ENTERZ DIRECT - USER GUIDE

The Entrez Utilities (EUtils) are URL-based methods for programmatically accessing the NCBI's Entrez data. They are easy to incorporate into Perl or UNIX shell scripts, though there is a non-trivial learning curve and an expectation of computer programming experience. A sample EUtils URL for an ESearch query is:

```
http://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi?db=pubmed&term=insulin
```

Entrez Direct (EDirect) constructs and executes an EUtils service request from commands typed into a terminal window. It takes search terms from command-line arguments and combines individual operations into multi-step queries by using UNIX pipes. The design of EDirect was influenced by the NCBI's EUtils Power Scripting course and EBot service, but it is meant to be easier to use and it has a wider target audience. A laboratory biologist can use EDirect to build a complex Entrez query without having to write a program, and without needing any formal training in bioinformatics.

INTRODUCTION

An EDirect query starts with the name of the function to perform (e.g., esearch, esummary) followed by command-line arguments specific to the operation. For example, in the query:

```
efetch -db nucleotide -id U54469.1 -format fasta
```

the "efetch" function is used for fetching records or reports in various formats. For efetch operations, the -format argument is required. The value of "fasta" tells the program to download sequence records in FASTA format, which is suitable for input into many sequence analysis programs. The -db argument specifies the database (e.g., pubmed, nucleotide, protein, structure, etc.), and the -id argument supplies a sequence accession number or the record's "unique identifier" (UID) tracking number (e.g., PMID for PubMed articles or GI number for sequences). Multiple accessions or identifiers can be supplied, separated by commas.

EDirect commands are actually wrapper scripts that send arguments to the "edirect.pl" program. The scripts must all be kept in the same directory. If that directory is not in the user's "PATH", EDirect commands can be run from within that location by prefixing the program name with the "./" UNIX current directory path symbol. Or the PATH environment variable can be temporarily adjusted to add the edirect directory by starting each terminal session with:

```
PATH=$PATH:$HOME/edirect
```

ENTREZ DIRECT FUNCTION SUMMARY

EDirect operations can be grouped into several categories. Navigation functions support exploration within the Entrez interconnected databases:

esearch	Performs a new Entrez search using terms in indexed fields.
elink	Looks up neighbors (within a database) or links (between databases).
efilter	Filters or restricts the results of a previous query.

The desired product of many searches are document summaries or downloaded records:

esummary	Obtains document summaries for the query results in XML format.
efetch	Retrieves records or generates reports in a designated format.

Information about available Entrez databases, and search fields and link types within each database, can be retrieved in XML format:

```
  einfo       Obtains information on indexed fields in an Entrez database.
```

Lists of UIDs or sequence accession numbers can be uploaded by piping them into:

```
  epost -format uid
  epost -format acc
```

or downloaded by piping the previous query into:

```
  efetch -format uid
  efetch -format acc
```

(The database must be supplied with a -db command-line argument or piped in from the previous query step.)

Desired fields from XML results can be extracted without writing a program:

```
  xtract      Converts XML into a table of data values.
```

External web resources can also be queried with command-line arguments:

```
  nquire      Sends a URL request to a web page or CGI service.
```

ENTERING MULTIPLE-STEP QUERIES

EDirect gains flexibility by allowing individual operations to be described separately, combining them into a multi-step query by using the vertical bar ("|") UNIX pipe symbol. Individual steps can be placed on separate lines by using the backslash ("\") UNIX line continuation character. This makes query development and editing easier, allowing common query patterns to be saved in a text editor document and then copied and pasted into a terminal for execution. A sample EDirect query starting with unqualified search terms:

```
  esearch -db pubmed -query "Casadaban AND transposition immunity Tn3" | \
  elink -related | \
  elink -target protein | \
  efilter -query "NOT (bacteria [ORGN] OR fungi [ORGN])" | \
  efetch -format fasta | grep '.'
```

searches by author name and indexed text words to retrieve a particular paper, finds the related articles (pubmed neighbors), links to all protein sequences published in those articles, restricts the results to organisms other than bacteria or fungi (in this case it hits a few plasmids), and downloads the FASTA sequence reports of those records for further analysis:

```
  >gi|151827|gb|AAA26080.1| recombinase [Plasmid R46]
  MRLFGYARVSTSQQSLDIQIKGLKEAGVKASRIFTDKASGSSTDRKGLDLLRMKVEEGDVTLVKKLDRLG
  RDTADMIQLTKEFDAQGVAVRFIDDGISTDGEMGKMVVTILSAVAQAERRRILERTNEGRQEAKLKGIRF
  GRKRIIDRNSVLALHQQGTGATDIARRLSIARSTVYKILEDESRVNLSKI
  >gi|152542|gb|AAA26415.1| parB [Plasmid RP4]
  MPARLGPNGYTQFDQGGRDEGGQALPTTSIHRLQHRGAGLRAWTRRHTRPAHPRPGARGPRPARPGRVPF
  ...
```

(The UNIX "grep" command at the end removes blank lines between FASTA records.)

QUALIFYING QUERIES BY INDEXED FIELD

Search terms can be restricted to a specific indexed field by placing the field abbreviation inside brackets. Commonly-used fields for pubmed queries include:

[AFFL]	Affiliation	[FILT]	Filter	[MESH]	MeSH Terms
[ALL]	All Fields	[JOUR]	Journal	[PTYP]	Publication Type
[AUTH]	Author	[LANG]	Language	[WORD]	Text Word
[FAUT]	Author - First	[MAJR]	MeSH Major Topic	[TITL]	Title
[LAUT]	Author - Last	[SUBH]	MeSH Subheading	[TIAB]	Title/Abstract
[PDAT]	Date - Publication			[UID]	UID

Some examples of filters that limit search results to subsets of pubmed are:

humans [MESH]	historical article [FILT]
pharmacokinetics [MESH]	loprovflybase [FILT]
chemically induced [SUBH]	randomized controlled trial [FILT]
all child [FILT]	clinical trial, phase ii [PTYP]
free full text [FILT]	review [PTYP]

Sequence databases are indexed with a different set of search fields, including:

[ACCN]	Accession	[GENE]	Gene Name	[PROT]	Protein Name
[ALL]	All Fields	[JOUR]	Journal	[SQID]	SeqID String
[AUTH]	Author	[KYWD]	Keyword	[SLEN]	Sequence Length
[GPRJ]	BioProject	[MLWT]	Molecular Weight	[SUBS]	Substance Name
[ECNO]	EC/RN Number	[ORGN]	Organism	[WORD]	Text Word
[FKEY]	Feature Key	[PACC]	Primary Accession	[TITL]	Title
[FILT]	Filter	[PROP]	Properties	[UID]	UID

Specifying the field can also be useful in the sequence databases, both in the initial query and for filtering the results. Examples are:

mammalia [ORGN]	dbxref flybase [PROP]
mammalia [ORGN:noexp]	gbdiv phg [PROP]
cds [FKEY]	src cultivar [PROP]
lacz [GENE]	srcdb refseq validated [PROP]
beta galactosidase [PROT]	150:200 [SLEN]
protein snp [FILT]	2000:4000 [MLWT]
biomol genomic [PROP]	

(The calculated molecular weight (MLWT) field is only indexed for proteins (and structures), not nucleotides.)

Results can also be filtered by time with the -days argument. For example, use "-days 60 -datetype PDAT" to restrict results to the previous two months.

CONVERSION OF XML DATA INTO TABULAR FORM

EDirect results are frequently returned in XML format. The xtract function simplifies interpretation of the data by selectively converting XML values into a tab-delimited table. (Xtract is described in detail later in this document.)

Unless otherwise directed, the -pattern argument divides the results into rows, while placement of data into columns is controlled by -element. To see the full list of indexed pubmed field abbreviations and names, for example, run:

```
einfo -db pubmed | xtract -pattern Field -element Name FullName
```

EXAMINING INTERMEDIATE RESULT COUNTS

EDirect uses the Entrez history mechanism to store intermediate results on the EUtils server, and passes a custom XML message with the relevant fields (database, web environment, query key, record count, step number, and optional error string, e-mail address, and software tool name) to the next command in the pipeline.

The results of each step in a query can be examined to confirm expected behavior before adding the next step. The Count field in the ENTREZ_DIRECT XML object contains the number of records returned by the previous step. A good measure of query success is a reasonable (non-zero) count value. For example:

```
  esearch -db protein -query "NP_567004" | \
  elink -related | \
  efilter -query "protein structure [FILT]" | \
  efilter -query "28000:30000 [MLWT]" | \
  elink -target structure | \
  efilter -query "0:2 [RESO]"
```

produces:

```
  <ENTREZ_DIRECT>
    <Db>structure</Db>
    <WebEnv>NCID_1_545606712_172.16.22.25_5555_1348089299_358182861</WebEnv>
    <QueryKey>8</QueryKey>
    <Count>39</Count>
    <Step>6</Step>
  </ENTREZ_DIRECT>
```

with 39 protein structures being within the specified molecular weight range and having the desired (X-ray crystallographic) atomic position resolution.

(The QueryKey value is 8 instead of 6 because the elink function obtains the record count by running an ESearch query after the ELink operation.)

The count can be printed by piping the query through:

```
  xtract -pattern ENTREZ_DIRECT -element Count
```

COMBINING INDEPENDENT QUERY RESULTS

Independent esearch, elink, and efilter operations can be performed and then combined at the end by using the history server's "#" convention to indicate query key numbers. Subsequent esearch functions can take a -db argument to override the database piped in from the previous step. (Piping the queries together is necessary for sharing the same web environment.) For example:

```
  esearch -db protein -query "amyloid* [PROT]" | \
  elink -target pubmed | \
  esearch -db gene -query "apo* [GENE]" | \
  elink -target pubmed | \
  esearch -query "(#3) AND (#6)" | \
  esummary | \
  xtract -pattern DocumentSummary -element Id Title
```

uses truncation searching to return titles of papers with links to amyloid protein sequence and apolipoprotein gene records:

```
  23071163    Stronger effect of amyloid load than APOE genotype on cognitive ...
  22935789    Down's syndrome and Alzheimer's disease: towards secondary ...
  22560297    Congenital asplenia in mice and humans with mutations in a Pbx/ ...
  ...
```

The use of (#3) AND (#6) instead of (#2) AND (#4) above reflects the need for each elink command to execute a separate ESearch query, which increments the QueryKey, in order to obtain the record count. The -label argument can be used to get around this artifact. The label value is prefixed by a "#" symbol and placed in parentheses in the final search. Thus:

```
  esearch -db structure -query "insulin [TITL]" | \
  elink -target pubmed -label struc_cit | \
  esearch -db protein -query "insulin [PROT]" | \
  elink -target pubmed -label prot_cit | \
  esearch -query "(#struc_cit) AND (#prot_cit)" | \
  efetch -format url
```

will return:

```
  http://www.ncbi.nlm.nih.gov/pubmed/15299880
  http://www.ncbi.nlm.nih.gov/pubmed/9235985
  http://www.ncbi.nlm.nih.gov/pubmed/9141131
  ...
```

without the need to keep track of the internal QueryKey values.

INTERACTION WITH UNIX UTILITIES

Queries can move seamlessly between EDirect commands and UNIX utilities to perform actions that cannot be accomplished entirely within the Entrez environment.

For example:

```
  esearch -db protein -query "NP_567004" | \
  elink -related | \
  efilter -query "protein structure [FILT]" | \
  esummary | \
  xtract -pattern DocumentSummary -element AccessionVersion Slen
```

will return a table of sequence accession numbers and sequence lengths:

```
  2EKC_B            262
  1KFB_A            268
  YP_002809892.1    268
  YP_004704.1       271
  AFW60211.1        347
  NP_460686.1       268
  Q8U094.1          248
  P00929.1          268
  C3LLL5.1          268
  1WXJ_A            271
  NP_001105219.1    347
  ACP05441.1        268
  CAD37460.1        346
  ...
```

Accessions from the PDB database of 3-dimensional structures begin with a digit (e.g., 1DOI_A), and can be excluded by piping through a UNIX grep command:

```
grep -v '^[0123456789]'
```

Additional commands:

```
sort -t $'\t' -k 2,2nr | \
head -n 10 | \
cut -f 1 - | \
epost -db protein -format acc | \
efetch -format fasta | \
grep '.'
```

will sort by sequence length, take the 10 longest sequences, cut away the length column, and upload the resulting accessions for retrieval in FASTA format.

SENDING RESULTS TO SCRIPTS OR SPREADSHEETS

Specific fields can be extracted from esummary or efetch XML results and passed to other programs or scripts for further computational analysis. For example:

```
esearch -db nuccore -query "U49897" | \
elink -target gene | \
elink -target homologene | \
elink -target gene | \
esummary | \
xtract -pattern DocumentSummary -element Id \
  -block GenomicInfo -element ChrAccVer ChrStart ChrStop
```

obtains a series of homologous genes and writes a tab-delimited data stream:

```
1276381    NT_078265.2    19282703    19277264
741097     NC_006479.3    103242093   103150284
698696     NC_007868.1    104081543   104008954
...
```

(The use of -block limits the subsequent -element search to fields within the GenomicInfo block, and prevents it from capturing an additional ChrStart item that occurs elsewhere in the DocumentSummary XML.)

Given a shell script named "upstream.sh":

```
#!/bin/bash -norc

bases=1500
if [ -n "$1" ]; then
  bases=$1
fi

while read id accn start stop; do
  if [[ $start -eq 0 || $stop -eq 0 || $start -eq $stop ]]; then
    echo "Skipping $id due to ambiguous coordinates"
    continue
  fi
  if [ $start -gt $stop ]; then
    stop=$(( $start + $bases ))
    start=$(( $start + 1 ))
    strand=2
```

```
    else
      stop=$(( $start - 1 ))
      start=$(( $start - $bases ))
      strand=1
    fi
    rslt=`efetch -db nuccore -id $accn -format fasta \
          -seq_start $start -seq_stop $stop -strand $strand < /dev/null`
    echo "$rslt"
  done
```

the data lines can be piped through:

```
  upstream.sh 500
```

to extract and print the 500 nucleotides immediately upstream of each gene. (Without the number after the script name it will default to 1500 nucleotides.)

(The use of the "< /dev/null" input redirection construct prevents the efetch command from "draining" the remaining lines from stdin.)

Tab-delimited data can also be saved directly to a file with the right angle bracket (">") UNIX output redirection character. The resulting file is suitable for importing into a database or spreadsheet program.

ADVANCED ELINK COMMANDS

Elink has several command modes, and these can be specified with the -cmd argument. When not using the default "neighbor_history" command, elink will return an eLinkResult XML object, with the links for each UID presented in separate blocks.

For example:

```
  esearch -db pubmed -query "Garber ED [AUTH]" | \
  elink -related -cmd neighbor | \
  xtract -pattern LinkSetDb -element Id
```

will show the original PMID in the first column and related article PMIDs in subsequent columns:

```
  12236055    10415498    7337690     10568785    12796294    16339722    ...
  10777446    9045760     19226747    16674592    12207223    15887033    ...
  10415498    1358468     10786938    1366759     1979162     9586052     ...
  ...
```

When the elink command "prlinks" is used with "ref" mode, it can obtain HTML containing or referencing full text articles directly from the publishers. The UNIX "xargs" command calls elink separately for each identifier:

```
  epost -db pubmed -id 22966225,19880848 | \
  efilter -query "free full text [FILT]" | \
  efetch -format uid | \
  xargs -n 1 elink -db pubmed -cmd prlinks -mode ref -http get -id
```

The elink -batch flag is available to bypass the EUtils server history mechanism for large queries.

STORING COMMON PHRASES IN ALIAS FILES

Frequently used, long, or complicated search phrases can be saved in a file to avoid having to retype (or copy and paste) the full text for each query. Each line of the file has a shortcut keyword, a tab character, and the expanded search term. The shortcut is used with the "#" convention in esearch or efilter queries. For example, given a file named "query_aliases" containing:

```
jour_filt    [MULT] AND ncbijournals [FILT]
trans_imm    (transposition OR target) immunity
```

the esearch line in:

```
esearch -alias query_aliases -db nlmcatalog -query "Science (#jour_filt)" | \
esummary | \
xtract -pattern DocumentSummary -element ISOAbbreviation \
  -subset ISSNInfo -sep "|" -element issn,issntype | \
column -s $'\t' -t
```

will be expanded to:

```
esearch -db nlmcatalog -query "Science [MULT] AND ncbijournals [FILT]"
```

with the query producing:

```
J. Zhejiang Univ. Sci.  1009-3095|Print  1009-3095|Linking
Science (80- )          0193-4511|Print  0193-4511|Linking
Science                 0036-8075|Print  1095-9203|Electronic  ...
```

EXTRACTION OF DATA FROM XML RESULTS

The EDirect xtract function is driven by command-line arguments to provide flexible extraction of information from esummary, efetch, and einfo XML results without the need for writing a program. The function first reads its entire input as a single line and removes unnecessary white space. This allows Perl regular expression pattern matching to be used reliably on XML data, and helps eliminate the need for specifying hierarchical paths within the XML.

An XML input stream is split into separate output lines by the -pattern argument. Within each occurrence of the pattern, specific field names containing the desired data are indicated with the -element argument, and chosen values are normally displayed in columns separated by tab characters. These commands are comparable to "for each" loops in programming languages in that they automatically explore each occurrence of a specified XML tag in the data.

Additional xtract arguments limit exploration to specified XML regions (-group, -block, and -subset), filter by data content (-match and -avoid) or elements (-present and -absent), and customize the report presentation. These allow a simple list of arguments to control sophisticated data extraction. Advanced features of xtract will be discussed in greater detail in the next section.

A query for publications that cite specific PubMed records:

```
esearch -db pubmed -query "Beadle GW [AUTH]" | \
elink -related -name pubmed_pubmed_citedin | \
esummary
```

will generate a raw XML result:

```
  <eSummaryResult>
    <DocumentSummarySet status="OK">
      <DocumentSummary uid="23055940">
        <Id>23055940</Id>
        <PubDate>2012 Oct</PubDate>
        <EPubDate>2012 Oct 4</EPubDate>
        <Source>PLoS Genet</Source>
        <Authors>
          <Author>
            <Name>Sun Y</Name>
            <AuthType>Author</AuthType>
            <ClusterID>0</ClusterID>
          </Author>
          <Author>
            <Name>Ambrose JH</Name>
            <AuthType>Author</AuthType>
            <ClusterID>0</ClusterID>
          </Author>
          ...
```

that can be piped through:

```
  xtract -pattern Author -element Name
```

to produce a series of compressed author blocks:

```
  <Author><Name>Sun Y</Name><AuthType> ... </ClusterID></Author>
  <Author><Name>Ambrose JH</Name><AuthType> ... </ClusterID></Author>
  <Author><Name>Haughey BS</Name><AuthType> ... </ClusterID></Author>
  ...
```

that are then processed to extract the author name values:

```
  Sun Y
  Ambrose JH
  ...
```

(The pubmed_pubmed_citedin link is available only from publications with full text deposited in PubMed Central.)

This use of command-line arguments that alter the order of data exploration, and extract information from within the current scope, is in contrast to other query methods that require explicit paths to specific data elements, such as:

```
  /eSummaryResult/DocumentSummarySet/DocumentSummary[1]/Authors/Author[1]/Name
```

to select the first author name from the first publication.

The author list can be analyzed by piping the result through a series of standard UNIX commands. Continuing the previous example, author names are first sorted alphabetically, duplicates are removed, and each unique name is prefixed by an occurrence count. Perl regular expression pattern matching and text substitution then separate the count and name columns with a tab character:

```
  sort | uniq -c | \
  perl -pe 's/\s*(\d+)\s(.+)/$1\t$2/'
```

(The perl substitution form is "s/pattern/replacement/". Groups in parentheses are "captured" if the pattern matches, and are referenced by position with the $1, $2, etc., variables when replacing the string.)

Using the tab to distinguish columns, the list is then sorted by count in reverse numerical order, and lines with the same count are sorted alphabetically by author name (case-insensitive, so that "de la" and "van der" prefixed names are not moved to the end). Finally, the columns are aligned for cleaner printing:

```
sort -t $'\t' -k 1,1rn -k 2,2f | \
column -s $'\t' -t
```

and the end result is a list of authors ranked by frequency:

```
13  Beadle GW
8   Ephrussi B
7   Glass NL
7   Mitchell MB
7   Tatum EL
...
```

The full set of author names per article can be shown by visiting each DocumentSummary object and extracting the Id and Name values:

```
esearch -db pubmed -query "Beadle GW [AUTH]" | \
elink -related -name pubmed_pubmed_citedin | \
esummary | \
xtract -pattern DocumentSummary -element Id Name | \
column -s $'\t' -t
```

resulting in:

```
23055940  Sun Y            Ambrose JH         Haughey BS    ...
22912788  Jia Y            Jia MH             Wang X        ...
22822426  Emmert-Streib F  de Matos Simoes R  Tripathi S    ...
...
```

(Using -first or -last with Name, instead of using -element, would write only the first or last author name for each publication.)

XML with inconsistent line-wrapping can be reformatted for easier visual inspection of the data structure and content by piping it through:

```
xmllint --format -
```

ADVANCED XML DATA EXTRACTION

The xtract function has arguments that allow it to perform sophisticated data transformation that would otherwise require writing a script. By default, reported elements in a row are separated by a tab character, but this behavior can be customized. For example:

```
esearch -db gene -query "deuteranopia" | \
efetch -format xml | \
xtract -pattern Entrezgene \
  -element Gene-track_geneid Gene-ref_locus \
  -sep "|" -element Gene-ref_syn_E
```

combines all synonyms for a gene into a single column, separated by vertical bars:

```
2652    OPN1MW    CBD|GCP|GOP|CBBM|COD5|OPN1MW1
5956    OPN1LW    CBP|RCP|ROP|CBBM|COD5
```

(The "-sep" value separates multiple elements of the same type. It also separates unrelated elements that are grouped with commas, as shown in the next example. Otherwise the "-tab" value separates individual fields.)

A -block argument will visit every instance of the XML block for sequential processing. Individual fields can be grouped with commas and given specific prefix, suffix, and separator characters. The normal tab and return characters used to separate fields and lines can also be overridden.

For example:

```
efetch -db pubmed -id 11449725 -format xml | \
xtract -pattern PubmedArticle -tab "\n" -element "MedlineCitation/PMID" \
  -block MeshHeading -pfx "[" -sfx "]" -sep "|" -tab "\n" \
  -element DescriptorName@MajorTopicYN,DescriptorName
```

will visit each MeshHeading object in the data:

```
...
<MeshHeadingList>
  <MeshHeading>
    <DescriptorName MajorTopicYN="N">Animals</DescriptorName>
  </MeshHeading>
  <MeshHeading>
    <DescriptorName MajorTopicYN="N">Computational Biology</DescriptorName>
  </MeshHeading>
  <MeshHeading>
    <DescriptorName MajorTopicYN="N">Computer Simulation</DescriptorName>
  </MeshHeading>
  <MeshHeading>
    <DescriptorName MajorTopicYN="Y">Databases, Factual</DescriptorName>
  </MeshHeading>
  ...
```

and extract Medical Subject Headings ("MeSH" terms) and their associated major topic attributes:

```
11449725
[N|Animals]
[N|Computational Biology]
[N|Computer Simulation]
[Y|Databases, Factual]
[N|Humans]
...
```

into a form that can be further transformed by piping through a series of UNIX "sed" (stream editor) commands:

```
sed -e 's/\[N|/\[/g' -e 's/\[Y|/\[*/g' -e 's/\[//g' -e 's/\]//g'
```

to display MeSH terms where an asterisk indicates a major topic ("starred" term):

```
11449725
Animals
```

```
  Computational Biology
  Computer Simulation
  *Databases, Factual
  Humans
  ...
```

(In this example the tab separator between fields was replaced by a return (newline) "\n" character. The tab character itself is represented by "\t".)

(The sed pattern replacement nomenclature is similar to perl. The "g" at the end of the substitution instruction indicates "global" replacement. Multiple sequential editing steps can be performed by using several "-e" arguments.)

Without the -block argument, the MeSH major topic attributes would all be visited before any of the MeSH term values were seen:

```
  [N|N|N|Y|N|N|Y|N|N|N|Animals|Computational Biology| ... ]
```

Within an expanded block, -subset can be used for internal (nested) exploration. A little experimentation with prefix, suffix, separator, and tab replacement can coerce the output to group data elements properly. For example:

```
  efetch -db pubmed -id 1937004 -format xml | \
  xtract -pattern PubmedArticle -tab "" -element "MedlineCitation/PMID" \
    -block MeshHeading -pfx "\n[" -sfx "]" -sep "|" -tab "" \
    -element DescriptorName@MajorTopicYN,DescriptorName \
      -subset QualifierName -pfx "{" -sfx "}" -sep "|" -tab "" \
      -element "@MajorTopicYN,QualifierName"
```

will process multiple QualifierName elements within a MeshHeading:

```
  ...
  <MeshHeading>
    <DescriptorName MajorTopicYN="N">Saccharomyces cerevisiae</DescriptorName>
    <QualifierName MajorTopicYN="Y">genetics</QualifierName>
    <QualifierName MajorTopicYN="N">radiation effects</QualifierName>
  </MeshHeading>
  ...
```

to produce an intermediate form:

```
  ...
  [N|Saccharomyces cerevisiae]{Y|genetics}{N|radiation effects}
  ...
```

that keeps the major topic indicators correctly associated with each item. The temporary punctuation symbols can then be processed by:

```
  sed -e 's/\[N|/\[/g' -e 's/\[Y|/\[*/g' -e 's/{N|/{/g' -e 's/{Y|/{*/g' | \
  sed -e 's/\]{/\]\/{/g' -e 's/}{/}\/{/g' | \
  sed -e 's/\[//g' -e 's/\]//g' -e 's/{//g' -e 's/}//g'
```

to generate a traditional presentation of MeSH terms and subheadings:

```
  1937004
  Adenosine Triphosphatases
  Amino Acid Sequence
  Base Sequence
  Binding Sites/genetics
```

```
  DNA Repair/*genetics
  DNA Repair Enzymes
  Fungal Proteins/*genetics
  Genes, Fungal/*genetics
  Meiosis/genetics
  Molecular Sequence Data
  Open Reading Frames/genetics
  Recombination, Genetic/genetics
  Saccharomyces cerevisiae/*genetics/radiation effects
  *Saccharomyces cerevisiae Proteins
```

Multiple -block arguments can be used in a single xtract command to obtain data from distinct regions of the XML. For example:

```
  efetch -db pubmed -id 781293,2678811,6301692,8332518 -format xml | \
  xtract -pattern PubmedArticle -element "MedlineCitation/PMID" \
    -block AuthorList -sep "|" -element LastName "#Author" \
    -block PubDate -sep " " -element Year,Month MedlineDate \
    -block DateCreated -sep "-" -element Year,Month,Day | \
  sort -t $'\t' -k 3,3n | column -s $'\t' -t
```

produces a table that allows easy parsing of author names, counts the number of authors present, and prints the date each record was published and the date it was entered into PubMed, sorting the results by the computed author count:

```
  781293   Casadaban                          1  1976 Jul      1976-10-02
  6301692  Krasnow|Cozzarelli                 2  1983 Apr      1983-06-17
  8332518  Benson|Lipman|Ostell               3  1993 Jul      1993-08-17
  2678811  Mortimer|Schild|Contopoulou|Kans   4  1989 Sep-Oct  1989-11-22
```

(Note that the PubDate object can exist either in a structured form:

```
  <PubDate>
    <Year>1976</Year>
    <Month>Jul</Month>
    <Day>5</Day>
  </PubDate>
```

(with the Day field frequently absent), or in a string form:

```
  <PubDate>
    <MedlineDate>1989 Sep-Oct</MedlineDate>
  </PubDate>
```

but would not contain a mixture of both types, so the directive:

```
  -element Year,Month MedlineDate
```

will only contribute a single column to the output.)

The -match argument can be used to restrict by content. For example:

```
  esearch -db nuccore -query "src country [PROP] AND src lat_lon [PROP]" | \
  efilter -query "(Ustilago [ORGN] OR Neurospora [ORGN])" | \
  efetch -format gbc -mode xml
```

generates INSDSeq XML (an XML structured version of the GenBank flatfile):

```
  ...
  <INSDFeature_quals>
    <INSDQualifier>
      <INSDQualifier_name>organism</INSDQualifier_name>
      <INSDQualifier_value>Neurospora sp. E10527a</INSDQualifier_value>
    </INSDQualifier>
    <INSDQualifier>
      <INSDQualifier_name>country</INSDQualifier_name>
      <INSDQualifier_value>Ecuador</INSDQualifier_value>
    </INSDQualifier>
    <INSDQualifier>
      <INSDQualifier_name>lat_lon</INSDQualifier_name>
      <INSDQualifier_value>0.68 S 76.40 W</INSDQualifier_value>
    </INSDQualifier>
    ...
```

where qualifier names are data values, not XML tags. Piping it through:

```
  xtract -pattern INSDFeature_quals \
    -block INSDQualifier -match ">country<" -element INSDQualifier_value \
    -block INSDQualifier -match ">lat_lon<" -element INSDQualifier_value | \
  sort | uniq | column -s $'\t' -t
```

will filter by content to separately identify country and latitude-longitude qualifiers, producing:

```
  Brazil:Rio de Janeiro, Seropedica  22.45 S 43.4 W
  Ecuador                            0.68 S 76.40 W
  Ecuador                            2.18 S 80.02 W
  USA: Texas, near Vernon            33.855556 N 99.447220 W
```

(Incorporating fragments of the surrounding XML tags in -match arguments prevents coincidental matches to free text in note qualifiers.)

The -match command must immediately follow -pattern, -group, -block, or -subset.

The -avoid command excludes blocks containing the indicated text, and must immediately follow -pattern, -group, -block, -subset, or -match.

Multiple -avoid or -match conditions are specified with -and and -or commands:

```
  -group INSDFeature -avoid ">proprotein<" -and ">sig_peptide<" \
    -block INSDQualifier -match ">calculated_mol_wt<" -or ">peptide<" \
```

Self-closing tags of the standard form:

```
  <Na-strand/>
```

or alternative form:

```
  <Na-strand></Na-strand>
```

can be indicated by appending parentheses after the tag:

```
  -element "Na-strand()"
```

and the tag name will be printed when found. If a string is placed in the parentheses, it will be printed instead of the name. If the tag contains an attribute:

```
  <Seq-interval_strand>
    <Na-strand value="plus"/>
  </Seq-interval_strand>
```

the -match filter can be used to print a particular string based on the value:

```
  -block Seq-interval_strand -match plus -element "Na-strand(+)" \
  -block Seq-interval_strand -match minus -element "Na-strand(-)"
```

The -lbl argument can be used to print arbitrary text:

```
  -group MeshHeading \
    -block MeshHeading -present QualifierName -lbl "subheadings present" \
    -block MeshHeading -absent QualifierName -lbl "subheadings missing"
```

A value can be recorded in a variable and then displayed multiple times as needed. Variables are indicated by a hyphen followed by a string of capital letters or digits. The variable "-ACCN" is referred to as "&ACCN" in an -element argument.

For example:

```
  efetch -db protein -id "NP_000198.1" -format gpc -mode xml
```

retrieves the XML of a protein with multiple mature peptide features:

```
  ...
  <INSDFeature>
    <INSDFeature_key>mat_peptide</INSDFeature_key>
    <INSDFeature_location>25..54</INSDFeature_location>
    <INSDFeature_intervals>
      <INSDInterval>
        <INSDInterval_from>25</INSDInterval_from>
        <INSDInterval_to>54</INSDInterval_to>
        <INSDInterval_accession>NP_000198.1</INSDInterval_accession>
      </INSDInterval>
    </INSDFeature_intervals>
    <INSDFeature_quals>
      <INSDQualifier>
        <INSDQualifier_name>product</INSDQualifier_name>
        <INSDQualifier_value>insulin B chain</INSDQualifier_value>
      </INSDQualifier>
      <INSDQualifier>
        <INSDQualifier_name>calculated_mol_wt</INSDQualifier_name>
        <INSDQualifier_value>3430</INSDQualifier_value>
      </INSDQualifier>
      <INSDQualifier>
        <INSDQualifier_name>peptide</INSDQualifier_name>
        <INSDQualifier_value>FVNQHLCGSHLVEALYLVCGERGFFYTPKT</INSDQualifier_value>
      </INSDQualifier>
    </INSDFeature_quals>
  </INSDFeature>
  <INSDFeature>
    <INSDFeature_key>mat_peptide</INSDFeature_key>
    <INSDFeature_location>57..87</INSDFeature_location>
    ...
```

Piping the data to:

```
  xtract -pattern INSDSeq -ACCN INSDSeq_accession-version \
    -group INSDFeature -match ">mat_peptide<" \
      -avoid "<INSDFeature_partial" -pfx "\n" -element "&ACCN" \
    -block INSDQualifier -match ">peptide<" -element INSDQualifier_value \
    -block INSDQualifier -match ">peptide<" -element "%INSDQualifier_value" \
    -block INSDQualifier -match ">product<" -element INSDQualifier_value \
    -block INSDQualifier -match ">calculated_mol_wt<" -element INSDQualifier_value
```

saves the accession for use with complete mature peptide features. The prefix causes each mat_peptide entry to be placed on a separate line. Each line begins with the accession number, even though it is outside the scope of the -group.

(Note the common pattern used to extract each qualifier value. An example in the scripting section later in this document takes command-line arguments for feature key and qualifier names to automate the production of xtract instructions for obtaining data from INSDSeq XML.)

(Two GenBank features (gene and operon) have qualifiers with identical names, so -match/-avoid arguments for those terms should start with "_key>" or "_name>" to ensure that searching from INSDFeature correctly targets data in INSDFeature_key or INSDQualifier_name elements.)

The table is sorted by the calculated peptide sequence length column:

```
  sort -t $'\t' -k 3,3nr
```

to produce:

```
  NP_000198.1  EAEDLQVGQVELGGGPGAGSLQPLALEGSLQ  31  C-peptide        3020
  NP_000198.1  FVNQHLCGSHLVEALYLVCGERGFFYTPKT   30  insulin B chain  3430
  NP_000198.1  GIVEQCCTSICSLYQLENYCN            21  insulin A chain  2384
```

The "awk" programming language can perform comparisons on tab-delimited table contents, deleting lines that do not pass the indicated test. Piping to:

```
  awk '{if ( $3 >= 21 && $3 <= 30 ) print}'
```

would filter by the peptide sequence length and remove the C peptide row.

Setting a variable default value, by placing text in parentheses, can be used to prevent columns from shifting when data fields are missing:

```
  -MLWT "(-)"
```

Parallel -present and -absent (or -match and -avoid) statements are used to handle alternative conditions. For example, in:

```
  efetch -db pubmed -id 1937004,2539356 -format xml | \
  xtract -pattern PubmedArticle -PMID "MedlineCitation/PMID" \
    -group MeshHeading \
      -block MeshHeading -present QualifierName \
        -subset DescriptorName -TERM "DescriptorName" -MAJR "@MajorTopicYN" \
        -subset QualifierName -pfx "\n" -element "&PMID" -sep "|" -tab "" \
          -pfx "[" -sfx "]" -element "&MAJR,&TERM" \
          -pfx "{" -sfx "}" -element "@MajorTopicYN,QualifierName" \
      -block MeshHeading -absent QualifierName \
        -subset DescriptorName -pfx "\n" -element "&PMID" -sep "|" -tab "" \
          -pfx "[" -sfx "]" -element "@MajorTopicYN,DescriptorName"
```

the PubMed identifier is memorized by the "-PMID" variable, and there are separate -block sections to handle MeSH terms with and without subheadings. In the former case, the "-TERM" and "-MAJR" variables are set to the MeSH term value and major topic attribute, respectively, for subsequent use with each subheading.

The data line with two MeSH subheadings in the earlier example:

```
[N|Saccharomyces cerevisiae]{Y|genetics}{N|radiation effects}
```

now produces an intermediate form with a separate line for each subheading:

```
1937004    [N|Saccharomyces cerevisiae]{Y|genetics}
1937004    [N|Saccharomyces cerevisiae]{N|radiation effects}
```

The -pattern, -group, -block, and -subset commands provide a nested hierarchy of loop organizers for exploration of XML objects. Each pattern can contain multiple groups, each group can encompass multiple blocks, and each block can have multiple subsets. Use of different argument names allows a linear representation of loop nesting, and provides sufficient flexibility to identify and extract arbitrary data from XML records in Entrez.

The extraction command:

```
xtract -pattern PubmedArticle \
  -block Author -element Initials,LastName \
  -block MeshHeading -element DescriptorName \
  -subset QualifierName -element QualifierName
```

could be represented as a computer program in pseudo code by:

```
for each PubmedArticle {
  for each Author {
    print Initials LastName
  }
  for each MeshHeading {
    print DescriptorName
    for each QualifierName {
      print QualifierName
    }
  }
}
```

Sketching an extraction in pseudo code can clarify relative nesting levels when building complex commands.

Extra arguments (-division, -branch, -section, and -unit) are held in reserve to provide additional levels of organization, should the need arise in the future for complex, deeply-nested XML data. The full set of commands, in order of rank, are:

```
-pattern
-division
-group
-branch
-block
-section
-subset
-unit
```

Starting xtract exploration with -block, and expanding with -group and -subset, leaves additional level names that can be used wherever needed without having to redesign the entire command.

XML DATA EXTRACTION SPECIAL TOPICS

Certain XML objects returned by efetch are recursively defined, including Taxon in TaxaSet (-db taxonomy) and Gene-commentary in Entrezgene_comments (-db gene). Thus, they can have nested objects with the same XML tag. Retrieval of taxonomy records for human and fruit fly:

```
  efetch -db taxonomy -id 9606,7227 -format xml
```

produces XML with nested Taxon objects (marked below with line references) for each rank in the taxonomic lineage:

```
    <TaxaSet>
1     <Taxon>
        <TaxId>9606</TaxId>
        <ScientificName>Homo sapiens</ScientificName>
        ...
        <LineageEx>
2         <Taxon>
            <TaxId>131567</TaxId>
            <ScientificName>cellular organisms</ScientificName>
            <Rank>no rank</Rank>
3         </Taxon>
4         <Taxon>
            <TaxId>2759</TaxId>
            <ScientificName>Eukaryota</ScientificName>
            <Rank>superkingdom</Rank>
5         </Taxon>
          ...
        </LineageEx>
        ...
6     </Taxon>
7     <Taxon>
        <TaxId>7227</TaxId>
        <ScientificName>Drosophila melanogaster</ScientificName>
        ...
8     </Taxon>
    </TaxaSet>
```

Although <Taxon> on line 1 is actually closed by </Taxon> on line 6, use of "-group Taxon" to visit data between <Taxon> and </Taxon> pairs will incorrectly match it with the first </Taxon> on line 3.

Xtract circumvents this artifact with an alternative search algorithm that tracks XML object depth, instead of using regular expression pattern matching, and which is triggered by capitalized versions of the exploration commands:

```
  -Division
  -Group
  -Branch
  -Block
  -Section
  -Subset
  -Unit
```

Extraction of data with a capitalized -Group argument:

```
  efetch -db taxonomy -id 9606,7227 -format xml | \
  xtract -pattern TaxaSet -Group Taxon -sfx "\n" -tab "" -ret "" \
    -first TaxId,ScientificName,GenbankCommonName
```

behaves as desired and returns information for the main entries:

```
  9606    Homo sapiens               human
  7227    Drosophila melanogaster    fruit fly
```

Use of lower-case "-group" erroneously visits the entries for each taxonomic lineage level (skipping the first because of the pattern matching artifact):

```
  9606     Homo sapiens    human
  2759     Eukaryota
  33154    Opisthokonta
  33208    Metazoa
  ...
```

Similarly, use of a capitalized -Block argument:

```
  efetch -db gene -id 837025,837031 -format xml | \
  xtract -pattern Entrezgene_comments \
    -Block Gene-commentary -match "Related Sequences" \
    -element Gene-commentary_accession
```

correctly finds all appropriate accessions in Entrezgene records:

```
  CP002684    AEE27818    DQ446230    ABE65601
  CP002684    AEE27823    CP002684    AEE27824    CP002684    AEE27825
```

(The alternative search algorithm is significantly slower than the regular expression version in the current implementation, so capitalized exploration arguments should only be used when necessary to handle recursive elements.)

(There is no capitalized version of the top-level -pattern organizer.)

The -trim argument will remove the first tag (e.g., <Taxon>) so that subsequent explorations are able to visit the internal objects:

```
  efetch -db taxonomy -id 9606,7227 -format xml | \
  xtract -pattern TaxaSet -Group Taxon -trim -block Taxon -sfx "\n" \
    -tab "" -ret "" -first TaxId,ScientificName,GenbankCommonName
```

returns:

```
  131567    cellular organisms
  2759      Eukaryota
  33154     Opisthokonta
  33208     Metazoa
  ...
```

Visual inspection of the XML structure with the -outline argument helps determine the appropriate arguments to employ. A pubmed summary query:

```
  esummary -db pubmed -id 16588492
```

retrieves an XML document summary:

```
  <eSummaryResult>
    <DocumentSummarySet status="OK">
      <DocumentSummary uid="16588492">
        <Id>16588492</Id>
        <PubDate>1941 Nov 15</PubDate>
        <EPubDate/>
        <Source>Proc Natl Acad Sci U S A</Source>
        <Authors>
          <Author>
            <Name>Beadle GW</Name>
            <AuthType>Author</AuthType>
            <ClusterID>518804</ClusterID>
          </Author>
          <Author>
            <Name>Tatum EL</Name>
            <AuthType>Author</AuthType>
            <ClusterID>6857925</ClusterID>
          </Author>
        </Authors>
        <LastAuthor>Tatum EL</LastAuthor>
        <Title>Genetic Control of Biochemical Reactions in Neurospora.</Title>
        ...
```

that can be piped through:

```
  xtract -pattern DocumentSummary -outline
```

to return an outline that shows XML structure and tag names but without attributes, closing tags, and content values:

```
  DocumentSummary
    Id
    PubDate
  EPubDate/
    Source
    Authors
      Author
        Name
        AuthType
        ClusterID
      Author
        Name
        AuthType
        ClusterID
    LastAuthor
    Title
    SortTitle
    Volume
    Issue
    Pages
    Lang
      string
    NlmUniqueID
    ISSN
    ESSN
    PubType
    ...
```

The outline view presents a clear, uncluttered picture of the XML hierarchy that is useful in designing the appropriate command for actual data extraction. Copy and paste from the -outline output to xtract arguments (-element, -group, -match, etc.) can help avoid typographical errors.

QUERYING EXTERNAL WEB SERVICES

The EDirect nquire function can be used to obtain data from an arbitrary URL. Queries are built up from command-line arguments. For example:

```
  nquire -url "http://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi" \
    -db pubmed -term insulin
```

reads the URL and then tag/value pairs to generate an EUtils query:

```
  http://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi?db=pubmed&term=insulin
```

Paths can be separated into components, which are combined with slashes, so:

```
  -url http://eutils.ncbi.nlm.nih.gov entrez/eutils efetch.fcgi
```

is converted to:

```
  http://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi
```

Multiple values between tags are combined with commas. Thus:

```
  -db nuccore -id U54469 V00328 -rettype fasta
```

is transformed into:

```
  db=nuccore&id=U54469,V00328&rettype=fasta
```

A value that starts with a hyphen (or minus sign) can be distinguished from a tag by prefixing it with a backslash. For example:

```
  nquire -url http://ws.geonames.org/countryCode -lat 41.796 -lng "\-87.577"
```

will be sent as:

```
  http://ws.geonames.org/countryCode?lat=41.796&lng=-87.577
```

and will return "US" for coordinates within Chicago, which has a negative (western hemisphere) longitude value.

The -alias argument can read a file of shortcut keywords and URL aliases. The following aliases are always available:

```
  ncbi_url      http://www.ncbi.nlm.nih.gov
  eutils_url    http://eutils.ncbi.nlm.nih.gov/entrez/eutils
```

so the command:

```
  nquire -url "(#eutils_url)" esearch.fcgi \
    -db gds -term "GSE22309 [ACCN] AND gse [ETYP]" -retmax 200
```

will run an ESearch query and return an eSearchResult XML object.

ENTREZ DIRECT COMMANDS WITHIN SCRIPTS

(The next sections are on advanced automation with UNIX shell scripts. Most biologists using EDirect should find the methods described above sufficient for their needs, and thus can safely choose to ignore these remaining topics.)

Taking an adventurous plunge into the world of programming, a shell script can be written when each output line of one step needs to be processed independently, instead of output being piped in its entirety to the next command. (The simplest shell script can be merely a copy of the set of commands that are typed into the terminal for execution.)

In shell scripts, variables can be set to the results of a command by enclosing the statements in backtick ("`") characters. The variable name is prefixed by a dollar sign ("$") to use its value as an argument in another command. Comments start with a pound sign ("#") and are ignored. Quotation marks within quoted strings are "escaped" with a backslash ("\"). Subroutines can be used to collect common code or simplify the organization of the script.

For example:

```
  #!/bin/bash -norc

  parse_fields() {
    echo "$1" | \
    xtract -pattern Field -element Name FullName Description | \
    sort -t $'\t' -k 2,2f | \
    perl -pe 's/(.+?)\t(.+?)\t(.+)/\[$1\]\t$2\t$3/' | \
    column -s $'\t' -t
  }

  dbs=`einfo -dbs | xtract -pattern DbName -element DbName | sort`

  for db in $dbs; do
    eix=`einfo -db $db`

    echo "$db"
    echo ""
    flds=`parse_fields "$eix"`
    echo "$flds"
    echo ""

    sleep 1
  done
```

will obtain the list of Entrez databases, and then return the abbreviations, names, and descriptions of indexed search fields, for each individual database:

```
  ...
  epigenomics

  [ACCN]  Accession      Accession number of sequence
  [ALL]   All Fields     All terms from all searchable fields
  [AUTH]  Author         Author
  [CDAT]  Create Date    CreateDate
  [DOCT]  Document Type  DocType
  [FILT]  Filter         Limits the records
  [KYWD]  Keyword        Keyword
  [ORGN]  Organism       scientific and common names of organism
```

```
  [PRID]  Project ID     ProjectId
  [TXID]  Taxonomy ID    TaxId
  [WORD]  Text Word      Text
  [TITL]  Title          Title
  [UID]   UID            Unique number assigned to publication
  [MDAT]  Update Date    UpdateDate
  ...
```

A shell script can read a variable number of command-line arguments and use them to automate complex operations. Given a script named "qualifiers.sh":

```
  #!/bin/bash -norc

  if [ "$#" -lt 2 ]; then
    echo "Must supply a feature key and at least one qualifier name"
    exit 1
  fi

  first=1
  query="xtract -pattern INSDSeq -ACCN INSDSeq_accession-version"

  for arg in "$@"; do
    if [ "$first" -eq 1 ]; then
      query=`echo "$query" "-group INSDFeature -match \"_key>$arg<\"" \
                    "-pfx \"\n\" -element \"&ACCN\""`
      first=0
    else
      query=`echo "$query" "-block INSDQualifier -match \"_name>$arg<\"" \
                    "-element INSDQualifier_value"`
    fi
  done

  if [ -t 0 ]; then
    echo "$query"
  else
    inp=`cat /dev/stdin`
    res=`eval "echo -e \"$inp\" | $query"`
    echo "$res"
  fi
```

if INSDSeq XML data from NP_000198.1 in the earlier example is piped to:

```
  qualifiers.sh mat_peptide peptide calculated_mol_wt product
```

it will read the first argument as the feature key, dynamically add to the growing xtract command for each qualifier argument, and execute it to display:

```
  NP_000198.1  FVNQHLCGSHLVEALYLVCGERGFFYTPKT   3430  insulin B chain
  NP_000198.1  EAEDLQVGQVELGGGPGAGSLQPLALEGSLQ  3020  C-peptide
  NP_000198.1  GIVEQCCTSICSLYQLENYCN            2384  insulin A chain
```

This removes the tedium of custom-coding the xtract statement. (Running without piped input will print the generated xtract command for use in another script.)

COMPLEX BEHAVIOR WITHOUT SCRIPTING

Writing a script to process every database or UID can sometimes be avoided by creative use of the UNIX xargs command. For example, in:

```
einfo -dbs | xtract -pattern DbName -element DbName | sort | \
xargs -n 1 einfo -db | \
xtract -pattern DbInfo -pfx "\n" -tab "\n" -element DbName \
  -block Field -pfx "" -tab "\n" -element Name,FullName | \
perl -pe 's/(.+?)\t(.+?)/\[$1\]\t$2/' | \
expand
```

the xargs line passes each Entrez database name to a separate einfo request. The combined XML results are then sent as one unit to the remaining commands:

```
...
epigenomics
[ALL]   All Fields
[UID]   UID
[FILT]  Filter
[WORD]  Text Word
[TITL]  Title
[TXID]  Taxonomy ID
...
```

With this simple use of xargs, however, subsequent processing per database is not possible, so fields cannot be sorted in alphabetical order.

Combining xargs with a UNIX shell statement can get around the per-database processing limitation while still avoiding a script. Within the "sh" command string, the database argument passed by xargs is substituted at the "$0" variable. All of the sh commands are run separately on each database.

Because the sh instructions are enclosed between single quote ("'") characters (apostrophes), in-line perl commands cannot be used, nor can sort be told to use tabs (instead of tabs or spaces) to delimit columns. But a combination of xtract customization and the trick of temporarily replacing spaces with asterisks:

```
einfo -dbs | xtract -pattern DbName -element DbName | sort | \
xargs -n 1 sh -c 'einfo -db $0 | \
  xtract -pattern DbInfo -tab "\n\n" -element DbName \
    -block Field -pfx "[" -sep "]\t" -tab "\n" -element Name,FullName | \
  sed "s/ /*/g" | sort -k 2,2f | sed "s/*/ /g" | \
  expand'
```

allows the desired sorting by full field name and produces results that are alphabetized for each database:

```
...
epigenomics
[ACCN]  Accession
[ALL]   All Fields
[AUTH]  Author
[CDAT]  Create Date
[DOCT]  Document Type
[FILT]  Filter
...
```

Xargs can also help with large esummary XML results that cannot be handled by xtract as a single unit. The xargs "-n" argument determines how many words (database names, accessions, UIDs, etc.) are sent to each command. Thus:

```
  ...
  esummary | \
  xtract -pattern Author -element Name | \
  sort | uniq -c | sort -rn
```

can be replaced by:

```
  ...
  efetch -format uid | \
  xargs -n 50 echo | sed 's/ /,/g' | \
  xargs -n 1 sh -c 'esummary -db pubmed -id "$0" | \
  xtract -pattern Author -element Name' | \
  sort | uniq -c | sort -rn
```

to process summaries in groups of 50, and then sort the combined results.

HOUSEKEEPING

EDirect automatically obtains the current user's e-mail address from the system, to have someone to notify in case a runaway script causes problems with an EUtils server, but if another contact address is desired (e.g., that of a system administrator or software developer) it can be explicitly set at the beginning of a pipeline or script:

```
  econtact -email author@institution.edu -tool name_of_script
```

That way the NCBI has information on who to contact if a runaway loop in the script accidentally abuses NCBI resources. (For convenience, the preferred e-mail address and software tool name can also be set in all EUtils-calling operations.)

An alias file of shortcut keys and query phrases can be read by all operations that call EUtils functions, but it can also be read in a separate instruction at the beginning of a pipeline or script:

```
  eproxy -alias query_alias_file_name
```

For maximum flexibility, separate eproxy commands can be piped together to load multiple shortcut files, as long as the shortcut strings are all unique.

The shell script command:

```
  sleep 1
```

adds a one second delay between steps in a loop, and can be used to help prevent overuse of the EUtils servers by advanced scripts.

APPENDICES

Command-line arguments for the EDirect functions are shown below:

```
  esearch

    -db
    -query
    -days
    -mindate
```

```
  -maxdate
  -datetype
  -label

efilter

  -query
  -days
  -mindate
  -maxdate
  -datetype
  -label

elink

  -db
  -id
  -target
  -name
  -related
  -cmd
  -mode
  -batch
  -holding
  -label

esummary

  -db
  -id

efetch

  -db
  -id
  -format
  -mode
  -seq_start
  -seq_stop
  -strand
  -complexity
  -pipe

einfo

  -db
  -dbs

epost

  -db
  -id
  -format
  -label

eproxy

  -alias
  -pipe
```

econtact

  -email
  -tool

nquire

  -url
  -http

xtract

  -pattern
  -division
  -group
  -branch
  -block
  -section
  -subset
  -unit
  -trim
  -match
  -avoid
  -present
  -absent
  -and
  -or
  -element
  -first
  -last
  -pfx
  -sfx
  -sep
  -tab
  -ret
  -lbl
  -outline
  -pipe

In addition, -email and -tool are available in all EUtils functions to override default values, -silent will suppress link failure retry messages, -verbose will print the <ENTREZ_DIRECT> field values at each step, -debug can be used to assist with debugging a script by printing the internal URL query and XML results of each step, -http get will force the use of GET instead of POST, -alias will specify a file of shortcut keywords and query strings or URL sections, and -base will specify a particular server for quality assurance testing.

Efetch -format and -mode values for each database are shown below:

-db	-format	-mode	Record Type
(all)			
	full		Same as native except for mesh
	uid		Unique Identifier List
	url		Entrez URL
	xml		Converted to -format full -mode xml

bioproject			
	native		BioProject Report
	native	xml	RecordSet XML
biosample			
	native		BioSample Report
	native	xml	BioSampleSet XML
biosystems			
	native	xml	Sys-set XML
gds			
	native	xml	RecordSet XML
	summary		Summary
gene			
	gene_table		Gene Table
	native		Gene Report
	native	asn.1	Entrezgene ASN.1
	native	xml	Entrezgene-Set XML
homologene			
	alignmentscores		Alignment Scores
	fasta		FASTA
	homologene		Homologene Report
	native		Homologene List
	native	asn.1	HG-Entry ASN.1
	native	xml	Entrez-Homologene-Set XML
mesh			
	full		Full Record
	native	xml	RecordSet XML
	native		MeSH Report
nlmcatalog			
	native		Full Record
	native	xml	NLMCatalogRecordSet XML
pmc			
	medline		MEDLINE
	native	xml	pmc-articleset XML
pubmed			
	abstract		Abstract
	medline		MEDLINE
	native	asn.1	Pubmed-entry ASN.1
	native	xml	PubmedArticleSet XML
(sequences)			
	acc		Accession Number
	est		EST Report
	fasta		FASTA
	fasta	xml	TinySeq XML
	fasta_cds_aa		FASTA of CDS Products
	fasta_cds_na		FASTA of Coding Regions
	ft		Feature Table
	gb		GenBank Flatfile
	gb	xml	GBSet XML
	gbc	xml	INSDSet XML
	gbwithparts		GenBank with Contig Sequences

```
                gp                          GenPept Flatfile
                gp                  xml     GBSet XML
                gpc                 xml     INSDSet XML
                gss                         GSS Report
                native              asn.1   Seq-entry ASN.1
                native              xml     Bioseq-set XML
                seqid                       Seq-id ASN.1

  snp
                chr                         Chromosome Report
                docset                      Summary
                fasta                       FASTA
                flt                         Flat File
                native              asn.1   Rs ASN.1
                native              xml     ExchangeSet XML
                rsr                         RS Cluster Report
                ssexemplar                  SS Exemplar List

  sra
                native              xml     EXPERIMENT_PACKAGE_SET XML

  structure
                mmdb                        Ncbi-mime-asn1 strucseq ASN.1
                native                      MMDB Report
                native              xml     RecordSet XML

  taxonomy
                native                      Taxonomy List
                native              xml     TaxaSet XML
```

EInfo field data contains status flags for several indexing properties:

```
  ...
  <Field>
    <Name>ALL</Name>
    <FullName>All Fields</FullName>
    <Description>All terms from all searchable fields</Description>
    <TermCount>138982028</TermCount>
    <IsDate>N</IsDate>
    <IsNumerical>N</IsNumerical>
    <SingleToken>N</SingleToken>
    <Hierarchy>N</Hierarchy>
    <IsHidden>N</IsHidden>
    <IsTruncatable>Y</IsTruncatable>
    <IsRangable>N</IsRangable>
  </Field>
  ...
```

Documentation for the Entrez Utilities can be found at:

  http://www.ncbi.nlm.nih.gov/books/NBK25501/