Entrez Direct Reference

Searching, Retrieving, and Parsing Data from NCBI Databases through the Unix Command Line

National Library of Medicine
National Center for Biotechnology Information

Abstract

Entrez Direct was designed to make it easy for biologists to automate Entrez queries and analyze structured data results, all without requiring extensive prior computer programming experience.

EDirect supports both online queries and local archives by providing two equivalent sets of search, link, filter, and fetch scripts. The same structured message format is used for communication between steps, allowing query pipelines to seamlessly operate on a mixture of local and remote data sources.

Central to EDirect's success is a utility that uses command-line arguments to control data extraction from structured records, without requiring explicit object paths or complicated path formulas. Instead, the input stream is first partitioned into separate records by parent object name. Then, within each record, data items are found by element name, using a depth-first recursive search.

Context is supplied by exploration commands that redirect the search to separately visit each instance of a named container. These are equivalent to nested for-loops, and can keep related fields together in the output. For example, exploration by Author limits the scope of subsequent searches for Initials and LastName elements to nodes within the current author object.

Element extraction variants perform text modifications or integer calculations on the data values. Format customization commands control record, field, and instance separators, and control optional prefix and suffix strings that can surround extracted element values. A wrapping shortcut generates reformatting instructions that automatically save the extracted information inside new XML objects.

An unexpected benefit of wrapping is that it encourages breaking up complex tasks into a series of simpler operations. Multistep processing chains can replace much larger programs that are written in traditional computer languages. They combine XML rewrapping commands and selected element variants to place modified values into new XML structures for stepwise transformation. No custom reading or writing code is needed, since the data in each step remains in XML format.

EDirect local archives rely on modern solid-state drives and file systems. If SSD price and capacity trends continue, large local data repositories should become increasingly affordable to research groups over the next decade. This provides a viable long-term alternative to remote public storage. EDirect archives also avoid common system administration and infrastructure support costs.

Prototyped in Perl, EDirect was incrementally refactored into compiled Go language programs and portable Unix shell scripts. Self-contained native binary executables avoid dynamic-link library startup delays and version skew, and the redesign gave a 150-fold speed boost to structured data extraction.

Acknowledgments

This work was supported by the National Center for Biotechnology Information of the National Library of Medicine (NLM), National Institutes of Health (NIH). The contributions of the NIH author(s) are considered Works of the United States Government. The findings and conclusions presented in this paper are those of the author(s) and do not necessarily reflect the views of the NIH or the U.S. Department of Health and Human Services.

Introduction

Entrez Direct (EDirect) provides access to Entrez, the NCBI's suite of interconnected databases, from a Unix terminal window. Search terms are entered as command-line arguments. Individual operations are connected with Unix pipes to construct multi-step queries. Selected records can then be retrieved in a variety of formats.

Programmatic Access

EDirect connects to Entrez through the Entrez Programming Utilities interface. It supports searching by indexed terms, looking up precomputed neighbors or links, filtering results by date or category, and downloading record summaries or reports.

Navigation programs (**esearch**, **elink**, **efilter**, and **efetch**) communicate by means of a small structured message, which can be passed invisibly between operations with a Unix pipe. The message includes the current database, so it does not need to be given as an argument after the first step.

Accessory programs (**nquire**, **transmute**, and **xtract**) can help eliminate the need for writing custom software to answer ad hoc questions. Queries can move effortlessly between EDirect programs and Unix utilities or scripts to perform actions that cannot be accomplished entirely within Entrez.

Navigation Functions

Esearch performs a new Entrez search using terms in indexed fields. It requires a **-db** argument for the database name and uses **-query** for the search terms. For PubMed, without field qualifiers, the server uses automatic term mapping to compose a search strategy by translating the supplied query:

```
esearch -db pubmed -query "selective serotonin reuptake inhibitor"
```

Search terms can also be qualified with a bracketed field name to match within the specified index:

```
esearch -db nuccore -query "insulin [PROT] AND rodents [ORGN]"
```

Elink looks up precomputed neighbors within a database, or finds associated records in other databases, or uses the NIH Open Citation Collection service (PMID 31600197) to follow reference lists:

```
elink -related
```

```
elink -target gene
```

```
elink -cited
```

```
elink -cites
```

Efilter limits the results of a previous query, with shortcuts that can also be used in esearch:

```
efilter -molecule genomic -location chloroplast -country sweden -mindate 1985
```

Efetch downloads selected records or reports in a style designated by **-format**:

```
efetch -format abstract
```

Individual query commands are connected by a Unix **vertical bar** pipe symbol:

```
esearch -db pubmed -query "tn3 transposition immunity" | efetch -format apa
```

The vertical bar also allows query steps to be placed on separate lines:

```
esearch -db pubmed -query "raynaud disease AND fish oil" |
efetch -format medline
```

Einfo can report the fields and links that are indexed for each database:

```
einfo -db protein -fields
```

This will return a table of field abbreviations and names indexed for proteins:

```
ACCN    Accession
ALL     All Fields
ASSM    Assembly
…
```

Each program has a **-help** command that prints detailed information about available arguments.

EDirect programs are designed to work on large sets of data. There is no need to use a script to loop over records in small groups, or write code to retry a query after a transient network or server failure, or add a time delay between requests. All of those features are already built into the system.

Accessory Programs

Nquire retrieves data from remote servers with URLs constructed from command line arguments:

```
nquire -get https://icite.od.nih.gov api/pubs -pmids 2539356 |
```

Transmute converts a concatenated stream of JSON objects or other structured formats into XML:

```
transmute -j2x |
```

Xtract uses waypoints to navigate complex XML hierarchies, and obtains data values by field name:

```
xtract -pattern data -element cited_by |
```

The resulting output can be post-processed by Unix utilities or scripts:

```
fmt -w 1 | sort -V | uniq
```

Mixed-Content Records

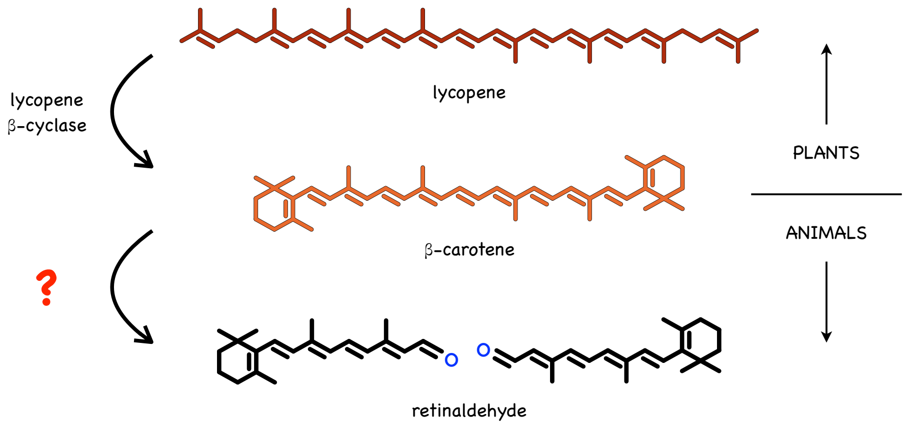
The **-mixed** flag is needed when processing XML or HTML records that contain embedded markup instructions. Using this argument, transmute -format can reformat the records for easier reading, and xtract -verify can confirm that all opening and closing tags are properly balanced:

```
efetch -db pmc -id 4305238 -format xml |
transmute -mixed -format |
xtract -mixed -verify
```

Discovery by Navigation

PubMed related articles are identified by a text retrieval algorithm that examines the title, abstract, and medical subject headings (MeSH terms). The connections between papers can be used to help make discoveries. An example of this is finding the last enzymatic step in the vitamin A biosynthetic pathway.

Lycopene cyclase in plants converts lycopene into β-carotene, the immediate biochemical precursor of vitamin A. β-Carotene is an essential nutrient, required in the diet of herbivores. This indicates that lycopene cyclase is not present in animals (with a few exceptions caused by horizontal gene transfer), and that the enzyme responsible for converting β-carotene into vitamin A is not present in plants.



An initial search on the lycopene cyclase enzyme finds 306 articles. Looking up precomputed neighbors returns 19,146 papers, some of which might be expected to discuss other enzymes in the pathway:

```
esearch -db pubmed -query "lycopene cyclase" | elink -related |
```

We cannot reliably limit the results to animals in PubMed, but we can for sequence records, which are indexed by the NCBI taxonomy. Linking the publication neighbors to their associated protein records finds 604,878 sequences. Restricting those to mice excludes plants, fungi, and bacteria, thus eliminating the earlier enzymes:

```
elink -target protein | efilter -organism mouse -source refseq |
```

This matches only 32 sequences, which is small enough to examine by retrieving the individual records:

```
efetch -format fasta
```

As expected, the results include the enzyme that splits β-carotene into two molecules of retinal:

```
…
>NP_067461.2 beta,beta-carotene 15,15'-dioxygenase isoform 1 [Mus musculus]
MEIIFGQNKKEQLEPVQAKVTGSIPAWLQGTLLRNGPGMHTVGESKYNHWFDGLALLHSFSIRDGEVFYR
SKYLQSDTYIANIEANRIVVSEFGTMAYPDPCKNIFSKAFSYLSHTIPDFTDNCLINIMKCGEDFYATTE
…
```

A better example used Entrez protein neighbors to instantly rediscover the similarity between a human colon cancer gene and microbial DNA repair genes. Unfortunately, precomputed BLAST links were discontinued due to the exponential growth of the sequence databases.

XML Data Extraction

The ability to obtain Entrez records in structured format, and to easily extract the underlying data, allows the user to ask novel questions that are not addressed by existing analysis software.

The xtract program uses command-line arguments to direct the conversion of data in eXtensible Markup Language format. It allows record detection, path exploration, element selection, conditional processing, and report formatting to be controlled independently.

The **-pattern** command partitions an XML stream by object name into individual records that are processed separately. Within each record, the **-element** command does an exhaustive, depth-first search to find data content by field name.

Neither explicit object paths nor complicated path formulas are needed for element identification.

Format Customization

By default, the -pattern argument divides the results into rows, while placement of data into columns is controlled by -element, to create a tab-delimited table.

Formatting commands allow extensive customization of the output. The line break between -pattern rows is changed with **-ret**, while the tab character between -element columns is modified by **-tab**.

Multiple instances of the same element are distinguished using **-sep**, which controls their separation independently of the -tab command. The following query:

```
efetch -db pubmed -id 6271474,24178092 -format docsum |
xtract -pattern DocumentSummary -sep "|" -element Id PubDate Name
```

returns a tab-delimited table with individual author names separated by vertical bars:

```
6271474     1981        Casadaban MJ|Chou J|Lemaux P|Tu CP|Cohen SN
24178092    1994 Dec    Garber ED|Ruddat M
```

The -sep value also applies to distinct -element arguments that are grouped with **commas**. This can be used to keep data from multiple related fields in the same column:

```
-sep " " -element Initials,LastName
```

The **-def** command sets a default placeholder to be printed when none of the comma-separated fields in an -element clause are present:

```
-def "-" -sep " " -element Year,Month,MedlineDate
```

Limit by Parent

An -element argument can use the **parent / child** construct to limit selection when items can only be disambiguated by position, not by name. In this case, it prevents the display of additional PMIDs that might be present in CommentsCorrections objects deeper within the MedlineCitation container:

```
xtract -pattern PubmedArticle -element MedlineCitation/PMID
```

Element Variants

Derivatives of -element were initially created to avoid having to write post-processing scripts just to perform trivial modifications or integer calculations on extracted data. Other variants were added for content normalization, report formatting, or index generation. The commands are in several categories:

Positional:	**-first**, **-last**, **-even**, **-odd**, **-backward**
Numeric:	**-num**, **-len**, **-inc**, **-dec**, **-mod**, **-bin**, **-hex**, **-bit**, **-sqt**, **-lge**, **-lg2**, **-log**
Statistics:	**-sum**, **-acc**, **-min**, **-max**, **-dev**, **-med**, **-avg**, **-geo**, **-hrm**, **-rms**
Character:	**-encode**, **-upper**, **-lower**, **-title**, **-mirror**, **-alpha**, **-alnum**
Text:	**-terms**, **words**, **-pairs**, **-letters**, **-split**, **-order**, **-reverse**, **-prose**
Sequence:	**-revcomp**, **-fasta**, **-ncbi2na**, **-cds2prot**, **-molwt**, **-ucsc-based**, **-nucleic**
Citation:	**-year**, **-month**, **-date**, **-auth**, **-initials**, **-page**, **-author**, **-journal**
Other:	**-doi**, **-wct**, **-trim**, **-pad**, **-mask**, **-pept**, **-accession**, **-numeric**

The original -element prefix shortcuts, "**#**" and "**%**", are redirected to **-num** and **-len**, respectively.

Substring Extraction

Following an element name with square brackets allows selection by numeric range (with a colon between the character start and stop positions) or removal of leading and trailing text (with a vertical bar separating the interior prefix and suffix endpoints):

```
-author Initials[1:1] -numeric "ArticleId[PMC|]" -upper "Accession[|.]"
```

XML Repackaging

Repackaging commands (**-wrp**, **-enc**, and **-pkg**) wrap extracted data values with bracketed XML tags given only the object name. For example, **-wrp Word** issues the following formatting instructions:

```
-pfx "<Word>" -sep "</Word><Word>" -sfx "</Word>"
```

It also sets an internal flag to ensure that data values containing encoded symbols (ampersands, angle brackets, apostrophes, and quotation marks) remain properly encoded inside the new XML.

Combining -wrp commands and -element variants can break up complex tasks into a series of simpler operations. No custom reading or writing code is needed, since the data remains in XML format.

Generating Attributes

Additional commands (**-tag**, **-att**, **-atr**, **-cls**, **-slf**, and **-end**) allow generation of XML tags with attributes. The following will produce regular and self-closing XML objects, respectively:

```
-tag Item -att type journal -cls -element Source -end Item
```

```
<Item type="journal">J Bacteriol</Item>
```

```
-tag Item -att type journal -atr name Source -slf
```

```
<Item type="journal" name="J Bacteriol" />
```

Exploration Control

Exploration commands control the order in which XML record contents are examined, by separately presenting each instance of the chosen subregion. This limits what subsequent commands "see" at any one time, and can allow related fields in an object to be kept together.

Unlike the simpler DocumentSummary format, records retrieved as PubmedArticle XML:

```
efetch -db pubmed -id 1413997 -format xml |
```

have authors with separate fields for last name and initials:

```
<Author>
  <LastName>Mortimer</LastName>
  <Initials>RK</Initials>
</Author>
```

Without being given any guidance about context, an -element command on initials and last names:

```
xtract -pattern PubmedArticle -element Initials LastName
```

will explore the current record for each argument in turn, printing all initials followed by all last names:

```
RK     CR     JS     Mortimer     Contopoulou     King
```

Inserting a **-block** command adds a new exploration layer between -pattern and -element, which redirects data exploration to present the authors one at a time:

```
xtract -pattern PubmedArticle -block Author -element Initials LastName
```

Each time through the loop, the -element command only sees the current author's values. This restores the correct association of initials and last names in the output:

```
RK     Mortimer     CR     Contopoulou     JS     King
```

Grouping the two author subfields with a comma, and adjusting the -sep and -tab values:

```
xtract -pattern PubmedArticle -block Author \
  -sep " " -tab ", " -element Initials,LastName
```

produces a more traditional formatting of author names:

```
RK Mortimer, CR Contopoulou, JS King
```

Nested Exploration

Exploration command names (**-group**, **-block**, and **-subset**) are assigned to a precedence hierarchy:

```
-pattern > -group > -block > -subset > -element
```

and are combined in ranked order to control object iteration at progressively deeper levels in the XML data structure. Each command argument acts as a "nested for-loop" control variable, retaining information about the context, or state of exploration, at its level.

A nucleotide or protein sequence record can have multiple features. Each feature can have multiple qualifiers. And every qualifier has separate name and value nodes. Exploring this natural data hierarchy, with **-pattern** for the sequence, **-group** for the feature, and **-block** for the qualifier:

```
efetch -db nuccore -id NM_021486.4 -format gbc |
xtract -pattern INSDSeq -element INSDSeq_accession-version \
  -group INSDFeature -deq "\n\t" -element INSDFeature_key \
    -block INSDQualifier -deq "\n\t\t" \
      -element INSDQualifier_name INSDQualifier_value
```

keeps qualifiers, such as gene and product, associated with their parent features, and keeps qualifier names and values together on the same line:

```
NM_021486.4
    source
              organism       Mus musculus
              mol_type       mRNA
    gene
              gene           Bco1
    CDS
              gene           Bco1
              product        beta,beta-carotene 15,15'-dioxygenase isoform 1
              protein_id     NP_067461.2
              translation    MEIIFGQNKKEQLEPVQAKVTGSIPAWLQGTLLRNGPGM …
              …
```

Saving Data in Variables

A value can be recorded in a variable and used wherever needed. Variables are created by a hyphen followed by a name consisting of a string of capital letters or digits (e.g., **-KEY**). Stored values are retrieved by placing an ampersand before the variable name (e.g., **"&KEY"**) in an -element statement:

```
efetch -db nuccore -id NM_021486.4 -format gbc |
xtract -pattern INSDSeq -element INSDSeq_accession-version \
  -group INSDFeature -KEY INSDFeature_key \
    -block INSDQualifier -deq "\n\t" \
      -element "&KEY" INSDQualifier_name INSDQualifier_value
```

This prints the feature key on each line before the qualifier name and value, even though the feature key is now outside of the visibility scope (which is the current qualifier):

```
NM_021486.4
    source    organism       Mus musculus
    source    mol_type       mRNA
    gene      gene           Bco1
    CDS       gene           Bco1
    CDS       product        beta,beta-carotene 15,15'-dioxygenase isoform 1
    CDS       protein_id     NP_067461.2
    CDS       translation    MEIIFGQNKKEQLEPVQAKVTGSIPAWLQGTLLRNGPGM …
    …
```

Variables can be (re)initialized with an explicit literal value inside parentheses. A double-hyphen appends a value to the variable. A variable can also save the modified data resulting from an -element variant operation. This can allow multiple sequential transitions within a single xtract command:

```
-COM "(, )" --KYWDS kwd -END -sum "Start,Length" -MID -avg "Start,&END"
```

Conditional Execution

Conditional processing commands (**-if** and **-unless**) restrict object exploration by data content. They check to see if the named field is within the scope, and may be used in conjunction with string, numeric, or object constraints to require an additional match by value. Use **-and** and **-or** to build compound tests. Streaming input through **-select** removes records that do not satisfy the condition:

```
esearch -db pubmed -query "Havran W [AUTH]" | efetch -format xml |
xtract -pattern PubmedArticle -select Language -equals eng |
xtract -pattern PubmedArticle -block Author -if LastName -is-not Havran \
  -sep ", " -tab "\n" -author LastName,Initials[1:1] |
sort-uniq-count-rank
```

This limits the results to papers written in English and prints a table of the most frequent collaborators, using a range to keep only the first initial so that variants like "Berg, C" and "Berg, CM" are combined:

```
35    Witherden, D
15    Boismenu, R
12    Jameson, J
10    Allison, J
10    Fitch, F
…
```

Numeric constraints can compare the integer values of two fields. This can be used to find genes that are encoded on the minus strand of a particular chromosome:

```
-if ChrLoc -equals X -and ChrStart -gt ChrStop
```

Object constraints will compare the string values of two named fields, and can look for internal inconsistencies between fields whose contents should (in most cases) be identical:

```
-if Chromosome -differs-from ChrLoc
```

The **-position** command restricts presentation of objects by relative location or index number:

```
-block Author -position last -sep ", " -element Lastname,Initials
```

The **-else** command can run alternative -element or **-lbl** instructions if the condition is not satisfied:

```
-if Strand -contains "-" -lbl "minus strand" -else -lbl "plus strand"
```

Copying XML Objects

The **-element "*"** construct prints the entirety of the current XML container, including all XML tags. (A period or a percent sign write ASN.1 or JSON, respectively. Tag names with leading, trailing, or internal underscores control item-specific formats, such as unquoted values or unnamed brackets.)

Automatic Format Conversion

Xtract can now detect and convert input data in JSON, text ASN.1, and GenBank/GenPept flatfile formats. Explicit transmute or shortcut commands are only needed to view the intermediate XML's field names or override the default conversion settings.

Advanced xtract Operations

Text Modification

For custom editing, target and replacement strings are set with **-reg** and **-exp**, respectively. Regular expression text matching and substitution is then performed on an element with a **-replace** command:

```
-reg "-" -exp "." -replace Phone
```

Value Substitution

External values can be loaded by reading a two-column, precomputed file or ad hoc conversion table with **-transform**, with lookup requested by applying **-translate** to an element:

```
xtract -transform accn-to-uid.txt  …  -translate Accession

xtract -transform <( echo -e "Genomic\t1\nCoding\t2\nProtein\t3\n" )  …
```

Multi-Step Transformations

Although xtract provides -element derivatives to do simple data manipulation, more complex tasks may be broken up into a series of simpler transformations, or "processing chains".

BioSample document summaries:

```
efetch -db biosample -id SAMN38051082 -format docsum |
```

store preferred qualifier names in a "harmonized_name" XML attribute:

```
<Attribute harmonized_name="strain">BALB/c</Attribute>
<Attribute harmonized_name="isolate">Mtb infected Spleen MZB-2</Attribute>
<Attribute harmonized_name="geo_loc_name">Singapore</Attribute>
```

Piping the data to the first xtract command, and using the "**@**" sign to select the attribute:

```
xtract -rec BioSampleInfo -pattern DocumentSummary \
  -wrp Accession -element Accession \
  -group Attribute -if @harmonized_name \
    -TAG -lower @harmonized_name -wrp "&TAG" -element Attribute |
```

generates an intermediate form, with XML tag names taken from the original XML attributes:

```
<BioSampleInfo>
  <Accession>SAMN38051082</Accession>
  <strain>BALB/c</strain>
  <isolate>Mtb infected Spleen MZB-2</isolate>
  <geo_loc_name>Singapore</geo_loc_name>
  …
```

Desired fields can then be selected by name in the second xtract command:

```
xtract -pattern BioSampleInfo -def "-" -first Accession \
  geo_loc_name strain isolate
```

Parsing XML Elements

GFF3 lines can be converted to XML with transmute **-t2x** (or **tbl2xml**), which gets field names from the remaining arguments (or, with **-header**, from the first row of the input data):

```
tbl2xml -rec Rec SeqID Source Type Start End Score Strand Phase Attributes |
```

The Attributes element consist of multiple clauses that are delimited by semicolons:

```
<Attributes>ID=gene-XXXX_016554;Name=XXXX_016554; … </Attributes>
```

Piping to xtract creates a container, and uses **-with** and **-split** to parse the Attribute field:

```
xtract -pattern Rec -pkg Context -wrp Item -with ";" -split Attributes |
```

into individual "**`tag=value`**" elements within the new package:

```
<Context>
  <Item>ID=gene-XXXX_016554</Item>
  <Item>Name=XXXX_016554</Item>
  …
```

A second xtract can now use element prefix and suffix removal to isolate the tag and value. The tag is saved in a variable to become the name of a new XML object in which the value is stored:

```
xtract -pattern Context -pkg Attributes -block Item \
  -TAG "Item[|=]" -wrp "&TAG" -element "Item[=|]" |
```

The final XML can be queried by field names that were extracted from attribute tags:

```
<Attributes>
  <ID>gene-XXXX_016554</ID>
  <Name>XXXX_016554</Name>
  …
```

XML Namespaces

Namespace prefixes are followed by a colon, while a leading colon matches any prefix:

```
nquire -url https://webservice.wikipathways.org getPathway -pwId WP455 |
xtract -pattern "ns1:getPathwayResponse" -decode ":gpml" |
```

The embedded Graphical Pathway Markup Language object can then be processed:

```
xtract -pattern Pathway -block Xref \
  -if @Database -equals "Entrez Gene" -tab "\n" -element @ID
```

Record Filtering

The **filter-record** script can extract an arbitrary range of text records using **-min** and **-max**:

```
filter-record -pattern "<PubmedArticle>" -min 13 -max 24
```

It also has arguments to **-require** or **-exclude** specific text content.

Partitioning Sets of Records

Given a sample record set containing around 1000 articles:

```
esearch -db pubmed -query "conotoxin [TITL]" | efetch -format xml |
```

you can sort the data and keep the results in a single file, or distribute the records into separate files.

To sort the records by a particular field, pass the element name to xtract **-sort-fwd** or **-sort-rev**:

```
xtract -pattern PubmedArticle -sort-rev MedlineCitation/PMID
```

To partition the records, give the number of records per output file to xtract **-split-by-num**:

```
xtract -set PubmedArticleSet -pattern PubmedArticle \
  -split-by-num 100 -prefix subset -suffix xml
```

or use the maximum size for each output file as the argument to xtract **-split-by-len**:

```
xtract -set PubmedArticleSet -pattern PubmedArticle \
  -split-by-len 1000000 -prefix subset -suffix xml
```

Output file names will be of the form "`subset001.xml`", "`subset002.xml`", etc.

Viewing an XML Hierarchy

Piping a PubmedArticle XML object to xtract **-outline**:

```
esearch -db pubmed -query "Boguski MS [AUTH] AND Boguski RM [AUTH]" |
efetch -format xml |
xtract -outline 3
```

will give an indented overview of the XML hierarchy, optionally limited to a specified object depth:

```
PubmedArticle
  MedlineCitation
    PMID
    DateCompleted
    DateRevised
    Article
    MedlineJournalInfo
  PubmedData
    History
    PublicationStatus
    ArticleIdList
    ReferenceList
```

Using xtract **-synopsis** or **-contour** will show the full paths to all nodes or just the terminal (leaf) nodes, respectively. Piping those results to **sort-uniq-count** will produce a table of unique paths.

Using a **caret** as an element argument prefix will print its absolute depth in the XML hierarchy:

```
xtract -pattern PubmedArticle -element "^PMID"
```

Expanding Horizons

The nquire program uses command-line arguments to obtain data from external RESTful, CGI, or FTP servers. (Xtract can read JSON, ASN.1, and GenBank formats directly, but previously-required conversion commands - now for inspecting XML or overriding defaults - are shown below in light text.)

JSON Arrays

Human β-globin information from a Scripps Research data integration project (PMID 23175613):

```
  nquire -get https://mygene.info/v3 gene 3043 | json2xml |
```

contains a multi-dimensional JavaScript Object Notation array of exon coordinates:

```
  "position": [
    [ 5225463, 5225726 ],
    [ 5226576, 5226799 ],
    [ 5226929, 5227071 ]
  ],
  "strand": -1,
```

Conversion to XML assigns distinct tag names to each level with the **-nest element** default:

```
  <position>
    <position_E>5225463</position_E>
    <position_E>5225726</position_E>
  </position>
  …
```

Heterogeneous Data

A query for the human green-sensitive opsin gene:

```
  nquire -get https://mygene.info/v3/gene/2652 | json2xml |
```

returns data containing a heterogeneous mixture of objects in the pathway section:

```
  <pathway>
    <reactome>
      <id>R-HSA-162582</id>
      <name>Signal Transduction</name>
    </reactome>
    …
    <wikipathways>
      <id>WP455</id>
      <name>GPCRs, Class A Rhodopsin-like</name>
    </wikipathways>
  </pathway>
```

The **parent / star** construct is used to visit the individual components of a parent object without needing to explicitly specify their names. For printing, the name of a child object is indicated by a **question mark**, while a **tilde** would return the anonymous object's value:

```
  xtract -pattern opt -group "pathway/*" \
    -pfc "\n" -element "?,name,id"
```

This displays a table of pathway database references:

```
reactome       Signal Transduction                R-HSA-162582
reactome       Disease                            R-HSA-1643685
…
reactome       Diseases of the neuronal system    R-HSA-9675143
wikipathways   GPCRs, Class A Rhodopsin-like      WP455
```

Exhaustive Exploration

PubMed Central full text records consist of recursive "sec" objects that contain section title and paragraph text elements. A record obtained in XML format:

```
esearch -db pmc -query "Kitts PA [AUTH] AND type strains [TITL]" |
efetch -format xml |
```

is first processed by xtract **-mask**, to remove complex formula and table mark-up instructions:

```
xtract -mixed -pattern article -mask "table-wrap,alternatives,inline-formula,\
disp-formula,pub-history,related-article,list-item,fig" -element "*" |
```

Extracting the title and abstract fields is straightforward:

```
xtract -mixed -rec PMCData -pattern article \
  -division front \
    -wrp Title -prose title-group/article-title \
    -wrp Abstract -prose abstract/p \
```

The full text is obtained by exhaustive exploration using the **parent / double star** construct:

```
  -division "body/**" \
```

Multiple -if clauses use a **question mark** to test the child object's name and select only paragraph and title elements. A **caret** returns the nesting depth of the element:

```
    -if "?" -equals "p" -pkg Paragraph \
      -wrp Level -element "^" -wrp Text -prose p \
    -if "?" -equals "title" -pkg Section \
      -wrp Level -element "^" -wrp Title -prose sec/title
```

Like **-else**, the multiple -if convention can only be used for element extraction operations. Multiple explorations below the double star would require the use of individual **-group "*"** commands.

```
<PMCData>
  <Title>Collection and curation of prokaryotic genome assemblies … </Title>
  <Abstract>The public sequence databases are entrusted with the … </Abstract>
  …
  <Section>
    <Level>5</Level>
    <Title>Assemblies not used as types</Title>
  </Section>
  <Paragraph>
    <Level>5</Level>
    <Text>NCBI evaluates all assemblies, including type assemblies … </Text>
  </Paragraph>
  …
```

GenBank Download

The most recent GenBank virus release file can be downloaded from NCBI servers:

```
nquire -lst ftp.ncbi.nlm.nih.gov genbank |
grep "^gbvrl" | grep ".seq.gz" | sort -V |
tail -n 1 | skip-if-file-exists |
nquire -dwn ftp.ncbi.nlm.nih.gov genbank
```

GenBank flatfile records can be selected by organism name or taxon identifier, or by presence or absence of an arbitrary text string, with transmute **-gbf** (or **filter-genbank**):

```
gunzip -c *.seq.gz | filter-genbank -taxid 11292 |
```

The full set of filter-genbank arguments are **-accession**, **-accessions**, **-taxid**, **-taxids**, **-organism**, **-truncate**, **-exclude**, **-require**, **-min**, and **-max**.

While GenBank format can be read directly by xtract, explicit conversion to INSDSeq XML with transmute **-g2x** (or **gbf2xml**) may be up to three times faster for large sets of records:

```
gbf2xml |
```

Feature location intervals and underlying sequences of individual coding regions are then obtained by:

```
xtract -insd CDS gene product feat_location sub_sequence
```

Table Operations

Tab-delimited tables can be piped through **filter-columns** for numeric and substring matching:

```
filter-columns '10 <= $2 && $2 <= 30 && $5 ~ peptide'
```

and sent to **print-columns** for flexible modification of text and numbers in the final output:

```
print-columns '$1, $2+1, $3+$4-1, "\042" $5 "\042", tolower($6), total += $2'
```

Both scripts are simple front-ends to the **awk** data manipulation utility.

In order to avoid misinterpretation of dollar signs by the Unix shell, both scripts require their argument to be inside apostrophes instead of double quotation marks. The standard NF and NR awk variables can also be used, as can YR (for year) and DT (for date in YYYY-MM-DD format).

Along with **sort-table**, these scripts allow novice users to do useful things on tabular data without first having to know arcane setup details (such as how to specify the tab character as the column separator).

The **align-columns** script has several arguments to reformat a table and make it easier to interpret.

The **intersect-uids**, **combine-uids**, and **exclude-uids** scripts merge unique identifier files with Boolean operations (AND, OR, and NOT, respectively). Use **compare-uids** to see UID differences.

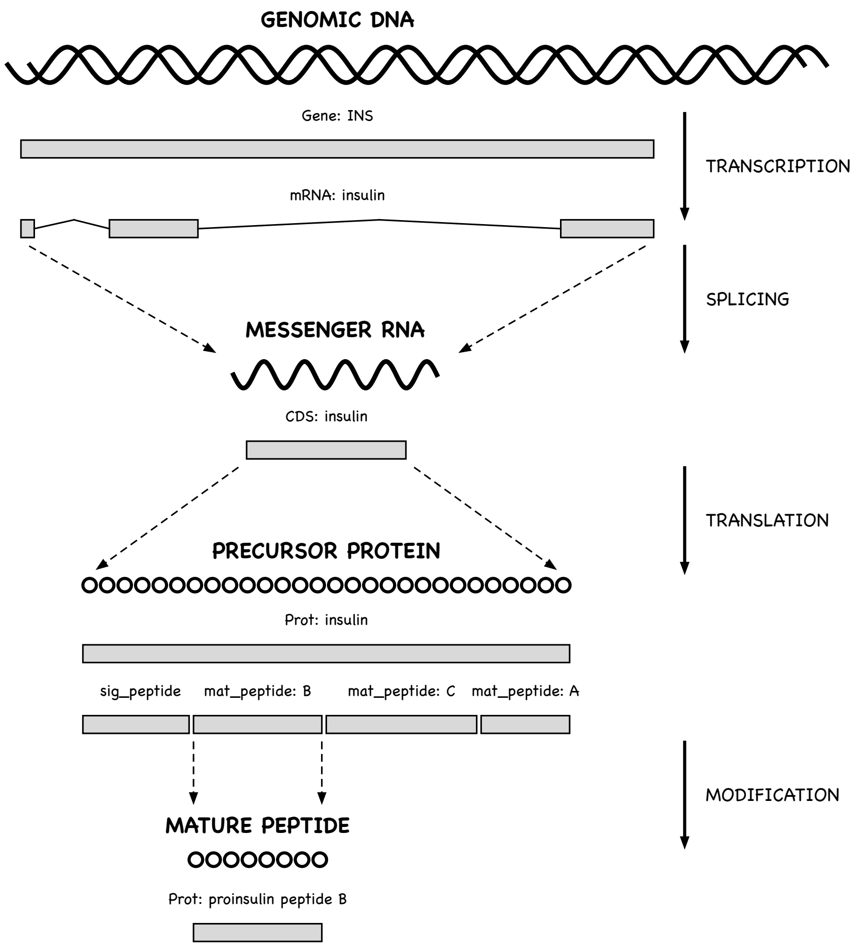
Biological Data in Entrez

EDirect provides additional functions, scripts, and exploration constructs to simplify the extraction of complex data obtained from the interconnected Entrez biological databases.

Sequence Qualifiers

The NCBI data model for sequence records (PMID 11449725) is based on the central dogma of molecular biology. Sequences, including genomic DNA, messenger RNAs, and protein products, are "instantiated" with the actual sequence letters, and are assigned accession numbers for reference.

Features contain information about the biology of a region, including the transformations involved in gene expression. Qualifiers store specific details about a feature, such as the name of the gene, genetic code used for protein translation, or accession of the product sequence.



A gene feature indicates the location of a heritable region of nucleic acid that confers a measurable phenotype. An mRNA feature on genomic DNA represents the exonic and untranslated regions that remain after message transcription and intron splicing. A coding region (CDS) feature has a product reference to the translated protein sequence record.

As a convenience for exploring sequence records, the xtract **-insd** helper function generates the appropriate nested extraction commands from feature and qualifier names on the command line. (Two computed qualifiers, **feat_location** and **sub_sequence**, are also supported.)

Snail Venom Peptide Sequences

A search for cone snail venom mature peptides:

```
esearch -db protein -query "conotoxin" -feature mat_peptide |
efetch -format gpc |
xtract -insd complete mat_peptide "%peptide" product mol_wt peptide |
grep -i conotoxin | sort-table -u -k 2,2n
```

uses the xtract -insd function to print the accession number, mature peptide length, product name, calculated molecular weight, and amino acid sequence for a sample of neurotoxic peptides:

```
ADB43131.1    15    conotoxin Cal 1b      1708    LCCKRHHGCHPCGRT
ADB43128.1    16    conotoxin Cal 5.1     1829    DPAPCCQHPIETCCRR
AIC77105.1    17    conotoxin Lt1.4       1705    GCCSHPACDVNNPDICG
ADB43129.1    18    conotoxin Cal 5.2     2008    MIQRSQCCAVKKNCCHVG
ADD97803.1    20    conotoxin Cal 1.2     2206    AGCCPTIMYKTGACRTNRCR
AIC77085.1    21    conotoxin Bt14.8      2574    NECDNCMRSFCSMIYEKCRLK
ADB43125.1    22    conotoxin Cal 14.2    2157    GCPADCPNTCDSSNKCSPGFPG
AIC77154.1    23    conotoxin Bt14.19     2578    VREKDCPPHPVPGMHKCVCLKTC
…
```

Recursive Taxonomy Data

To accommodate recursively-defined data, entry to an internal object is blocked when its name matches the current exploration container. The **star / child** construct bypasses the search constraint to allow controlled descent one level at a time. Using **double star / child** recursively visits every object regardless of depth, and can flatten a complex structure into a linear set of elements in a single step:

```
efetch -db taxonomy -id 9606 -format xml |
xtract -pattern Taxon \
  -first TaxId -tab "\n" -element ScientificName \
  -block "**/Taxon" -if Rank -is-not "no rank" -and Rank -excludes "root" \
    -tab "\n" -element Rank,ScientificName
```

This prints all of the individual internal lineage nodes:

```
9606          Homo sapiens
domain        Eukaryota
clade         Opisthokonta
kingdom       Metazoa
clade         Eumetazoa
clade         Bilateria
clade         Deuterostomia
phylum        Chordata
subphylum     Craniata
clade         Vertebrata
clade         Gnathostomata
clade         Teleostomi
clade         Euteleostomi
superclass    Sarcopterygii
clade         Dipnotetrapodomorpha
clade         Tetrapoda
clade         Amniota
class         Mammalia
…
```

Genes in a Region

Records for protein-coding genes on the human X chromosome are retrieved by running:

```
esearch -db gene -query "Homo sapiens [ORGN] AND X [CHR]" |
efilter -status alive -type coding | efetch -format docsum |
```

Gene names and chromosomal positions are extracted by piping the records to:

```
xtract -pattern DocumentSummary -NAME Name -DESC Description \
  -block GenomicInfoType -if ChrLoc -equals X \
    -min ChrStart,ChrStop -element "&NAME" "&DESC" |
```

The -if statement eliminates coordinates from pseudoautosomal gene copies present on the Y chromosome telomeres. Results can now be sorted by position, and then filtered and partitioned:

```
sort-table -k 1,1n | cut -f 2- |
grep -v pseudogene | grep -v uncharacterized | grep -v hypothetical |
between-two-genes AMER1 FAAH2
```

to produce an ordered table of known genes located between two markers flanking the centromere:

```
FAAH2      fatty acid amide hydrolase 2
SPIN2A     spindlin family member 2A
ZXDB       zinc finger X-linked duplicated B
NLRP2B     NLR family pyrin domain containing 2B
ZXDA       zinc finger X-linked duplicated A
SPIN4      spindlin family member 4
ARHGEF9    Cdc42 guanine nucleotide exchange factor 9
AMER1      APC membrane recruitment protein 1
```

SNP-Modified Product Pairs

Single nucleotide polymorphisms can represent different substitutions at the same position, but variation records do not explicitly match a specific CDS modification to its altered protein product:

```
efetch -db snp -id 11549407 -format docsum |
```

The hgvs2spdi script converts 1-based HGVS data ("`NM_000518.5:c.118C>T`") into 0-based SPDI format ("`NM_000518.5:167:C:T`"). For SNPs on cDNA transcripts the position is CDS-relative, and the script retrieves the GenBank record in order to calculate the absolute sequence offset:

```
snp2hgvs | hgvs2spdi | spdi2tbl | tbl2prod
```

The tbl2prod step translates the coding region locations (after nucleotide substitution), and sorts them with protein sequences (after residue replacement) to produce adjacent matching CDS/protein pairs:

```
rs11549407    NM_000518.5:167:C:T    MVHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWT*R …
rs11549407    NP_000509.1:39:Q:*     MVHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWT*R …
rs11549407    NM_000518.5:167:C:G    MVHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWTER …
rs11549407    NP_000509.1:39:Q:E     MVHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWTER …
rs11549407    NM_000518.5:167:C:A    MVHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWTKR …
rs11549407    NP_000509.1:39:Q:K     MVHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWTKR …
…
```

Sequence Analysis

EDirect sequence processing functions are provided by the transmute program. They can handle huge sequences as normal strings, without requiring any special coding techniques or custom data structures.

Reverse Complementation

A GenBank sequence can be converted to FASTA, reverse-complemented, and printed as FASTA with:

```
efetch -db nuccore -id U00096 -format gb |
gbf2fsa | transmute -revcomp | transmute -fasta -width 50
```

Sequence Editing

The pBR322 cloning vector is a circular plasmid with unique restriction sites in two antibiotic resistance genes. The transmute **-replace** function introduces a second BamHI restriction enzyme recognition site (based on a site-directed mutagenesis experiment) by modifying two bases in the rop gene:

```
efetch -db nuccore -id J01749 -format fasta |
transmute -replace -offset 1907 -delete GG -insert TC |
…
```

Pattern Searching

The modified sequence from above is then passed to transmute **-search**, which takes a list of sequence patterns (with optional labels) and uses a finite-state algorithm to simultaneously search for all patterns:

```
…
transmute -search -circular GGATCC:BamHI GAATTC:EcoRI CTGCAG:PstI |
align-columns -g 4 -a rl
```

The (0-based) starting positions and restriction enzyme names for each match are printed in a table:

```
 374    BamHI
1904    BamHI
3606    PstI
4358    EcoRI
```

The **disambiguate-nucleotides** and **systematic-mutations** scripts can generate all possible single-base substitutions in a pattern for use in a relaxed-stringency search.

Six-Frame Translation

Protein translation uses a finite-state machine to slide a triplet-codon window along the sequence. State values are offsets into an amino acid lookup table. The next-state and reverse-complement transition tables, and an amino acid table for every genetic code, are all precomputed. A nucleotide sequence can then be translated in all six possible reading frames by an extremely fast loop of indexed lookups:

```
efetch -db nuccore -id U54469.1 -format fasta |
transmute -cds2prot -gcode 1 -all |
```

This produces regions of translated protein separated by asterisks representing stop codons:

```
>U54469.1-1+
RLLGFYNISQ*QAFPELPCSTIDSCLWPPKSQT*LKN*IIRIIIKPSNLR …
>U54469.1-2+
GCLGFITSVSDRHFQSCPVQQSIAAFGHQNPKLN*RIK*FE**LSPVTYA …
…
```

The FASTA is then converted to XML, and xtract **-pept** splits the translated frames at each stop codon indicator (asterisk), sequence gap (hyphen), or ambiguous translation ("X") character:

```
fsa2xml |
xtract -rec FASTA -pattern FASTA \
  -group "FASTA/*" -element "*" \
  -group FASTA -wrp FRAG -pept Seq
```

That appends individual peptide fragments to the original fields and creates a new XML record:

```
…
<FRAG>rllgfynisq</FRAG>
<FRAG>qafpelpcstidsclwppksqt</FRAG>
<FRAG>lkn</FRAG>
…
```

Feature Locations

A table of coding region locations and gene names can be saved directly as XML with xtract **-insdx**:

```
efetch -db nuccore -id NC_000011 -format gb -style master |
xtract -insdx CDS gene feat_location > cds_loc.xml
```

The human β-globin coding region location is retrieved and stored in a Unix shell variable with:

```
loc=$( xtract -input cds_loc.xml -pattern Rec \
         -if gene -equals HBB -element feat_location )
```

Location intervals are shown in biological order, where start is greater than stop on the minus strand:

```
5227021..5226930,5226799..5226577,5225726..5225598
```

Sequence Transformations

The consequences of a genomic SNP can be reproduced with transmute functions: **-replace** applies the substitution, **-extract** uses the location intervals from above to isolate the altered coding sequence, and **-cds2prot** translates the modified CDS into protein with the designated genetic code:

```
efetch -db nuccore -id NC_000011 -format fasta |
transmute -replace -offset 5226773 -delete G -insert A |
transmute -extract -1-based "$loc" |
transmute -cds2prot -gcode 1 -frame 0 -every -trim
```

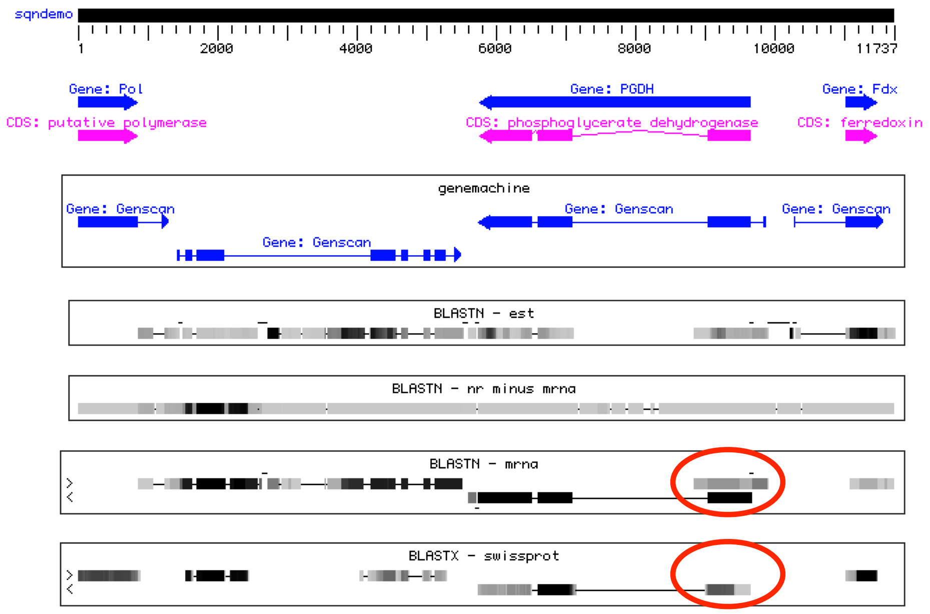
The genomic G to A transition on human chromosome 11 corresponds to the C to T substitution on the minus-strand-encoded β-globin mRNA in the earlier SNP example:

```
MVHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWT*R …
```

Alignment Excavation

NHGRI's GeneMachine program could run several gene prediction algorithms and BLAST searches, combining the results in a richly-annotated sequence record.

NCBI's Sequin program could read the record and display the combined ranges of separate alignments in a "smear" that made it possible to see an emerging image of the underlying gene structures:



While interesting patterns (e.g., possible antisense regulation transcripts) might occasionally be noticed by visual inspection, examination by eye is tedious, error-prone, and does not scale to chromosome size. However, the process of converting raw BLAST alignment data in ASN.1 format to a computable, tab-delimited data table, is easily automated with a chain of simple xtract commands.

The first xtract reads the ASN.1 sequence record, converts it internally to XML, and uses **-select** to conditionally filter out everything but the BLASTN-mRNA annotation:

```
xtract -pattern annot_E -select label/str -equals "BLASTN - mrna" |
```

For each alignment, the segment start positions (and strands) are implicitly paired - one for the genomic sequence followed by one for the mRNA sequence. One particular alignment has three segments (here showing the original ASN.1 on the right, and the automatically-derived XML on the left):

```
<starts>                                                              starts {
  <starts_E>5613</starts_E>                                             5613,
  <starts_E>842</starts_E>                                              842,
  <starts_E>-1</starts_E>                                               -1,
  <starts_E>961</starts_E>                                              961,
  <starts_E>5599</starts_E>                                             5599,
  <starts_E>962</starts_E>                                              962
</starts>                                                             },
```

The "-1" start position indicates an internal gap in the genomic assembly relative to the cloned mRNA sequence, and the length of "1" (below) says that it is a single base deletion. The "underscore-E" suffix indicates that the object was derived from an unlabeled element within a named ASN.1 sequence or set.

A second xtract explores each alignment, using **-odd** to get the first (genomic) value in each start pair:

```
  … -pattern align_E -block starts -wrp Start -odd starts_E … |
```

Each element is wrapped with a descriptive name, and the fields are packed into a novel structure:

```
  <Start>5613</Start>
  <Start>-1</Start>
  <Start>5599</Start>
  <Length>119</Length>
  <Length>1</Length>
  <Length>14</Length>
```

The range of an alignment can be calculated from the first and last start positions, and either the first or the last segment length value, depending upon the strand. Those values are obtained using positional **-first** and **-last** commands:

```
  … -wrp FirstPos -first Start -wrp LastPos -last Start … |
```

and the new fields are packaged in a different ad hoc XML structure:

```
  <FirstPos>5613</FirstPos>
  <LastPos>5599</LastPos>
  <FirstLen>119</FirstLen>
  <LastLen>14</LastLen>
  <Strand>minus</Strand>
```

A final xtract command performs two parallel explorations, one for each strand:

```
  …
  -block Rec -if Strand -equals plus -def "-" \
    -element Accn Score FirstPos LastPos LastLen Strand \
  -block Rec -if Strand -equals minus -def "-" \
    -element Accn Score LastPos FirstPos FirstLen Strand |
```

producing a tab-delimited intermediate table:

```
  AY046051.1    126    5599    5613    119    minus
```

A print-columns command calculates the end position and total length of each alignment, the results are sorted by strand and genomic location, and the "0-based" position fields are incremented to produce "1-based" final values:

```
  print-columns '$1, $2, $3, $4 + $5 - 1, $4 + $5 - $3, $6' |
  sort-table -k 6,6fr -k 3,3n -k 4,4nr -k 1,1f |
  print-columns '$1, $2, $3 + 1, $4 + 1, $5, $6'
```

The adjusted table values, suitable for further computation:

```
  AY046051.1    126    5600    5732    133    minus
```

now match the coordinates in the standard BLAST report:

```
  5600  AACTTAATACATAA-GTTGGTAGCCCACAATGTGAAAGATTAAATTAAAACTCATCCATT  5658
        |||||||||||||| ||||||||||||||||||||||||||||||||||||||||||||
  976   AACTTAATACATAATGTTGGTAGCCCACAATGTGAAAGATTAAATTAAAACTCATCCATT  917
                                   …
```

Local PubMed Archive

Fetching data from Entrez works well when a few thousand records are needed, but it does not scale for much larger sets of data, where the time it takes to download becomes a limiting factor.

Local Record Cache

EDirect can now preload over 39 million live PubMed records onto an inexpensive external 1 TB solid-state drive as individual files for rapid retrieval. For example, PMID 2539356 would be stored at:

```
/pubmed/Archive/02/53/93/2539356.xml.gz
```

using a hierarchy of folders to organize the data for random access to any record.

The local archive is a completely self-contained turnkey product, with no need to download, configure, and maintain complicated third-party database software.

Set an environment variable in your configuration file(s) to reference a section of your external drive:

```
export EDIRECT_LOCAL_ARCHIVE=/Volumes/external_drive_name/
```

Then run **archive-pubmed** to download the PubMed release files and save each record on the drive. The initial download may take several hours, depending on your network connection, with initial archiving taking another few hours. Subsequent updates are incremental, and should finish in minutes.

Retrieving and decompressing over 135,000 PubMed records from the local archive with **xfetch**:

```
esearch -db pubmed -query "PNAS [JOUR]" -pub abstract | xfetch |
```

takes about 70 seconds. Retrieving those records from NCBI's network service, with efetch -format xml, would take around 40 minutes.

Even modest sets of PubMed query results can benefit from using the local cache. A reverse citation lookup on 191 papers requires 5 seconds to match 9784 subsequent articles:

```
esearch -db pubmed -query "Cozzarelli NR [AUTH]" | elink -cited |
```

Piping to xfetch **-stream** returns the original compressed records, for a roughly four-fold size reduction of data to send over a network. This would be followed by decompression on the client with **gunzip -c**.

Alternatively, using xfetch **-turbo** precedes each record with an object containing its size in bytes:

```
<NEXT_RECORD_SIZE>4027</NEXT_RECORD_SIZE>
```

Piping to xtract **-turbo** then approximately doubles the speed of the rate-limiting record partitioning step, which otherwise uses the Boyer-Moore-Horspool fast string search algorithm.

All of the xfetch forms take about one second to retrieve the records from the archive. Fetching from the network service would extend the 6 second running time to over 2 minutes.

Local Search Index

A similar strategy was used to create a local information retrieval system suitable for large data mining queries. Run **archive-pubmed -index** to populate retrieval index files from records stored in the local archive. The initial indexing will also take a few hours. Since PubMed updates are released once per day, it may be convenient to schedule reindexing to start in the late evening and run during the night.

For PubMed titles and primary abstracts, the indexing process deletes hyphens after specific prefixes, removes accents and diacritical marks, splits words at punctuation characters, corrects encoding artifacts, and spells out Greek letters for easier searching on scientific terms. It then prepares inverted indices with term positions, and uses them to build distributed term lists and postings files.

For example, the term list that includes "cancer" in the title or abstract would be located at:

```
/pubmed/Postings/TIAB/c/a/n/c/canc.TIAB.trm
```

A query on cancer thus only needs to load a very small subset of the total index. The software supports expression evaluation, wildcard truncation, phrase queries, proximity searches, and partial matches.

The **xinfo**, **xsearch**, **xlink**, and **xfilter** scripts provide access to the local search system.

Names of indexed fields, all terms for a given field, and terms plus record counts, are shown by:

```
xinfo -fields

xinfo -terms SUBH

xinfo -totals PROP
```

Terms are truncated with a trailing asterisk, and can be expanded to show individual postings counts:

```
xinfo -count "catabolite repress*"

xinfo -counts "catabolite repress*"
```

Query evaluation includes Boolean operations and parenthetical expressions:

```
xsearch -query "(literacy AND numeracy) NOT (adolescent OR child)"
```

Adjacent words in title or abstract fields are treated as a contiguous phrase:

```
xsearch -query "selective serotonin reuptake inhibitor [TITL]"
```

Each plus sign will replace a single word inside a phrase, and runs of tildes indicate the maximum distance between sequential phrases:

```
xsearch -query "vitamin c + + common cold"

xsearch -query "vitamin c ~ ~ common cold"
```

Ranked partial term matching is available in any field with **-match**:

```
xsearch -match "tn3 transposition immunity [PAIR]" | just-top-hits 1
```

An exact substring match, without special processing of Boolean operators or indexed field names, can be obtained with -title (on the article title) or -exact (on the title or abstract):

```
  xsearch -title "Genetic Control of Biochemical Reactions in Neurospora."
```

MeSH identifier code, MeSH hierarchy key, and year of publication are also indexed, and MESH field queries are supported by internally mapping to the appropriate CODE or TREE entries:

```
  xsearch -db pubmed -query "C14.907.617.812* [TREE] AND 2015:2019 [YEAR]"
```

PMIDs processed through an external source can be reintroduced to a local query pipeline with xfilter:

```
  | xfilter -db pubmed -query "Monoamine Oxidase [MESH] AND Deficiency [SUBH]" |
```

Data Analysis and Visualization

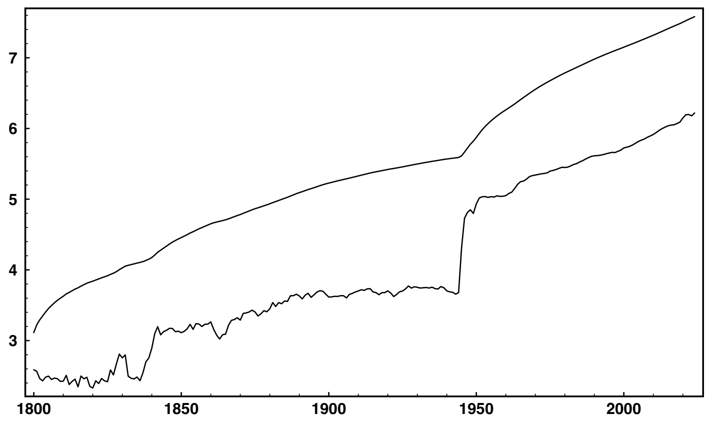
All query commands return a structured message containing the database name and a list of UIDs, which can be piped directly to xfetch to retrieve the uncompressed records. For example:

```
  xsearch -db pubmed -query "selective serotonin ~ ~ ~ reuptake inhibit*" |
  xfetch | xtract -pattern PubmedArticle -num AuthorList/Author |
  sort-uniq-count -n | reorder-columns 2 1 | align-columns -g 4 -a lr
```

performs a proximity search with dynamic wildcard expansion (matching phrases like "selective serotonin and norepinephrine reuptake inhibitors") and fetches 14,978 PubMed records from the local archive. It then counts the number of authors for each paper (a consortium is treated as a single author), printing a frequency table of the number of papers per number of authors.

The cumulative size of PubMed can be calculated with a running sum of the annual record counts. Exponential growth over time will appear as a roughly linear curve on a semi-logarithmic graph:

```
  xinfo -db pubmed -totals YEAR | print-columns '$2, $1, total += $1' |
  print-columns '$1, log($2)/log(10), log($3)/log(10)' |
  filter-columns '$1 >= 1800 && $1 < YR' | xy-plot annual-and-cumulative.png
```



The sharp jump after World War II was caused by several factors, including the release of declassified papers, a policy of expanding biomedical research in postwar America, and the introduction of computers that could keep up with the indexing of articles from a broader range of subjects.

Natural Language Processing

NLM's Biomedical Text Mining Group performs computational analysis to extract chemical, disease, and gene references from article contents (PMID 31114887). NLM indexing of PubMed records assigns Gene Reference into Function (GeneRIF) mappings (PMID 14728215).

Running **archive-nlmnlp -index** periodically (monthly) will automatically refresh any out-of-date support files and then index the connections in CHEM, DISZ, GENE, GRIF, GSYN, and PREF fields:

```
xinfo -terms DISZ | grep -i Raynaud
```

```
xinfo -counts "Raynaud* [DISZ]"
```

```
xsearch -query "Raynaud Disease [DISZ]"
```

Following Citation Links

Running **archive-nihocc -index** will download the latest NIH Open Citation Collection monthly release and build CITED and CITES indices, the local equivalent of elink -cited and -cites commands.

Citation links are retrieved by piping one or more PMIDs to xlink **-target**:

```
xsearch -db pubmed -query "Haselkorn R* [AUTH]" |
xlink -target CITED |
```

This returns PMIDs for 8356 articles that cite the original 225 papers. The results are then restricted to a range of recent years, and those records are fetched. The xtract **-histogram** shortcut builds a journal frequency table from the subsequent articles:

```
xfilter -query "2020:2025 [YEAR]" | xfetch |
xtract -pattern PubmedArticle -histogram Journal/ISOAbbreviation |
sort-table -nr | head -n 10
```

The **archive-pids -index** command reads the PubMed local archive's incremental inverted index files, and builds a PMCID index that allows xlink to return PubMed Central identifiers from PMIDs:

```
xlink -db pubmed -id 12372140 -target PMCID | xfetch
```

The xlink **-ranked** flag produces the same ranked output format as xsearch **-match**.

Additional Experimental Archives

Running **archive-pmc -index** downloads PMC release files, and collects primary author names, citation details, section titles, and full-text paragraphs. It then converts them to the more tractable PMCInfo object, and builds an archive from those derived records.

Similarly, **archive-taxonomy -index** archives novel records assembled from NCBI taxonomy data tables retrieved from the FTP site.

A new database is ready for use once its archiving script has finished all downloading, conversion, validation, caching, indexing, inversion, merging, and postings steps.

Adding a New Local Archive

New database domains can be added using records obtained from external public data resources, or with private information taken from a laboratory's research results or a hospital's clinical trial data.

Archive Scripts

The first step is to create a one-line archive script that sends command-line arguments to **xbuild**. A sample database of peptide sequence fragments is built with **archive-peptide**:

```
  xbuild -db peptide -project peptide "$@"
```

A primary database has the same values for the **-db** and **-project** arguments. It may import structured records, or it may assemble records from tabular data. These may be cached directly, or it may store simpler records that are derived from the originals.

Secondary projects add new indexed fields or links to primary databases. The **archive-nihocc** script:

```
  xbuild -db pubmed -project nihocc "$@"
```

adds reciprocal citation reference links to the pubmed database, but does not create its own records.

Project Directories

A new database or project next requires adding a project directory within the edirect/external folder. The **edirect /extern /peptide** subfolder contains several helper scripts. By convention, their names are composed of a formal step in the build process, a hyphen, and the project name:

```
  download-peptide
  populate-peptide
  index-peptide
```

A single configuration file named for the project is also present:

```
  peptide.ini
```

Secondary project files go in an **edirect /extern /{database}-{project}** subfolder.

Configuration Files

Project-specific configuration files guide the build process. The **peptide.ini** file provides the names of fields to be indexed, as well as information used by xfetch to retrieve the derived records:

```
  [info]
  db=peptide
  project=peptide

  [posting]
  fields="ACCN ORGN PENT PROD UID"

  [fetch]
  name=PeptInfo
  set=PeptInfoSet
```

In contrast, the **nihocc.ini** configuration file adds link flags in the merge and posting sections:

```
[info]
db=pubmed
project=nihocc

[merge]
link=true

[posting]
fields="CITED CITES"
link=true
```

while **pubmed.ini** has a links section with link-specific target databases, for use by xlink:

```
…
[links]
CITED=pubmed
CITES=pubmed
PMCID=pmc
```

"Download" Helper

The actual helper files needed for any given project depends on the source of the records, and on any accessory data files that must be retrieved separately. For each archive step, the helper file is "sourced" by an intermediate script, which executes it in the context of properly initialized variables.

An abridged version of the **download-peptide** file is shown below:

```
FilterByDivision() {

  dir="$1"

  if [ "$dir" = "archaea" ]
  then
    grep "wp_"
  elif [ "$dir" = "bacteria" ] || [ "$dir" = "plasmid" ]
  then
    grep "."
  else
    grep -v "wp_"
  fi
}

if [ -d "${sourceBase}" ]
then
  cd "${sourceBase}"

  if [ -n "$custom" ]
  then
    nquire -lst ftp.ncbi.nih.gov/refseq/release "$custom" < /dev/null |
    grep gpff | FilterByDivision "$custom" | sort -V | skip-if-file-exists |
    while read fl
    do
      echo "$fl" | nquire -asp ftp.ncbi.nih.gov/refseq/release "$custom"
    done
  fi
fi
```

For this example, a RefSeq division folder is passed by archive-peptide using the **-custom** argument:

```
  archive-peptide -custom "viral"
```

In the real download-peptide file, the custom variable contains one or more division abbreviations:

```
  archive-peptide -custom "ARC FUN PRO VRL"
```

Any single step in the build process can be executed by using the **-step** argument:

```
  archive-peptide -step DWN
```

For the full download-peptide helper file, running just the download step without a -custom argument prints a table displaying the division abbreviation, actual subfolder name, number of files, and total number of bytes, for each taxonomic division. Final formatting was done by align-columns:

```
  ARC    archaea                  9      1,930,250,045
  BCT    bacteria               810    177,206,022,924
  FUN    fungi                   35      4,004,535,659
  INV    invertebrate            91      6,044,182,842
  MAM    vertebrate_mammalian   120      5,867,723,698
  MIT    mitochondrion            1        113,489,001
  PLN    plant                   67      4,339,393,382
  PRO    protozoa                 7        786,468,826
  PSM    plasmid                  9      1,359,056,776
  PST    plastid                  3        512,714,453
  VRL    viral                    1        226,768,975
  VRT    vertebrate_other       173     10,509,364,620
```

The counts refer to data files run through the FilterByDivision function. This favors the non-redundant "WP" protein sequence files for archaea and bacteria, which were introduced for prokaryotes in 2013.

"Populate" Helper

Unless the original source records are perfect for your intended use, you can design a more convenient structure containing just the information you want.

A modified excerpt from **populate-peptide**:

```
  base=${fl%.gz}
  if [ ! -f "${archiveBase}/Sentinels/$base.snt" ] && [ -s "$fl" ]
  then

    gunzip -c "$fl" |
    gbf2xml |
    xtract -rec PeptInfo -pattern INSDSeq \
      -wrp UID -element "+" \
      -wrp Accession -element INSDSeq_primary-accession \
      …
      -wrp Sequence -element INSDSeq_sequence |
    transmute -format |
    rchive -gzip -db "$dbase" \
      -archive "${archiveBase}" "${indexBase}" "${invertBase}" \
      -index UID -pattern PeptInfo

    touch "${archiveBase}/Sentinels/$base.snt"
  fi
```

reads compressed GenPept release files, and converts each record into a novel PeptInfo structure:

```
<PeptInfo>
  <UID>1</UID>
  <Accession>ALJ91870</Accession>
  <Product>DNA topoisomerase I</Product>
  <Organism div="ARC" taxid="498848">Thermus aquaticus Y51MC23</Organism>
  <Sequence>mpkkpktqgaahlgeggpkaearattlvvvespakarsiqkmlgp … </Sequence>
</PeptInfo>
…
```

Using a plus sign for the -element argument right after -pattern returns the record count in the input stream. Here it creates a unique integer access key for caching the record (and also deletes any incremental index files that just became stale). [The real code adds the previous maximum UID value.]

"Index" Helper

Given a protein sequence, **-element** will show it as a polypeptide, **-letters** will split it into individual amino acids, and **-pentamers** will create overlapping oligopeptides (residues 1-5, 2-6, 3-7, etc.):

```
-element      mpkkpktqgaahlgeggpkaearattlvvvespakarsiqk …
-letters      m p k k p k t q g a a h l g e g g p k a e …
-pentamers    mpkkp pkkpk kkpkt kpktq pktqg ktqga tqgaa …
```

The -letters option resembles words in a sentence, which is the starting point for building the positional indices that support phrase and proximity searches. But a standard phrase search will drop candidates that have even a single mismatch to the query sequence.

A better approach for finding the most similar sequences is to use xsearch **-match** on the pentamers. Indexing short overlapping peptides would make more efficient use of the local archive's directory hierarchy. For convenience, when peptide fragments are indexed using the PENT field, -match will internally split the protein query sequence into overlapping pentamers before running the search:

```
xsearch -db peptide -match "rlgrdtadmiqlikefdaqgvavrfiddgistdgdmgqmv [PENT]"
```

The result is a set of protein record keys ranked by the number of overlapping fragments matched.

For primary databases, index helpers embed an xtract command in a Unix "HERE" document. This is then passed as an argument for execution within an incremental indexing function. (Secondary projects do not need this invalidated index repopulation mechanism.) The complete **index-peptide** script is:

```
read -r -d '' idxtxt <<- EOS
xtract -set IdxDocumentSet -rec IdxDocument \
  -pattern PeptInfo -UID PeptInfo/UID \
    -wrp IdxUid -element "&UID" -clr -rst -tab "" \
    -group PeptInfo -pkg IdxSearchFields \
      -block PeptInfo -wrp UID -pad "&UID" \
      -block Accession -wrp ACCN -element Accession \
      -block Product -wrp PROD -element Product \
      -block Organism -wrp ORGN -element Organism -wrp DIV -element @div \
      -block Sequence -wrp PENT -pentamers Sequence
EOS

IncrementalIndex "${idxtxt}"
```

This will split the overlapping pentamers into separate terms for regular indexing:

```
…
<IdxDocument>
  <IdxUid>1</IdxUid>
  <IdxSearchFields>
    <UID>00000001</UID>
    <ACCN>ALJ91870</ACCN>
    <PROD>DNA topoisomerase I</PROD>
    <ORGN>Thermus aquaticus Y51MC23</ORGN>
    <DIV>ARC</DIV>
    <PENT>mpkkp</PENT>
    <PENT>pkkpk</PENT>
    <PENT>kkpkt</PENT>
    <PENT>kpktq</PENT>
    …
```

Text fields can use **-indexer** to create positional indices, but that will currently also remove stop words, which would discard valid pentapeptides like "apply", "every", and "never". Those attributes will likely be decoupled, with field-specific behavior put under the control of configuration settings in the future.

Aside from storing positions in attributes, EDirect uses the same index format as the Entrez indexer.

Later build steps, starting with index inversion, are standardized, and do not need external helper files.

Importing Data Tables

Large data tables are processed by programs written in the Go programming language. They are compiled and executed on-the-fly by helpers calling "go run". The central loop in **prep-nihocc.go**, simplified for brevity, and without the usual reality checks and periodic output buffer flushing, is:

```
var bldr strings.Builder

wrtr := bufio.NewWriter(os.Stdout)
scanr := bufio.NewScanner(os.Stdin)

for scanr.Scan() {

    line := scanr.Text()
    if line == "citing,referenced" {
        continue
    }

    cols := strings.Split(line, ",")
    fst, scd := cols[0], cols[1]

    pdFst, pdScd := padNumericID(fst), padNumericID(scd)

    bldr.WriteString(fst + "\tCITED\t" + pdScd + "\n")
    bldr.WriteString(scd + "\tCITES\t" + pdFst + "\n")

    txt := bldr.String()
    wrtr.WriteString(txt[:])
    bldr.Reset()
}

wrtr.Flush()
```

Archive Configuration File

More flexible drive management uses an environment variable that points to a configuration file:

```
export EDIRECT_LOCAL_CONFIG="${HOME}/edirect.ini"
```

The configuration file has one section per database, with entries pointing to solid-state drives:

```
[pubmed]
```

An **ARCHIVE** entry is required, and defaults to having all folders in the absence of other entries, but it is primarily for the Archive, Data, and Posting folders needed for record retrieval and indexed queries:

```
ARCHIVE=/Volumes/archive/
```

If set, a **WORKING** entry will move Extras, Index, Invert, Merged, Scratch, and Source folders, used for building the archive and search indices but not for active queries, to another drive:

```
WORKING=/Volumes/builder-pool-pm/
```

The Scratch directory contains Current, Indexed, and Inverted subfolders, which prevents secondary project builds from colliding with primary database persistent incremental indexing and inversion files.

Independently, **POSTING** and **SOURCES** can move their respective folders to other drives, and the source file repository can even use a large, rotating hard disk, since it only needs sequential streaming access:

```
POSTING=/Volumes/pm-posting/
SOURCES=/Volumes/common-sources-1/
```

With this configuration mechanism, each separate database can point to its own set of unique drives.

Solid-State Drive Preparation

To initialize a solid-state drive for hosting the local archive on a Mac, log into an admin account, run Disk Utility, choose View → Show All Devices, select the top-level external drive, and press the Erase icon. Set the Scheme popup to GUID Partition Map, and APFS will appear as a format choice. Set the Format popup to APFS, enter the desired name for the volume, and click the Erase button.

To finish the drive configuration, disable Spotlight indexing on the drive with:

```
sudo mdutil -i off "${EDIRECT_LOCAL_ARCHIVE}"
sudo mdutil -E "${EDIRECT_LOCAL_ARCHIVE}"
```

and turn off FSEvents logging with:

```
sudo touch "${EDIRECT_LOCAL_ARCHIVE}/.fseventsd/no_log"
```

Also exclude the drive from being backed up by Time Machine or scanned by a virus checker.

Finally, in Apple → System Settings → Privacy & Security → Full Disk Access, turn on the Terminal slide switch.

Python Integration

Controlling EDirect from Python scripts is easily done with assistance from the **edirect.py** library file, which is included in the EDirect archive.

At the beginning of your program, import the edirect module with the following commands:

```
#!/usr/bin/env python3

import sys
import os
import shutil

sys.path.insert(1, os.path.dirname(shutil.which('xtract')))
import edirect
```

The first argument to **edirect.execute** is the Unix command you wish to run. It can be a string:

```
("efetch -db nuccore -id NM_000518.5 -format fasta")
```

or a sequence of strings, which allows a variable's value to be substituted for a specific parameter:

```
accession = "NM_000518.5"
(('efetch', '-db', 'nuccore', '-id', accession, '-format', 'fasta'))
```

An optional second argument accepts data to be passed to the Unix command through stdin. Multiple steps are chained together by using the result of the previous command as the data argument in the next command:

```
seq = edirect.execute("efetch -db nuccore -id NM_000518.5 -format fasta")
sub = edirect.execute("transmute -extract -1-based -loc 51..494", seq)
prt = edirect.execute(('transmute', '-cds2prot', '-every', '-trim'), sub)
```

Data piped to the script itself is relayed by using **sys.stdin.read**() as the second argument.

Alternatively, the **edirect.pipeline** function can execute a string containing several piped commands:

```
edirect.pipeline('''efetch -db nuccore -id NM_000518.5 -format gb |
                    xtract -insd CDS gene product feat_location''')
```

or can accept a sequence of individual command strings to be piped together for execution:

```
edirect.pipeline(('efetch -db protein -id NP_000509.1 -format gp',
                  'xtract -insd Protein mol_wt sub_sequence'))
```

An **edirect.efetch** shortcut that uses named arguments is also available:

```
edirect.efetch(db="nuccore", id="NM_000518.5", format="fasta")
```

To run a custom shell script, make sure the execute permission bit is set, supply the full execution path, and follow it with any command-line arguments:

```
db = "pubmed"
res = edirect.execute(("./datefields.sh", db), "")
```

Compiled Programs

A program written in a compiled language is translated into a computer's native machine instruction code, and will run much faster than an interpreted script. Piping FASTA data to the basecount binary executable (compiled from the basecount.go source code file, below):

```
  efetch -db nuccore -id J01749,U54469 -format fasta | basecount
```

will return rows containing an accession number followed by counts for each base:

```
  J01749.1     A 983     C 1210     G 1134     T 1034
  U54469.1     A 849     C 699      G 585      T 748
```

Programs in Google's Go language ("golang") start with **package main** and then **import** additional software libraries (many included with Go, others residing in commercial repositories like github.com):

```
  package main

  import (
      "cmp"
      "eutils"
      "fmt"
      "maps"
      "os"
      "slices"
  )
```

Each compiled Go binary has a single **main** function, which is where program execution begins:

```
  func main() {
```

The fsta **variable** is assigned to a data **channel** that streams individual FASTA records one at a time:

```
      fsta := eutils.FASTAConverter(os.Stdin, false)
```

The countLetters **subroutine** will be called with the identifier and sequence of each FASTA record:

```
      countLetters := func(id, seq string) {
```

An empty counts **map** is created for each sequence, and its memory is freed when the subroutine exits:

```
          counts := make(map[rune]int)
```

A **for** loop on the **range** of the sequence string visits each sequence letter. The map keeps a running count for each base or residue, with "**++**" incrementing the current value of the letter's map entry:

```
          for _, base := range seq {
              counts[base]++
          }
```

A sorted keys **array** is produced by calling **slices.SortedFunc**. The alphabetical sort order is determined by the second argument, which is is an **anonymous** function literal:

```
          keys := slices.SortedFunc(maps.Keys(counts),
              func(i, j rune) int { return cmp.Compare(i, j) })
```

(Swapping the cmp.Compare arguments to "**j, i**" will return keys in reverse alphabetical order.)

The sequence identifier is printed in the first column:

```
          fmt.Fprintf(os.Stdout, "%s", id)
```

Iterating over the array prints letters and base counts in alphabetical order, with tabs between columns:

```
          for _, base := range keys {
              num := counts[base]
              fmt.Fprintf(os.Stdout, "\t%c %d", base, num)
          }
```

A newline is printed at the end of the row, and then the subroutine exits, clearing the map and array:

```
          fmt.Fprintf(os.Stdout, "\n")
      }
```

The remainder of the main function uses a loop to **drain** the fsta channel, passing the identifier and sequence string of each successive FASTA record to the countLetters function. The main function then ends with a final closing brace:

```
      for fsa := range fsta {
          countLetters(fsa.SeqID, fsa.Sequence)
      }
  }
```

Save the following script to a file named **build.sh**, in the same directory as the **basecount.go** file. Adjust optional GOOS and GOARCH environment variables to cross-compile for a different platform:

```
  #!/bin/bash

  if [ ! -f "go.mod" ]
  then
    go mod init "$( basename $PWD )"
    echo "replace eutils => $HOME/edirect/eutils" >> go.mod
    go get eutils
  fi
  if [ ! -f "go.sum" ]
  then
    go mod tidy
  fi

  for fl in *.go
  do
    env GOOS=darwin GOARCH=arm64 go build -o "${fl%.go}" "$fl"
  done
```

The build script creates module files used to track dependencies and retrieve imported packages. It also computes the path for finding the local **eutils** helper library included with EDirect. Set the Unix execution permission bit for the build script and compile the program(s) by running:

```
  chmod +x build.sh
  ./build.sh
```

Installation

EDirect consists of a set of scripts and programs that are downloaded to the user's computer. To install the software, open a terminal window and execute one of the following two commands:

```
sh -c "$(curl -fsSL https://ftp.ncbi.nlm.nih.gov/entrez/entrezdirect/install-edirect.sh)"

sh -c "$(wget -q https://ftp.ncbi.nlm.nih.gov/entrez/entrezdirect/install-edirect.sh -O -)"
```

One installation is complete, run the following to set the PATH for the current terminal session:

```
export PATH=${HOME}/edirect:${PATH}
```

For best performance of Entrez network service requests, obtain an API Key from NCBI, and place the following line in your .bash_profile and .zshrc configuration files:

```
export NCBI_API_KEY=unique_api_key
```

For faster downloading of NCBI release files, install the free Aspera Connect file transfer utility on your computer. It is available from the IBM Aspera Connect subsection at:

```
https://www.ibm.com/products/aspera/downloads#cds
```

Documentation

Documentation for EDirect is on the web at:

```
https://www.ncbi.nlm.nih.gov/books/NBK179288
```

Additional examples, organized by Entrez database, are at:

```
https://www.ncbi.nlm.nih.gov/books/NBK565821
```

Information on how to obtain an API Key is described in this NCBI blogpost:

```
https://ncbiinsights.ncbi.nlm.nih.gov/2017/11/02/new-api-keys-for-the-e-utilities
```

Introductions to shell scripting for non-programmers, and to the Go programming language, are at:

```
https://missing.csail.mit.edu/2020/shell-tools/
https://cacm.acm.org/research/the-go-programming-language-and-environment/
```

Instructions for downloading and installing the Go compiler are at:

```
https://golang.org/doc/install#download
```

Questions or comments on EDirect may be sent to info@ncbi.nlm.nih.gov.