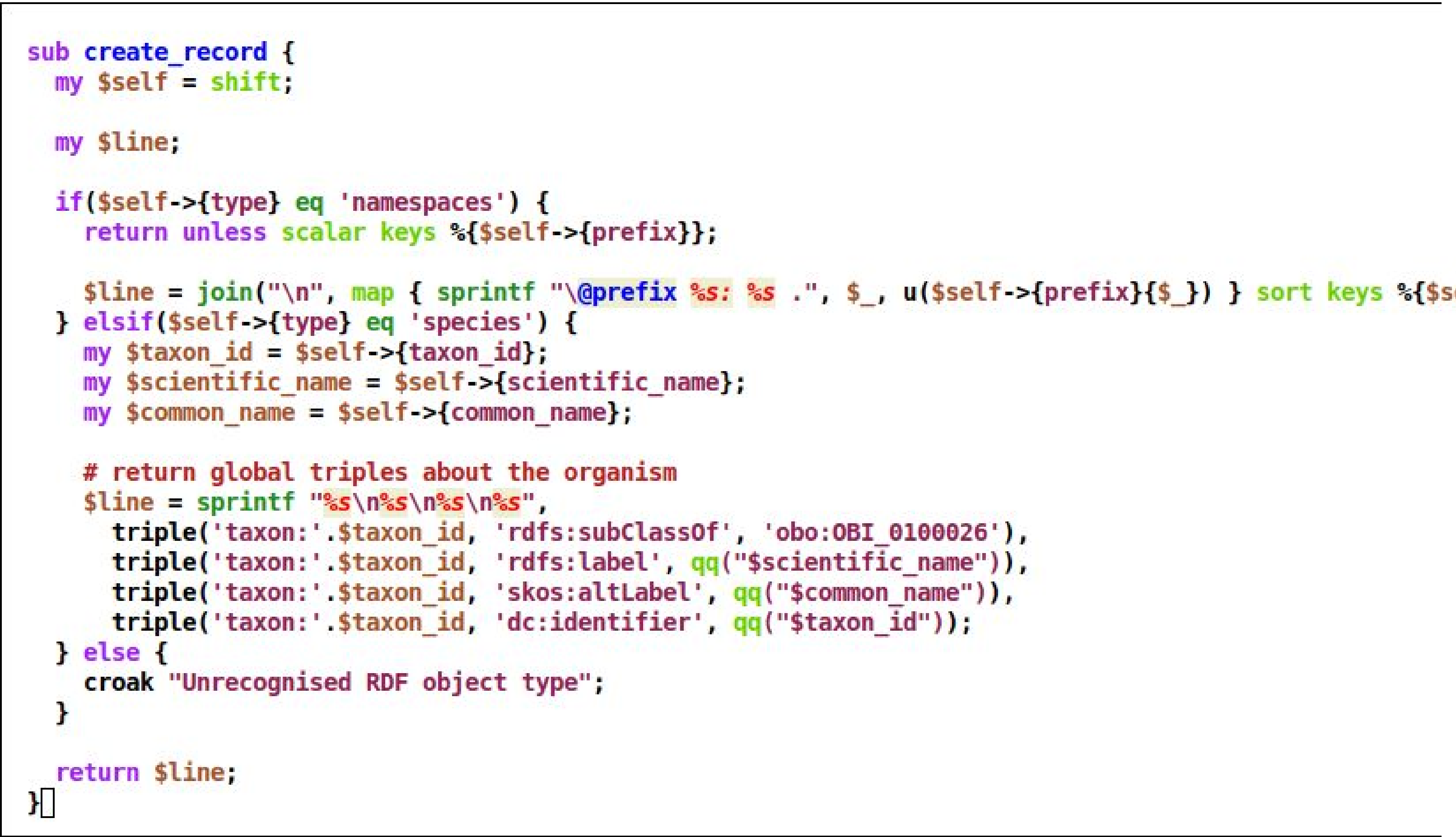
RDF Header

- Follow the GTF/GFF3/VCF approach
- Define self-reflective class
  - methods to create object special cases
  - create_record handles the details, used by writer

Bio::EnsEMBL::IO::Object::RDF

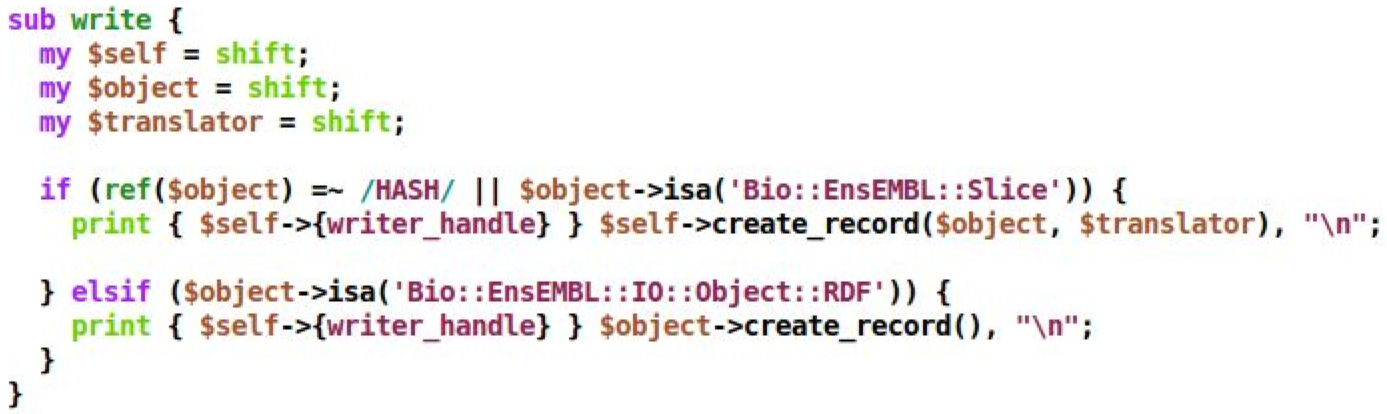


```perl
sub namespaces {
  my ($class, %prefix) = @_;
  %prefix = %Bio::EnsEMBL::Utils::RDF::prefix unless %prefix;

  return bless { type => 'namespaces', prefix => \%prefix }, $class;
}

sub species {
  my $class = shift;
  my %args = @_;
  exists $args{taxon_id} or croak "Undefined species taxon_id";
  exists $args{scientific_name} or croak "Undefined species scientific name";
  exists $args{common_name} or croak "Undefined species common name";

  return bless { type => 'species', %args }, $class;
}
```

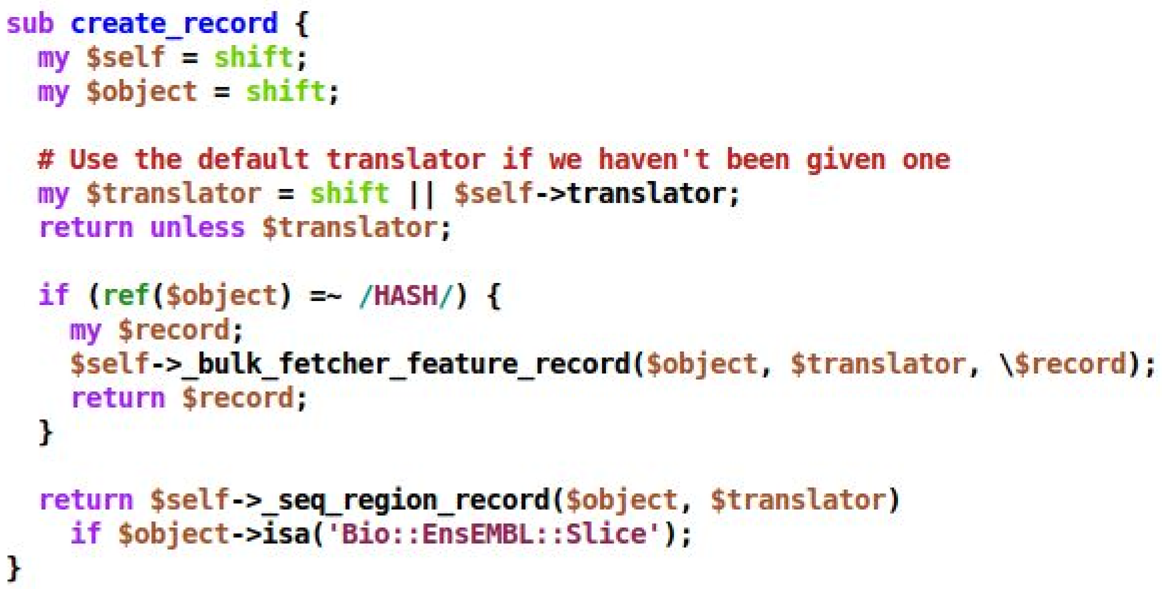


```perl
sub create_record {
  my $self = shift;

  my $line;

  if($self->{type} eq 'namespaces') {
    return unless scalar keys %{$self->{prefix}};

    $line = join("\n", map { sprintf "\@prefix %s: %s .", $_, u($self->{prefix}{$_}) } sort keys %{$self->{prefix}});
  } elsif($self->{type} eq 'species') {
    my $taxon_id = $self->{taxon_id};
    my $scientific_name = $self->{scientific_name};
    my $common_name = $self->{common_name};

    # return global triples about the organism
    $line = sprintf "%s\n%s\n%s\n%s",
      triple('taxon:'.$taxon_id, 'rdfs:subClassOf', 'obo:OBI_0100026'),
      triple('taxon:'.$taxon_id, 'rdfs:label', qq("$scientific_name")),
      triple('taxon:'.$taxon_id, 'skos:altLabel', qq("$common_name")),
      triple('taxon:'.$taxon_id, 'dc:identifier', qq("$taxon_id"));
  } else {
    croak "Unrecognised RDF object type";
  }

  return $line;
}
```

Bio::EnsEMBL::IO::Writer::RDF

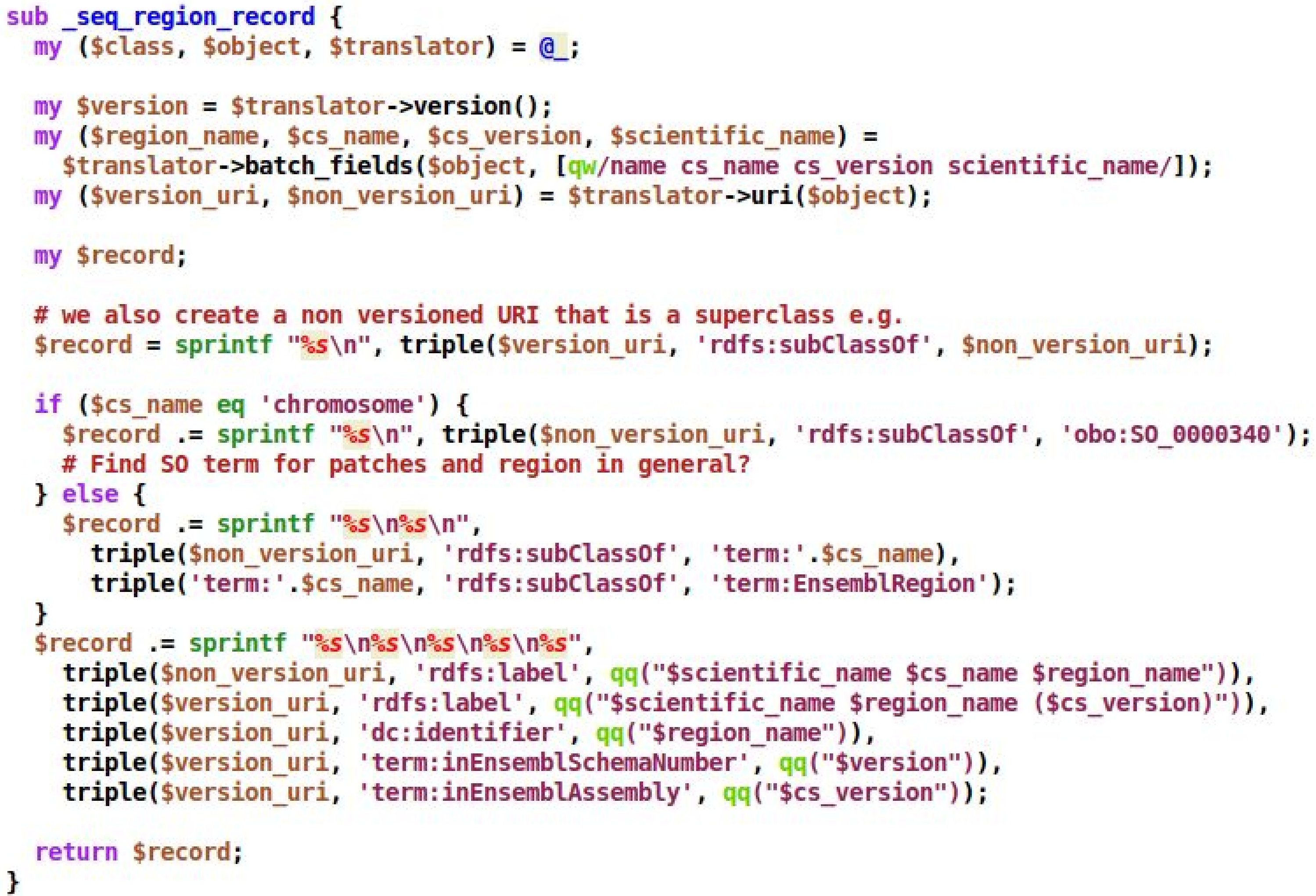


```perl
sub write {
  my $self = shift;
  my $object = shift;
  my $translator = shift;

  if (ref($object) =~ /HASH/ || $object->isa('Bio::EnsEMBL::Slice')) {
    print { $self->{writer_handle} } $self->create_record($object, $translator), "\n";

  } elsif ($object->isa('Bio::EnsEMBL::IO::Object::RDF')) {
    print { $self->{writer_handle} } $object->create_record(), "\n";
  }
}
```



```perl
sub create_record {
  my $self = shift;
  my $object = shift;

  # Use the default translator if we haven't been given one
  my $translator = shift || $self->translator;
  return unless $translator;

  if (ref($object) =~ /HASH/) {
    my $record;
    $self->_bulk_fetcher_feature_record($object, $translator, \$record);
    return $record;
  }

  return $self->_seq_region_record($object, $translator)
    if $object->isa('Bio::EnsEMBL::Slice');
}
```

wellcome trust sanger institute
EMBL-EBI



```perl
sub _seq_region_record {
  my ($class, $object, $translator) = @_;

  my $version = $translator->version();
  my ($region_name, $cs_name, $cs_version, $scientific_name) =
    $translator->batch_fields($object, [qw/name cs_name cs_version scientific_name/]);
  my ($version_uri, $non_version_uri) = $translator->uri($object);

  my $record;

  # we also create a non versioned URI that is a superclass e.g.
  $record = sprintf "%s\n", triple($version_uri, 'rdfs:subClassOf', $non_version_uri);

  if ($cs_name eq 'chromosome') {
    $record .= sprintf "%s\n", triple($non_version_uri, 'rdfs:subClassOf', 'obo:SO_0000340');
    # Find SO term for patches and region in general?
  } else {
    $record .= sprintf "%s\n%s\n",
      triple($non_version_uri, 'rdfs:subClassOf', 'term:'.$cs_name),
      triple('term:'.$cs_name, 'rdfs:subClassOf', 'term:EnsemblRegion');
  }
  $record .= sprintf "%s\n%s\n%s\n%s\n%s",
    triple($non_version_uri, 'rdfs:label', qq("$scientific_name $cs_name $region_name")),
    triple($version_uri, 'rdfs:label', qq("$scientific_name $region_name ($cs_version)")),
    triple($version_uri, 'dc:identifier', qq("$region_name")),
    triple($version_uri, 'term:inEnsemblSchemaNumber', qq("$version")),
    triple($version_uri, 'term:inEnsemblAssembly', qq("$cs_version"));

  return $record;
}
```

Altogether

```perl
my $strans = Bio::EnsEMBL::IO::Translator::Slice->new(%sargs);
my $ftrans = Bio::EnsEMBL::IO::Translator::BulkFetcherFeature->new(%fargs);

my $writer = Bio::EnsEMBL::IO::Writer::RDF->new();
$writer->open('outfile.ttl');

# namespaces and species prelude
$writer->write(Bio::EnsEMBL::IO::Object::RDF->namespaces());
$writer->write(Bio::EnsEMBL::IO::Object::RDF->species(taxon_id => 9606, ...));

# seq regions and bulkfetcher items
for $slice in (@slices) { $writer->write($slice, $strans); }
for $feature in (@bfeatures) { $writer->write($feature, $ftrans); }
```

TODO - RDF Xrefs

`Bio::EnsEMBL::IO::Writer::RDF::Xrefs`

- acts on BulkFetcher items
- can use the same translator
- 'write' invoke xrefs focused callbacks

```
my $ftrans = Bio::EnsEMBL::IO::Translator::BulkFetcherFeature->new(%fargs);

my $writer = Bio::EnsEMBL::IO::Writer::RDF::Xrefs->new();
$writer->open('xrefs.ttl');
for $feature in (@bfeatures) { $writer->write($feature, $ftrans); }
```

Open Issues

- No e! features
  - would need to extend existing translator (callbacks/adaptors)
  - an intermediate layer converting BulkFetcher features
- Design flaw?!
  - writing involves lots of type-checking, can we improve?
  - presentation
  - can it be used in other contexts, REST/Web?