Chapter 17

KEGG

KEGG (http://www.kegg.jp/) is a database resource for understanding high-level functions and utilities of the biological system, such as the cell, the organism and the ecosystem, from molecular-level information, especially large-scale molecular datasets generated by genome sequencing and other high-throughput experimental technologies.

Please note that the KEGG parser implementation in Biopython is incomplete. While the KEGG website indicates many flat file formats, only parsers and writers for compound, enzyme, and map are currently implemented. However, a generic parser is implemented to handle the other formats.

17.1 Parsing KEGG records

Parsing a KEGG record is as simple as using any other file format parser in Biopython.

```
>>> from Bio.KEGG import Enzyme
>>> records = Enzyme.parse(open("ec:5.4.2.2.txt"))
>>> record = list(records)[0]
>>> record.classname
['Isomerases;', 'Intramolecular transferases;', 'Phosphotransferases (phosphomutases)']
>>> record.entry
'5.4.2.2'
```

The following section will shows how to download the above enzyme using the KEGG api as well as how to use the generic parser with data that does not have a custom parser implemented.

17.2 Querying the KEGG API

Biopython has full support for the querying of the KEGG api. Querying all KEGG endpoints are supported; all methods documented by KEGG (http://www.kegg.jp/kegg/rest/keggapi.html) are supported. The interface has some validation of queries which follow rules defined on the KEGG site. However, invalid queries which return a 400 or 404 must be handled by the user.

First, here is how to extend the above example by downloading the relevant enzyme and passing it through the Enzyme parser.

```
>>> from Bio import KEGG
>>> from Bio.KEGG import Enzyme
>>> request = KEGG.query("get", "ec:5.4.2.2")
>>> open("ec:5.4.2.2.txt", 'w').write(request)
>>> records = Enzyme.parse(open("ec:5.4.2.2.txt"))
>>> record = list(records)[0]
>>> record.classname
['Isomerases;', 'Intramolecular transferases;', 'Phosphotransferases (phosphomutases)']
>>> record.entry
'5.4.2.2'
```

Now, here's a more realistic example which shows a combination of querying the KEGG API and use of the generic parser. This will demonstrate how to extract a unique set of all human pathway gene symbols which relate to DNA repair. The steps that need to be taken to do so are as follows. First, we need to get a list of all human pathways. Secondly, we need to filter those for ones which relate to ”repair”. Lastly, we need to get a list of all the gene symbols in all repair pathways.

```
from Bio import KEGG

human_pathways = KEGG.query("list", "pathway", "hsa")
human_pathways = KEGG.generic_parser("list", human_pathways)

# Filter all human pathways for repair pathways
repair_pathways = []
for id_, description in human_pathways:
    if "repair" in description:
        repair_pathways.append(id_)

# Get the genes for all three of these pathways and add them to a list
repair_genes = []
for pathway in repair_pathways:
    pathway = KEGG.query("get", pathway)
    pathway = KEGG.generic_parser("get", pathway)
    pathway_genes = pathway[0]["gene"]
    for gene in pathway_genes:
        gene_identifiers, gene_description = gene.split("; ")
        gene_id, gene_symbol = gene_identifiers.split()
        repair_genes.append(gene_symbol)

# Make this list of genes unique
repair_genes = list(set(repair_genes))
print ", ".join(repair_genes)
```

The KEGG API wrapper is compatible with all endpoints. Usage is essentially replacing all slashes in the url with commas and using that list as arguments to KEGG.query. Here are a few examples from the api documentation (http://www.kegg.jp/kegg/docs/keggapi.html).

```
/list/hsa:10458+ece:Z5100          -> KEGG.query("list", ["hsa:10458", "ece:Z510"])
/find/compound/300-310/mol_weight  -> KEGG.query("find", "compound", "300-310", "mol_weight")
/get/hsa:10458+ece:Z5100/aaseq     -> KEGG.query("get", ["hsa:10458", "ece:Z5100"], "aaseq")
```

The generic parser deonstrated above requires the operation to be specified as well since depending on the operation, different responses from KEGG are genereated. As specified in the KEGG documentation, the generic parser acts upon the different operations as follows:

- List, find, conv, and link operations return tab-delimited text, which is parsed into a list of lists.
- Info returns plain text, which is parsed into a list (of lines).

- Get reutrns a database format. Since each database format is different, it's parsed into a dict with the section as the key, and a list of lines corresponding to that section as the value. It's up to the user to then take that value and parse out the information needed.