**Course: Advanced Bioinformatics**

**Module title: Utils Entrez**

**Module no. : 206**

**Accessing NCBI’s Entrez databases**

Entrez (<http://www.ncbi.nlm.nih.gov/Entrez>) is a data retrieval system that provides users access to NCBI’s databases such as PubMed, GenBank, GEO, and many others. You can access Entrez from a web browser to manually enter queries, or you can use Biopython’s Bio.Entrez module for programmatic access to Entrez. The latter allows you for example to search PubMed or download GenBank records from within a Python script.

The Bio.Entrez module makes use of the Entrez Programming Utilities (also known as EUtils), consisting of eight tools that are described in detail on NCBI’s page at <http://www.ncbi.nlm.nih.gov/entrez/utils/>. Each of these tools corresponds to one Python function in the Bio.Entrezmodule, as described in the sections below. This module makes sure that the correct URL is used for the queries, and that not more than one request is made every three seconds, as required by NCBI.

The output returned by the Entrez Programming Utilities is typically in XML format. To parse such output, you have several options:

1. Use Bio.Entrez’s parser to parse the XML output into a Python object;
2. Use the DOM (Document Object Model) parser in Python’s standard library;
3. Use the SAX (Simple API for XML) parser in Python’s standard library;
4. Read the XML output as raw text, and parse it by string searching and manipulation.

For the DOM and SAX parsers, see the Python documentation. The parser in Bio.Entrez is discussed below.

NCBI uses DTD (Document Type Definition) files to describe the structure of the information contained in XML files. Most of the DTD files used by NCBI are included in the Biopython distribution. The Bio.Entrez parser makes use of the DTD files when parsing an XML file returned by NCBI Entrez.

Occasionally, you may find that the DTD file associated with a specific XML file is missing in the Biopython distribution. In particular, this may happen when NCBI updates its DTD files. If this happens, Entrez.read will show a warning message with the name and URL of the missing DTD file. The parser will proceed to access the missing DTD file through the internet, allowing the parsing of the XML file to continue. However, the parser is much faster if the DTD file is available locally. For this purpose, please download the DTD file from the URL in the warning message and place it in the directory ...site-packages/Bio/Entrez/DTDs, containing the other DTD files. If you don’t have write access to this directory, you can also place the DTD file in ~/.biopython/Bio/Entrez/DTDs, where ~ represents your home directory. Since this directory is read before the directory ...site-packages/Bio/Entrez/DTDs, you can also put newer versions of DTD files there if the ones in ...site-packages/Bio/Entrez/DTDs become outdated. Alternatively, if you installed Biopython from source, you can add the DTD file to the source code’s Bio/Entrez/DTDs directory, and reinstall Biopython. This will install the new DTD file in the correct location together with the other DTD files.

**Entrez Guidelines**

Before using Biopython to access the NCBI’s online resources (via Bio.Entrez or some of the other modules), please read the [NCBI’s Entrez User Requirements](http://www.ncbi.nlm.nih.gov/books/NBK25497/#chapter2.Usage_Guidelines_and_Requiremen). If the NCBI finds you are abusing their systems, they can and will ban your access!

To paraphrase:

* For any series of more than 100 requests, do this at weekends or outside USA peak times. This is up to you to obey.
* Use the [http://eutils.ncbi.nlm.nih.gov](http://eutils.ncbi.nlm.nih.gov/) address, not the standard NCBI Web address. Biopython uses this web address.
* Make no more than three requests every seconds (relaxed from at most one request every three seconds in early 2009). This is automatically enforced by Biopython.
* Use the optional email parameter so the NCBI can contact you if there is a problem. You can either explicitly set this as a parameter with each call to Entrez (e.g. include email="A.N.Other@example.com" in the argument list), or you can set a global email address:
* >>> from Bio import Entrez
* >>> Entrez.email = "A.N.Other@example.com"

Bio.Entrez will then use this email address with each call to Entrez. The example.com address is a reserved domain name specifically for documentation (RFC 2606). Please DO NOT use a random email – it’s better not to give an email at all. The email parameter will be mandatory from June 1, 2010. In case of excessive usage, NCBI will attempt to contact a user at the e-mail address provided prior to blocking access to the E-utilities.

* If you are using Biopython within some larger software suite, use the tool parameter to specify this. You can either explicitly set the tool name as a parameter with each call to Entrez (e.g. include tool="MyLocalScript" in the argument list), or you can set a global tool name:
* >>> from Bio import Entrez
* >>> Entrez.tool = "MyLocalScript"

The tool parameter will default to Biopython.

**EInfo: Obtaining information about the Entrez databases**

EInfo provides field index term counts, last update, and available links for each of NCBI’s databases. In addition, you can use EInfo to obtain a list of all database names accessible through the Entrez utilities:

>>> from Bio import Entrez

>>> Entrez.email = "A.N.Other@example.com" # Always tell NCBI who you are

>>> handle = Entrez.einfo()

>>> result = handle.read()

The variable result now contains a list of databases in XML format:

>>> print(result)

<?xml version="1.0"?>

<!DOCTYPE eInfoResult PUBLIC "-//NLM//DTD eInfoResult, 11 May 2002//EN"

"http://www.ncbi.nlm.nih.gov/entrez/query/DTD/eInfo\_020511.dtd">

<eInfoResult>

<DbList>

<DbName>pubmed</DbName>

<DbName>protein</DbName>

<DbName>nucleotide</DbName>

<DbName>nuccore</DbName>

<DbName>nucgss</DbName>

<DbName>nucest</DbName>

<DbName>structure</DbName>

<DbName>genome</DbName>

<DbName>books</DbName>

<DbName>cancerchromosomes</DbName>

<DbName>cdd</DbName>

<DbName>gap</DbName>

<DbName>domains</DbName>

<DbName>gene</DbName>

<DbName>genomeprj</DbName>

<DbName>gensat</DbName>

<DbName>geo</DbName>

<DbName>gds</DbName>

<DbName>homologene</DbName>

<DbName>journals</DbName>

<DbName>mesh</DbName>

<DbName>ncbisearch</DbName>

<DbName>nlmcatalog</DbName>

<DbName>omia</DbName>

<DbName>omim</DbName>

<DbName>pmc</DbName>

<DbName>popset</DbName>

<DbName>probe</DbName>

<DbName>proteinclusters</DbName>

<DbName>pcassay</DbName>

<DbName>pccompound</DbName>

<DbName>pcsubstance</DbName>

<DbName>snp</DbName>

<DbName>taxonomy</DbName>

<DbName>toolkit</DbName>

<DbName>unigene</DbName>

<DbName>unists</DbName>

</DbList>

</eInfoResult>

Since this is a fairly simple XML file, we could extract the information it contains simply by string searching. Using Bio.Entrez’s parser instead, we can directly parse this XML file into a Python object:

>>> from Bio import Entrez

>>> handle = Entrez.einfo()

>>> record = Entrez.read(handle)

Now record is a dictionary with exactly one key:

>>> record.keys()

[u'DbList']

The values stored in this key is the list of database names shown in the XML above:

>>> record["DbList"]

['pubmed', 'protein', 'nucleotide', 'nuccore', 'nucgss', 'nucest',

'structure', 'genome', 'books', 'cancerchromosomes', 'cdd', 'gap',

'domains', 'gene', 'genomeprj', 'gensat', 'geo', 'gds', 'homologene',

'journals', 'mesh', 'ncbisearch', 'nlmcatalog', 'omia', 'omim', 'pmc',

'popset', 'probe', 'proteinclusters', 'pcassay', 'pccompound',

'pcsubstance', 'snp', 'taxonomy', 'toolkit', 'unigene', 'unists']

For each of these databases, we can use EInfo again to obtain more information:

>>> handle = Entrez.einfo(db="pubmed")

>>> record = Entrez.read(handle)

>>> record["DbInfo"]["Description"]

'PubMed bibliographic record'

>>> record["DbInfo"]["Count"]

'17989604'

>>> record["DbInfo"]["LastUpdate"]

'2008/05/24 06:45'

Try record["DbInfo"].keys() for other information stored in this record. One of the most useful is a list of possible search fields for use with ESearch:

>>> for field in record["DbInfo"]["FieldList"]:

... print("%(Name)s, %(FullName)s, %(Description)s" % field)

ALL, All Fields, All terms from all searchable fields

UID, UID, Unique number assigned to publication

FILT, Filter, Limits the records

TITL, Title, Words in title of publication

WORD, Text Word, Free text associated with publication

MESH, MeSH Terms, Medical Subject Headings assigned to publication

MAJR, MeSH Major Topic, MeSH terms of major importance to publication

AUTH, Author, Author(s) of publication

JOUR, Journal, Journal abbreviation of publication

AFFL, Affiliation, Author's institutional affiliation and address

...

That’s a long list, but indirectly this tells you that for the PubMed database, you can do things like Jones[AUTH] to search the author field, or Sanger[AFFL] to restrict to authors at the Sanger Centre. This can be very handy - especially if you are not so familiar with a particular database.

**ESearch: Searching the Entrez databases**

To search any of these databases, we use Bio.Entrez.esearch(). For example, let’s search in PubMed for publications related to Biopython:

>>> from Bio import Entrez

>>> Entrez.email = "A.N.Other@example.com" # Always tell NCBI who you are

>>> handle = Entrez.esearch(db="pubmed", term="biopython")

>>> record = Entrez.read(handle)

>>> record["IdList"]

['19304878', '18606172', '16403221', '16377612', '14871861', '14630660', '12230038']

In this output, you see seven PubMed IDs (including 19304878 which is the PMID for the Biopython application note), which can be retrieved by EFetch

You can also use ESearch to search GenBank. Here we’ll do a quick search for the *matK* gene in *Cypripedioideae* orchids:

>>> handle = Entrez.esearch(db="nucleotide", term="Cypripedioideae[Orgn] AND matK[Gene]")

>>> record = Entrez.read(handle)

>>> record["Count"]

'25'

>>> record["IdList"]

['126789333', '37222967', '37222966', '37222965', ..., '61585492']

Each of the IDs (126789333, 37222967, 37222966, …) is a GenBank identifier.

Note that instead of a species name like Cypripedioideae[Orgn], you can restrict the search using an NCBI taxon identifier, here this would be txid158330[Orgn]. This isn’t currently documented on the ESearch help page - the NCBI explained this in reply to an email query. You can often deduce the search term formatting by playing with the Entrez web interface. For example, including complete[prop] in a genome search restricts to just completed genomes.

As a final example, let’s get a list of computational journal titles:

>>> handle = Entrez.esearch(db="journals", term="computational")

>>> record = Entrez.read(handle)

>>> record["Count"]

'16'

>>> record["IdList"]

['30367', '33843', '33823', '32989', '33190', '33009', '31986',

'34502', '8799', '22857', '32675', '20258', '33859', '32534',

'32357', '32249']

Again, we could use EFetch to obtain more information for each of these journal IDs.

ESearch has many useful options — see the [ESearch help page](http://www.ncbi.nlm.nih.gov/entrez/query/static/esearch_help.html) for more information.

**9.4  EPost: Uploading a list of identifiers**

EPost uploads a list of UIs for use in subsequent search strategies; see the [EPost help page](http://www.ncbi.nlm.nih.gov/entrez/query/static/epost_help.html) for more information. It is available from Biopython through the Bio.Entrez.epost() function.

To give an example of when this is useful, suppose you have a long list of IDs you want to download using EFetch (maybe sequences, maybe citations – anything). When you make a request with EFetch your list of IDs, the database etc, are all turned into a long URL sent to the server. If your list of IDs is long, this URL gets long, and long URLs can break (e.g. some proxies don’t cope well).

Instead, you can break this up into two steps, first uploading the list of IDs using EPost (this uses an “HTML post” internally, rather than an “HTML get”, getting round the long URL problem). With the history support, you can then refer to this long list of IDs, and download the associated data with EFetch.

Let’s look at a simple example to see how EPost works – uploading some PubMed identifiers:

>>> from Bio import Entrez

>>> Entrez.email = "A.N.Other@example.com" # Always tell NCBI who you are

>>> id\_list = ["19304878", "18606172", "16403221", "16377612", "14871861", "14630660"]

>>> print(Entrez.epost("pubmed", id=",".join(id\_list)).read())

<?xml version="1.0"?>

<!DOCTYPE ePostResult PUBLIC "-//NLM//DTD ePostResult, 11 May 2002//EN"

"http://www.ncbi.nlm.nih.gov/entrez/query/DTD/ePost\_020511.dtd">

<ePostResult>

<QueryKey>1</QueryKey>

<WebEnv>NCID\_01\_206841095\_130.14.22.101\_9001\_1242061629</WebEnv>

</ePostResult>

The returned XML includes two important strings, QueryKey and WebEnv which together define your history session. You would extract these values for use with another Entrez call such as EFetch:

>>> from Bio import Entrez

>>> Entrez.email = "A.N.Other@example.com" # Always tell NCBI who you are

>>> id\_list = ["19304878", "18606172", "16403221", "16377612", "14871861", "14630660"]

>>> search\_results = Entrez.read(Entrez.epost("pubmed", id=",".join(id\_list)))

>>> webenv = search\_results["WebEnv"]

>>> query\_key = search\_results["QueryKey"]

**ESummary: Retrieving summaries from primary IDs**

ESummary retrieves document summaries from a list of primary IDs (see the [ESummary help page](http://www.ncbi.nlm.nih.gov/entrez/query/static/esummary_help.html) for more information). In Biopython, ESummary is available as Bio.Entrez.esummary(). Using the search result above, we can for example find out more about the journal with ID 30367:

>>> from Bio import Entrez

>>> Entrez.email = "A.N.Other@example.com" # Always tell NCBI who you are

>>> handle = Entrez.esummary(db="journals", id="30367")

>>> record = Entrez.read(handle)

>>> record[0]["Id"]

'30367'

>>> record[0]["Title"]

'Computational biology and chemistry'

>>> record[0]["Publisher"]

'Pergamon,'

**EFetch: Downloading full records from Entrez**

EFetch is what you use when you want to retrieve a full record from Entrez. This covers several possible databases, as described on the main [EFetch Help page](http://eutils.ncbi.nlm.nih.gov/entrez/query/static/efetch_help.html).

For most of their databases, the NCBI support several different file formats. Requesting a specific file format from Entrez using Bio.Entrez.efetch() requires specifying the rettype and/or retmode optional arguments. The different combinations are described for each database type on the pages linked to on [NCBI efetch webpage](http://www.ncbi.nlm.nih.gov/entrez/query/static/efetch_help.html) (e.g. [literature](http://eutils.ncbi.nlm.nih.gov/corehtml/query/static/efetchlit_help.html), [sequences](http://eutils.ncbi.nlm.nih.gov/corehtml/query/static/efetchseq_help.html) and [taxonomy](http://eutils.ncbi.nlm.nih.gov/corehtml/query/static/efetchtax_help.html)).

One common usage is downloading sequences in the FASTA or GenBank/GenPept plain text formats (which can then be parsed with Bio.SeqIO, From the *Cypripedioideae* example above, we can download GenBank record 186972394 using Bio.Entrez.efetch:

>>> from Bio import Entrez

>>> Entrez.email = "A.N.Other@example.com" # Always tell NCBI who you are

>>> handle = Entrez.efetch(db="nucleotide", id="186972394", rettype="gb", retmode="text")

>>> print(handle.read())

LOCUS EU490707 1302 bp DNA linear PLN 05-MAY-2008

DEFINITION Selenipedium aequinoctiale maturase K (matK) gene, partial cds;

chloroplast.

ACCESSION EU490707

VERSION EU490707.1 GI:186972394

KEYWORDS .

SOURCE chloroplast Selenipedium aequinoctiale

ORGANISM Selenipedium aequinoctiale

Eukaryota; Viridiplantae; Streptophyta; Embryophyta; Tracheophyta;

Spermatophyta; Magnoliophyta; Liliopsida; Asparagales; Orchidaceae;

Cypripedioideae; Selenipedium.

REFERENCE 1 (bases 1 to 1302)

AUTHORS Neubig,K.M., Whitten,W.M., Carlsward,B.S., Blanco,M.A.,

Endara,C.L., Williams,N.H. and Moore,M.J.

TITLE Phylogenetic utility of ycf1 in orchids

JOURNAL Unpublished

REFERENCE 2 (bases 1 to 1302)

AUTHORS Neubig,K.M., Whitten,W.M., Carlsward,B.S., Blanco,M.A.,

Endara,C.L., Williams,N.H. and Moore,M.J.

TITLE Direct Submission

JOURNAL Submitted (14-FEB-2008) Department of Botany, University of

Florida, 220 Bartram Hall, Gainesville, FL 32611-8526, USA

FEATURES Location/Qualifiers

source 1..1302

/organism="Selenipedium aequinoctiale"

/organelle="plastid:chloroplast"

/mol\_type="genomic DNA"

/specimen\_voucher="FLAS:Blanco 2475"

/db\_xref="taxon:256374"

gene <1..>1302

/gene="matK"

CDS <1..>1302

/gene="matK"

/codon\_start=1

/transl\_table=11

/product="maturase K"

/protein\_id="ACC99456.1"

/db\_xref="GI:186972395"

/translation="IFYEPVEIFGYDNKSSLVLVKRLITRMYQQNFLISSVNDSNQKG

FWGHKHFFSSHFSSQMVSEGFGVILEIPFSSQLVSSLEEKKIPKYQNLRSIHSIFPFL

EDKFLHLNYVSDLLIPHPIHLEILVQILQCRIKDVPSLHLLRLLFHEYHNLNSLITSK

KFIYAFSKRKKRFLWLLYNSYVYECEYLFQFLRKQSSYLRSTSSGVFLERTHLYVKIE

HLLVVCCNSFQRILCFLKDPFMHYVRYQGKAILASKGTLILMKKWKFHLVNFWQSYFH

FWSQPYRIHIKQLSNYSFSFLGYFSSVLENHLVVRNQMLENSFIINLLTKKFDTIAPV

ISLIGSLSKAQFCTVLGHPISKPIWTDFSDSDILDRFCRICRNLCRYHSGSSKKQVLY

RIKYILRLSCARTLARKHKSTVRTFMRRLGSGLLEEFFMEEE"

ORIGIN

1 attttttacg aacctgtgga aatttttggt tatgacaata aatctagttt agtacttgtg

61 aaacgtttaa ttactcgaat gtatcaacag aattttttga tttcttcggt taatgattct

121 aaccaaaaag gattttgggg gcacaagcat tttttttctt ctcatttttc ttctcaaatg

181 gtatcagaag gttttggagt cattctggaa attccattct cgtcgcaatt agtatcttct

241 cttgaagaaa aaaaaatacc aaaatatcag aatttacgat ctattcattc aatatttccc

301 tttttagaag acaaattttt acatttgaat tatgtgtcag atctactaat accccatccc

361 atccatctgg aaatcttggt tcaaatcctt caatgccgga tcaaggatgt tccttctttg

421 catttattgc gattgctttt ccacgaatat cataatttga atagtctcat tacttcaaag

481 aaattcattt acgccttttc aaaaagaaag aaaagattcc tttggttact atataattct

541 tatgtatatg aatgcgaata tctattccag tttcttcgta aacagtcttc ttatttacga

601 tcaacatctt ctggagtctt tcttgagcga acacatttat atgtaaaaat agaacatctt

661 ctagtagtgt gttgtaattc ttttcagagg atcctatgct ttctcaagga tcctttcatg

721 cattatgttc gatatcaagg aaaagcaatt ctggcttcaa agggaactct tattctgatg

781 aagaaatgga aatttcatct tgtgaatttt tggcaatctt attttcactt ttggtctcaa

841 ccgtatagga ttcatataaa gcaattatcc aactattcct tctcttttct ggggtatttt

901 tcaagtgtac tagaaaatca tttggtagta agaaatcaaa tgctagagaa ttcatttata

961 ataaatcttc tgactaagaa attcgatacc atagccccag ttatttctct tattggatca

1021 ttgtcgaaag ctcaattttg tactgtattg ggtcatccta ttagtaaacc gatctggacc

1081 gatttctcgg attctgatat tcttgatcga ttttgccgga tatgtagaaa tctttgtcgt

1141 tatcacagcg gatcctcaaa aaaacaggtt ttgtatcgta taaaatatat acttcgactt

1201 tcgtgtgcta gaactttggc acggaaacat aaaagtacag tacgcacttt tatgcgaaga

1261 ttaggttcgg gattattaga agaattcttt atggaagaag aa

//

The arguments rettype="gb" and retmode="text" let us download this record in the GenBank format.

Note that until Easter 2009, the Entrez EFetch API let you use “genbank” as the return type, however the NCBI now insist on using the official return types of “gb” or “gbwithparts” (or “gp” for proteins) as described on online. Also not that until Feb 2012, the Entrez EFetch API would default to returning plain text files, but now defaults to XML.

Alternatively, you could for example use rettype="fasta" to get the Fasta-format; see the [EFetch Sequences Help page](http://www.ncbi.nlm.nih.gov/entrez/query/static/efetchseq_help.html) for other options. Remember – the available formats depend on which database you are downloading from - see the main [EFetch Help page](http://eutils.ncbi.nlm.nih.gov/entrez/query/static/efetch_help.html).

If you fetch the record in one of the formats accepted by Bio.SeqIO), you could directly parse it into a SeqRecord:

>>> from Bio import Entrez, SeqIO

>>> handle = Entrez.efetch(db="nucleotide", id="186972394", rettype="gb", retmode="text")

>>> record = SeqIO.read(handle, "genbank")

>>> handle.close()

>>> print(record)

ID: EU490707.1

Name: EU490707

Description: Selenipedium aequinoctiale maturase K (matK) gene, partial cds; chloroplast.

Number of features: 3

...

Seq('ATTTTTTACGAACCTGTGGAAATTTTTGGTTATGACAATAAATCTAGTTTAGTA...GAA', IUPACAmbiguousDNA())

Note that a more typical use would be to save the sequence data to a local file, and *then* parse it with Bio.SeqIO. This can save you having to re-download the same file repeatedly while working on your script, and places less load on the NCBI’s servers. For example:

import os

from Bio import SeqIO

from Bio import Entrez

Entrez.email = "A.N.Other@example.com" # Always tell NCBI who you are

filename = "gi\_186972394.gbk"

if not os.path.isfile(filename):

# Downloading...

net\_handle = Entrez.efetch(db="nucleotide",id="186972394",rettype="gb", retmode="text")

out\_handle = open(filename, "w")

out\_handle.write(net\_handle.read())

out\_handle.close()

net\_handle.close()

print("Saved")

print("Parsing...")

record = SeqIO.read(filename, "genbank")

print(record)

To get the output in XML format, which you can parse using the Bio.Entrez.read() function, use retmode="xml":

>>> from Bio import Entrez

>>> handle = Entrez.efetch(db="nucleotide", id="186972394", retmode="xml")

>>> record = Entrez.read(handle)

>>> handle.close()

>>> record[0]["GBSeq\_definition"]

'Selenipedium aequinoctiale maturase K (matK) gene, partial cds; chloroplast'

>>> record[0]["GBSeq\_source"]

'chloroplast Selenipedium aequinoctiale'

**ELink: Searching for related items in NCBI Entrez**

ELink, available from Biopython as Bio.Entrez.elink(), can be used to find related items in the NCBI Entrez databases. For example, you can us this to find nucleotide entries for an entry in the gene database, and other cool stuff.

Let’s use ELink to find articles related to the Biopython application note published in *Bioinformatics* in 2009. The PubMed ID of this article is 19304878:

>>> from Bio import Entrez

>>> Entrez.email = "A.N.Other@example.com"

>>> pmid = "19304878"

>>> record = Entrez.read(Entrez.elink(dbfrom="pubmed", id=pmid))

The record variable consists of a Python list, one for each database in which we searched. Since we specified only one PubMed ID to search for, record contains only one item. This item is a dictionary containing information about our search term, as well as all the related items that were found:

>>> record[0]["DbFrom"]

'pubmed'

>>> record[0]["IdList"]

['19304878']

The "LinkSetDb" key contains the search results, stored as a list consisting of one item for each target database. In our search results, we only find hits in the PubMed database (although sub-divided into categories):

>>> len(record[0]["LinkSetDb"])

5

>>> for linksetdb in record[0]["LinkSetDb"]:

... print(linksetdb["DbTo"], linksetdb["LinkName"], len(linksetdb["Link"]))

...

pubmed pubmed\_pubmed 110

pubmed pubmed\_pubmed\_combined 6

pubmed pubmed\_pubmed\_five 6

pubmed pubmed\_pubmed\_reviews 5

pubmed pubmed\_pubmed\_reviews\_five 5

The actual search results are stored as under the "Link" key. In total, 110 items were found under standard search. Let’s now at the first search result:

>>> record[0]["LinkSetDb"][0]["Link"][0]

{u'Id': '19304878'}

This is the article we searched for, which doesn’t help us much, so let’s look at the second search result:

>>> record[0]["LinkSetDb"][0]["Link"][1]

{u'Id': '14630660'}

This paper, with PubMed ID 14630660, is about the Biopython PDB parser.

We can use a loop to print out all PubMed IDs:

>>> for link in record[0]["LinkSetDb"][0]["Link"]:

... print(link["Id"])

19304878

14630660

18689808

17121776

16377612

12368254

......

**Parsing huge Entrez XML files**

The Entrez.read function reads the entire XML file returned by Entrez into a single Python object, which is kept in memory. To parse Entrez XML files too large to fit in memory, you can use the function Entrez.parse. This is a generator function that reads records in the XML file one by one. This function is only useful if the XML file reflects a Python list object (in other words, if Entrez.read on a computer with infinite memory resources would return a Python list).

For example, you can download the entire Entrez Gene database for a given organism as a file from NCBI’s ftp site. These files can be very large. As an example, on September 4, 2009, the file Homo\_sapiens.ags.gz, containing the Entrez Gene database for human, had a size of 116576 kB. This file, which is in the ASN format, can be converted into an XML file using NCBI’s gene2xml program (see NCBI’s ftp site for more information):

gene2xml -b T -i Homo\_sapiens.ags -o Homo\_sapiens.xml

The resulting XML file has a size of 6.1 GB. Attempting Entrez.read on this file will result in a MemoryError on many computers.

The XML file Homo\_sapiens.xml consists of a list of Entrez gene records, each corresponding to one Entrez gene in human. Entrez.parse retrieves these gene records one by one. You can then print out or store the relevant information in each record by iterating over the records. For example, this script iterates over the Entrez gene records and prints out the gene numbers and names for all current genes:

>>> from Bio import Entrez

>>> handle = open("Homo\_sapiens.xml")

>>> records = Entrez.parse(handle)

>>> for record in records:

... status = record['Entrezgene\_track-info']['Gene-track']['Gene-track\_status']

... if status.attributes['value']=='discontinued':

... continue

... geneid = record['Entrezgene\_track-info']['Gene-track']['Gene-track\_geneid']

... genename = record['Entrezgene\_gene']['Gene-ref']['Gene-ref\_locus']

... print(geneid, genename)

This will print:

1 A1BG

2 A2M

3 A2MP

8 AA

9 NAT1

10 NAT2

11 AACP

12 SERPINA3

13 AADAC

14 AAMP

15 AANAT

16 AARS

17 AAVS1

...

**Parsing Medline records**

You can find the Medline parser in Bio.Medline. Suppose we want to parse the file pubmed\_result1.txt, containing one Medline record. You can find this file in Biopython’s Tests\Medline directory. The file looks like this:

PMID- 12230038

OWN - NLM

STAT- MEDLINE

DA - 20020916

DCOM- 20030606

LR - 20041117

PUBM- Print

IS - 1467-5463 (Print)

VI - 3

IP - 3

DP - 2002 Sep

TI - The Bio\* toolkits--a brief overview.

PG - 296-302

AB - Bioinformatics research is often difficult to do with commercial software. The

Open Source BioPerl, BioPython and Biojava projects provide toolkits with

...

We first open the file and then parse it:

>>> from Bio import Medline

>>> with open("pubmed\_result1.txt") as handle:

... record = Medline.read(handle)

...

The record now contains the Medline record as a Python dictionary:

>>> record["PMID"]

'12230038'

>>> record["AB"]

'Bioinformatics research is often difficult to do with commercial software.

The Open Source BioPerl, BioPython and Biojava projects provide toolkits with

multiple functionality that make it easier to create customised pipelines or

analysis. This review briefly compares the quirks of the underlying languages

and the functionality, documentation, utility and relative advantages of the

Bio counterparts, particularly from the point of view of the beginning

biologist programmer.'

The key names used in a Medline record can be rather obscure; use

>>> help(record)

for a brief summary.

To parse a file containing multiple Medline records, you can use the parse function instead:

>>> from Bio import Medline

>>> with open("pubmed\_result2.txt") as handle:

... for record in Medline.parse(handle):

... print(record["TI"])

...

A high level interface to SCOP and ASTRAL implemented in python.

GenomeDiagram: a python package for the visualization of large-scale genomic data.

Open source clustering software.

PDB file parser and structure class implemented in Python.

Instead of parsing Medline records stored in files, you can also parse Medline records downloaded by Bio.Entrez.efetch. For example, let’s look at all Medline records in PubMed related to Biopython:

>>> from Bio import Entrez

>>> Entrez.email = "A.N.Other@example.com" # Always tell NCBI who you are

>>> handle = Entrez.esearch(db="pubmed", term="biopython")

>>> record = Entrez.read(handle)

>>> record["IdList"]

['19304878', '18606172', '16403221', '16377612', '14871861', '14630660', '12230038']

We now use Bio.Entrez.efetch to download these Medline records:

>>> idlist = record["IdList"]

>>> handle = Entrez.efetch(db="pubmed", id=idlist, rettype="medline", retmode="text")

Here, we specify rettype="medline", retmode="text" to obtain the Medline records in plain-text Medline format. Now we use Bio.Medline to parse these records:

>>> from Bio import Medline

>>> records = Medline.parse(handle)

>>> for record in records:

... print(record["AU"])

['Cock PJ', 'Antao T', 'Chang JT', 'Chapman BA', 'Cox CJ', 'Dalke A', ..., 'de Hoon MJ']

['Munteanu CR', 'Gonzalez-Diaz H', 'Magalhaes AL']

['Casbon JA', 'Crooks GE', 'Saqi MA']

['Pritchard L', 'White JA', 'Birch PR', 'Toth IK']

['de Hoon MJ', 'Imoto S', 'Nolan J', 'Miyano S']

['Hamelryck T', 'Manderick B']

['Mangalam H']

For comparison, here we show an example using the XML format:

>>> idlist = record["IdList"]

>>> handle = Entrez.efetch(db="pubmed", id=idlist, rettype="medline", retmode="xml")

>>> records = Entrez.read(handle)

>>> for record in records:

... print(record["MedlineCitation"]["Article"]["ArticleTitle"])

Biopython: freely available Python tools for computational molecular biology and

bioinformatics.

Enzymes/non-enzymes classification model complexity based on composition, sequence,

3D and topological indices.

A high level interface to SCOP and ASTRAL implemented in python.

GenomeDiagram: a python package for the visualization of large-scale genomic data.

Open source clustering software.

PDB file parser and structure class implemented in Python.

The Bio\* toolkits--a brief overview.

Note that in both of these examples, for simplicity we have naively combined ESearch and EFetch. In this situation, the NCBI would expect you to use their history feature, as illustrated in Section [9.15](http://biopython.org/DIST/docs/tutorial/Tutorial.html#sec:entrez-webenv).

**Examples**

**PubMed and Medline**

If you are in the medical field or interested in human issues (and many times even if you are not!), PubMed (<http://www.ncbi.nlm.nih.gov/PubMed/>) is an excellent source of all kinds of goodies. So like other things, we’d like to be able to grab information from it and use it in Python scripts.

In this example, we will query PubMed for all articles having to do with orchids. We first check how many of such articles there are:

>>> from Bio import Entrez

>>> Entrez.email = "A.N.Other@example.com" # Always tell NCBI who you are

>>> handle = Entrez.egquery(term="orchid")

>>> record = Entrez.read(handle)

>>> for row in record["eGQueryResult"]:

... if row["DbName"]=="pubmed":

... print(row["Count"])

463

Now we use the Bio.Entrez.efetch function to download the PubMed IDs of these 463 articles:

>>> handle = Entrez.esearch(db="pubmed", term="orchid", retmax=463)

>>> record = Entrez.read(handle)

>>> idlist = record["IdList"]

>>> print(idlist)

This returns a Python list containing all of the PubMed IDs of articles related to orchids:

['18680603', '18665331', '18661158', '18627489', '18627452', '18612381',

'18594007', '18591784', '18589523', '18579475', '18575811', '18575690',

...

Now that we’ve got them, we obviously want to get the corresponding Medline records and extract the information from them. Here, we’ll download the Medline records in the Medline flat-file format, and use the Bio.Medline module to parse them:

>>> from Bio import Medline

>>> handle = Entrez.efetch(db="pubmed", id=idlist, rettype="medline",

retmode="text")

>>> records = Medline.parse(handle)

NOTE - We’ve just done a separate search and fetch here, the NCBI much prefer you to take advantage of their history support in this situation.

Keep in mind that records is an iterator, so you can iterate through the records only once. If you want to save the records, you can convert them to a list:

>>> records = list(records)

Let’s now iterate over the records to print out some information about each record:

>>> for record in records:

... print("title:", record.get("TI", "?"))

... print("authors:", record.get("AU", "?"))

... print("source:", record.get("SO", "?"))

... print("")

The output for this looks like:

title: Sex pheromone mimicry in the early spider orchid (ophrys sphegodes):

patterns of hydrocarbons as the key mechanism for pollination by sexual

deception [In Process Citation]

authors: ['Schiestl FP', 'Ayasse M', 'Paulus HF', 'Lofstedt C', 'Hansson BS',

'Ibarra F', 'Francke W']

source: J Comp Physiol [A] 2000 Jun;186(6):567-74

Especially interesting to note is the list of authors, which is returned as a standard Python list. This makes it easy to manipulate and search using standard Python tools. For instance, we could loop through a whole bunch of entries searching for a particular author with code like the following:

>>> search\_author = "Waits T"

>>> for record in records:

... if not "AU" in record:

... continue

... if search\_author in record["AU"]:

... print("Author %s found: %s" % (search\_author, record["SO"]))

Hopefully this section gave you an idea of the power and flexibility of the Entrez and Medline interfaces and how they can be used together.

**Searching, downloading, and parsing Entrez Nucleotide records**

Here we’ll show a simple example of performing a remote Entrez query. We talked about using NCBI’s Entrez website to search the NCBI nucleotide databases for info on Cypripedioideae, our friends the lady slipper orchids. Now, we’ll look at how to automate that process using a Python script. In this example, we’ll just show how to connect, get the results, and parse them, with the Entrez module doing all of the work.

First, we use EGQuery to find out the number of results we will get before actually downloading them. EGQuery will tell us how many search results were found in each of the databases, but for this example we are only interested in nucleotides:

>>> from Bio import Entrez

>>> Entrez.email = "A.N.Other@example.com" # Always tell NCBI who you are

>>> handle = Entrez.egquery(term="Cypripedioideae")

>>> record = Entrez.read(handle)

>>> for row in record["eGQueryResult"]:

... if row["DbName"]=="nuccore":

... print(row["Count"])

814

So, we expect to find 814 Entrez Nucleotide records (this is the number I obtained in 2008; it is likely to increase in the future). If you find some ridiculously high number of hits, you may want to reconsider if you really want to download all of them, which is our next step:

>>> from Bio import Entrez

>>> handle = Entrez.esearch(db="nucleotide", term="Cypripedioideae", retmax=814)

>>> record = Entrez.read(handle)

Here, record is a Python dictionary containing the search results and some auxiliary information. Just for information, let’s look at what is stored in this dictionary:

>>> print(record.keys())

[u'Count', u'RetMax', u'IdList', u'TranslationSet', u'RetStart', u'QueryTranslation']

First, let’s check how many results were found:

>>> print(record["Count"])

'814'

which is the number we expected. The 814 results are stored in record['IdList']:

>>> len(record["IdList"])

814

Let’s look at the first five results:

>>> record["IdList"][:5]

['187237168', '187372713', '187372690', '187372688', '187372686']

We can download these records using efetch. While you could download these records one by one, to reduce the load on NCBI’s servers, it is better to fetch a bunch of records at the same time, shown below.

>>> idlist = ",".join(record["IdList"][:5])

>>> print(idlist)

187237168,187372713,187372690,187372688,187372686

>>> handle = Entrez.efetch(db="nucleotide", id=idlist, retmode="xml")

>>> records = Entrez.read(handle)

>>> len(records)

5

Each of these records corresponds to one GenBank record.

>>> print(records[0].keys())

[u'GBSeq\_moltype', u'GBSeq\_source', u'GBSeq\_sequence',

u'GBSeq\_primary-accession', u'GBSeq\_definition', u'GBSeq\_accession-version',

u'GBSeq\_topology', u'GBSeq\_length', u'GBSeq\_feature-table',

u'GBSeq\_create-date', u'GBSeq\_other-seqids', u'GBSeq\_division',

u'GBSeq\_taxonomy', u'GBSeq\_references', u'GBSeq\_update-date',

u'GBSeq\_organism', u'GBSeq\_locus', u'GBSeq\_strandedness']

>>> print(records[0]["GBSeq\_primary-accession"])

DQ110336

>>> print(records[0]["GBSeq\_other-seqids"])

['gb|DQ110336.1|', 'gi|187237168']

>>> print(records[0]["GBSeq\_definition"])

Cypripedium calceolus voucher Davis 03-03 A maturase (matR) gene, partial cds;

mitochondrial

>>> print(records[0]["GBSeq\_organism"])

Cypripedium calceolus