Entrez Programming Utilities Help

Last Updated: November 30, 2022

National Center for Biotechnology Information (US)
Bethesda (MD)

National Center for Biotechnology Information (US), Bethesda (MD)

NLM Citation: Entrez Programming Utilities Help [Internet]. Bethesda (MD): National Center for Biotechnology Information (US); 2010-.

- Please see the E-utilities Introduction video for a brief introduction.
- Please see the Release Notes for details and changes.

The Entrez Programming Utilities (E-utilities) are a set of eight server-side programs that provide a stable interface into the Entrez query and database system at the National Center for Biotechnology Information (NCBI). The E-utilities use a fixed URL syntax that translates a standard set of input parameters into the values necessary for various NCBI software components to search for and retrieve the requested data. The E-utilities are therefore the structured interface to the Entrez system, which currently includes 38 databases covering a variety of biomedical data, including nucleotide and protein sequences, gene records, three-dimensional molecular structures, and the biomedical literature.

Table of Contents

E-utilities Quick Start

Eric Sayers, PhD[1]

Created: December 12, 2008; Updated: October 24, 2018.

Release Notes

Please see our Release Notes for details on recent changes and updates.

Announcement

On December 1, 2018, NCBI will begin enforcing the use of new API keys for E-utility calls. Please see Chapter 2 for more details about this important change.

Introduction

This chapter provides a brief overview of basic E-utility functions along with examples of URL calls. Please see Chapter 2 for a general introduction to these utilities and Chapter 4 for a detailed discussion of syntax and parameters.

Examples include live URLs that provide sample outputs.

All E-utility calls share the same base URL:

`https://eutils.ncbi.nlm.nih.gov/entrez/eutils/`

Searching a Database

Basic Searching

`esearch.fcgi?db=<database>&term=<query>`

Input: Entrez database (&db); Any Entrez text query (&term)

Output: List of UIDs matching the Entrez query

Example: Get the PubMed IDs (PMIDs) for articles about breast cancer published in Science in 2008

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi?db=pubmed&term=science[journal]+AND+breast+cancer+AND+2008[pdat]

Storing Search Results

`esearch.fcgi?db=<database>&term=<query>&usehistory=y`

Input: Any Entrez text query (&term); Entrez database (&db); &usehistory=y

Output: Web environment (&WebEnv) and query key (&query_key) parameters specifying the location on the Entrez history server of the list of UIDs matching the Entrez query

Example: Get the PubMed IDs (PMIDs) for articles about breast cancer published in Science in 2008, and store them on the Entrez history server for later use

Author Affiliation: 1 NCBI; Email: sayers@ncbi.nlm.nih.gov.

Corresponding author.

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi?db=pubmed&term=science[journal]+AND+breast+cancer+AND+2008[pdat]&usehistory=y

Associating Search Results with Existing Search Results

```
esearch.fcgi?db=<database>&term=<query1>&usehistory=y

# esearch produces WebEnv value ($web1) and QueryKey value ($key1)

esearch.fcgi?db=<database>&term=<query2>&usehistory=y&WebEnv=$web1

# esearch produces WebEnv value ($web2) that contains the results
of both searches ($key1 and $key2)
```

Input: Any Entrez text query (&term); Entrez database (&db); &usehistory=y; Existing web environment (&WebEnv) from a prior E-utility call

Output: Web environment (&WebEnv) and query key (&query_key) parameters specifying the location on the Entrez history server of the list of UIDs matching the Entrez query

For More Information

Please see ESearch In-Depth for a full description of ESearch.

Sample ESearch Output

```
<?xml version="1.0" ?>
<!DOCTYPE eSearchResult PUBLIC "-//NLM//DTD eSearchResult, 11 May 2002//EN"
 "https://www.ncbi.nlm.nih.gov/entrez/query/DTD/eSearch_020511.dtd">
<eSearchResult>
<Count>255147</Count>   # total number of records matching query
<RetMax>20</RetMax># number of UIDs returned in this XML; default=20
<RetStart>0</RetStart># index of first record returned; default=0
<QueryKey>1</QueryKey># QueryKey, only present if &usehistory=y
<WebEnv>0l93yIkBjmM60UBXuvBvPfBIq8-9nIsldXuMP0hhuMH-
8GjCz7F_Dz1XL6z@397033B29A81FB01_0038SID</WebEnv>
                  # WebEnv; only present if &usehistory=y
      <IdList>
<Id>229486465</Id>    # list of UIDs returned
<Id>229486321</Id>
<Id>229485738</Id>
<Id>229470359</Id>
<Id>229463047</Id>
<Id>229463037</Id>
<Id>229463022</Id>
<Id>229463019</Id>
<Id>229463007</Id>
<Id>229463002</Id>
<Id>229463000</Id>
<Id>229462974</Id>
<Id>229462961</Id>
<Id>229462956</Id>
<Id>229462921</Id>
<Id>229462905</Id>
<Id>229462899</Id>
<Id>229462873</Id>
<Id>229462863</Id>
<Id>229462862</Id>
```

```
</IdList>
<TranslationSet>        # details of how Entrez translated the query
    <Translation>
     <From>mouse[orgn]</From>
     <To>"Mus musculus"[Organism]</To>
    </Translation>
</TranslationSet>
<TranslationStack>
   <TermSet>
    <Term>"Mus musculus"[Organism]</Term>
    <Field>Organism</Field>
    <Count>255147</Count>
    <Explode>Y</Explode>
   </TermSet>
   <OP>GROUP</OP>
</TranslationStack>
<QueryTranslation>"Mus musculus"[Organism]</QueryTranslation>
</eSearchResult>
```

Searching PubMed with Citation Data

```
ecitmatch.cgi?db=pubmed&rettype=xml&bdata=<citations>
```

Input: List of citation strings separated by a carriage return (%0D), where each citation string has the following format:

journal_title|year|volume|first_page|author_name|your_key|

Output: A list of citation strings with the corresponding PubMed ID (PMID) appended.

Example: Search PubMed for the following ciations:

Art1: Mann, BJ. (1991) *Proc. Natl. Acad. Sci. USA*. 88:3248

Art2: Palmenberg, AC. (1987) *Science* 235:182

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/ecitmatch.cgi?db=pubmed&retmode=xml&bdata=proc+natl+acad+sci+u+s+a|1991|88|3248|mann+bj|Art1|%0Dscience|1987|235|182|palmenberg+ac|Art2|

Sample Output (the PMIDs appear in the rightmost field):

```
proc natl acad sci u s a|1991|88|3248|mann bj|Art1|2014248
science|1987|235|182|palmenberg ac|Art2|3026048
```

Please see ECitMatch In-Depth for a full description of ECitMatch.

Uploading UIDs to Entrez

Basic Uploading

```
epost.fcgi?db=<database>&id=<uid_list>
```

Input: List of UIDs (&id); Entrez database (&db)

Output: Web environment (&WebEnv) and query key (&query_key) parameters specifying the location on the Entrez history server of the list of uploaded UIDs

Example: Upload five Gene IDs (7173,22018,54314,403521,525013) for later processing.

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/epost.fcgi?db=gene&id=7173,22018,54314,403521,525013

Associating a Set of UIDs with Previously Posted Sets

```
epost.fcgi?db=<database1>&id=<uid_list1>

# epost produces WebEnv value ($web1) and QueryKey value ($key1)

epost.fcgi?db=<database2>&id=<uid_list2>&WebEnv=$web1

# epost produces WebEnv value ($web2) that contains the results of both
posts ($key1 and $key2)
```

Input: List of UIDs (&id); Entrez database (&db); Existing web environment (&WebEnv)

Output: Web environment (&WebEnv) and query key (&query_key) parameters specifying the location on the Entrez history server of the list of uploaded UIDs

For More Information

Please see EPost In-Depth for a full description of EPost.

Sample EPost Output

```
<?xml version="1.0"?>
<!DOCTYPE ePostResult PUBLIC "-//NLM//DTD ePostResult, 11 May 2002//EN"
 "https://www.ncbi.nlm.nih.gov/entrez/query/DTD/ePost_020511.dtd">
<ePostResult>
<QueryKey>1</QueryKey>
<WebEnv>NCID_01_268116914_130.14.18.47_9001_1241798628</WebEnv>
</ePostResult>
```

Downloading Document Summaries

Basic Downloading

```
esummary.fcgi?db=<database>&id=<uid_list>
```

Input: List of UIDs (&id); Entrez database (&db)

Output: XML DocSums

Example: Download DocSums for these protein GIs: 6678417,9507199,28558982,28558984,28558988,28558990

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esummary.fcgi?db=protein&id=6678417,9507199,28558982,28558984,28558988,28558990

Downloading Data From a Previous Search

```
esearch.fcgi?db=<database>&term=<query>&usehistory=y

# esearch produces WebEnv value ($web1) and QueryKey value ($key1)

esummary.fcgi?db=<database>&query_key=$key1&WebEnv=$web1
```

Input: Web environment (&WebEnv) and query key (&query_key) representing a set of Entrez UIDs on the Entrez history server

Output: XML DocSums

Sample ESummary Output

The output of ESummary is a series of XML "DocSums" (Document Summaries), the format of which depends on the database. Below is an example DocSum for Entrez Protein.

```
<?xml version="1.0"?>
<!DOCTYPE eSummaryResult PUBLIC "-//NLM//DTD eSummaryResult, 29 October
 2004//EN" "https://www.ncbi.nlm.nih.gov/entrez/query/DTD/eSummary_
041029.dtd">
<eSummaryResult>
<DocSum>
<Id>15718680</Id>
<Item Name="Caption" Type="String">NP_005537</Item>
<Item Name="Title" Type="String">IL2-inducible T-cell kinase [Homo
 sapiens]</Item>
<Item Name="Extra"
Type="String">gi|15718680|ref|NP_005537.3|[15718680]</Item>
<Item Name="Gi" Type="Integer">15718680</Item>
<Item Name="CreateDate" Type="String">1999/06/09</Item>
<Item Name="UpdateDate" Type="String">2009/04/05</Item>
<Item Name="Flags" Type="Integer">512</Item>
<Item Name="TaxId" Type="Integer">9606</Item>
<Item Name="Length" Type="Integer">620</Item>
<Item Name="Status" Type="String">live</Item>
<Item Name="ReplacedBy" Type="String"></Item>
<Item Name="Comment" Type="String"><![CDATA[  ]]></Item>
</DocSum>
</eSummaryResult>
```

Sample ESummary version 2.0 Output

Version 2.0 of ESummary is an alternate XML presentation of Entrez DocSums. To retrieve version 2.0 DocSums, the URL should contain the &version parameter with an assigned value of '2.0'. Each Entrez database provides its own unique DTD for version 2.0 DocSums, and a link to the relevant DTD is provided in the header of the version 2.0 XML.

```
esummary.fcgi?db=<database>&id=<uid_list>&version=2.0
```

Below is an example version 2.0 DocSum from Entrez Protein (the same record as shown above in the default DocSum XML).

```
<?xml version="1.0"?>
<!DOCTYPE eSummaryResult PUBLIC "-//NLM//DTD eSummaryResult//EN" "https://
www.ncbi.nlm.nih.gov/entrez/query/DTD/eSummaryDTD/eSummary_protein.dtd">
<eSummaryResult>
    <DocumentSummarySet status="OK">
        <DocumentSummary uid="15718680">
            <Caption>NP_005537</Caption>
            <Title>tyrosine-protein kinase ITK/TSK [Homo sapiens]</Title>
            <Extra>gi|15718680|ref|NP_005537.3|</Extra>
            <Gi>15718680</Gi>

            <CreateDate>1999/06/09</CreateDate>
            <UpdateDate>2011/10/09</UpdateDate>
            <Flags>512</Flags>
            <TaxId>9606</TaxId>
```

```
<Slen>620</Slen>

<Biomol/>

<MolType>aa</MolType>
<Topology>linear</Topology>
<SourceDb>refseq</SourceDb>
<SegSetSize>0</SegSetSize>
<ProjectId>0</ProjectId>
<Genome>genomic</Genome>

<SubType>chromosome|map</SubType>
<SubName>5|5q31-q32</SubName>
<AssemblyGi>399658</AssemblyGi>
<AssemblyAcc>D13720.1</AssemblyAcc>
<Tech/>
<Completeness/>
<GeneticCode>1</GeneticCode>

<Strand/>
<Organism>Homo sapiens</Organism>
<Statistics>
    <Stat type="all" count="8"/>
    <Stat type="blob_size" count="16154"/>
    <Stat type="cdregion" count="1"/>
    <Stat type="cdregion" subtype="CDS" count="1"/>
    <Stat type="gene" count="1"/>
    <Stat type="gene" subtype="Gene" count="1"/>
    <Stat type="org" count="1"/>
    <Stat type="prot" count="1"/>
    <Stat type="prot" subtype="Prot" count="1"/>
    <Stat type="pub" count="14"/>
    <Stat type="pub" subtype="PubMed" count="10"/>
    <Stat type="pub" subtype="PubMed/Gene-rif" count="4"/>
    <Stat type="site" count="4"/>
    <Stat type="site" subtype="Site" count="4"/>
    <Stat source="CDD" type="all" count="15"/>
    <Stat source="CDD" type="region" count="6"/>
    <Stat source="CDD" type="region" subtype="Region" count="6"/>
    <Stat source="CDD" type="site" count="9"/>
    <Stat source="CDD" type="site" subtype="Site" count="9"/>
    <Stat source="HPRD" type="all" count="3"/>
    <Stat source="HPRD" type="site" count="3"/>
    <Stat source="HPRD" type="site" subtype="Site" count="3"/>
    <Stat source="SNP" type="all" count="31"/>
    <Stat source="SNP" type="imp" count="31"/>
    <Stat source="SNP" type="imp" subtype="variation" count="31"/>
    <Stat source="all" type="all" count="57"/>
    <Stat source="all" type="blob_size" count="16154"/>
    <Stat source="all" type="cdregion" count="1"/>
    <Stat source="all" type="gene" count="1"/>
    <Stat source="all" type="imp" count="31"/>
    <Stat source="all" type="org" count="1"/>
    <Stat source="all" type="prot" count="1"/>
    <Stat source="all" type="pub" count="14"/>
    <Stat source="all" type="region" count="6"/>
    <Stat source="all" type="site" count="16"/>
</Statistics>
<AccessionVersion>NP_005537.3</AccessionVersion>
```

```
            <Properties aa="2">2</Properties>
            <Comment/>
            <OSLT indexed="yes">NP_005537.3</OSLT>
            <IdGiClass mol="3" repr="2" gi_state="10" sat="4" sat_key="58760802"
owner="20"
                sat_name="NCBI" owner_name="NCBI-Genomes" defdiv="GNM" length="620"
extfeatmask="41"
            />
        </DocumentSummary>

    </DocumentSummarySet>
</eSummaryResult>
```

Downloading Full Records

Basic Downloading

```
efetch.fcgi?db=<database>&id=<uid_list>&rettype=<retrieval_type>
&retmode=<retrieval_mode>
```

Input: List of UIDs (&id); Entrez database (&db); Retrieval type (&rettype); Retrieval mode (&retmode)

Output: Formatted data records as specified

Example: Download nuccore GIs 34577062 and 24475906 in FASTA format

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=nuccore&id=34577062,24475906&rettype=fasta&retmode=text

Downloading Data From a Previous Search

```
esearch.fcgi?db=<database>&term=<query>&usehistory=y

# esearch produces WebEnv value ($web1) and QueryKey value ($key1)

efetch.fcgi?db=<database>&query_key=$key1&WebEnv=$web1&rettype=
<retrieval_type>&retmode=<retrieval_mode>
```

Input: Entrez database (&db); Web environment (&WebEnv) and query key (&query_key) representing a set of Entrez UIDs on the Entrez history server; Retrieval type (&rettype); Retrieval mode (&retmode)

Output: Formatted data records as specified

Downloading a Large Set of Records

Please see Application 3 **in Chapter 3**

Input: Entrez database (&db); Web environment (&WebEnv) and query key (&query_key) representing a set of Entrez UIDs on the Entrez history server; Retrieval start (&retstart), the first record of the set to retrieve; Retrieval maximum (&retmax), maximum number of records to retrieve

Output: Formatted data records as specified

For More Information

Please see EFetch In-Depth for a full description of EFetch.

Finding Related Data Through Entrez Links

Basic Linking

Batch mode – finds only one set of linked UIDs

```
elink.fcgi?dbfrom=<source_db>&db=<destination_db>&id=<uid_list>
```

Input: List of UIDs (&id); Source Entrez database (&dbfrom); Destination Entrez database (&db)

Output: XML containing linked UIDs from source and destination databases

Example: Find one set of Gene IDs linked to nuccore GIs 34577062 and 24475906

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?dbfrom=nuccore&db=gene&id=34577062,24475906

'By Id' mode – finds one set of linked UIDs for each input UID

```
elink.fcgi?dbfrom=<source_db>&db=<destination_db>&id=<uid1>&id=
<uid2>&id=<uid3>...
```

Example: Find separate sets of Gene IDs linked to nuccore GIs 34577062 and 24475906

**https://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?
dbfrom=nuccore&db=gene&id=34577062&id=24475906**

Note: &db may be a comma-delimited list of databases, so that elink returns multiple sets of linked UIDs in a single call

Finding Links to Data from a Previous Search

```
esearch.fcgi?db=<source_db>&term=<query>&usehistory=y

# esearch produces WebEnv value ($web1) and QueryKey value ($key1)

elink.fcgi?dbfrom=<source_db>&db=<destination_db>&query_key=
$key1&WebEnv=$web1&cmd=neighbor_history
```

Input: Source Entrez database (&dbfrom); Destination Entrez database (&db); Web environment (&WebEnv) and query key (&query_key) representing the set of source UIDs on the Entrez history server; Command mode (&cmd)

Output: XML containing Web environments and query keys for each set of linked UIDs

Note: To achieve 'By Id' mode, one must send each input UID as a separate &id parameter in the URL. Sending a WebEnv/query_key set always produces Batch mode behavior (one set of linked UIDs).

Finding Computational Neighbors Limited by an Entrez Search

```
elink.fcgi?dbfrom=<source_db>&db=<source_db>&id=<uid_list>&term=
<query>&cmd=neighbor_history
```

Input: Source Entrez database (&dbfrom); Destination Entrez database (&db); List of UIDs (&id); Entrez query (&term); Command mode (&cmd)

Output: XML containing Web environments and query keys for each set of linked UIDs

Example: Find protein UIDs that are rat Reference Sequences and that are sequence similar to GI 15718680

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?dbfrom=protein&db=protein&id=15718680&term=rat[orgn]+AND+srcdb+refseq[prop]&cmd=neighbor_history

For More Information

Please see ELink In-Depth for a full description of ELink.

Getting Database Statistics and Search Fields

`einfo.fcgi?db=<database>`

Input: Entrez database (&db)

Output: XML containing database statistics

Note: If no database parameter is supplied, einfo will return a list of all valid Entrez databases.

Example: Find database statistics for Entrez Protein.

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/einfo.fcgi?db=protein

For More Information

Please see EInfo In-Depth for a full description of EInfo.

Sample EInfo Output

```
<?xml version="1.0"?>
<!DOCTYPE eInfoResult PUBLIC "-//NLM//DTD eInfoResult, 11 May 2002//EN"
"https://www.ncbi.nlm.nih.gov/entrez/query/DTD/eInfo_020511.dtd">
<eInfoResult>
<DbInfo>
<DbName>protein</DbName>
<MenuName>Protein</MenuName>
<Description>Protein sequence record</Description>
<Count>26715092</Count>
<LastUpdate>2009/05/12 04:39</LastUpdate>
<FieldList>
<Field>
<Name>ALL</Name>
<FullName>All Fields</FullName>
<Description>All terms from all searchable fields</Description>
<TermCount>133639432</TermCount>
<IsDate>N</IsDate>
<IsNumerical>N</IsNumerical>
<SingleToken>N</SingleToken>
<Hierarchy>N</Hierarchy>
<IsHidden>N</IsHidden>
</Field>
...
<Field>
<Name>PORG</Name>
<FullName>Primary Organism</FullName>
<Description>Scientific and common names
of primary organism, and all higher levels of taxonomy</Description>
<TermCount>673555</TermCount>
<IsDate>N</IsDate>
```

```
<IsNumerical>N</IsNumerical>
<SingleToken>Y</SingleToken>
<Hierarchy>Y</Hierarchy>
<IsHidden>N</IsHidden>
</Field>
</FieldList>
<LinkList>
<Link>
<Name>protein_biosystems</Name>
<Menu>BioSystem Links</Menu>
<Description>BioSystems</Description>
<DbTo>biosystems</DbTo>
</Link>
...
<Link>
<Name>protein_unigene</Name>
<Menu>UniGene Links</Menu>
<Description>Related UniGene records</Description>
<DbTo>unigene</DbTo>
</Link>
</LinkList>
</DbInfo>
</eInfoResult>
```

Performing a Global Entrez Search

`egquery.fcgi?term=<query>`

Input: Entrez text query (&term)

Output: XML containing the number of hits in each database.

Example: Determine the number of records for mouse in Entrez.

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/egquery.fcgi?term=mouse[orgn]

For More Information

Please see EGQuery In-Depth for a full description of EGQuery.

Sample EGQuery Output

```
<?xml version="1.0"?>
<!DOCTYPE Result PUBLIC "-//NLM//DTD eSearchResult, January 2004//EN"
 "https://www.ncbi.nlm.nih.gov/entrez/query/DTD/egquery.dtd">
<!--
        $Id: egquery_template.xml 106311 2007-06-26 14:46:31Z osipov $
-->
<!-- ================================================================ -->
<Result>
        <Term>mouse[orgn]</Term>
        <eGQueryResult>
             <ResultItem>
                  <DbName>pubmed</DbName>
                  <MenuName>PubMed</MenuName>
                  <Count>0</Count>
                  <Status>Term or Database is not found</Status>
             </ResultItem>
             <ResultItem>
```

```
                <DbName>pmc</DbName>
                <MenuName>PMC</MenuName>
                <Count>3823</Count>
                <Status>Ok</Status>
            </ResultItem>
...
            <ResultItem>
                <DbName>nuccore</DbName>
                <MenuName>Nucleotide</MenuName>
                <Count>1739903</Count>
                <Status>Ok</Status>
            </ResultItem>
            <ResultItem>
                <DbName>nucgss</DbName>
                <MenuName>GSS</MenuName>
                <Count>2264567</Count>
                <Status>Ok</Status>
            </ResultItem>
            <ResultItem>
                <DbName>nucest</DbName>
                <MenuName>EST</MenuName>
                <Count>4852140</Count>
                <Status>Ok</Status>
            </ResultItem>
            <ResultItem>
                <DbName>protein</DbName>
                <MenuName>Protein</MenuName>
                <Count>255212</Count>
                <Status>Ok</Status>
            </ResultItem>
...
            <ResultItem>
                <DbName>proteinclusters</DbName>
                <MenuName>Protein Clusters</MenuName>
                <Count>13</Count>
                <Status>Ok</Status>
            </ResultItem>
        </eGQueryResult>
</Result>
```

Retrieving Spelling Suggestions

`espell.fcgi?term=<query>&db=<database>`

Input: Entrez text query (&term); Entrez database (&db)

Output: XML containing the original query and spelling suggestions.

Example: Find spelling suggestions for the PubMed Central query 'fiberblast cell grwth'.

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/espell.fcgi?term=fiberblast+cell+grwth&db=pmc

For More Information

Please see ESpell In-Depth for a full description of EGQuery.

Sample ESpell Output

```
<?xml version="1.0"?>
<!DOCTYPE eSpellResult PUBLIC "-//NLM//DTD eSpellResult, 23 November
2004//EN" "https://www.ncbi.nlm.nih.gov/entrez/query/DTD/eSpell.dtd">
<eSpellResult>
<Database>pmc</Database>
<Query>fiberblast cell grwth</Query>
<CorrectedQuery>fibroblast cell growth</CorrectedQuery>
<SpelledQuery>
 <Replaced>fibroblast</Replaced>
 <Original> cell </Original>
 <Replaced>growth</Replaced>
</SpelledQuery>
<ERROR/>
</eSpellResult>
```

Demonstration Programs

EBot

EBot is an interactive web tool that first allows users to construct an arbitrary E-utility analysis pipeline and then generates a Perl script to execute the pipeline. The Perl script can be downloaded and executed on any computer with a Perl installation. For more details, see the EBot page linked above.

Sample Perl Scripts

The two sample Perl scripts below demonstrate basic E-utility functions. Both scripts should be copied and saved as plain text files and can be executed on any computer with a Perl installation.

ESearch-EFetch demonstrates basic search and retrieval functions.

```
#!/usr/local/bin/perl -w
# ===========================================================================
#
#                            PUBLIC DOMAIN NOTICE
#               National Center for Biotechnology Information
#
#  This software/database is a "United States Government Work" under the
#  terms of the United States Copyright Act.  It was written as part of
#  the author's official duties as a United States Government employee and
#  thus cannot be copyrighted.  This software/database is freely available
#  to the public for use. The National Library of Medicine and the U.S.
#  Government have not placed any restriction on its use or reproduction.
#
#  Although all reasonable efforts have been taken to ensure the accuracy
#  and reliability of the software and data, the NLM and the U.S.
#  Government do not and cannot warrant the performance or results that
#  may be obtained by using this software or data. The NLM and the U.S.
#  Government disclaim all warranties, express or implied, including
#  warranties of performance, merchantability or fitness for any particular
#  purpose.
#
#  Please cite the author in any work or product based on this material.
#
# ===========================================================================
#
```

```
# Author:  Oleg Khovayko
#
# File Description: eSearch/eFetch calling example
#
# ---------------------------------------------------------------------
# Subroutine to prompt user for variables in the next section

sub ask_user {
  print "$_[0] [$_[1]]: ";
  my $rc = <>;
  chomp $rc;
  if($rc eq "") { $rc = $_[1]; }
  return $rc;
}

# ---------------------------------------------------------------------
# Define library for the 'get' function used in the next section.
# $utils contains route for the utilities.
# $db, $query, and $report may be supplied by the user when prompted;
# if not answered, default values, will be assigned as shown below.

use LWP::Simple;

my $utils = "https://www.ncbi.nlm.nih.gov/entrez/eutils";

my $db     = ask_user("Database", "Pubmed");
my $query  = ask_user("Query",    "zanzibar");
my $report = ask_user("Report",   "abstract");

# ---------------------------------------------------------------------
# $esearch cont?ins the PATH & parameters for the ESearch call
# $esearch_result containts the result of the ESearch call
# the results are displayed ?nd parsed into variables
# $Count, $QueryKey, and $WebEnv for later use and then displayed.

my $esearch = "$utils/esearch.fcgi?" .
              "db=$db&retmax=1&usehistory=y&term=";

my $esearch_result = get($esearch . $query);

print "\nESEARCH RESULT: $esearch_result\n";

$esearch_result =~
  m|<Count>(\d+)</Count>.*<QueryKey>(\d+)</QueryKey>.*<WebEnv>(\S+)</WebEnv>|s;

my $Count    = $1;
my $QueryKey = $2;
my $WebEnv   = $3;

print "Count = $Count; QueryKey = $QueryKey; WebEnv = $WebEnv\n";

# ---------------------------------------------------------------------
# this area defines a loop which will display $retmax citation results from
# Efetch each time the the Enter Key is pressed, after a prompt.

my $retstart;
my $retmax=3;

for($retstart = 0; $retstart < $Count; $retstart += $retmax) {
```

```
  my $efetch = "$utils/efetch.fcgi?" .
               "rettype=$report&retmode=text&retstart=$retstart&retmax=$retmax&" .
               "db=$db&query_key=$QueryKey&WebEnv=$WebEnv";

  print "\nEF_QUERY=$efetch\n";

  my $efetch_result = get($efetch);

  print "---------\nEFETCH RESULT(".
         ($retstart + 1) . ".." . ($retstart + $retmax) . "): ".
        "[$efetch_result]\n-----PRESS ENTER!!!-------\n";
  <>;
}
```

EPost-ESummary demonstrates basic uploading and document summary retrieval.

```
#!/usr/local/bin/perl -w
# ===========================================================================
#
#                            PUBLIC DOMAIN NOTICE
#               National Center for Biotechnology Information
#
#  This software/database is a "United States Government Work" under the
#  terms of the United States Copyright Act.  It was written as part of
#  the author's official duties as a United States Government employee and
#  thus cannot be copyrighted.  This software/database is freely available
#  to the public for use. The National Library of Medicine and the U.S.
#  Government have not placed any restriction on its use or reproduction.
#
#  Although all reasonable efforts have been taken to ensure the accuracy
#  and reliability of the software and data, the NLM and the U.S.
#  Government do not and cannot warrant the performance or results that
#  may be obtained by using this software or data. The NLM and the U.S.
#  Government disclaim all warranties, express or implied, including
#  warranties of performance, merchantability or fitness for any particular
#  purpose.
#
#  Please cite the author in any work or product based on this material.
#
# ===========================================================================
#
# Author:  Oleg Khovayko
#
# File Description: ePost/eSummary calling example
#

# --------------------------------------------------------------------
my $eutils_root  = "https://www.ncbi.nlm.nih.gov/entrez/eutils";
my $ePost_url    = "$eutils_root/epost.fcgi";
my $eSummary_url = "$eutils_root/esummary.fcgi";

my $db_name = "PubMed";

# --------------------------------------------------------------------
use strict;

use LWP::UserAgent;
use LWP::Simple;
use HTTP::Request;
```

```
use HTTP::Headers;
use CGI;

# ------------------------------------------------------------------
# Read input file into variable $file
# File name - forst argument $ARGV[0]

undef $/;  #for load whole file

open IF, $ARGV[0] || die "Can't open for read: $!\n";
my $file = <IF>;
close IF;
print "Loaded file: [$file]\n";

# Prepare file - substitute all separators to comma

$file =~ s/\s+/,/gs;
print "Prepared file: [$file]\n";

#Create CGI param line

my $form_data = "db=$db_name&id=$file";

# ------------------------------------------------------------------
# Create HTTP request

my $headers = new HTTP::Headers(
    Accept        => "text/html, text/plain",
    Content_Type    => "application/x-www-form-urlencoded"
);

my $request = new HTTP::Request("POST", $ePost_url, $headers );

$request->content($form_data);

# Create the user agent object

my $ua = new LWP::UserAgent;
$ua->agent("ePost/example");

# ------------------------------------------------------------------
# send file to ePost by HTTP

my $response = $ua->request($request);

# ------------------------------------------------------------------

print "Responce status message: [" . $response->message . "]\n";
print "Responce content: [" .        $response->content . "]\n";

# ------------------------------------------------------------------
# Parse response->content and extract QueryKey & WebEnv
$response->content =~
  m|<QueryKey>(\d+)</QueryKey>.*<WebEnv>(\S+)</WebEnv>|s;

my $QueryKey = $1;
my $WebEnv   = $2;

print "\nEXTRACTED:\nQueryKey = $QueryKey;\nWebEnv = $WebEnv\n\n";
```

```
# ------------------------------------------------------------------------
# Retrieve DocSum from eSummary by simple::get method and print it
#
print "eSummary result: [" .
  get("$eSummary_url?db=$db_name&query_key=$QueryKey&WebEnv=$WebEnv") .
  "]\n";
```

For More Information

Announcement Mailing List

NCBI posts general announcements regarding the E-utilities to the utilities-announce announcement mailing list. This mailing list is an announcement list only; individual subscribers may **not** send mail to the list. Also, the list of subscribers is private and is not shared or used in any other way except for providing announcements to list members. The list receives about one posting per month. Please subscribe at the above link.

Getting Help

Please refer to the PubMed and Entrez help documents for more information about search queries, database indexing, field limitations and database content.

Suggestions, comments, and questions specifically relating to the EUtility programs may be sent to eutilities@ncbi.nlm.nih.gov.

A General Introduction to the E-utilities

Eric Sayers, PhD[1]

Created: May 26, 2009; Updated: November 17, 2022.

Introduction

The Entrez Programming Utilities (E-utilities) are a set of nine server-side programs that provide a stable interface into the Entrez query and database system at the National Center for Biotechnology Information (NCBI). The E-utilities use a fixed URL syntax that translates a standard set of input parameters into the values necessary for various NCBI software components to search for and retrieve the requested data. The E-utilities are therefore the structured interface to the Entrez system, which currently includes 38 databases covering a variety of biomedical data, including nucleotide and protein sequences, gene records, three-dimensional molecular structures, and the biomedical literature.

To access these data, a piece of software first posts an E-utility URL to NCBI, then retrieves the results of this posting, after which it processes the data as required. The software can thus use any computer language that can send a URL to the E-utilities server and interpret the XML response; examples of such languages are Perl, Python, Java, and C++. Combining E-utilities components to form customized data pipelines within these applications is a powerful approach to data manipulation.

This chapter first describes the general function and use of the eight E-utilities, followed by basic usage guidelines and requirements, and concludes with a discussion of how the E-utilities function within the Entrez system.

Usage Guidelines and Requirements

Use the E-utility URL

All E-utility requests should be made to URLs beginning with the following string:

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/

These URLs direct requests to servers that are used only by the E-utilities and that are optimized to give users the best performance.

Frequency, Timing and Registration of E-utility URL Requests

In order not to overload the E-utility servers, NCBI recommends that users post no more than three URL requests per second and limit large jobs to either weekends or between 9:00 PM and 5:00 AM Eastern time during weekdays. Failure to comply with this policy may result in an IP address being blocked from accessing NCBI. If NCBI blocks an IP address, service will not be restored unless the developers of the software accessing the E-utilities register values of the **tool** and **email** parameters with NCBI. The value of **tool** should be a string with no internal spaces that uniquely identifies the software producing the request. The value of **email** should be a complete and valid e-mail address of the software developer and not that of a third-party end user. The value of **email** will be used only to contact developers if NCBI observes requests that violate our policies, and we will attempt such contact prior to blocking access. In addition, developers may request that the value of **email** be added to the E-utility mailing list that provides announcements of software updates, known bugs and other

Author Affiliation: 1 NCBI; Email: sayers@ncbi.nlm.nih.gov.

Corresponding author.

policy changes affecting the E-utilities. To register **tool** and **email** values, simply send an e-mail to eutilities@ncbi.nlm.nih.gov including the desired values along with the name of either a developer or the organization creating the software. Once NCBI establishes communication with a developer, receives values for **tool** and **email** and validates the e-mail address in **email**, the block will be lifted. Once **tool** and **email** values are registered, all subsequent E-utility requests from that software package should contain both values. Please be aware that merely providing values for **tool** and **email** in requests is not sufficient to comply with this policy; these values must be registered with NCBI. Requests from any IP that lack registered values for **tool** and **email** and that violate the above usage policies may be blocked. Software developers may register values of **tool** and **email** at any time, and are encouraged to do so.

API Keys

Since December 1, 2018, NCBI has provided API keys that offer enhanced levels of supported access to the E-utilities. Without an API key, any site (IP address) posting more than 3 requests per second to the E-utilities will receive an error message. By including an API key, a site can post up to 10 requests per second by default. Higher rates are available by request (eutilities@ncbi.nlm.nih.gov). Users can obtain an API key now from the Settings page of their NCBI account (to create an account, visit http://www.ncbi.nlm.nih.gov/account/). After creating the key, users should include it in each E-utility request by assigning it to the *api_key* parameter.

Example request including an API key:

```
esummary.fcgi?db=pubmed&id=123456&api_key=ABCDE12345
```

Example error message if rates are exceeded:

```
{"error":"API rate limit exceeded","count":"11"}
```

Only one API key is allowed per NCBI account; however, a user may request a new key at any time. Such a request will invalidate any existing API key associated with that NCBI account.

Minimizing the Number of Requests

If a task requires searching for and/or downloading a large number of records, it is much more efficient to use the Entrez History to upload and/or retrieve these records in batches rather than using separate requests for each record. Please refer to Application 3 in Chapter 3 for an example. Many thousands of IDs can be uploaded using a single EPost request, and several hundred records can be downloaded using one EFetch request.

Disclaimer and Copyright Issues

If you use the E-utilities within software, NCBI's Disclaimer and Copyright notice (https://www.ncbi.nlm.nih.gov/About/disclaimer.html) must be evident to users of your product. Please note that abstracts in PubMed may incorporate material that may be protected by U.S. and foreign copyright laws. All persons reproducing, redistributing, or making commercial use of this information are expected to adhere to the terms and conditions asserted by the copyright holder. Transmission or reproduction of protected items beyond that allowed by fair use (PDF) as defined in the copyright laws requires the written permission of the copyright owners. NLM provides no legal advice concerning distribution of copyrighted materials. Please consult your legal counsel. If you wish to do a large data mining project on PubMed data, you can download a local copy of the database at https://www.nlm.nih.gov/databases/download/pubmed_medline.html.

Handling Special Characters Within URLs

When constructing URLs for the E-utilities, please use lowercase characters for all parameters except &WebEnv. There is no required order for the URL parameters in an E-utility URL, and null values or inappropriate parameters are generally ignored. Avoid placing spaces in the URLs, particularly in queries. If a space is required, use a plus sign (+) instead of a space:

```
Incorrect: &id=352, 25125, 234
Correct:   &id=352,25125,234

Incorrect: &term=biomol mrna[properties] AND mouse[organism]
Correct:   &term=biomol+mrna[properties]+AND+mouse[organism]
```

Other special characters, such as quotation marks (“) or the # symbol used in referring to a query key on the History server, should be represented by their URL encodings (%22 for “; %23 for #).

```
Incorrect: &term=#2+AND+"gene in genomic"[properties]
Correct:   &term=%232+AND+%22gene+in+genomic%22[properties]
```

The Nine E-utilities in Brief

EInfo (database statistics)

eutils.ncbi.nlm.nih.gov/entrez/eutils/einfo.fcgi

Provides the number of records indexed in each field of a given database, the date of the last update of the database, and the available links from the database to other Entrez databases.

ESearch (text searches)

eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi

Responds to a text query with the list of matching UIDs in a given database (for later use in ESummary, EFetch or ELink), along with the term translations of the query.

EPost (UID uploads)

eutils.ncbi.nlm.nih.gov/entrez/eutils/epost.fcgi

Accepts a list of UIDs from a given database, stores the set on the History Server, and responds with a query key and web environment for the uploaded dataset.

ESummary (document summary downloads)

eutils.ncbi.nlm.nih.gov/entrez/eutils/esummary.fcgi

Responds to a list of UIDs from a given database with the corresponding document summaries.

EFetch (data record downloads)

eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi

Responds to a list of UIDs in a given database with the corresponding data records in a specified format.

ELink (Entrez links)

eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi

Responds to a list of UIDs in a given database with either a list of related UIDs (and relevancy scores) in the same database or a list of linked UIDs in another Entrez database; checks for the existence of a specified link from a list of one or more UIDs; creates a hyperlink to the primary LinkOut provider for a specific UID and database, or lists LinkOut URLs and attributes for multiple UIDs.

EGQuery (global query)

eutils.ncbi.nlm.nih.gov/entrez/eutils/egquery.fcgi

Responds to a text query with the number of records matching the query in each Entrez database.

ESpell (spelling suggestions)

eutils.ncbi.nlm.nih.gov/entrez/eutils/espell.fcgi

Retrieves spelling suggestions for a text query in a given database.

ECitMatch (batch citation searching in PubMed)

eutils.ncbi.nlm.nih.gov/entrez/eutils/ecitmatch.cgi

Retrieves PubMed IDs (PMIDs) corresponding to a set of input citation strings.

Understanding the E-utilities Within Entrez

The E-utilities Access Entrez Databases

The E-utilities access the core search and retrieval engine of the Entrez system and, therefore, are only capable of retrieving data that are already in Entrez. Although the majority of data at NCBI are in Entrez, there are several datasets that exist outside of the Entrez system. Before beginning a project with the E-utilities, check that the desired data can be found within an Entrez database.

The Entrez System Identifies Database Records Using UIDs

Each Entrez database refers to the data records within it by an integer ID called a UID (unique identifier). Examples of UIDs are GI numbers for Nucleotide and Protein, PMIDs for PubMed, or MMDB-IDs for Structure. The E-utilities use UIDs for both data input and output, and thus it is often critical, especially for advanced data pipelines, to know how to find the UIDs associated with the desired data before beginning a project with the E-utilities.

See Table 1 for a complete list of UIDs in Entrez.

Table 1 – Entrez Unique Identifiers (UIDs) for selected databases

Entrez Database	UID common name	E-utility Database Name
BioProject	BioProject ID	bioproject
BioSample	BioSample ID	biosample
Books	Book ID	books
Conserved Domains	PSSM-ID	cdd
dbGaP	dbGaP ID	gap
dbVar	dbVar ID	dbvar
Gene	Gene ID	gene
Genome	Genome ID	genome
GEO Datasets	GDS ID	gds
GEO Profiles	GEO ID	geoprofiles
HomoloGene	HomoloGene ID	homologene

Table 1 continued from previous page.

Entrez Database	UID common name	E-utility Database Name
MeSH	MeSH ID	mesh
NCBI C++ Toolkit	Toolkit ID	toolkit
NLM Catalog	NLM Catalog ID	nlmcatalog
Nucleotide	GI number	nuccore
PopSet	PopSet ID	popset
Probe	Probe ID	probe
Protein	GI number	protein
Protein Clusters	Protein Cluster ID	proteinclusters
PubChem BioAssay	AID	pcassay
PubChem Compound	CID	pccompound
PubChem Substance	SID	pcsubstance
PubMed	PMID	pubmed
PubMed Central	PMCID	pmc
SNP	rs number	snp
SRA	SRA ID	sra
Structure	MMDB-ID	structure
Taxonomy	TaxID	taxonomy

Accessing Sequence Records Using Accession.Version Identifiers

NCBI now uses the accession.version identifier rather that the GI number (UID) as the primary identifier for nucleotide and protein sequence records (records in the nuccore, nucest, nucgss, popset, and protein databases). Even so, the E-utilities continue to provide access to these records using either GI numbers or accession.version identifiers. Those E-utilities that accept UIDs as input will also accept accession.version identifiers (for the sequence databases listed above). Those E-utilities that output UIDs can output accession.version identifiers instead by setting the &idtype parameter to "acc". Finally, EFetch can retrieve *any* sequence record by its accession.version identifier, including sequences that do not have GI numbers. Please see Chapter 4 for more details about how each E-utility handles accession.version identifers.

The Entrez Core Engine: EGQuery, ESearch, and ESummary

The core of Entrez is an engine that performs two basic tasks for any Entrez database: 1) assemble a list of UIDs that match a text query, and 2) retrieve a brief summary record called a Document Summary (DocSum) for each UID. These two basic tasks of the Entrez engine are performed by ESearch and ESummary. ESearch returns a list of UIDs that match a text query in a given Entrez database, and ESummary returns DocSums that match a list of input UIDs. A text search in web Entrez is equivalent to ESearch-ESummary. EGQuery is a global version of ESearch that searches all Entrez databases simultaneously. Because these three E-utilities perform the two core Entrez functions, they function for all Entrez databases.

```
egquery.fcgi?term=query
esearch.fcgi?db=database&term=query
esummary.fcgi?db=database&id=uid1,uid2,uid3,...
```

Syntax and Initial Parsing of Entrez Queries

Text search strings entered into the Entrez system are converted into Entrez queries with the following format:

term1[field1] **Op** term2[field2] **Op** term3[field3] **Op** ...

where the terms are search terms, each limited to a particular Entrez field in square brackets, combined using one of three Boolean operators: Op = AND, OR, or NOT. These Boolean operators must be typed in all capital letters.

Example: human[organism] AND topoisomerase[protein name]

Entrez initially splits the query into a series of items that were originally separated by spaces in the query; therefore it is critical that spaces separate each term and Boolean operator. If the query consists *only* of a list of UID numbers (unique identifiers) or accession numbers, the Entrez system simply returns the corresponding records and no further parsing is performed. If the query contains any Boolean operators (AND, OR, or NOT), the query is split into the terms separated by these operators, and then each term is parsed independently. The results of these searches are then combined according to the Boolean operators.

A full account of how to search Entrez can be found in the Entrez Help Document. Additional information is available from Entrez Help.

Entrez Databases: EInfo, EFetch, and ELink

The NCBI Entrez system currently contains 38 databases. EInfo provides detailed information about each database, including lists of the indexing fields in the database and the available links to other Entrez databases.

`einfo.fcgi?db=`*`database`*

Each Entrez database includes two primary enhancements to the raw data records: 1) software for producing a variety of display formats appropriate to the given database, and 2) links to records in other Entrez databases manifested as lists of associated UIDs. The display format function is performed by EFetch, which generates formatted output for a list of input UIDs. For example, EFetch can produce abstracts from Entrez PubMed or FASTA format from Entrez Protein. EFetch does not yet support all Entrez databases; please see the EFetch documentation for details.

`efetch.fcgi?db=database&id=`*`uid1,uid2,uid3`*`&rettype=`*`report_type`*`&retmode=`
`data_mode`

The linking function is performed by ELink, which generates a list of UIDs in a specified Entrez database that are linked to a set of input UIDs in either the same or another database. For example, ELink can find Entrez SNP records linked to records in Entrez Nucleotide, or Entrez Domain records linked to records in Entrez Protein.

`elink.fcgi?dbfrom=`*`initial_database`*`db=`*`target_database`*`&id=`*`uid1,uid2,uid3`*

Using the Entrez History Server

A powerful feature of the Entrez system is that it can store retrieved sets of UIDs temporarily on the servers so that they can be subsequently combined or provided as input for other E-utility calls. The Entrez History server provides this service and is accessed on the Web using either the Preview/Index or History tabs on Entrez search pages. Each of the E-utilities can also use the History server, which assigns each set of UIDs an integer label called a query key (&query_key) and an encoded cookie string called a Web environment (&WebEnv). EPost allows any list of UIDs to be uploaded to the History Server and returns the query key and Web environment. ESearch can also post its output set of UIDs to the History Server, but only if the &usehistory parameter is set to “y”. ELink also can post its output to the History server if &cmd is set to "neighbor_history". The resulting query

key and Web environment from either EPost or ESearch can then be used in place of a UID list in ESummary, EFetch, and ELink.

In Entrez, a set of UIDs is represented on the History by three parameters:

```
&db = database; &query_key = query key; &WebEnv = web environment
```

Upload steps that generate a web environment and query key

```
esearch.fcgi?db=database&term=query&usehistory=y

epost.fcgi?db=database&id=uid1,uid2,uid3,...

elink.fcgi?dbfrom=source_db&db=destination_db&cmd=neighbor_history&id=
uid1,uid2,...
```

Download steps that use a web environment and query key

```
esummary.fcgi?db=database&WebEnv=webenv&query_key=key

efetch.fcgi?db=database&WebEnv=webenv&query_key=key&rettype=
report_type&retmode=data_mode
```

Link step that uses a web environment and query key

```
elink.fcgi?dbfrom=initial_databasedb=target_database&WebEnv=
webenv&query_key=key
```

Search step that uses a web environment and a query key in the &term parameter (preceded by #, encoded as %23)

```
esearch.fcgi?db=database&term=%23key+AND+query&WebEnv=webenv&usehistory=y
```

Generating Multiple Data Sets on the History Server

Each web environment on the History Server can be associated with any number of query keys. This allows different data sets to be combined with the Boolean operators AND, OR, and NOT, or with another Entrez query. It is important to remember that for two data sets (query keys) to be combined, they must be associated with the same web environment. By default, successive E-utility calls produce query keys that are *not* associated with the same web environment, and so to overcome this, each E-utility call after the initial call must set the &WebEnv parameter to the value of the pre-existing web environment.

Default behavior: These two URLs…

```
URL 1: epost.fcgi?db=database&id=uid1,uid2,uid3
URL 2: esearch.fcgi?db=database&term=query&usehistory=y
```

will produce two History sets associated with different web environments:

URL	WebEnv	query_key	UIDs
1	web1	1	uid1,uid2,uid3
2	web2	1	uids matching query

Desired behavior: These two URLs…

```
URL 1: epost.fcgi?db=database&id=uid1,uid2,uid3
(extract web1 from the output of URL 1)
URL 2: esearch.fcgi?db=database&term=query&usehistory=y&WebEnv=web1
```

will produce two sets associated with the same (new) web environment:

```
URL    WebEnv       query_key         UIDs
1       web2           1          uid1,uid2,uid3
2       web2           2          uids matching query
```

Combining E-utility Calls to Create Entrez Applications

The E-utilities are useful when used by themselves in single URLs; however, their full potential is realized when successive E-utility URLs are combined to create a data pipeline. When used within such pipelines, the Entrez History server simplifies complex retrieval tasks by allowing easy data transfer between successive E-utility calls. Listed below are several examples of pipelines produced by combining E-utilities, with the arrows representing the passing of db, WebEnv and query_key values from one E-utility to another. These and related pipelines are discussed in detail in Chapter 3.

Basic Pipelines

Retrieving data records matching an Entrez query

ESearch → ESummary

ESearch → EFetch

Retrieving data records matching a list of UIDs

EPost → ESummary

EPost → EFetch

Finding UIDs linked to a set of records

ESearch → ELink

EPost → ELink

Limiting a set of records with an Entrez query

EPost → ESearch

ELink → ESearch

Advanced Pipelines

Retrieving data records in database B linked to records in database A matching an Entrez query

ESearch → ELink → ESummary

ESearch → ELink → EFetch

Retrieving data records from a subset of an ID list defined by an Entrez query

EPost → ESearch → ESummary

EPost → ESearch → EFetch

Retrieving a set of data records, defined by an Entrez query, in database B from a larger set of records linked to a list of UIDs in database A

EPost → ELink → ESearch → ESummary

EPost → ELink → ESearch → EFetch

Demonstration Programs

Please see Chapter 1 for sample Perl scripts.

For More Information

Please see Chapter 1 for getting additional information about the E-utilities.

Sample Applications of the E-utilities

Eric Sayers, PhD[1]

Created: April 24, 2009; Updated: November 1, 2017.

Introduction

This chapter presents several examples of how the E-utilities can be used to build useful applications. These examples use Perl to create the E-utility pipelines, and assume that the LWP::Simple module is installed. This module includes the *get* function that supports HTTP GET requests. One example (Application 4) uses an HTTP POST request, and requires the LWP::UserAgent module. In Perl, scalar variable names are preceded by a "$" symbol, and array names are preceded by a "@". In several instances, results will be stored in such variables for use in subsequent E-utility calls. The code examples here are working programs that can be copied to a text editor and executed directly. Equivalent HTTP requests can be constructed in many modern programming languages; all that is required is the ability to create and post an HTTP request.

Basic Pipelines

All E-utility applications consist of a series of calls that we will refer to as a pipeline. The simplest E-utility pipelines consist of two calls, and any arbitrary pipeline can be assembled from these basic building blocks. Many of these pipelines conclude with either ESummary (to retrieve DocSums) or EFetch (to retrieve full records). The comments indicate those portions of the code that are required for either call.

ESearch – ESummary/EFetch

Input: Entrez text query

ESummary Output: XML Document Summaries

EFetch Output: Formatted data records (e.g. abstracts, FASTA)

```
use LWP::Simple;

# Download PubMed records that are indexed in MeSH for both asthma and
# leukotrienes and were also published in 2009.

$db = 'pubmed';
$query = 'asthma[mesh]+AND+leukotrienes[mesh]+AND+2009[pdat]';

#assemble the esearch URL
$base = 'https://eutils.ncbi.nlm.nih.gov/entrez/eutils/';
$url = $base . "esearch.fcgi?db=$db&term=$query&usehistory=y";

#post the esearch URL
$output = get($url);

#parse WebEnv and QueryKey
$web = $1 if ($output =~ /<WebEnv>(\S+)<\/WebEnv>/);
$key = $1 if ($output =~ /<QueryKey>(\d+)<\/QueryKey>/);
```

Author Affiliation: 1 NCBI; Email: sayers@ncbi.nlm.nih.gov.

Corresponding author.

```
### include this code for ESearch-ESummary
#assemble the esummary URL
$url = $base . "esummary.fcgi?db=$db&query_key=$key&WebEnv=$web";

#post the esummary URL
$docsums = get($url);
print "$docsums";

### include this code for ESearch-EFetch
#assemble the efetch URL
$url = $base . "efetch.fcgi?db=$db&query_key=$key&WebEnv=$web";
$url .= "&rettype=abstract&retmode=text";

#post the efetch URL
$data = get($url);
print "$data";
```

EPost – ESummary/EFetch

Input: List of Entrez UIDs (integer identifiers, e.g. PMID, GI, Gene ID)

ESummary Output: XML Document Summaries

EFetch Output: Formatted data records (e.g. abstracts, FASTA)

```
use LWP::Simple;

# Download protein records corresponding to a list of GI numbers.

$db = 'protein';
$id_list = '194680922,50978626,28558982,9507199,6678417';

#assemble the epost URL
$base = 'https://eutils.ncbi.nlm.nih.gov/entrez/eutils/';
$url = $base . "epost.fcgi?db=$db&id=$id_list";

#post the epost URL
$output = get($url);

#parse WebEnv and QueryKey
$web = $1 if ($output =~ /<WebEnv>(\S+)<\/WebEnv>/);
$key = $1 if ($output =~ /<QueryKey>(\d+)<\/QueryKey>/);

### include this code for EPost-ESummary
#assemble the esummary URL
$url = $base . "esummary.fcgi?db=$db&query_key=$key&WebEnv=$web";

#post the esummary URL
$docsums = get($url);
print "$docsums";

### include this code for EPost-EFetch
#assemble the efetch URL
$url = $base . "efetch.fcgi?db=$db&query_key=$key&WebEnv=$web";
$url .= "&rettype=fasta&retmode=text";

#post the efetch URL
$data = get($url);
print "$data";
```

Note: *To post a large number (more than a few hundred) UIDs in a single URL, please use the HTTP POST method for the EPost call (see Application 4).*

ELink – ESummary/Efetch

Input: List of Entrez UIDs in database A (integer identifiers, e.g. PMID, GI, Gene ID)

ESummary Output: Linked XML Document Summaries from database B

EFetch Output: Formatted data records (e.g. abstracts, FASTA) from database B

```
use LWP::Simple;

# Download gene records linked to a set of proteins corresponding to a list
# of GI numbers.

$db1 = 'protein';  # &dbfrom
$db2 = 'gene';     # &db
$linkname = 'protein_gene'; # desired link &linkname
#input UIDs in $db1 (protein GIs)
$id_list = '194680922,50978626,28558982,9507199,6678417';

#assemble the elink URL
$base = 'https://eutils.ncbi.nlm.nih.gov/entrez/eutils/';
$url = $base . "elink.fcgi?dbfrom=$db1&db=$db2&id=$id_list";
$url .= "&linkname=$linkname&cmd=neighbor_history";

#post the elink URL
$output = get($url);

#parse WebEnv and QueryKey
$web = $1 if ($output =~ /<WebEnv>(\S+)<\/WebEnv>/);
$key = $1 if ($output =~ /<QueryKey>(\d+)<\/QueryKey>/);

### include this code for ELink-ESummary
#assemble the esummary URL
$url = $base . "esummary.fcgi?db=$db&query_key=$key&WebEnv=$web";

#post the esummary URL
$docsums = get($url);
print "$docsums";

### include this code for ELink-EFetch
#assemble the efetch URL
$url = $base . "efetch.fcgi?db=$db2&query_key=$key&WebEnv=$web";
$url .= "&rettype=xml&retmode=xml";

#post the efetch URL
$data = get($url);
print "$data";
```

Notes: *To submit a large number (more than a few hundred) UIDs to ELink in one URL, please use the HTTP POST method for the Elink call (see Application 4). The &linkname parameter is used to force ELink to return only one set of links (one &query_key) to simplify parsing. If more than one link is desired, the above code must be altered to parse the multiple &query_key values from the ELink XML output. This code uses ELink in "batch" mode, in that only one set of gene IDs is returned and the one-to-one correspondence between protein GIs and gene IDs is lost. To preserve this one-to-one correspondence, please see Application 4 below.*

ESearch – ELink – ESummary/EFetch

Input: Entrez text query in database A

ESummary Output: Linked XML Document Summaries from database B

EFetch Output: Formatted data records (e.g. abstracts, FASTA) from database B

```
use LWP::Simple;
# Download protein FASTA records linked to abstracts published
# in 2009 that are indexed in MeSH for both asthma and
# leukotrienes.

$db1 = 'pubmed';
$db2 = 'protein';
$linkname = 'pubmed_protein';
$query = 'asthma[mesh]+AND+leukotrienes[mesh]+AND+2009[pdat]';

#assemble the esearch URL
$base = 'https://eutils.ncbi.nlm.nih.gov/entrez/eutils/';
$url = $base . "esearch.fcgi?db=$db1&term=$query&usehistory=y";
#post the esearch URL
$output = get($url);

#parse WebEnv and QueryKey
$web1 = $1 if ($output =~ /<WebEnv>(\S+)<\/WebEnv>/);
$key1 = $1 if ($output =~ /<QueryKey>(\d+)<\/QueryKey>/);

#assemble the elink URL
$base = 'https://eutils.ncbi.nlm.nih.gov/entrez/eutils/';
$url = $base . "elink.fcgi?dbfrom=$db1&db=$db2";
$url .= "&query_key=$key1&WebEnv=$web1";
$url .= "&linkname=$linkname&cmd=neighbor_history";
print "$url\n";

#post the elink URL
$output = get($url);
print "$output\n";

#parse WebEnv and QueryKey
$web2 = $1 if ($output =~ /<WebEnv>(\S+)<\/WebEnv>/);
$key2 = $1 if ($output =~ /<QueryKey>(\d+)<\/QueryKey>/);

### include this code for ESearch-ELink-ESummary
#assemble the esummary URL
$url = $base . "esummary.fcgi?db=$db2&query_key=$key2&WebEnv=$web2";
#post the esummary URL
$docsums = get($url);
print "$docsums";

### include this code for ESearch-ELink-EFetch
#assemble the efetch URL
$url = $base . "efetch.fcgi?db=$db2&query_key=$key2&WebEnv=$web2";
$url .= "&rettype=fasta&retmode=text";
#post the efetch URL
$data = get($url);
print "$data";
```

Notes: *The &linkname parameter is used to force ELink to return only one set of links (one &query_key) to simplify parsing. If more than one link is desired, the above code must be altered to parse the multiple &query_key values from the ELink XML output. This code uses ELink in "batch" mode, in that only one set of PubMed IDs is returned and the one-to-one correspondence between PubMed IDs and their related PubMed IDs is lost. To preserve this one-to-one correspondence, please see Application 4 below.*

EPost – ELink – ESummary/EFetch

Input: List of Entrez UIDs (integer identifiers, e.g. PMID, GI, Gene ID) in database A

ESummary Output: Linked XML Document Summaries from database B

EFetch Output: Formatted data records (e.g. abstracts, FASTA) from database B

```
use LWP::Simple;

# Downloads gene records linked to a set of proteins corresponding
# to a list of protein GI numbers.

$db1 = 'protein';  # &dbfrom
$db2 = 'gene';     # &db
$linkname = 'protein_gene';
#input UIDs in $db1 (protein GIs)
$id_list = '194680922,50978626,28558982,9507199,6678417';

#assemble the epost URL
$base = 'https://eutils.ncbi.nlm.nih.gov/entrez/eutils/';
$url = $base . "epost.fcgi?db=$db1&id=$id_list";

#post the epost URL
$output = get($url);

#parse WebEnv and QueryKey
$web1 = $1 if ($output =~ /<WebEnv>(\S+)<\/WebEnv>/);
$key1 = $1 if ($output =~ /<QueryKey>(\d+)<\/QueryKey>/);

#assemble the elink URL
$base = 'https://eutils.ncbi.nlm.nih.gov/entrez/eutils/';
$url = $base . "elink.fcgi?dbfrom=$db1&db=$db2&query_key=$key1";
$url .= "&WebEnv=$web1&linkname=$linkname&cmd=neighbor_history";

#post the elink URL
$output = get($url);

#parse WebEnv and QueryKey
$web2 = $1 if ($output =~ /<WebEnv>(\S+)<\/WebEnv>/);
$key2 = $1 if ($output =~ /<QueryKey>(\d+)<\/QueryKey>/);

### include this code for ESearch-ELink-ESummary
#assemble the esummary URL
$url = $base . "esummary.fcgi?db=$db2&query_key=$key2&WebEnv=$web2";

#post the esummary URL
$docsums = get($url);
print "$docsums";

### include this code for ESearch-ELink-EFetch
#assemble the efetch URL
```

```
$url = $base . "efetch.fcgi?db=$db2&query_key=$key2&WebEnv=$web2";
$url .= "&rettype=xml&retmode=xml";

#post the efetch URL
$data = get($url);
print "$data";
```

Notes: *To post a large number (more than a few hundred) UIDs in a single URL, please use the HTTP POST method for the EPost call (see Application 4 below). The &linkname parameter is used to force ELink to return only one set of links (one &query_key) to simplify parsing. If more than one link is desired, the above code must be altered to parse the multiple &query_key values from the ELink XML output. This code uses ELink in "batch" mode, in that only one set of gene IDs is returned and the one-to-one correspondence between protein GIs and Gene IDs is lost. To preserve this one-to-one correspondence, please see Application 4 below.*

EPost – ESearch

Input: List of Entrez UIDs (integer identifiers, e.g. PMID, GI, Gene ID)

Output: History set consisting of the subset of posted UIDs that match an Entrez text query

```
use LWP::Simple;

# Given an input set of protein GI numbers, this script creates
# a history set containing the members of the input set that
# correspond to human proteins.
#(Which of these proteins are from human?)

$db = 'protein';
$query = 'human[orgn]';
$id_list = '194680922,50978626,28558982,9507199,6678417';

#assemble the epost URL
$base = 'https://eutils.ncbi.nlm.nih.gov/entrez/eutils/';
$url = $base . "epost.fcgi?db=$db&id=$id_list";

#post the epost URL
$output = get($url);

#parse WebEnv and QueryKey
$web = $1 if ($output =~ /<WebEnv>(\S+)<\/WebEnv>/);
$key = $1 if ($output =~ /<QueryKey>(\d+)<\/QueryKey>/);

#assemble the esearch URL
$term = "%23$key+AND+$query";
# %23 places a '#' before the query key
$url = $base . "esearch.fcgi?db=$db&term=$term";
$url .= "&WebEnv=$web&usehistory=y";

#post esearch URL
$limited = get($url);

print "$limited\n";

# Output remains on the history server (&query_key, &WebEnv)
# Use ESummary or EFetch as above to retrieve them
```

Note: *To post a large number (more than a few hundred) UIDs in a single URL, please use the HTTP POST method for the EPost call (see Application 4).*

ELink – ESearch

Input: List of Entrez UIDs (integer identifiers, e.g. PMID, GI, Gene ID) in database A

Output: History set consisting of the subset of linked UIDs in database B that match an Entrez text query

```
use LWP::Simple;

# Given an input set of protein GI numbers, this script creates a
# history set containing the gene IDs linked to members of the input
# set that also are on human chromosome X.
#(Which of the input proteins are encoded by a gene on human
# chromosome X?)

$db1 = 'protein';  # &dbfrom
$db2 = 'gene';     # &db
$linkname = 'protein_gene'; # desired link &linkname
$query = 'human[orgn]+AND+x[chr]';
#input UIDs in $db1 (protein GIs)
$id_list = '148596974,42544182,187937179,4557377,6678417';

#assemble the elink URL
$base = 'https://eutils.ncbi.nlm.nih.gov/entrez/eutils/';
$url = $base . "elink.fcgi?dbfrom=$db1&db=$db2&id=$id_list";
$url .= "&linkname=$linkname&cmd=neighbor_history";

#post the elink URL
$output = get($url);

#parse WebEnv and QueryKey
$web = $1 if ($output =~ /<WebEnv>(\S+)<\/WebEnv>/);
$key = $1 if ($output =~ /<QueryKey>(\d+)<\/QueryKey>/);

#assemble the esearch URL
$term = "%23$key+AND+$query";  # %23 places a '#' before the query key
$url = $base . "esearch.fcgi?db=$db2&term=$term&WebEnv=$web&usehistory=y";

#post esearch URL
$limited = get($url);

print "$limited\n";

# Output remains on the history server (&query_key, &WebEnv)
# Use ESummary or EFetch as in previous examples to retrieve them
```

Note: *To submit a large number (more than a few hundred) UIDs to ELink in one URL, please use the HTTP POST method for the Elink call (see Application 4). The &linkname parameter is used to force ELink to return only one set of links (one &query_key) to simplify parsing. If more than one link is desired, the above code must be altered to parse the multiple &query_key values from the ELink XML output. This code uses ELink in "batch" mode, in that only one set of gene IDs is returned and the one-to-one correspondence between protein GIs and Gene IDs is lost. To preserve this one-to-one correspondence, please see Application 4 below.*

Application 1: Converting GI numbers to accession numbers

Goal: Starting with a list of nucleotide GI numbers, prepare a set of corresponding accession numbers.

Solution: Use EFetch with &retttype=acc

Input: $gi_list – comma-delimited list of GI numbers

Output: List of accession numbers.

```
use LWP::Simple;
$gi_list = '24475906,224465210,50978625,9507198';

#assemble the URL
$base = 'https://eutils.ncbi.nlm.nih.gov/entrez/eutils/';
$url = $base . "efetch.fcgi?db=nucleotide&id=$gi_list&rettype=acc";

#post the URL
$output = get($url);
print "$output";
```

Notes: *The order of the accessions in the output will be the same order as the GI numbers in $gi_list.*

Application 2: Converting accession numbers to data

Goal: Starting with a list of protein accession numbers, return the sequences in FASTA format.

Solution: Create a string consisting of items separated by 'OR', where each item is an accession number followed by '[accn]'.

Example: accn1[accn]+OR+accn2[accn]+OR+accn3[accn]+OR+…

Submit this string as a &term in ESearch, then use EFetch to retrieve the FASTA data.

Input: $acc_list – comma-delimited list of accessions

Output: FASTA data

```
use LWP::Simple;
$acc_list = 'NM_009417,NM_000547,NM_001003009,NM_019353';
@acc_array = split(/,/, $acc_list);

#append [accn] field to each accession
for ($i=0; $i < @acc_array; $i++) {
   $acc_array[$i] .= "[accn]";
}

#join the accessions with OR
$query = join('+OR+',@acc_array);

#assemble the esearch URL
$base = 'https://eutils.ncbi.nlm.nih.gov/entrez/eutils/';
$url = $base . "esearch.fcgi?db=nuccore&term=$query&usehistory=y";

#post the esearch URL
$output = get($url);

#parse WebEnv and QueryKey
$web = $1 if ($output =~ /<WebEnv>(\S+)<\/WebEnv>/);
$key = $1 if ($output =~ /<QueryKey>(\d+)<\/QueryKey>/);

#assemble the efetch URL
$url = $base . "efetch.fcgi?db=nuccore&query_key=$key&WebEnv=$web";
$url .= "&rettype=fasta&retmode=text";
```

```
#post the efetch URL
$fasta = get($url);
print "$fasta";
```

Notes: *For large numbers of accessions, use HTTP POST to submit the esearch request (see Application 4), and see Application 3 below for downloading the large set in batches.*

Application 3: Retrieving large datasets

Goal: Download all chimpanzee mRNA sequences in FASTA format (>50,000 sequences).

Solution: First use ESearch to retrieve the GI numbers for these sequences and post them on the History server, then use multiple EFetch calls to retrieve the data in batches of 500.

Input: $query – chimpanzee[orgn]+AND+biomol+mrna[prop]

Output: A file named "chimp.fna" containing FASTA data.

```
use LWP::Simple;
$query = 'chimpanzee[orgn]+AND+biomol+mrna[prop]';

#assemble the esearch URL
$base = 'https://eutils.ncbi.nlm.nih.gov/entrez/eutils/';
$url = $base . "esearch.fcgi?db=nucleotide&term=$query&usehistory=y";

#post the esearch URL
$output = get($url);

#parse WebEnv, QueryKey and Count (# records retrieved)
$web = $1 if ($output =~ /<WebEnv>(\S+)<\/WebEnv>/);
$key = $1 if ($output =~ /<QueryKey>(\d+)<\/QueryKey>/);
$count = $1 if ($output =~ /<Count>(\d+)<\/Count>/);

#open output file for writing
open(OUT, ">chimp.fna") || die "Can't open file!\n";

#retrieve data in batches of 500
$retmax = 500;
for ($retstart = 0; $retstart < $count; $retstart += $retmax) {
        $efetch_url = $base ."efetch.fcgi?db=nucleotide&WebEnv=$web";
        $efetch_url .= "&query_key=$key&retstart=$retstart";
        $efetch_url .= "&retmax=$retmax&rettype=fasta&retmode=text";
        $efetch_out = get($efetch_url);
        print OUT "$efetch_out";
}
close OUT;
```

Application 4: Finding unique sets of linked records for each member of a large dataset

Goal: Download separately the SNP rs numbers (identifiers) for each current gene on human chromosome 20.

Solution: First use ESearch to retrieve the Gene IDs for the genes, and then assemble an ELink URL where each Gene ID is submitted as a separate &id parameter.

Input: $query – human[orgn]+AND+20[chr]+AND+alive[prop]

Output: A file named "snp_table" containing on each line the gene id followed by a colon (":") followed by a comma-delimited list of the linked SNP rs numbers.

```
use LWP::Simple;
use LWP::UserAgent;
$query = 'human[orgn]+AND+20[chr]+AND+alive[prop]';
$db1 = 'gene';
$db2 = 'snp';
$linkname = 'gene_snp';

#assemble the esearch URL
$base = 'https://eutils.ncbi.nlm.nih.gov/entrez/eutils/';
$url = $base . "esearch.fcgi?db=$db1&term=$query&usehistory=y&retmax=5000";

#post the esearch URL
$output = get($url);

#parse IDs retrieved
while ($output =~ /<Id>(\d+?)<\/Id>/sg) {
   push(@ids, $1);
}

#assemble  the elink URL as an HTTP POST call
$url = $base . "elink.fcgi";

$url_params = "dbfrom=$db1&db=$db2&linkname=$linkname";
foreach $id (@ids) {
   $url_params .= "&id=$id";
}

#create HTTP user agent
$ua = new LWP::UserAgent;
$ua->agent("elink/1.0 " . $ua->agent);

#create HTTP request object
$req = new HTTP::Request POST => "$url";
$req->content_type('application/x-www-form-urlencoded');
$req->content("$url_params");

#post the HTTP request
$response = $ua->request($req);
$output = $response->content;

open (OUT, ">snp_table") || die "Can't open file!\n";

while ($output =~ /<LinkSet>(.*?)<\/LinkSet>/sg) {

   $linkset = $1;
   if ($linkset =~ /<IdList>(.*?)<\/IdList>/sg) {
      $input = $1;
      $input_id = $1 if ($input =~ /<Id>(\d+)<\/Id>/sg);
   }

   while ($linkset =~ /<Link>(.*?)<\/Link>/sg) {
      $link = $1;
      push (@output, $1) if ($link =~ /<Id>(\d+)<\/Id>/);
   }
```

```
   print OUT "$input_id:" . join(',', @output) . "\n";

}

close OUT;
```

Notes: *This example uses an HTTP POST request for the elink call, as the number of Gene IDs is over 500. The &retmax parameter in the ESearch call is set to 5000, as this is a reasonable limit to the number of IDs to send to ELink in one request (if you send 5000 IDs, you are effectively performing 5000 ELink operations). If you need to link more than 5000 records, add &retstart to the ESearch call and repeat the entire procedure for each batch of 5000 IDs, incrementing &retstart for each batch.*

Demonstration Programs

Please see Chapter 1 for sample Perl scripts.

For More Information

Please see Chapter 1 for getting additional information about the E-utilities.

The E-utilities In-Depth: Parameters, Syntax and More

Eric Sayers, PhD[1]

Created: May 29, 2009; Updated: November 30, 2022.

Introduction

This chapter serves as a reference for all supported parameters for the E-utilities, along with accepted values and usage guidelines. This information is provided for each E-utility in sections below, and parameters and/or values specific to particular databases are discussed within each section. Most E-utilities have a set of parameters that are required for any call, in addition to several additional optional parameters that extend the tool's functionality. These two sets of parameters are discussed separately in each section.

General Usage Guidelines

Please see Chapter 2 for a detailed discussion of E-utility usage policy. The following two parameters should be included in all E-utility requests.

tool

Name of application making the E-utility call. Value must be a string with no internal spaces.

email

E-mail address of the E-utility user. Value must be a string with no internal spaces, and should be a valid e-mail address.

If you expect to post more than 3 E-utility requests per second from a single IP address, consider including the following parameter:

api_key

Value of the API key for sites that post more than 3 requests per second. Please see Chapter 2 for a full discussion of this policy.

E-utilities DTDs

With the exception of EFetch, the E-utilities each generate a single XML output format that conforms to a DTD specific for that utility. Links to these DTDs are provided in the XML headers of the E-utility returns.

ESummary version 2.0 produces unique XML DocSums for each Entrez database, and as such each Entrez database has a unique DTD for version 2.0 DocSums. Links to these DTDs are provided in the version 2.0 XML.

EFetch produces output in a variety of formats, some of which are XML. Most of these XML formats also conform to DTDs or schema specific to the relevant Entrez database. Please follow the appropriate link below for the PubMed DTD:

- PubMed DTD June 2018 – current PubMed DTD
- PubMed DTD January 2019 – forthcoming DTD

Author Affiliation: 1 NCBI; Email: sayers@ncbi.nlm.nih.gov.

Corresponding author.

EInfo

Base URL

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/einfo.fcgi

Functions

- Provides a list of the names of all valid Entrez databases
- Provides statistics for a single database, including lists of indexing fields and available link names

Required Parameters

None. If no **db** parameter is provided, einfo will return a list of the names of all valid Entrez databases.

Optional Parameters

db

Target database about which to gather statistics. Value must be a valid Entrez database name.

version

Used to specify version 2.0 EInfo XML. The only supported value is '2.0'. When present, EInfo will return XML that includes two new fields: <IsTruncatable> and <IsRangeable>. Fields that are truncatable allow the wildcard character '*' in terms. The wildcard character will expand to match any set of characters up to a limit of 600 unique expansions. Fields that are rangeable allow the range operator ':' to be placed between a lower and upper limit for the desired range (e.g. 2008:2010[pdat]).

retmode

Retrieval type. Determines the format of the returned output. The default value is 'xml' for EInfo XML, but 'json' is also supported to return output in JSON format.

Examples

Return a list of all Entrez database names:

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/einfo.fcgi

Return version 2.0 statistics for Entrez Protein:

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/einfo.fcgi?db=protein&version=2.0

ESearch

Base URL

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi

Functions

- Provides a list of UIDs matching a text query
- Posts the results of a search on the History server
- Downloads all UIDs from a dataset stored on the History server

- Combines or limits UID datasets stored on the History server
- Sorts sets of UIDs

API users should be aware that some NCBI products contain search tools that generate content from searches on the web interface that are not available to ESearch. For example, the PubMed web interface (pubmed.ncbi.nlm.nih.gov) contains citation matching and spelling correction tools that are only available through that interface. Please see ECitMatch and ESpell below for API equivalents.

Required Parameters

db

Database to search. Value must be a valid Entrez database name (default = pubmed).

term

Entrez text query. All special characters must be URL encoded. Spaces may be replaced by '+' signs. For very long queries (more than several hundred characters long), consider using an HTTP POST call. See the PubMed or Entrez help for information about search field descriptions and tags. Search fields and tags are database specific.

```
esearch.fcgi?db=pubmed&term=asthma
```

PubMed also offers "proximity searching" for multiple terms appearing in any order within a specified number of words from one another in the [Title] or [Title/Abstract] fields.

```
esearch.fcgi?db=pubmed&term="asthma treatment"[Title:~3]
```

Optional Parameters – History Server

usehistory

When **usehistory** is set to 'y', ESearch will post the UIDs resulting from the search operation onto the History server so that they can be used directly in a subsequent E-utility call. Also, **usehistory** must be set to 'y' for ESearch to interpret query key values included in **term** or to accept a **WebEnv** as input.

WebEnv

Web environment string returned from a previous ESearch, EPost or ELink call. When provided, ESearch will post the results of the search operation to this pre-existing WebEnv, thereby appending the results to the existing environment. In addition, providing **WebEnv** allows query keys to be used in **term** so that previous search sets can be combined or limited. As described above, if **WebEnv** is used, **usehistory** must be set to 'y'.

```
esearch.fcgi?db=pubmed&term=asthma&WebEnv=<webenv string>&usehistory=y
```

query_key

Integer query key returned by a previous ESearch, EPost or ELink call. When provided, ESearch will find the intersection of the set specified by **query_key** and the set retrieved by the query in **term** (i.e. joins the two with AND). For **query_key** to function, **WebEnv** must be assigned an existing WebEnv string and **usehistory** must be set to 'y'.

Values for query keys may also be provided in **term** if they are preceeded by a '#' (%23 in the URL). While only one **query_key** parameter can be provided to ESearch, any number of query keys can be combined in **term**. Also, if query keys are provided in **term**, they can be combined with OR or NOT in addition to AND.

```
The following two URLs are functionally equivalent:

esearch.fcgi?db=pubmed&term=asthma&query_key=1&WebEnv=
<webenv string>&usehistory=y

esearch.fcgi?db=pubmed&term=%231+AND+asthma&WebEnv=
<webenv string>&usehistory=y
```

Optional Parameters – Retrieval

retstart

Sequential index of the first UID in the retrieved set to be shown in the XML output (default=0, corresponding to the first record of the entire set). This parameter can be used in conjunction with **retmax** to download an arbitrary subset of UIDs retrieved from a search.

retmax

Total number of UIDs from the retrieved set to be shown in the XML output (default=20). By default, ESearch only includes the first 20 UIDs retrieved in the XML output. If **usehistory** is set to 'y', the remainder of the retrieved set will be stored on the History server; otherwise these UIDs are lost. Increasing **retmax** allows more of the retrieved UIDs to be included in the XML output, up to a maximum of 10,000 records.

To retrieve more than 10,000 UIDs from databases other than PubMed, submit multiple esearch requests while incrementing the value of **retstart** (see Application 3). For PubMed, ESearch can only retrieve the first 10,000 records matching the query. To obtain more than 10,000 PubMed records, consider using <EDirect> that contains additional logic to batch PubMed search results automatically so that an arbitrary number can be retrieved.

rettype

Retrieval type. There are two allowed values for ESearch: 'uilist' (default), which displays the standard XML output, and 'count', which displays only the <Count> tag.

retmode

Retrieval type. Determines the format of the returned output. The default value is 'xml' for ESearch XML, but 'json' is also supported to return output in JSON format.

sort

Specifies the method used to sort UIDs in the ESearch output. The available values vary by database (**db**) and may be found in the Display Settings menu on an Entrez search results page. If **usehistory** is set to 'y', the UIDs are loaded onto the History Server in the specified sort order and will be retrieved in that order by ESummary or EFetch. Example values are 'relevance' and 'name' for Gene. Users should be aware that the default value of **sort** varies from one database to another, and that the default value used by ESearch for a given database may differ from that used on NCBI web search pages.

Values of **sort** for PubMed are as follows:

- *pub_date* – descending sort by publication date
- *Author* – ascending sort by first author
- *JournalName* – ascending sort by journal name
- *relevance* – default sort order, ("Best Match") on web PubMed

field

Search field. If used, the entire search term will be limited to the specified Entrez field. The following two URLs are equivalent:

```
esearch.fcgi?db=pubmed&term=asthma&field=title

esearch.fcgi?db=pubmed&term=asthma[title]
```

idtype

Specifies the type of identifier to return for sequence databases (nuccore, popset, protein). By default, ESearch returns GI numbers in its output. If **idtype** is set to 'acc', ESearch will return accession.version identifiers rather than GI numbers.

Optional Parameters – Dates

datetype

Type of date used to limit a search. The allowed values vary between Entrez databases, but common values are 'mdat' (modification date), 'pdat' (publication date) and 'edat' (Entrez date). Generally an Entrez database will have only two allowed values for **datetype**.

reldate

When **reldate** is set to an integer *n*, the search returns only those items that have a date specified by **datetype** within the last *n* days.

mindate, maxdate

Date range used to limit a search result by the date specified by **datetype**. These two parameters (**mindate, maxdate**) must be used together to specify an arbitrary date range. The general date format is YYYY/MM/DD, and these variants are also allowed: YYYY, YYYY/MM.

Examples

Search in PubMed with the term *cancer* for abstracts that have an Entrez date within the last 60 days; retrieve the first 100 PMIDs and translations; post the results on the History server and return a **WebEnv** and **query_key**:

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi?db=pubmed&term=cancer&reldate=60&datetype=edat&retmax=100&usehistory=y

Search in PubMed for the journal PNAS, Volume 97, and retrieve six PMIDs starting with the seventh PMID in the list:

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi?db=pubmed&term=PNAS[ta]+AND+97[vi]&retstart=6&retmax=6&tool=biomed3

Search in the NLM Catalog for journals matching the term *obstetrics*:

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi?db=nlmcatalog&term=obstetrics+AND+ncbijournals[filter]

Search PubMed Central for free full text articles containing the query *stem cells*:

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi?db=pmc&term=stem+cells+AND+free+fulltext[filter]

Search in Nucleotide for all tRNAs:

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi?db=nucleotide&term=biomol+trna[prop]

Search in Protein for a molecular weight range:

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esearch.fcgi?db=protein&term=70000:90000[molecular+weight]

EPost

Base URL

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/epost.fcgi

Functions

- Uploads a list of UIDs to the Entrez History server
- Appends a list of UIDs to an existing set of UID lists attached to a Web Environment

Required Parameters

db

Database containing the UIDs in the input list. The value must be a valid Entrez database name (default = pubmed).

id

UID list. Either a single UID or a comma-delimited list of UIDs may be provided. All of the UIDs must be from the database specified by **db**. For PubMed, no more than 10,000 UIDs can be included in a single URL request. For other databases there is no set maximum for the number of UIDs that can be passed to epost, but if more than about 200 UIDs are to be posted, the request should be made using the HTTP POST method.

For sequence databases (nuccore, popset, protein), the UID list may be a mixed list of GI numbers and accession.version identifiers. **Note:** When using accession.version identifiers, there is a conversion step that takes place that causes large lists of identifiers to time out, even when using POST. Therefore, we recommend batching these types of requests in sizes of about 500 UIDs or less, to avoid retrieving only a partial amount of records from your original POST input list.

```
epost.fcgi?db=pubmed&id=19393038,30242208,29453458
epost.fcgi?db=protein&id=15718680,NP_001098858.1,119703751
```

Optional Parameter

WebEnv

Web Environment. If provided, this parameter specifies the Web Environment that will receive the UID list sent by post. EPost will create a new query key associated with that Web Environment. Usually this WebEnv value is obtained from the output of a previous ESearch, EPost or ELink call. If no **WebEnv** parameter is provided, EPost will create a new Web Environment and post the UID list to **query_key** 1.

```
epost.fcgi?db=protein&id=15718680,157427902,119703751&WebEnv=
<webenv string>
```

Example

Post records to PubMed:

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/epost.fcgi?db=pubmed&id=11237011,12466850

ESummary

Base URL

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esummary.fcgi

Functions

- Returns document summaries (DocSums) for a list of input UIDs
- Returns DocSums for a set of UIDs stored on the Entrez History server

Required Parameter

db

Database from which to retrieve DocSums. The value must be a valid Entrez database name (default = pubmed).

Required Parameter – Used only when input is from a UID list

id

UID list. Either a single UID or a comma-delimited list of UIDs may be provided. All of the UIDs must be from the database specified by **db**. There is no set maximum for the number of UIDs that can be passed to ESummary, but if more than about 200 UIDs are to be provided, the request should be made using the HTTP POST method.

For sequence databases (nuccore, popset, protein), the UID list may be a mixed list of GI numbers and accession.version identifiers.

```
esummary.fcgi?db=pubmed&id=19393038,30242208,29453458
esummary.fcgi?db=protein&id=15718680,NP_001098858.1,119703751
```

Required Parameters – Used only when input is from the Entrez History server

query_key

Query key. This integer specifies which of the UID lists attached to the given Web Environment will be used as input to ESummary. Query keys are obtained from the output of previous ESearch, EPost or ELink calls. The **query_key** parameter must be used in conjunction with **WebEnv**.

WebEnv

Web Environment. This parameter specifies the Web Environment that contains the UID list to be provided as input to ESummary. Usually this WebEnv value is obtained from the output of a previous ESearch, EPost or ELink call. The **WebEnv** parameter must be used in conjunction with **query_key**.

```
esummary.fcgi?db=protein&query_key=<key>&WebEnv=<webenv string>
```

Optional Parameters – Retrieval

retstart

Sequential index of the first DocSum to be retrieved (default=1, corresponding to the first record of the entire set). This parameter can be used in conjunction with **retmax** to download an arbitrary subset of DocSums from the input set.

retmax

Total number of DocSums from the input set to be retrieved, up to a maximum of 10,000. If the total set is larger than this maximum, the value of **retstart** can be iterated while holding **retmax** constant, thereby downloading the entire set in batches of size **retmax**.

retmode

Retrieval type. Determines the format of the returned output. The default value is 'xml' for ESummary XML, but 'json' is also supported to return output in JSON format.

version

Used to specify version 2.0 ESummary XML. The only supported value is '2.0'. When present, ESummary will return version 2.0 DocSum XML that is unique to each Entrez database and that often contains more data than the default DocSum XML.

Examples

PubMed:

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esummary.fcgi?db=pubmed&id=11850928,11482001

PubMed, version 2.0 XML:

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esummary.fcgi?db=pubmed&id=11850928,11482001&version=2.0

Protein:

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esummary.fcgi?db=protein&id=28800982,28628843

Nucleotide:

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esummary.fcgi?db=nucleotide&id=28864546,28800981

Structure:

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esummary.fcgi?db=structure&id=19923,12120

Taxonomy:

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esummary.fcgi?db=taxonomy&id=9913,30521

UniSTS:

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/esummary.fcgi?db=unists&id=254085,254086

EFetch

Base URL

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi

Functions

- Returns formatted data records for a list of input UIDs
- Returns formatted data records for a set of UIDs stored on the Entrez History server

Required Parameters

db

Database from which to retrieve records. The value must be a valid Entrez database name (default = pubmed). Currently EFetch does not support all Entrez databases. Please see Table 1 in Chapter 2 for a list of available databases.

Required Parameter – Used only when input is from a UID list

id

UID list. Either a single UID or a comma-delimited list of UIDs may be provided. All of the UIDs must be from the database specified by **db**. There is no set maximum for the number of UIDs that can be passed to EFetch, but if more than about 200 UIDs are to be provided, the request should be made using the HTTP POST method.

For sequence databases (nuccore, popset, protein), the UID list may be a mixed list of GI numbers and accession.version identifiers.

```
efetch.fcgi?db=pubmed&id=19393038,30242208,29453458
efetch.fcgi?db=protein&id=15718680,NP_001098858.1,119703751
```

Special note for sequence databases.

NCBI is no longer assigning GI numbers to a growing number of new sequence records. As such, these records are not indexed in Entrez, and so cannot be retrieved using ESearch or ESummary, and have no Entrez links accessible by ELink. EFetch *can* retrieve these records by including their accession.version identifier in the **id** parameter.

Required Parameters – Used only when input is from the Entrez History server

query_key

Query key. This integer specifies which of the UID lists attached to the given Web Environment will be used as input to EFetch. Query keys are obtained from the output of previous ESearch, EPost or ELInk calls. The **query_key** parameter must be used in conjunction with **WebEnv**.

WebEnv

Web Environment. This parameter specifies the Web Environment that contains the UID list to be provided as input to EFetch. Usually this WebEnv value is obtained from the output of a previous ESearch, EPost or ELink call. The **WebEnv** parameter must be used in conjunction with **query_key**.

```
efetch.fcgi?db=protein&query_key=<key>&WebEnv=<webenv string>
```

Optional Parameters – Retrieval

retmode

Retrieval mode. This parameter specifies the data format of the records returned, such as plain text, HMTL or XML. See Table 1 for a full list of allowed values for each database.

Table 1 – Valid values of &retmode and &rettype for EFetch (null = empty string)

Record Type	&rettype	&retmode
All Databases		
Document summary	docsum	xml, *default*
List of UIDs in XML	uilist	xml
List of UIDs in plain text	uilist	text
db = bioproject		
Full record XML	xml, *default*	xml, *default*
db = biosample		
Full record XML	full, *default*	xml, *default*
Full record text	full, *default*	text
db = gds		
Summary	summary, *default*	text, *default*
db = gene		
text ASN.1	*null*	asn.1, *default*
XML	*null*	xml
Gene table	gene_table	text
db = homologene		

Table 1 continued from previous page.

text ASN.1	*null*	asn.1, *default*
XML	*null*	xml
Alignment scores	alignmentscores	text
FASTA	fasta	text
HomoloGene	homologene	text
db = mesh		
Full record	full, *default*	text, *default*
db = nlmcatalog		
Full record	*null*	text, *default*
XML	*null*	xml
db = nuccore, protein or popset		
text ASN.1	*null*	text, *default*
binary ASN.1	*null*	asn.1
Full record in XML	native	xml
Accession number(s)	acc	text
FASTA	fasta	text
TinySeq XML	fasta	xml
SeqID string	seqid	text
Additional options for db = nuccore or popset		
GenBank flat file	gb	text

Table 1 continued from previous page.

GBSeq XML	gb	xml
INSDSeq XML	gbc	xml
Additional option for db = nuccore and protein		
Feature table	ft	text
Additional option for db = nuccore		
GenBank flat file with full sequence (contigs)	gbwithparts	text
CDS nucleotide FASTA	fasta_cds_na	text
CDS protein FASTA	fasta_cds_aa	text
Additional options for db = protein		
GenPept flat file	gp	text
GBSeq XML	gp	xml
INSDSeq XML	gpc	xml
Identical Protein XML	ipg	xml
db = pmc		
XML	*null*	xml, *default*
MEDLINE	medline	text
db = pubmed		
XML	*null*	xml, *default*
MEDLINE	medline	text
PMID list	uilist	text

Table 1 continued from previous page.

Abstract	abstract	text
db = sequences		
text ASN.1	*null*	text, *default*
Accession number(s)	acc	text
FASTA	fasta	text
SeqID string	seqid	text
db = snp		
text ASN.1	*null*	asn.1, *default*
XML	*null*	xml
Flat file	flt	text
FASTA	fasta	text
RS Cluster report	rsr	text
SS Exemplar list	ssexemplar	text
Chromosome report	chr	text
Summary	docset	text
UID list	uilist	text or xml
db = sra		
XML	full, *default*	xml, *default*
db = taxonomy		
XML	*null*	xml, *default*

Table 1 continued from previous page.

TaxID list	uilist	text or xml
db = clinvar		
ClinVar Set	clinvarset	xml, *default*
UID list	uilist	text or xml
db = gtr		
GTR Test Report	gtracc	xml, *default*

rettype

Retrieval type. This parameter specifies the record view returned, such as Abstract or MEDLINE from PubMed, or GenPept or FASTA from protein. Please see Table 1 for a full list of allowed values for each database.

retstart

Sequential index of the first record to be retrieved (default=0, corresponding to the first record of the entire set). This parameter can be used in conjunction with **retmax** to download an arbitrary subset of records from the input set.

retmax

Total number of records from the input set to be retrieved, up to a maximum of 10,000. Optionally, for a large set the value of **retstart** can be iterated while holding **retmax** constant, thereby downloading the entire set in batches of size **retmax**.

Optional Parameters – Sequence Databases

strand

Strand of DNA to retrieve. Available values are "1" for the plus strand and "2" for the minus strand.

seq_start

First sequence base to retrieve. The value should be the integer coordinate of the first desired base, with "1" representing the first base of the seqence.

seq_stop

Last sequence base to retrieve. The value should be the integer coordinate of the last desired base, with "1" representing the first base of the seqence.

complexity

Data content to return. Many sequence records are part of a larger data structure or "blob", and the **complexity** parameter determines how much of that blob to return. For example, an mRNA may be stored together with its protein product. The available values are as follows:

Value of complexity	Data returned for each requested GI
0	entire blob
1	bioseq
2	minimal bioseq-set
3	minimal nuc-prot
4	minimal pub-set

Examples

PubMed

Fetch PMIDs 17284678 and 9997 as text abstracts:

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=pubmed&id=17284678,9997&retmode=text&rettype=abstract

Fetch PMIDs in XML:

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=pubmed&id=11748933,11700088&retmode=xml

PubMed Central

Fetch XML for PubMed Central ID 212403:

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=pmc&id=212403

Nucleotide/Nuccore

Fetch the first 100 bases of the plus strand of GI 21614549 in FASTA format:

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=nuccore&id=21614549&strand=1&seq_start=1&seq_stop=100&rettype=fasta&retmode=text

Fetch the first 100 bases of the minus strand of GI 21614549 in FASTA format:

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=nuccore&id=21614549&strand=2&seq_start=1&seq_stop=100&rettype=fasta&retmode=text

Fetch the nuc-prot object for GI 21614549:

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=nuccore&id=21614549&complexity=3

Fetch the full ASN.1 record for GI 5:

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=nucleotide&id=5

Fetch FASTA for GI 5:

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=nucleotide&id=5&rettype=fasta

Fetch the GenBank flat file for GI 5:

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=nucleotide&id=5&rettype=gb

Fetch GBSeqXML for GI 5:

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=nucleotide&id=5&rettype=gb&retmode=xml

Fetch TinySeqXML for GI 5:

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=nucleotide&id=5&rettype=fasta&retmode=xml

Popset

Fetch the GenPept flat file for Popset ID 12829836:

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=popset&id=12829836&rettype=gp

Protein

Fetch the GenPept flat file for GI 8:

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=protein&id=8&rettype=gp

Fetch GBSeqXML for GI 8:

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=protein&id=8&rettype=gp&retmode=xml

Sequences

Fetch FASTA for a transcript and its protein product (GIs 312836839 and 34577063)

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=sequences&id=312836839,34577063&rettype=fasta&retmode=text

Gene

Fetch full XML record for Gene ID 2:

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=gene&id=2&retmode=xml

ELink

Base URL

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi

Functions

- Returns UIDs linked to an input set of UIDs in either the same or a different Entrez database
- Returns UIDs linked to other UIDs in the same Entrez database that match an Entrez query
- Checks for the existence of Entrez links for a set of UIDs within the same database
- Lists the available links for a UID
- Lists LinkOut URLs and attributes for a set of UIDs
- Lists hyperlinks to primary LinkOut providers for a set of UIDs
- Creates hyperlinks to the primary LinkOut provider for a single UID

Required Parameters

db

Database from which to retrieve UIDs. The value must be a valid Entrez database name (default = pubmed). This is the destination database for the link operation.

dbfrom

Database containing the input UIDs. The value must be a valid Entrez database name (default = pubmed). This is the origin database of the link operation. If **db** and **dbfrom** are set to the same database value, then ELink will return computational neighbors within that database. Please see the full list of Entrez links for available computational neighbors. Computational neighbors have linknames that begin with *dbname_dbname* (examples: protein_protein, pcassay_pcassay_activityneighbor).

cmd

ELink command mode. The command mode specified which function ELink will perform. Some optional parameters only function for certain values of &cmd (see below).

cmd=neighbor (default)

ELink returns a set of UIDs in **db** linked to the input UIDs in **dbfrom.**

Example: Link from protein to gene

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?dbfrom=protein&db=gene&id=15718680,157427902

cmd=neighbor_score

ELink returns a set of UIDs within the same database as the input UIDs along with computed similarity scores**.**

Example: Find related articles to PMID 20210808

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?
dbfrom=pubmed&db=pubmed&id=20210808&cmd=neighbor_score

cmd=neighbor_history

ELink posts the output UIDs to the Entrez History server and returns a **query_key** and **WebEnv** corresponding to the location of the output set.

Example: Link from protein to gene and post the results on the Entrez History

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?
dbfrom=protein&db=gene&id=15718680,157427902&cmd=neighbor_history

cmd=acheck

ELink lists all links available for a set of UIDs.

Example: List all possible links from two protein GIs

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?
dbfrom=protein&id=15718680,157427902&cmd=acheck

Example: List all possible links from two protein GIs to PubMed

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?dbfrom=protein&db=pubmed&id=15718680,157427902&cmd=acheck

cmd=ncheck

ELink checks for the existence of links *within the same database* for a set of UIDs. These links are equivalent to setting **db** and **dbfrom** to the same value.

Example: Check whether two nuccore sequences have "related sequences" links.

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?dbfrom=nuccore&id=21614549,219152114&cmd=ncheck

cmd=lcheck

Elink checks for the existence of external links (LinkOuts) for a set of UIDs.

Example: Check whether two protein sequences have any LinkOut providers.

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?dbfrom=protein&id=15718680,157427902&cmd=lcheck

cmd=llinks

For each input UID, ELink lists the URLs and attributes for the LinkOut providers that are not libraries.

Example: List the LinkOut URLs for non-library providers for two pubmed abstracts.

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?dbfrom=pubmed&id=19880848,19822630&cmd=llinks

cmd=llinkslib

For each input UID, ELink lists the URLs and attributes for *all* LinkOut providers including libraries.

Example: List all LinkOut URLs for two PubMed abstracts.

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?dbfrom=pubmed&id=19880848,19822630&cmd=llinkslib

cmd=prlinks

ELink lists the primary LinkOut provider for each input UID, or links directly to the LinkOut provider's web site for a single UID if **retmode** is set to *ref*.

Example: Find links to full text providers for two PubMed abstracts.

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?dbfrom=pubmed&id=19880848,19822630&cmd=prlinks

Example: Link directly to the full text for a PubMed abstract at the provider's web site.

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?dbfrom=pubmed&id=19880848&cmd=prlinks&retmode=ref

Required Parameter – Used only when input is from a UID list

id

UID list. Either a single UID or a comma-delimited list of UIDs may be provided. All of the UIDs must be from the database specified by **dbfrom**. There is no set maximum for the number of UIDs that can be passed to ELink, but if more than about 200 UIDs are to be provided, the request should be made using the HTTP POST method.

Link from protein to gene.

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?dbfrom=protein&db=gene&id=15718680,157427902,119703751

Find related sequences (link from nuccore to nuccore).

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?dbfrom=nuccore&db=nuccore&id=34577062

If more than one **id** parameter is provided, ELink will perform a separate link operation for the set of UIDs specified by each **id** parameter. This effectively accomplishes "one-to-one" links and preserves the connection between the input and output UIDs.

Find one-to-one links from protein to gene.

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?dbfrom=protein&db=gene&id=15718680&id=157427902&id=119703751

For sequence databases (nuccore, popset, protein), the UID list may be a mixed list of GI numbers and accession.version identifiers.

Find one-to-one links from protein to gene.

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?dbfrom=protein&db=gene&id=15718680&id=NP_001098858.1&id=119703751

Required Parameters – Used only when input is from the Entrez History server

query_key

Query key. This integer specifies which of the UID lists attached to the given Web Environment will be used as input to ELink. Query keys are obtained from the output of previous ESearch, EPost or ELInk calls. The **query_key** parameter must be used in conjunction with **WebEnv**.

WebEnv

Web Environment. This parameter specifies the Web Environment that contains the UID list to be provided as input to ELink. Usually this WebEnv value is obtained from the output of a previous ESearch, EPost or ELink call. The **WebEnv** parameter must be used in conjunction with **query_key**.

```
Link from protein to gene:
elink.fcgi?dbfrom=protein&db=gene&query_key=<key>&WebEnv=<webenv string>

Find related sequences (link from protein to protein):
elink.fcgi?dbfrom=protein&db=protein&query_key=<key>&WebEnv=
<webenv string>
```

Optional Parameter – Retrieval

retmode

Retrieval type. Determines the format of the returned output. The default value is 'xml' for ELink XML, but 'json' is also supported to return output in JSON format.

idtype

Specifies the type of identifier to return for sequence databases (nuccore, popset, protein). By default, ELink returns GI numbers in its output. If **idtype** is set to 'acc', ELink will return accession.version identifiers rather than GI numbers.

Optional Parameters – Limiting the Output Set of Links

linkname

Name of the Entrez link to retrieve. Every link in Entrez is given a name of the form

dbfrom_db_subset.

The values of *subset* vary depending on the values of *dbfrom* and *db*. Many *dbfrom/db* combinations have no *subset* values. See the list of Entrez links for a listing of all available linknames. When **linkname** is used, only the links with that name will be retrieved.

The **linkname** parameter only functions when **cmd** is set to *neighbor* or *neighbor_history*.

Find all links from gene to snp.

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?dbfrom=gene&db=snp&id=93986

Find snps with genotype data linked to genes.

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?dbfrom=gene&db=snp&id=93986&linkname=gene_snp_genegenotype

term

Entrez query used to limit the output set of linked UIDs. The query in the **term** parameter will be applied after the link operation, and only those UIDs matching the query will be returned by ELink. The **term** parameter only functions when **db** and **dbfrom** are set to the same database value.

Find all related articles for a PMID.

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?dbfrom=pubmed&db=pubmed&id=19879512

Find all related review articles published in 2008 for a PMID.

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?dbfrom=pubmed&db=pubmed&id=19879512&term=review%5Bfilter%5D+AND+2008%5Bpdat%5Dh

holding

Name of LinkOut provider. Only URLs for the LinkOut provider specified by **holding** will be returned. The value provided to **holding** should be the abbreviation of the LinkOut provider's name found in the <NameAbbr> tag of the ELink XML output when **cmd** is set to *llinks* or *llinkslib*. The **holding** parameter only functions when **cmd** is set to *llinks* or *llinkslib*.

Find information for all LinkOut providers for a PMID.

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?dbfrom=pubmed&cmd=llinkslib&id=16210666

Find information from clinicaltrials.gov for a PMID.

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?dbfrom=pubmed&cmd=llinkslib&id=16210666&holding=CTgov

Optional Parameters – Dates

These parameters only function when **cmd** is set to *neighbor* or *neighbor_history* and **dbfrom** is *pubmed*.

datetype

Type of date used to limit a link operation. The allowed values vary between Entrez databases, but common values are 'mdat' (modification date), 'pdat' (publication date) and 'edat' (Entrez date). Generally an Entrez database will have only two allowed values for **datetype**.

reldate

When **reldate** is set to an integer *n*, ELink returns only those items that have a date specified by **datetype** within the last *n* days.

mindate, maxdate

Date range used to limit a link operation by the date specified by **datetype**. These two parameters (**mindate, maxdate**) must be used together to specify an arbitrary date range. The general date format is YYYY/MM/DD, and these variants are also allowed: YYYY, YYYY/MM.

EGQuery

Base URL

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/egquery.fcgi

Function

Provides the number of records retrieved in all Entrez databases by a single text query.

Required Parameter

term

Entrez text query. All special characters must be URL encoded. Spaces may be replaced by '+' signs. For very long queries (more than several hundred characters long), consider using an HTTP POST call. See the PubMed or Entrez help for information about search field descriptions and tags. Search fields and tags are database specific.

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/egquery.fcgi?term=asthma

ESpell

Base URL

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/espell.fcgi

Function

Provides spelling suggestions for terms within a single text query in a given database.

Required Parameters

db

Database to search. Value must be a valid Entrez database name (default = pubmed).

term

Entrez text query. All special characters must be URL encoded. Spaces may be replaced by '+' signs. For very long queries (more than several hundred characters long), consider using an HTTP POST call. See the PubMed or Entrez help for information about search field descriptions and tags. Search fields and tags are database specific.

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/espell.fcgi?db=pubmed&term=asthmaa+OR+alergies

ECitMatch

Base URL

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/ecitmatch.cgi

Function

Retrieves PubMed IDs (PMIDs) that correspond to a set of input citation strings.

Required Parameters

db

Database to search. The only supported value is 'pubmed'.

rettype

Retrieval type. The only supported value is 'xml'.

bdata

Citation strings. Each input citation must be represented by a citation string in the following format:

journal_title|year|volume|first_page|author_name|your_key|

Multiple citation strings may be provided by separating the strings with a carriage return character (%0D). The *your_key* value is an arbitrary label provided by the user that may serve as a local identifier for the citation, and it will be included in the output. Be aware that all spaces must be replaced by '+' symbols and that citation strings should end with a final vertical bar '|'.

https://eutils.ncbi.nlm.nih.gov/entrez/eutils/ecitmatch.cgi?db=pubmed&retmode=xml&bdata=proc+natl+acad+sci+u+s+a|1991|88|3248|mann+bj|Art1|%0Dscience|1987|235|182|palmenberg+ac|Art2|

Release Notes

EFetch; ELink JSON ouput: June 24, 2015

- EFetch now supports ClinVar and GTR
- ELink now provides output in JSON format

ESearch &sort; JSON output format: February 14, 2014

- ESearch now provides a supported **sort** parameter
- EInfo, ESearch and ESummary now provide output data in JSON format

ECitMatch, EInfo Version 2.0, EFetch: August 9, 2013

- ECitMatch is a new E-utility that serves as an API to the PubMed batch citation matcher
- EInfo has an updated XML output that includes two new fields: <IsTruncatable> and <IsRangeable>
- EFetch now supports the BioProject database.

EFetch Version 2.0. Target release date: February 15, 2012

- EFetch now supports the following databases: biosample, biosystems and sra
- EFetch now has defined default values for &retmode and &rettype for all supported databases (please see Table 1 for all supported values of these parameters)
- EFetch no longer supports &retmode=html; requests containing &retmode=html will return data using the default &retmode value for the specified database (&db)
- EFetch requests including &rettype=docsum return XML data equivalent to ESummary output

Release of new Genome database: November 9, 2011

- Entrez Genome has been completely redesigned, and database records now correspond to a species rather than an individual chromosome sequence. Please see full details of the change at https://www.ncbi.nlm.nih.gov/About/news/17Nov2011.html
- Old Genome IDs are no longer valid. A file is available on the NCBI FTP site that maps old Genome IDs to Nucleotide GIs: ftp.ncbi.nih.gov/genomes/old_genomeID2nucGI
- EFetch no longer supports retrievals from Genome (db=genome).
- The ESummary XML for Genome has been recast to reflect the new data model.
- To view the new search fields and links supported for the new Genome database, please see https://eutils.ncbi.nlm.nih.gov/entrez/eutils/einfo.fcgi?db=genome

ESummary Version 2.0. November 4, 2011

- ESummary now supports a new, alternative XML presentation for Entrez document summaries (DocSums). The new XML is unique to each Entrez database and generally contains more extensive data about the record than the original DocSum XML.
- There are no plans at present to discontinue the original DocSum XML, so developers can continue to use this presentation, which will remain the default.
- Version 2.0 XML is returned when &version=2.0 is included in the ESummary URL.

Demonstration Programs

Please see Chapter 1 for sample Perl scripts.

For More Information

Please see Chapter 1 for getting additional information about the E-utilities.

The E-utility Web Service (SOAP)

Eric Sayers, PhD[✉1] and Vadim Miller[2]

Created: January 21, 2010; Updated: January 23, 2015.

Termination Announcement

The SOAP web service for the E-utilities will be **TERMINATED** permanently on July 1, 2015. All requests made to this service after that date will fail.

If you have software that is currently using the E-utility SOAP web service, please plan to transition to using the standard URL interface described in Chapters 1-4 of this book.

Please contact info@ncbi.nlm.nih.gov if you have questions about this change.

For More Information

E-utility DTDs

Please see Chapter 1 for getting additional information about the E-utilities.

Author Affiliations: 1 NCBI; Email: sayers@ncbi.nlm.nih.gov. 2 NCBI; Email: miller@ncbi.nlm.nih.gov.

✉ Corresponding author.

Entrez Direct: E-utilities on the Unix Command Line

Jonathan Kans, PhD[1]

Created: April 23, 2013; Updated: November 23, 2022.

Getting Started

Introduction

Entrez Direct (EDirect) provides access to the NCBI's suite of interconnected databases (publication, sequence, structure, gene, variation, expression, etc.) 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.

Installation

EDirect will run on Unix and Macintosh computers, and under the Cygwin Unix-emulation environment on Windows PCs. To install the EDirect software, open a terminal window and execute one of the following two commands:

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

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

This will download a number of scripts and several precompiled programs into an "edirect" folder in the user's home directory. It may then print an additional command for updating the PATH environment variable in the user's configuration file. The editing instructions will look something like:

```
echo "export PATH=\$PATH:\$HOME/edirect" >> $HOME/.bash_profile
```

As a convenience, the installation process ends by offering to run the PATH update command for you. Answer "y" and press the Return key if you want it run. If the PATH is already set correctly, or if you prefer to make any editing changes manually, just press Return.

One installation is complete, run:

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

to set the PATH for the current terminal session.

Quick Start

The **readme.pdf** file included in the edirect folder contains a highly-abridged version of this document. It is intended to convey the most important points in the least amount of time for the new user, while still presenting the minimal essential details. It also covers subtle issues in several Entrez biological databases, demonstrates integration of data from external sources, and has a brief introduction to scripting and programming.

The full documentation gives a much more in-depth exploration of the underlying topics, especially in the Complex Objects section, and in the Additional Examples web page, which is organized by Entrez database. This document also introduces other worthy topics, such as identifier lookup and sequence coordinate conversions, and has a more thorough treatment of automation.

Author Affiliation: 1 NCBI; Email: kans@ncbi.nlm.nih.gov.

Corresponding author.

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 seamlessly between EDirect programs and Unix utilities or scripts to perform actions that cannot be accomplished entirely within Entrez.

All EDirect programs are designed to work on large sets of data. They handle many technical details behind the scenes (avoiding the learning curve normally required for E-utilities programming), and save intermediate results on the Entrez history server. For best performance, 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
```

Unix programs are run by typing the name of the program and then supplying any required or optional arguments on the command line. Argument names are letters or words that start with a dash ("**–**") character.

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

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:

```
elink -related
```

```
elink -target gene
```

Elink also connects to the NIH Open Citation Collection dataset to find publications that cite the selected PubMed articles, or to follow the reference lists of PubMed records:

```
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
```

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

Constructing Multi-Step Queries

EDirect allows individual operations to be described separately, combining them into a multi-step query by using the vertical bar ("|") Unix pipe symbol:

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

Writing Commands on Multiple Lines

A query can be continued on the next line by typing the backslash ("\") Unix escape character immediately before pressing the Return key.

```
esearch -db pubmed -query "opsin gene conversion" | \
```

Continuing the query looks up precomputed neighbors of the original papers, next links to all protein sequences published in the related articles, then limits those to the rodent division of GenBank, and finally retrieves the records in FASTA format:

```
elink -related | \
elink -target protein | \
efilter -division rod | \
efetch -format fasta
```

In most modern versions of Unix the vertical bar pipe symbol also allows the query to continue on the next line, without the need for an additional backslash.

Accessory Programs

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

```
nquire -url http://www.wikidata.org/entity Q22679758
```

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

```
transmute -j2x
```

Xtract can use waypoints to navigate a complex XML hierarchy and obtain data values by field name:

```
xtract -pattern entities -group P527/mainsnak -block datavalue -element id
```

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

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

Discovery by Navigation

PubMed related articles are calculated by a statistical text retrieval algorithm using the title, abstract, and medical subject headings (MeSH terms). The connections between papers can be used for making 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. An initial search on the enzyme finds 280 articles. Looking up precomputed neighbors returns 17,288 papers, some of which might be expected to discuss other enzymes in the pathway:

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

β-carotene is known to be 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.

Applying this knowledge, by linking the publication neighbors to their associated protein records and then filtering those candidates using the NCBI taxonomy, can help locate the desired enzyme.

Linking from pubmed to the protein database finds 657,677 protein sequences:

```
elink -target protein |
```

Limiting to mice excludes plants, fungi, and bacteria, which eliminates the earlier enzymes:

```
efilter -organism mouse -source refseq |
```

This matches only 43 sequences, which is small enough to examine by retrieving the individual records:

```
efetch -format fasta
```

As anticipated, 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
TNYIRKIDPQTLETLEKVDYRKYVAVNLATSHPHYDEAGNVLNMGTSVVDKGRTKYVIFKIPATVPDSKK
...
```

Retrieving PubMed Reports

Piping PubMed query results to efetch and specifying the "abstract" format:

```
esearch -db pubmed -query "lycopene cyclase" |
efetch -format abstract
```

returns a set of reports that can be read by a person:

```
...
85. PLoS One. 2013;8(3):e58144. doi: 10.1371/journal.pone.0058144. Epub ...

Levels of lycopene β-cyclase 1 modulate carotenoid gene expression and
accumulation in Daucus carota.

Moreno JC(1), Pizarro L, Fuentes P, Handford M, Cifuentes V, Stange C.

Author information:
(1)Departamento de Biología, Facultad de Ciencias, Universidad de Chile,
Santiago, Chile.

Plant carotenoids are synthesized and accumulated in plastids through a
highly regulated pathway. Lycopene β-cyclase (LCYB) is a key enzyme
involved directly in the synthesis of α-carotene and β-carotene through
...
```

If "medline" format is used instead:

```
esearch -db pubmed -query "lycopene cyclase" |
efetch -format medline
```

the output can be entered into common bibliographic management software packages:

```
...
PMID- 23555569
OWN - NLM
STAT- MEDLINE
DA  - 20130404
DCOM- 20130930
LR  - 20131121
IS  - 1932-6203 (Electronic)
IS  - 1932-6203 (Linking)
VI  - 8
IP  - 3
DP  - 2013
TI  - Levels of lycopene beta-cyclase 1 modulate carotenoid gene expression
      and accumulation in Daucus carota.
PG  - e58144
LID - 10.1371/journal.pone.0058144 [doi]
AB  - Plant carotenoids are synthesized and accumulated in plastids
      through a highly regulated pathway. Lycopene beta-cyclase (LCYB) is a
      key enzyme involved directly in the synthesis of alpha-carotene and
      ...
```

Retrieving Sequence Reports

Nucleotide and protein records can be downloaded in FASTA format:

```
esearch -db protein -query "lycopene cyclase" |
efetch -format fasta
```

which consists of a definition line followed by the sequence:

```
...
>gi|735882|gb|AAA81880.1| lycopene cyclase [Arabidopsis thaliana]
MDTLLKTPNKLDFFIPQFHGFERLCSNNPYPSRVRLGVKKRAIKIVSSVVSGSAALLDLVPETKKENLDF
ELPLYDTSKSQVVDLAIVGGGPAGLAVAQQVSEAGLSVCSIDPSPKLIWPNNYGVWVDEFEAMDLLDCLD
TTWSGAVVYVDEGVKKDLSRPYGRVNRKQLKSKMLQKCITNGVKFHQSKVTNVVHEEANSTVVCSDGVKI
QASVVLDATGFSRCLVQYDKPYNPGYQVAYGIIAEVDGHPFDVDKMVFMDWRDKHLDSYPELKERNSKIP
TFLYAMPFSSNRIFLEETSLVARPGLRMEDIQERMAARLKHLGINVKRIEEDERCVIPMGGPLPVLPQRV
VGIGGTAGMVHPSTGYMVARTLAAAPIVANAIVRYLGSPSSNSLRGDQLSAEVWRDLWPIERRRQREFFC
FGMDILLKLDLDATRRFFDAFFDLQPHYWHGFLSSRLFLPELLVFGLSLFSHASNTSRLEIMTKGTVPLA
KMINNLVQDRD
...
```

Sequence records can also be obtained as GenBank or GenPept flatfiles:

```
esearch -db protein -query "lycopene cyclase" |
efetch -format gp
```

which have features annotating particular regions of the sequence:

```
...
LOCUS       AAA81880                 501 aa            linear   PLN ...
DEFINITION  lycopene cyclase [Arabidopsis thaliana].
ACCESSION   AAA81880
VERSION     AAA81880.1  GI:735882
DBSOURCE    locus ATHLYC accession L40176.1
KEYWORDS    .
SOURCE      Arabidopsis thaliana (thale cress)
  ORGANISM  Arabidopsis thaliana
            Eukaryota; Viridiplantae; Streptophyta; Embryophyta;
            Tracheophyta; Spermatophyta; Magnoliophyta; eudicotyledons;
```

```
            Brassicales; Brassicaceae; Camelineae; Arabidopsis.
REFERENCE   1  (residues 1 to 501)
  AUTHORS   Scolnik,P.A. and Bartley,G.E.
  TITLE     Nucleotide sequence of lycopene cyclase (GenBank L40176) from
            Arabidopsis (PGR95-019)
  JOURNAL   Plant Physiol. 108 (3), 1343 (1995)
...
FEATURES             Location/Qualifiers
     source          1..501
                     /organism="Arabidopsis thaliana"
                     /db_xref="taxon:3702"
     Protein         1..501
                     /product="lycopene cyclase"
     transit_peptide 1..80
     mat_peptide     81..501
                     /product="lycopene cyclase"
     CDS             1..501
                     /gene="LYC"
                     /coded_by="L40176.1:2..1507"
ORIGIN
        1 mdtllktpnk ldffipqfhg ferlcsnnpy psrvrlgvkk raikivssvv sgsaalldlv
       61 petkkenldf elplydtsks qvvdlaivgg gpaglavaqq vseaglsvcs idpspkliwp
      121 nnygvwvdef eamdlldcld ttwsgavvyv degvkkdlsr pygrvnrkql kskmlqkcit
      181 ngvkfhqskv tnvvheeans tvvcsdgvki qasvvldatg fsrclvqydk pynpgyqvay
      241 giiaevdghp fdvdkmvfmd wrdkhldsyp elkernskip tflyampfss nrifleetsl
      301 varpglrmed iqermaarlk hlginvkrie edercvipmg gplpvlpqrv vgiggtagmv
      361 hpstgymvar tlaaapivan aivrylgsps snslrgdqls aevwrdlwpi errrqreffc
      421 fgmdillkld ldatrrffda ffdlqphywh gflssrlflp ellvfglslf shasntsrle
      481 imtkgtvpla kminnlvqdr d
//
...
```

Searching and Filtering

Restricting Query Results

The current results can be refined by further term searching in Entrez (useful in the protein database for limiting BLAST neighbors to a taxonomic subset):

```
esearch -db pubmed -query "opsin gene conversion" |
elink -related |
efilter -query "tetrachromacy"
```

Limiting by Date

Results can also be filtered by date. For example, the following statements:

```
efilter -days 60 -datetype PDAT

efilter -mindate 2000

efilter -maxdate 1985

efilter -mindate 1990 -maxdate 1999
```

restrict results to articles published in the previous two months, since the beginning of 2000, through the end of 1985, or in the 1990s, respectively. YYYY/MM and YYYY/MM/DD date formats are also accepted.

Fetch by Identifier

Efetch and elink can take a list of numeric identifiers or accessions in an **−id** argument:

```
efetch -db pubmed -id 7252148,1937004 -format xml

efetch -db nuccore -id 1121073309 -format acc

efetch -db protein -id 3OQZ_a -format fasta

efetch -db bioproject -id PRJNA257197 -format docsum

efetch -db pmc -id PMC209839 -format medline

elink -db pubmed -id 2539356 -cites
```

without the need for a preceding esearch command.

Non-integer accessions will be looked up with an internal search, using the appropriate field for the database:

```
esearch -db bioproject -query "PRJNA257197 [PRJA]" |
efetch -format uid | ...
```

Most databases use the [ACCN] field for identifier lookup, but there are a few exceptions:

```
annotinfo       [ASAC]
assembly        [ASAC]
bioproject      [PRJA]
books           [AID]
clinvar         [VACC]
gds             [ALL]
genome          [PRJA]
geoprofiles     [NAME]
gtr             [GTRACC]
mesh            [MHUI]
nuccore         [ACCN] or [PACC]
pcsubstance     [SRID]
snp             [RS] or [SS]
```

(For −db pmc it merely removes any "PMC" prefix from the integer identifier.)

For backward compatibility, **esummary** is a shortcut for esearch −format docsum:

```
esummary -db bioproject -id PRJNA257197

esummary -db sra -id SRR5437876
```

Reading Large Lists of Identifiers

Efetch and elink can also read a large list of identifiers or accessions piped in through stdin:

```
cat "file_of_identifiers.txt" |
efetch -db pubmed -format docsum
```

or from a file indicated by an **−input** argument:

```
efetch -input "file_of_identifiers.txt" -db pubmed -format docsum
```

As mentioned above, there is no need to use a script to split the identifiers into smaller groups or add a time delay between individual requests, since that functionality is already built into EDirect.

Indexed Fields

The **einfo** command 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
AUTH    Author
BRD     Breed
CULT    Cultivar
DIV     Division
ECNO    EC/RN Number
FILT    Filter
FKEY    Feature key
...
```

Qualifying Queries by Indexed Field

Query terms in esearch or efilter can be qualified by entering an indexed field abbreviation in brackets. Boolean operators and parentheses can also be used in the query expression for more complex searches.

Commonly-used fields for PubMed queries include:

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

and a qualified query looks like:

```
"Tager HS [AUTH] AND glucagon [TIAB]"
```

Filters that limit search results to subsets of PubMed include:

```
humans [MESH]
pharmacokinetics [MESH]
chemically induced [SUBH]
all child [FILT]
english [FILT]
freetext [FILT]
has abstract [FILT]
historical article [FILT]
randomized controlled trial [FILT]
clinical trial, phase ii [PTYP]
review [PTYP]
```

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

```
[ACCN]    Accession       [MLWT]    Molecular Weight
[ALL]     All Fields      [ORGN]    Organism
[AUTH]    Author          [PACC]    Primary Accession
```

```
[GPRJ]    BioProject      [PROP]    Properties
[BIOS]    BioSample       [PROT]    Protein Name
[ECNO]    EC/RN Number    [SQID]    SeqID String
[FKEY]    Feature key     [SLEN]    Sequence Length
[FILT]    Filter          [SUBS]    Substance Name
[GENE]    Gene Name       [WORD]    Text Word
[JOUR]    Journal         [TITL]    Title
[KYWD]    Keyword         [UID]     UID
```

and a sample query in the protein database is:

```
"alcohol dehydrogenase [PROT] NOT (bacteria [ORGN] OR fungi [ORGN])"
```

Additional examples of subset filters in sequence databases are:

```
mammalia [ORGN]
mammalia [ORGN:noexp]
txid40674 [ORGN]
cds [FKEY]
lacz [GENE]
beta galactosidase [PROT]
protein snp [FILT]
reviewed [FILT]
country united kingdom glasgow [TEXT]
biomol genomic [PROP]
dbxref flybase [PROP]
gbdiv phg [PROP]
phylogenetic study [PROP]
sequence from mitochondrion [PROP]
src cultivar [PROP]
srcdb refseq validated [PROP]
150:200 [SLEN]
```

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

See **efilter –help** for a list of filter shortcuts available for several Entrez databases.

Examining Intermediate Results

EDirect stores intermediate results on the Entrez history server. EDirect navigation functions produce a custom XML message with the relevant fields (database, web environment, query key, and record count) that can be read by 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 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 "tryptophan synthase alpha chain [PROT]" |
efilter -query "28000:30000 [MLWT]" |
elink -target structure |
efilter -query "0:2 [RESO]"
```

produces:

```
<ENTREZ_DIRECT>
  <Db>structure</Db>
  <WebEnv> MCID_5fac27e119f45d4eca20b0e6</WebEnv>
  <QueryKey>32</QueryKey>
  <Count>58</Count>
```

```
    <Step>4</Step>
  </ENTREZ_DIRECT>
```

with 58 protein structures being within the specified molecular weight range and having the desired (X-ray crystallographic) atomic position resolution.

(The QueryKey value differs from Step because the elink command splits its query into smaller chunks to avoid server truncation limits and timeout errors.)

Combining Independent Queries

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. (The steps to be combined must be in the same database.) Subsequent esearch commands 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 history thread.)

Because elink splits a large query into multiple smaller link requests, the new QueryKey value cannot be predicted in advance. The −label argument is used to get around this artifact. The label value is prefixed by a "#" symbol and placed in parentheses in the final search. For example, the query:

```
esearch -db protein -query "amyloid* [PROT]" |
elink -target pubmed -label prot_cit |
esearch -db gene -query "apo* [GENE]" |
elink -target pubmed -label gene_cit |
esearch -query "(#prot_cit) AND (#gene_cit)" |
efetch -format docsum |
xtract -pattern DocumentSummary -element Id Title
```

uses truncation searching (entering the beginning of a word followed by an asterisk) to return titles of papers with links to amyloid protein sequence and apolipoprotein gene records:

```
23962925    Genome analysis reveals insights into physiology and ...
23959870    Low levels of copper disrupt brain amyloid-β homeostasis ...
23371554    Genomic diversity and evolution of the head crest in the ...
23251661    Novel genetic loci identified for the pathophysiology of ...
...
```

Structured Data

Advantages of XML Format

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

The advantage of XML is that information is in specific locations in a well-defined data hierarchy. Accessing individual units of data that are fielded by name, such as:

```
<PubDate>2013</PubDate>
<Source>PLoS One</Source>
<Volume>8</Volume>
<Issue>3</Issue>
<Pages>e58144</Pages>
```

requires matching the same general pattern, differing only by the element name. This is much simpler than parsing the units from a long, complex string:

```
1. PLoS One. 2013;8(3):e58144 ...
```

The disadvantage of XML is that data extraction usually requires programming. But EDirect relies on the common pattern of XML value representation to provide a simplified approach to interpreting XML data.

Conversion of XML into Tables

The xtract program uses command-line arguments to direct the selective conversion of data in XML format. It allows 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. Explicit paths to objects are not needed.

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.

Format Customization

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 6271474,6092233,16589597 -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
6092233     1984 Jul-Aug    Calderon IL|Contopoulou CR|Mortimer RK
16589597    1954 Dec        Garber ED
```

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
```

Groups of fields are preceded by the **–pfx** value and followed by the **–sfx** value, both of which are initially empty.

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
```

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>"
```

and also ensures that data values containing encoded angle brackets, ampersands, quotation marks, or apostrophes remain properly encoded inside the new XML.

Element Variants

Derivatives of −element were created to eliminate the inconvenience of having to write post-processing scripts to perform otherwise trivial modifications or analyses on extracted data. They are subdivided into several categories. Substitute for −element as needed. A representative selection is shown below:

```
Positional:       -first, -last, -backward

Numeric:          -num, -len, -inc, -dec, -bin, -hex, -bit

Statistics:       -sum, -acc, -min, -max, -avg, -dev, -med

Character:        -encode, -upper, -title, -mirror, -alnum

String:           -basic, -plain, -simple, -author, -prose

Text:             -words, -pairs, -letters, -order, -reverse

Citation:         -year, -month, -page, -auth, -jour, -trim

Sequence:         -revcomp, -fasta, -ncbi2na, -molwt

Coordinate:       -0-based, -1-based, -ucsc-based

Variation:        -hgvs

Frequency:        -histogram

Expression:       -reg, -exp, -replace

Substitution:     -transform, -translate

Indexing:         -aliases, -classify

Miscellaneous:    -doi, -wct
```

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

See xtract **−help** for a brief description of each command.

Exploration Control

Exploration commands provide fine control over 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 allows related fields in an object to be kept together.

In contrast to 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:

```
efetch -db pubmed -id 1413997 -format xml |
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 another exploration layer between -pattern and -element , and redirects data exploration to present the authors one at a time:

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

```
efetch -db pubmed -id 1413997 -format xml |
xtract -pattern PubmedArticle -block Author \
  -sep " " -tab ", " -element Initials,LastName
```

produces a more traditional formatting of author names:

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

Sequential Exploration

Multiple -block statements can be used in a single xtract to explore different areas of the XML. This limits element extraction to the desired subregions, and allows disambiguation of fields with identical names. For example:

```
efetch -db pubmed -id 6092233,4640931,4296474 -format xml |
xtract -pattern PubmedArticle -element MedlineCitation/PMID \
  -block PubDate -sep " " -element Year,Month,MedlineDate \
  -block AuthorList -num Author -sep "/" -element LastName |
sort-table -k 3,3n -k 4,4f
```

generates a table that allows easy parsing of author last names, and sorts the results by author count:

```
4296474    1968 Apr        1    Friedmann
4640931    1972 Dec        2    Tager/Steiner
6092233    1984 Jul-Aug    3    Calderon/Contopoulou/Mortimer
```

Like -element arguments, the individual -block statements are executed sequentially, in order of appearance.

Note that "-element MedlineCitation/PMID" uses the **parent / child** construct to prevent the display of additional PMID items that might be present later in CommentsCorrections objects.

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

```
<PubDate>
  <Year>1968</Year>
  <Month>Apr</Month>
  <Day>25</Day>
</PubDate>
```

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

```
<PubDate>
  <MedlineDate>1984 Jul-Aug</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.

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.

(Hypothetical) census data would need several nested loops to visit each unique address in context:

```
-pattern State -group City -block Street -subset Number -element Resident
```

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 NG_008030.1 -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:

```
NG_008030.1
    source
              organism          Homo sapiens
              mol_type          genomic DNA
              db_xref           taxon:9606
    gene
              gene              COL5A1
    mRNA
              gene              COL5A1
              product           collagen type V alpha 1 chain, transcript variant 1
              transcript_id     NM_000093.4
    CDS
              gene              COL5A1
              product           collagen alpha-1(V) chain isoform 1 preproprotein
              protein_id        NP_000084.3
              translation       MDVHTRWKARSALRPGAPLLPPLLLLLLWAPPPSRAAQP...
              ...
```

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**). Variable values are retrieved by placing an ampersand before the variable name (e.g., "**&KEY**") in an -element statement:

```
efetch -db nuccore -id NG_008030.1 -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 version 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):

```
NG_008030.1
     source     organism          Homo sapiens
     source     mol_type          genomic DNA
     source     db_xref           taxon:9606
     gene       gene              COL5A1
     mRNA       gene              COL5A1
     mRNA       product           collagen type V alpha 1 chain, transcript variant 1
     mRNA       transcript_id     NM_000093.4
     CDS        gene              COL5A1
     CDS        product           collagen alpha-1(V) chain isoform 1 preproprotein
     CDS        protein_id        NP_000084.3
     CDS        translation       MDVHTRWKARSALRPGAPLLPPLLLLLLWAPPPSRAAQP...
     ...
```

Variables can be (re)initialized with an explicit literal value inside parentheses:

```
-block Author -sep " " -tab "" -element "&COM" Initials,LastName -COM "(, )"
```

They can also be used as the first argument in a conditional statement:

```
-CHR Chromosome -block GenomicInfoType -if "&CHR" -differs-from ChrLoc
```

Using a double-hyphen (e.g., **--STATS**) appends a value to the variable.

All variables are reset when the next record is processed.

Conditional Execution

Conditional processing commands (**-if**, **-unless**, **-and**, **-or**, and **-else**) 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. For example:

```
esearch -db pubmed -query "Havran W [AUTH]" |
efetch -format xml |
xtract -pattern PubmedArticle -if "#Author" -lt 14 \
  -block Author -if LastName -is-not Havran \
    -sep ", " -tab "\n" -element LastName,Initials[1:1] |
sort-uniq-count-rank
```

selects papers with fewer than 14 authors and prints a table of the most frequent collaborators, using a range to keep only the first initial so that variants like "Beadle, GW" and "Beadle, G" are combined:

```
34    Witherden, D
15    Boismenu, R
12    Jameson, J
```

```
10    Allison, J
10    Fitch, F
...
```

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

```
-if 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
```

Multiple conditions are specified with **-and** and **-or** commands:

```
-if @score -equals 1 -or @score -starts-with 0.9
```

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

```
-if MapLocation -element MapLocation -else -lbl "\-"
```

but setting a default value with -def may be more convenient in simple cases.

Parallel -if and -unless statements can be used to provide a more complex response to alternative conditions that include nested explorations.

Post-processing Functions

Elink **-cited** can perform a reverse citation lookup, thanks to a data service provided by the NIH Open Citation Collection. The extracted author names can be processed by piping to a chain of Unix utilities:

```
esearch -db pubmed -query "Beadle GW [AUTH]" |
elink -cited |
efetch -format docsum |
xtract -pattern Author -element Name |
sort -f | uniq -i -c
```

which produces an alphabetized count of authors who cited the original papers:

```
1 Abellan-Schneyder I
1 Abramowitz M
1 ABREU LA
1 ABREU RR
1 Abril JF
1 Abächerli E
1 Achetib N
1 Adams CM
2 ADELBERG EA
1 Adrian AB
...
```

Rather than always having to retype a series of common post-processing instructions, frequently-used combinations of Unix commands can be placed in a function, stored in an alias file (e.g., the user's .bash_profile), and executed by name. For example:

```
SortUniqCountRank() {
  grep '.' |
  sort -f |
  uniq -i -c |
  awk '{ n=$1; sub(/[ \t]*[0-9]+[ \t]/, ""); print n "\t" $0 }' |
  sort -t "$(printf '\t')" -k 1,1nr -k 2f
}
alias sort-uniq-count-rank='SortUniqCountRank'
```

(An enhanced version of **sort-uniq-count-rank** that accepts customization arguments is now included with EDirect as a stand-alone script.)

The raw author names can be passed directly to the sort-uniq-count-rank script:

```
esearch -db pubmed -query "Beadle GW [AUTH]" |
elink -cited |
efetch -format docsum |
xtract -pattern Author -element Name |
sort-uniq-count-rank
```

to produce a tab-delimited ranked list of authors who most often cited the original papers:

```
17    Hawley RS
13    Beadle GW
13    PERKINS DD
11    Glass NL
11    Vécsei L
10    Toldi J
9     TATUM EL
8     Ephrussi B
8     LEDERBERG J
...
```

Similarly, elink **–cites** uses NIH OCC data to return an article's reference list.

Other scripts for tab-delimited files include **sort-table**, **reorder-columns**, and **align-columns**. Unix parameter expansion requires **filter-columns** and **print-columns** arguments to be in single quotes.

Note that EDirect commands can also be used inside Unix functions or scripts.

Viewing an XML Hierarchy

Piping a PubmedArticle XML object to xtract **–outline** will give an indented overview of the XML hierarchy:

```
PubmedArticle
  MedlineCitation
    PMID
    DateCompleted
      Year
      Month
      Day
    ...
    Article
      Journal
        ...
        Title
        ISOAbbreviation
      ArticleTitle
      ...
      Abstract
```

```
      AbstractText
    AuthorList
      Author
        LastName
        ForeName
        Initials
        AffiliationInfo
          Affiliation
      Author
        ...
```

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.

Code Nesting Comparison

Sketching with indented pseudo code can clarify relative nesting levels. The extraction command:

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

where the rank of the argument name controls the nesting depth, could be represented as a computer program in pseudo code by:

```
for pat = each PubmedArticle {
  for blk = each pat.Author {
    print blk.Initials blk.LastName
  }
  for blk = each pat.MeSHTerm {
    if blk.Qual is present {
      print blk.MeshName
      for sbs = each blk.Qual {
        print sbs.QualName
      }
    }
  }
}
```

where the brace indentation count controls the nesting depth.

Extra arguments are held in reserve to provide additional levels of organization, should the need arise in the future for processing complex, deeply-nested XML data. The exploration commands below –pattern, in order of rank, are:

```
-path
  -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.

Complex Objects

Author Exploration

What's in a name? That which we call an author by any other name may be a consortium, investigator, or editor:

```
<PubmedArticle>
  <MedlineCitation>
    <PMID>99999999</PMID>
    <Article>
      <AuthorList>
        <Author>
          <LastName>Tinker</LastName>
        </Author>
        <Author>
          <LastName>Evers</LastName>
        </Author>
        <Author>
          <LastName>Chance</LastName>
        </Author>
        <Author>
          <CollectiveName>FlyBase Consortium</CollectiveName>
        </Author>
      </AuthorList>
    </Article>
    <InvestigatorList>
      <Investigator>
        <LastName>Alpher</LastName>
      </Investigator>
      <Investigator>
        <LastName>Bethe</LastName>
      </Investigator>
      <Investigator>
        <LastName>Gamow</LastName>
      </Investigator>
    </InvestigatorList>
  </MedlineCitation>
</PubmedArticle>
```

Within the record, –element exploration on last name:

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

prints each last name, but does not match the consortium:

```
Tinker    Evers    Chance    Alpher    Bethe    Gamow
```

Limiting to the author list:

```
xtract -pattern PubmedArticle -block AuthorList -element LastName
```

excludes the investigators:

```
Tinker    Evers    Chance
```

Using –num on each type of object:

```
xtract -pattern PubmedArticle -num Author Investigator LastName CollectiveName
```

displays the various object counts:

```
4    3    6    1
```

Date Selection

Dates come in all shapes and sizes:

```
<PubmedArticle>
  <MedlineCitation>
    <PMID>99999999</PMID>
    <DateCompleted>
      <Year>2011</Year>
    </DateCompleted>
    <DateRevised>
      <Year>2012</Year>
    </DateRevised>
    <Article>
      <Journal>
        <JournalIssue>
          <PubDate>
            <Year>2013</Year>
          </PubDate>
        </JournalIssue>
      </Journal>
      <ArticleDate>
        <Year>2014</Year>
      </ArticleDate>
    </Article>
  </MedlineCitation>
  <PubmedData>
    <History>
      <PubMedPubDate PubStatus="received">
        <Year>2015</Year>
      </PubMedPubDate>
      <PubMedPubDate PubStatus="accepted">
        <Year>2016</Year>
      </PubMedPubDate>
      <PubMedPubDate PubStatus="entrez">
        <Year>2017</Year>
      </PubMedPubDate>
      <PubMedPubDate PubStatus="pubmed">
        <Year>2018</Year>
      </PubMedPubDate>
      <PubMedPubDate PubStatus="medline">
        <Year>2019</Year>
      </PubMedPubDate>
    </History>
  </PubmedData>
</PubmedArticle>
```

Within the record, −element exploration on the year:

```
xtract -pattern PubmedArticle -element Year
```

finds and prints all nine instances:

```
2011    2012    2013    2014    2015    2016    2017    2018    2019
```

Using −block to limit the scope:

```
xtract -pattern PubmedArticle -block History -element Year
```

prints only the five years within the History object:

```
2015    2016    2017    2018    2019
```

Inserting a conditional statement to limit element selection to a date with a specific attribute:

```
xtract -pattern PubmedArticle -block History \
  -if @PubStatus -equals "pubmed" -element Year
```

surprisingly still prints all five years within History:

```
2015    2016    2017    2018    2019
```

This is because the -if command uses the same exploration logic as -element, but is designed to declare success if it finds a match anywhere within the current scope. There is indeed a "pubmed" attribute within History, in one of the five PubMedPubDate child objects, so the test succeeds. Thus, -element is given free rein to do its own exploration in History, and prints all five years.

The solution is to explore the individual PubMedPubDate objects:

```
xtract -pattern PubmedArticle -block PubMedPubDate \
  -if @PubStatus -equals "pubmed" -element Year
```

This visits each PubMedPubDate separately, with the -if test matching only the indicated date type, thus returning only the desired year:

```
2018
```

PMID Extraction

Because of the presence of a CommentsCorrections object:

```
<PubmedArticle>
  <MedlineCitation>
    <PMID>99999999</PMID>
    <CommentsCorrectionsList>
      <CommentsCorrections RefType="ErratumFor">
        <PMID>88888888</PMID>
      </CommentsCorrections>
    </CommentsCorrectionsList>
  </MedlineCitation>
</PubmedArticle>
```

attempting to print the record's PubMed Identifier:

```
xtract -pattern PubmedArticle -element PMID
```

also returns the PMID of the comment:

```
99999999    88888888
```

Using an exploration command cannot exclude the second instance, because it would need a parent node unique to the first element, and the chain of parents to the first PMID:

```
PubmedArticle/MedlineCitation
```

is a subset of the chain of parents to the second PMID:

```
PubmedArticle/MedlineCitation/CommentsCorrectionList/CommentsCorrections
```

Although **-first** PMID will work in this particular case, the more general solution is to limit by subpath with the parent / child construct:

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

That would work even if the order of objects were reversed.

Heterogeneous Data

XML objects can contain a heterogeneous mix of components. For example:

```
efetch -db pubmed -id 21433338,17247418 -format xml
```

returns a mixture of book and journal records:

```
<PubmedArticleSet>
  <PubmedBookArticle>
    <BookDocument>
    ...
    </PubmedBookData>
  </PubmedBookArticle>
  <PubmedArticle>
    <MedlineCitation>
    ...
    </PubmedData>
  </PubmedArticle>
</PubmedArticleSet>
```

The **parent / star** construct is used to visit the individual components, even though they may have different names. Piping the output to:

```
xtract -pattern "PubmedArticleSet/*" -element "*"
```

separately prints the entirety of each XML component:

```
<PubmedBookArticle><BookDocument> ... </PubmedBookData></PubmedBookArticle>
<PubmedArticle><MedlineCitation> ... </PubmedData></PubmedArticle>
```

Use of the parent / child construct can isolate objects of the same name that differ by their location in the XML hierarchy. For example:

```
efetch -db pubmed -id 21433338,17247418 -format xml |
xtract -pattern "PubmedArticleSet/*" \
  -group "BookDocument/AuthorList" -tab "\n" -element LastName \
  -group "Book/AuthorList" -tab "\n" -element LastName \
  -group "Article/AuthorList" -tab "\n" -element LastName
```

writes separate lines for book/chapter authors, book editors, and article authors:

```
Fauci         Desrosiers
Coffin        Hughes         Varmus
Lederberg     Cavalli        Lederberg
```

Simply exploring with individual arguments:

```
-group BookDocument -block AuthorList -element LastName
```

would visit the editors (at BookDocument/Book/AuthorList) as well as the authors (at BookDocument/AuthorList), and print names in order of appearance in the XML:

```
Coffin    Hughes    Varmus    Fauci    Desrosiers
```

(In this particular example the book author lists could be distinguished by using −if "@Type" −equals authors or −if "@Type" −equals editors, but exploring by parent / child is a general position-based approach.)

Recursive Definitions

Certain XML objects returned by efetch are recursively defined, including Taxon in −db taxonomy and Gene-commentary in −db gene. Thus, they can contain nested objects with the same XML tag.

Retrieving a set of taxonomy records:

```
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>
```

Xtract tracks XML object nesting to determine that the <Taxon> start tag on line 1 is closed by the </Taxon> stop tag on line 6, and not by the first </Taxon> encountered on line 3.

When a recursive object (e.g., Taxon) is given to an exploration command:

```
efetch -db taxonomy -id 9606,7227,10090 -format xml |
xtract -pattern Taxon \
  -element TaxId ScientificName GenbankCommonName Division
```

subsequent −element commands are blocked from descending into the internal objects, and return information only for the main entries:

```
9606     Homo sapiens               human          Primates
7227     Drosophila melanogaster    fruit fly      Invertebrates
10090    Mus musculus               house mouse    Rodents
```

The **star / child** construct will skip past the outer start tag:

```
efetch -db taxonomy -id 9606,7227,10090 -format xml |
xtract -pattern Taxon -block "*/Taxon" \
  -tab "\n" -element TaxId,ScientificName
```

to visit the next level of nested objects individually:

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

Recursive objects can be fully explored with a **double star / child** construct:

```
esearch -db gene -query "DMD [GENE] AND human [ORGN]" |
efetch -format xml |
xtract -pattern Entrezgene -block "**/Gene-commentary" \
  -tab "\n" -element Gene-commentary_type@value,Gene-commentary_accession
```

which visits every child object regardless of nesting depth:

```
genomic    NC_000023
mRNA       XM_006724469
peptide    XP_006724532
mRNA       XM_011545467
peptide    XP_011543769
...
```

Additional Elink Options

Elink has several additional modes that 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, the "neighbor" command:

```
esearch -db pubmed -query "Hoffmann PC [AUTH] AND dopamine [MAJR]" |
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:

```
1504781    11754494    3815119    1684029    14614914    12128255    ...
1684029    3815119     1504781    8097798    17161385    14755628    ...
2572612    2903614     6152036    2905789    9483560     1352865     ...
...
```

The "acheck" command returns all available link names for each record:

```
esearch -db pubmed -query "Federhen S [AUTH]" |
elink -cmd acheck |
xtract -pattern LinkSet -tab "\n" -element IdLinkSet/Id \
  -block LinkInfo -tab "\n" -element LinkName
```

printing each on its own line:

```
25510495
pubmed_images
pubmed_pmc
pubmed_pmc_local
pubmed_pmc_refs
pubmed_pubmed
pubmed_pubmed_citedin
...
```

The "prlinks" command can obtain the URL reference to the publisher web page for an article. The Unix "xargs" command calls elink separately for each identifier:

```
epost -db pubmed -id 22966225,19880848 |
efetch -format uid |
xargs -n 1 elink -db pubmed -cmd prlinks -id |
xtract -pattern LinkSet -first Id -element ObjUrl/Url
```

Repackaging XML Results

Splitting abstract paragraphs into individual words, while using XML reformatting commands:

```
efetch -db pubmed -id 2539356 -format xml |
xtract -stops -rec Rec -pattern PubmedArticle \
  -enc Paragraph -wrp Word -words AbstractText
```

generates:

```
...
<Paragraph>
  <Word>the</Word>
  <Word>tn3</Word>
  <Word>transposon</Word>
  <Word>inserts</Word>
  ...
  <Word>was</Word>
  <Word>necessary</Word>
  <Word>for</Word>
  <Word>immunity</Word>
</Paragraph>
...
```

with the words from each abstract instance encased in a separate parent object. Word counts for each paragraph could then be calculated by piping to:

```
xtract -pattern Rec -block Paragraph -num Word
```

Multi-Step Transformations

Although xtract provides −element variants to do simple data manipulation, more complex tasks are sometimes best handled by being broken up into a series of simpler transformations.

Document summaries for two bacterial chromosomes:

```
efetch -db nuccore -id U00096,CP002956 -format docsum |
```

contain several individual fields and a complex series of self-closing Stat objects:

```
<DocumentSummary>
  <Id>545778205</Id>
  <Caption>U00096</Caption>
  <Title>Escherichia coli str. K-12 substr. MG1655, complete genome</Title>
  <CreateDate>1998/10/13</CreateDate>
  <UpdateDate>2020/09/23</UpdateDate>
  <TaxId>511145</TaxId>
  <Slen>4641652</Slen>
  <Biomol>genomic</Biomol>
  <MolType>dna</MolType>
  <Topology>circular</Topology>
  <Genome>chromosome</Genome>
  <Completeness>complete</Completeness>
  <GeneticCode>11</GeneticCode>
  <Organism>Escherichia coli str. K-12 substr. MG1655</Organism>
```

```
    <Strain>K-12</Strain>
    <BioSample>SAMN02604091</BioSample>
    <Statistics>
      <Stat type="Length" count="4641652"/>
      <Stat type="all" count="9198"/>
      <Stat type="cdregion" count="4302"/>
      <Stat type="cdregion" subtype="CDS" count="4285"/>
      <Stat type="cdregion" subtype="CDS/pseudo" count="17"/>
      <Stat type="gene" count="4609"/>
      <Stat type="gene" subtype="Gene" count="4464"/>
      <Stat type="gene" subtype="Gene/pseudo" count="145"/>
      <Stat type="rna" count="187"/>
      <Stat type="rna" subtype="ncRNA" count="79"/>
      <Stat type="rna" subtype="rRNA" count="22"/>
      <Stat type="rna" subtype="tRNA" count="86"/>
      <Stat source="all" type="Length" count="4641652"/>
      <Stat source="all" type="all" count="13500"/>
      <Stat source="all" type="cdregion" count="4302"/>
      <Stat source="all" type="gene" count="4609"/>
      <Stat source="all" type="prot" count="4302"/>
      <Stat source="all" type="rna" count="187"/>
    </Statistics>
    <AccessionVersion>U00096.3</AccessionVersion>
  </DocumentSummary>
  <DocumentSummary>
    <Id>342852136</Id>
    <Caption>CP002956</Caption>
    <Title>Yersinia pestis A1122, complete genome</Title>
    ...
```

which make extracting the single "best" value for gene count a non-trivial exercise.

In addition to repackaging commands that surround extracted values with XML tags, the −element "*" construct prints the entirety of the current scope, including its XML wrapper. Piping the document summaries to:

```
xtract -set Set -rec Rec -pattern DocumentSummary \
  -block DocumentSummary -pkg Common \
    -wrp Accession -element AccessionVersion \
    -wrp Organism -element Organism \
    -wrp Length -element Slen \
    -wrp Title -element Title \
    -wrp Date -element CreateDate \
    -wrp Biomol -element Biomol \
    -wrp MolType -element MolType \
  -block Stat -if @type -equals gene -pkg Gene -element "*" \
  -block Stat -if @type -equals rna -pkg RNA -element "*" \
  -block Stat -if @type -equals cdregion -pkg CDS -element "*" |
```

encloses several fields in a Common block, and packages statistics on gene, RNA, and coding region features into separate sections of a new XML object:

```
...
<Rec>
  <Common>
    <Accession>U00096.3</Accession>
    <Organism>Escherichia coli str. K-12 substr. MG1655</Organism>
    <Length>4641652</Length>
    <Title>Escherichia coli str. K-12 substr. MG1655, complete genome</Title>
    <Date>1998/10/13</Date>
```

```
    <Biomol>genomic</Biomol>
    <MolType>dna</MolType>
  </Common>
  <Gene>
    <Stat type="gene" count="4609"/>
    <Stat type="gene" subtype="Gene" count="4464"/>
    <Stat type="gene" subtype="Gene/pseudo" count="145"/>
    <Stat source="all" type="gene" count="4609"/>
  </Gene>
  <RNA>
    <Stat type="rna" count="187"/>
    <Stat type="rna" subtype="ncRNA" count="79"/>
    <Stat type="rna" subtype="rRNA" count="22"/>
    <Stat type="rna" subtype="tRNA" count="86"/>
    <Stat source="all" type="rna" count="187"/>
  </RNA>
  <CDS>
    <Stat type="cdregion" count="4302"/>
    <Stat type="cdregion" subtype="CDS" count="4285"/>
    <Stat type="cdregion" subtype="CDS/pseudo" count="17"/>
    <Stat source="all" type="cdregion" count="4302"/>
  </CDS>
</Rec>
...
```

With statistics from different types of feature now segregated in their own substructures, total counts for each can be extracted with the −first command:

```
xtract -set Set -rec Rec -pattern Rec \
  -block Common -element "*" \
  -block Gene -wrp GeneCount -first Stat@count \
  -block RNA -wrp RnaCount -first Stat@count \
  -block CDS -wrp CDSCount -first Stat@count |
```

This rewraps the data into a third XML form containing specific feature counts:

```
...
<Rec>
  <Common>
    <Accession>U00096.3</Accession>
    <Organism>Escherichia coli str. K-12 substr. MG1655</Organism>
    <Length>4641652</Length>
    <Title>Escherichia coli str. K-12 substr. MG1655, complete genome</Title>
    <Date>1998/10/13</Date>
    <Biomol>genomic</Biomol>
    <MolType>dna</MolType>
  </Common>
  <GeneCount>4609</GeneCount>
  <RnaCount>187</RnaCount>
  <CDSCount>4302</CDSCount>
</Rec>
...
```

without requiring extraction commands for the individual elements in the Common block to be repeated at each step.

Assuming the contents are satisfactory, passing the last structured form to:

```
xtract \
  -head accession organism length gene_count rna_count \
```

```
  -pattern Rec -def "-" \
    -element Accession Organism Length GeneCount RnaCount
```

produces a tab-delimited table with the desired values:

```
accession     organism                 length     gene_count     rna_count
U00096.3      Escherichia coli ...     4641652    4609           187
CP002956.1    Yersinia pestis A1122    4553770    4217           86
```

If a different order of fields is desired after the final xtract has been run, piping to:

```
reorder-columns 1 3 5 4
```

will rearrange the output, including the column headings:

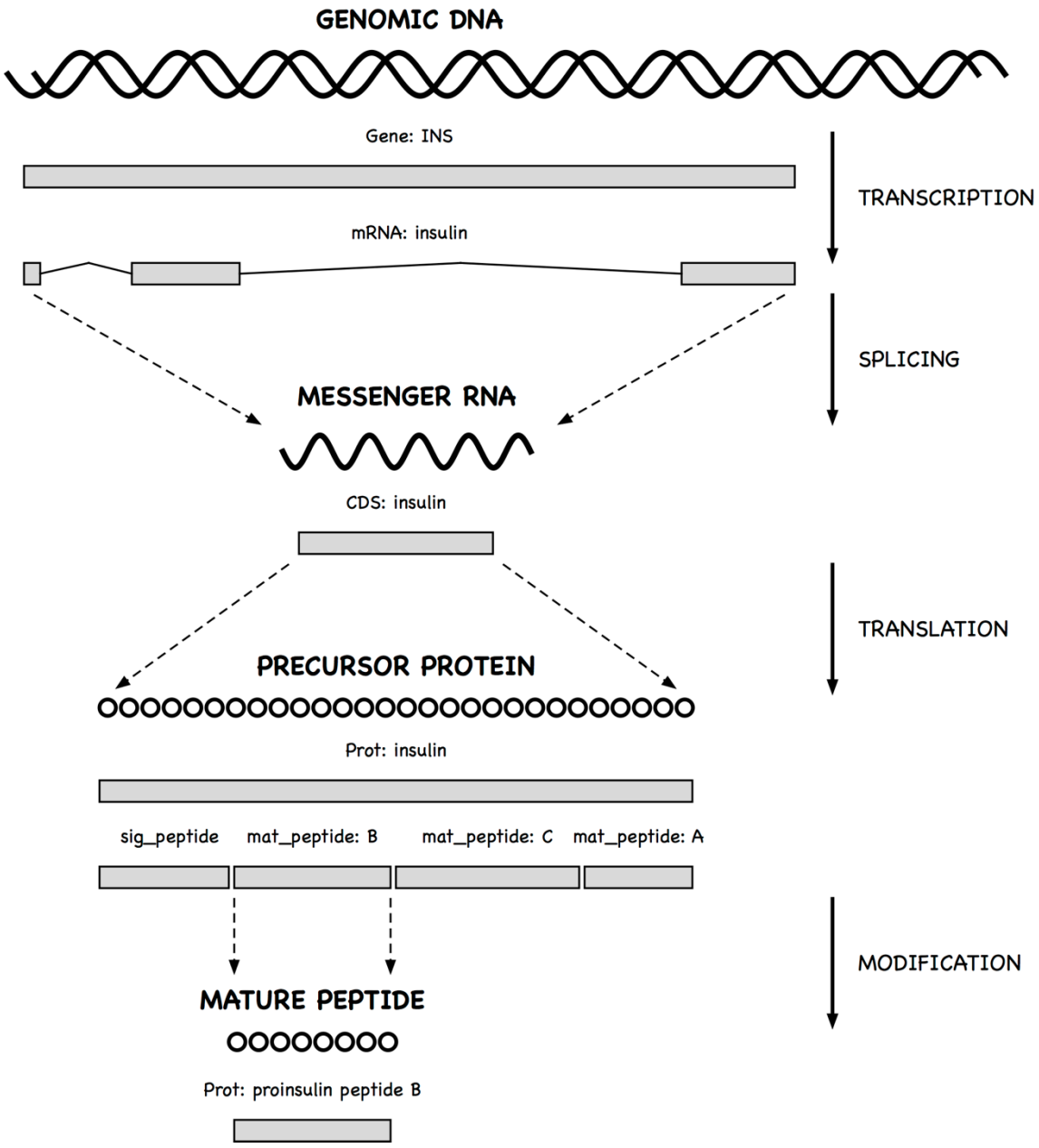
```
accession     length     rna_count     gene_count
U00096.3      4641652    187           4609
CP002956.1    4553770    86            4217
```

Sequence Records

NCBI Data Model for Sequence Records

The NCBI data model for sequence records 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 identifiers (e.g., accession numbers) for reference.

Each sequence can have multiple features, which contain information about the biology of a given region, including the transformations involved in gene expression. Each feature can have multiple qualifiers, which store specific details about that feature (e.g., name of the gene, genetic code used for protein translation, accession of the product sequence, cross-references to external databases).



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 of the message that remain after transcription and splicing. A coding region (CDS) feature has a product reference to the translated protein.

Since messenger RNA sequences are not always submitted with a genomic region, CDS features (which model the travel of ribosomes on transcript molecules) are traditionally annotated on the genomic sequence, with locations that encode the exonic intervals.

A qualifier can be dynamically generated from underlying data for the convenience of the user. Thus, the sequence of a mature peptide may be extracted from the mat_peptide feature's location on the precursor protein and displayed in a /peptide qualifier, even if a mature peptide is not instantiated.

Sequence Records in INSDSeq XML

Sequence records can be retrieved in an XML version of the GenBank or GenPept flatfile. The query:

```
efetch -db protein -id 26418308,26418074 -format gpc
```

returns a set of INSDSeq objects:

```
<INSDSet>
  <INSDSeq>
    <INSDSeq_locus>AAN78128</INSDSeq_locus>
    <INSDSeq_length>17</INSDSeq_length>
    <INSDSeq_moltype>AA</INSDSeq_moltype>
    <INSDSeq_topology>linear</INSDSeq_topology>
    <INSDSeq_division>INV</INSDSeq_division>
    <INSDSeq_update-date>03-JAN-2003</INSDSeq_update-date>
    <INSDSeq_create-date>10-DEC-2002</INSDSeq_create-date>
    <INSDSeq_definition>alpha-conotoxin ImI precursor, partial [Conus
       imperialis]</INSDSeq_definition>
    <INSDSeq_primary-accession>AAN78128</INSDSeq_primary-accession>
    <INSDSeq_accession-version>AAN78128.1</INSDSeq_accession-version>
    <INSDSeq_other-seqids>
      <INSDSeqid>gb|AAN78128.1|</INSDSeqid>
      <INSDSeqid>gi|26418308</INSDSeqid>
    </INSDSeq_other-seqids>
    <INSDSeq_source>Conus imperialis</INSDSeq_source>
    <INSDSeq_organism>Conus imperialis</INSDSeq_organism>
    <INSDSeq_taxonomy>Eukaryota; Metazoa; Lophotrochozoa; Mollusca;
       Gastropoda; Caenogastropoda; Hypsogastropoda; Neogastropoda;
       Conoidea; Conidae; Conus</INSDSeq_taxonomy>
    <INSDSeq_references>
      <INSDReference>
      ...
```

Biological features and qualifiers (shown here in GenPept format):

```
FEATURES             Location/Qualifiers
     source          1..17
                     /organism="Conus imperialis"
                     /db_xref="taxon:35631"
                     /country="Philippines"
     Protein         <1..17
                     /product="alpha-conotoxin ImI precursor"
     mat_peptide     5..16
                     /product="alpha-conotoxin ImI"
                     /note="the C-terminal glycine of the precursor is post
                     translationally removed"
                     /calculated_mol_wt=1357
                     /peptide="GCCSDPRCAWRC"
     CDS             1..17
                     /coded_by="AY159318.1:<1..54"
                     /note="nAChR antagonist"
```

are presented in INSDSeq XML as structured objects:

```
...
<INSDFeature>
  <INSDFeature_key>mat_peptide</INSDFeature_key>
  <INSDFeature_location>5..16</INSDFeature_location>
  <INSDFeature_intervals>
    <INSDInterval>
      <INSDInterval_from>5</INSDInterval_from>
      <INSDInterval_to>16</INSDInterval_to>
      <INSDInterval_accession>AAN78128.1</INSDInterval_accession>
    </INSDInterval>
  </INSDFeature_intervals>
```

```
    <INSDFeature_quals>
      <INSDQualifier>
        <INSDQualifier_name>product</INSDQualifier_name>
        <INSDQualifier_value>alpha-conotoxin ImI</INSDQualifier_value>
      </INSDQualifier>
      <INSDQualifier>
        <INSDQualifier_name>note</INSDQualifier_name>
        <INSDQualifier_value>the C-terminal glycine of the precursor is
           post translationally removed</INSDQualifier_value>
      </INSDQualifier>
      <INSDQualifier>
        <INSDQualifier_name>calculated_mol_wt</INSDQualifier_name>
        <INSDQualifier_value>1357</INSDQualifier_value>
      </INSDQualifier>
      <INSDQualifier>
        <INSDQualifier_name>peptide</INSDQualifier_name>
        <INSDQualifier_value>GCCSDPRCAWRC</INSDQualifier_value>
      </INSDQualifier>
    </INSDFeature_quals>
  </INSDFeature>
  ...
```

The data hierarchy is easily explored using a **–pattern** {sequence} **–group** {feature} **–block** {qualifier} construct. However, feature and qualifier names are indicated in data values, not XML element tags, and require –if and –equals to select the desired object and content.

Generating Qualifier Extraction Commands

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, **sub_sequence** and **feat_location**, are also supported.)

Running xtract –insd in an isolated command prints a new xtract statement that can then be copied, edited if necessary, and pasted into other queries. Running the –insd command within a multi-step pipe dynamically executes the automatically-constructed query.

Providing an optional (complete/partial) location indication, a feature key, and then one or more qualifier names:

```
xtract -insd complete mat_peptide product peptide
```

creates a new xtract statement that will produce a table of qualifier values from mature peptide features with complete locations. The statement starts with instructions to record the accession and find features of the indicated type:

```
xtract -pattern INSDSeq -ACCN INSDSeq_accession-version -SEQ INSDSeq_sequence \
  -group INSDFeature -if INSDFeature_key -equals mat_peptide \
    -unless INSDFeature_partial5 -or INSDFeature_partial3 \
    -clr -pfx "\n" -element "&ACCN" \
```

Each qualifier then generates custom extraction code that is appended to the growing query. For example:

```
-block INSDQualifier \
  -if INSDQualifier_name -equals product \
    -element INSDQualifier_value
```

Snail Venom Peptide Sequences

Incorporating the xtract −insd command in a search on cone snail venom:

```
esearch -db pubmed -query "conotoxin" |
elink -target protein |
efilter -query "mat_peptide [FKEY]" |
efetch -format gpc |
xtract -insd complete mat_peptide "%peptide" product mol_wt peptide |
```

prints the accession number, mature peptide length, product name, calculated molecular weight, and amino acid sequence for a sample of neurotoxic peptides:

```
AAN78128.1    12    alpha-conotoxin ImI    1357    GCCSDPRCAWRC
ADB65789.1    20    conotoxin Cal 16       2134    LEMQGCVCNANAKFCCGEGR
ADB65788.1    20    conotoxin Cal 16       2134    LEMQGCVCNANAKFCCGEGR
AGO59814.1    32    del13b conotoxin       3462    DCPTSCPTTCANGWECCKGYPCVRQHCSGCNH
AAO33169.1    16    alpha-conotoxin GIC    1615    GCCSHPACAGNNQHIC
AAN78279.1    21    conotoxin Vx-II        2252    WIDPSHYCCCGGGCTDDCVNC
AAF23167.1    31    BeTX toxin             3433    CRAEGTYCENDSQCCLNECCWGGCGHPCRHP
ABW16858.1    15    marmophin              1915    DWEYHAHPKPNSFWT
...
```

Piping the results to a series of Unix commands and EDirect scripts:

```
grep -i conotoxin |
filter-columns '10 <= $2 && $2 <= 30' |
sort-table -u -k 5 |
sort-table -k 2,2n |
align-columns -
```

filters by product name, limits the results to a specified range of peptide lengths, removes redundant sequences, sorts the table by peptide length, and aligns the columns for cleaner printing:

```
AAN78127.1    12    alpha-conotoxin ImII             1515    ACCSDRRCRWRC
AAN78128.1    12    alpha-conotoxin ImI              1357    GCCSDPRCAWRC
ADB43130.1    15    conotoxin Cal 1a                 1750    KCCKRHHGCHPCGRK
ADB43131.1    15    conotoxin Cal 1b                 1708    LCCKRHHGCHPCGRT
AAO33169.1    16    alpha-conotoxin GIC              1615    GCCSHPACAGNNQHIC
ADB43128.1    16    conotoxin Cal 5.1                1829    DPAPCCQHPIETCCRR
AAD31913.1    18    alpha A conotoxin Tx2            2010    PECCSHPACNVDHPEICR
ADB43129.1    18    conotoxin Cal 5.2                2008    MIQRSQCCAVKKNCCHVG
ADB65789.1    20    conotoxin Cal 16                 2134    LEMQGCVCNANAKFCCGEGR
ADD97803.1    20    conotoxin Cal 1.2                2206    AGCCPTIMYKTGACRTNRCR
AAD31912.1    21    alpha A conotoxin Tx1            2304    PECCSDPRCNSSHPELCGGRR
AAN78279.1    21    conotoxin Vx-II                  2252    WIDPSHYCCCGGGCTDDCVNC
ADB43125.1    22    conotoxin Cal 14.2               2157    GCPADCPNTCDSSNKCSPGFPG
ADD97802.1    23    conotoxin Cal 6.4                2514    GCWLCLGPNACCRGSVCHDYCPR
AAD31915.1    24    O-superfamily conotoxin TxO2     2565    CYDSGTSCNTGNQCCSGWCIFVCL
AAD31916.1    24    O-superfamily conotoxin TxO3     2555    CYDGGTSCDSGIQCCSGWCIFVCF
AAD31920.1    24    omega conotoxin SVIA mutant 1    2495    CRPSGSPCGVTSICCGRCYRGKCT
AAD31921.1    24    omega conotoxin SVIA mutant 2    2419    CRPSGSPCGVTSICCGRCSRGKCT
ABE27006.1    25    conotoxin p114a                  2917    FPRPRICNLACRAGIGHKYPFCHCR
ABE27007.1    25    conotoxin p114.1                 2645    GPGSAICNMACRLGQGHMYPFCNCN
...
```

Missing Qualifiers

For records where a particular qualifier is missing:

```
esearch -db protein -query "RAG1 [GENE] AND Mus musculus [ORGN]" |
efetch -format gpc |
xtract -insd source organism strain |
sort-table -u -k 2,3
```

a dash is inserted as a placeholder:

```
P15919.2       Mus musculus               -
AAO61776.1     Mus musculus               129/Sv
NP_033045.2    Mus musculus               C57BL/6
EDL27655.1     Mus musculus               mixed
BAD69530.1     Mus musculus castaneus     -
BAD69531.1     Mus musculus domesticus    BALB/c
BAD69532.1     Mus musculus molossinus    MOA
```

Sequence Coordinates

Gene Positions

An understanding of sequence coordinate conventions is necessary in order to use gene positions to retrieve the corresponding chromosome subregion with efetch or with the UCSC browser.

Sequence records displayed in GenBank or GenPept formats use a "one-based" coordinate system, with sequence position numbers starting at "1":

```
  1 catgccattc gttgagttgg aaacaaactt gccggctagc cgcatacccg cggggctgga
 61 gaaccggctg tgtgcggcca cagccaccat cctggacaaa cccgaagacg tgagtgaggg
121 tcggcgagaa cttgtgggct agggtcggac ctcccaatga cccgttccca tccccaggga
181 ccccactccc ctggtaacct ctgaccttcc gtgtcctatc ctcccttcct agatcccttc
...
```

Under this convention, positions refer to the sequence letters themselves:

```
C   A   T   G   C   C   A   T   T   C
1   2   3   4   5   6   7   8   9  10
```

and the position of the last base or residue is equal to the length of the sequence. The ATG initiation codon above is at positions 2 through 4, inclusive.

For computer programs, however, using "zero-based" coordinates can simplify the arithmetic used for calculations on sequence positions. The ATG codon in the 0-based representation is at positions 1 through 3. (The UCSC browser uses a hybrid, half-open representation, where the start position is 0-based and the stop position is 1-based.)

Software at NCBI will typically convert positions to 0-based coordinates upon input, perform whatever calculations are desired, and then convert the results to a 1-based representation for display. These transformations are done by simply subtracting 1 from the 1-based value or adding 1 to the 0-based value.

Coordinate Conversions

Retrieving the docsum for a particular gene:

```
esearch -db gene -query "BRCA2 [GENE] AND human [ORGN]" |
efetch -format docsum |
```

returns the chromosomal position of that gene in "zero-based" coordinates:

```
...
<GenomicInfoType>
```

```
    <ChrLoc>13</ChrLoc>
    <ChrAccVer>NC_000013.11</ChrAccVer>
    <ChrStart>32315479</ChrStart>
    <ChrStop>32399671</ChrStop>
    <ExonCount>27</ExonCount>
  </GenomicInfoType>
  ...
```

Piping the document summary to an xtract command using −element:

```
xtract -pattern GenomicInfoType -element ChrAccVer ChrStart ChrStop
```

obtains the accession and 0-based coordinate values:

```
NC_000013.11    32315479    32399671
```

Efetch has **−seq_start** and **−seq_stop** arguments to retrieve a gene segment, but these expect the sequence subrange to be in 1-based coordinates.

To address this problem, two additional efetch arguments, **−chr_start** and **−chr_stop**, were created to allow direct use of the 0-based coordinates:

```
efetch -db nuccore -format gb -id NC_000013.11 \
  -chr_start 32315479 -chr_stop 32399671
```

Xtract now has numeric extraction commands to assist with coordinate conversion. Selecting fields with an −inc argument:

```
xtract -pattern GenomicInfoType -element ChrAccVer -inc ChrStart ChrStop
```

obtains the accession and 0-based coordinates, then increments the positions to produce 1-based values:

```
NC_000013.11    32315480    32399672
```

EDirect knows the policies for sequence positions in all relevant Entrez databases (e.g., gene, snp, dbvar), and provides additional shortcuts for converting these to other conventions. For example:

```
xtract -pattern GenomicInfoType -element ChrAccVer -1-based ChrStart ChrStop
```

understands that gene docsum ChrStart and ChrStop fields are 0-based, sees that the desired output is 1-based, and translates the command to convert coordinates internally using the −inc logic. Similarly:

```
-element ChrAccVer -ucsc-based ChrStart ChrStop
```

leaves the 0-based start value unchanged but increments the original stop value to produce the half-open form that can be passed to the UCSC browser:

```
NC_000013.11    32315479    32399672
```

Gene Records

Genes in a Region

To list all genes between two markers flanking the human X chromosome centromere, first retrieve the protein-coding gene records on that chromosome:

```
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" |
```

Exploring each GenomicInfoType is needed because of pseudoautosomal regions at the ends of the X and Y chromosomes:

```
...
<GenomicInfo>
  <GenomicInfoType>
    <ChrLoc>X</ChrLoc>
    <ChrAccVer>NC_000023.11</ChrAccVer>
    <ChrStart>155997630</ChrStart>
    <ChrStop>156013016</ChrStop>
    <ExonCount>14</ExonCount>
  </GenomicInfoType>
  <GenomicInfoType>
    <ChrLoc>Y</ChrLoc>
    <ChrAccVer>NC_000024.10</ChrAccVer>
    <ChrStart>57184150</ChrStart>
    <ChrStop>57199536</ChrStop>
    <ExonCount>14</ExonCount>
  </GenomicInfoType>
</GenomicInfo>
...
```

Without limiting to chromosome X, the copy of IL9R near the "q" telomere of chromosome Y would be erroneously placed with genes that are near the X chromosome centromere, shown here in between SPIN2A and ZXDB:

```
...
57121860    FAAH2     fatty acid amide hydrolase 2
57133042    SPIN2A    spindlin family member 2A
57184150    IL9R      interleukin 9 receptor
57592010    ZXDB      zinc finger X-linked duplicated B
...
```

With genes restricted to the X chromosome, results can be sorted by position, and then filtered and partitioned:

```
sort -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 the two markers:

```
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
```

Genes in a Pathway

A gene can be linked to the biochemical pathways in which it participates:

```
esearch -db gene -query "PAH [GENE]" -organism human |
elink -target biosystems |
efilter -pathway wikipathways |
```

Linking from a pathway record back to the gene database:

```
elink -target gene |
efetch -format docsum |
xtract -pattern DocumentSummary -element Name Description |
grep -v pseudogene | grep -v uncharacterized | grep -v hypothetical |
sort -f
```

returns the set of all genes known to be involved in the pathway:

```
AANAT    aralkylamine N-acetyltransferase
ACADM    acyl-CoA dehydrogenase medium chain
ACHE     acetylcholinesterase (Cartwright blood group)
...
```

Gene Sequence

Genes encoded on the minus strand of a sequence:

```
esearch -db gene -query "DDT [GENE] AND mouse [ORGN]" |
efetch -format docsum |
xtract -pattern GenomicInfoType -element ChrAccVer ChrStart ChrStop |
```

have coordinates ("zero-based" in docsums) where the start position is greater than the stop:

```
NC_000076.6    75773373    75771232
```

These values can be read into Unix variables by a "while" loop:

```
while IFS=$'\t' read acn str stp
do
  efetch -db nuccore -format gb \
    -id "$acn" -chr_start "$str" -chr_stop "$stp"
done
```

The variables can then be used to obtain the reverse-complemented subregion in GenBank format:

```
LOCUS       NC_000076               2142 bp    DNA     linear   CON 08-AUG-2019
DEFINITION  Mus musculus strain C57BL/6J chromosome 10, GRCm38.p6 C57BL/6J.
ACCESSION   NC_000076 REGION: complement(75771233..75773374)
...
     gene            1..2142
                     /gene="Ddt"
     mRNA            join(1..159,462..637,1869..2142)
                     /gene="Ddt"
                     /product="D-dopachrome tautomerase"
                     /transcript_id="NM_010027.1"
     CDS             join(52..159,462..637,1869..1941)
                     /gene="Ddt"
                     /codon_start=1
                     /product="D-dopachrome decarboxylase"
                     /protein_id="NP_034157.1"
                     /translation="MPFVELETNLPASRIPAGLENRLCAATATILDKPEDRVSVTIRP
                     GMTLLMNKSTEPCAHLLVSSIGVVGTAEQNRTHSASFFKFLTEELSLDQDRIVIRFFP
                     ...
```

The reverse complement of a plus-strand sequence range can be selected with efetch −revcomp

External Data

Querying External Services

The nquire program uses command-line arguments to obtain data from RESTful, CGI, or FTP servers. Queries are built up from command-line arguments. Paths can be separated into components, which are combined with slashes. Remaining arguments (starting with a dash) are tag/value pairs, with multiple values between tags combined with commas.

For example, a POST request:

```
nquire -url http://w1.weather.gov/xml/current_obs/KSFO.xml |
xtract -pattern current_observation -tab "\n" \
  -element weather temp_f wind_dir wind_mph
```

returns the current weather report at the San Francisco airport:

```
A Few Clouds
54.0
Southeast
5.8
```

and a GET query:

```
nquire -get http://collections.mnh.si.edu/services/resolver/resolver.php \
  -voucher "Birds:321082" |
xtract -pattern Result -tab "\n" -element ScientificName StateProvince Country
```

returns information on a ruby-throated hummingbird specimen:

```
Archilochus colubris
Maryland
United States
```

while an FTP request:

```
nquire -ftp ftp.ncbi.nlm.nih.gov pub/gdp ideogram_9606_GCF_000001305.14_850_V1 |
grep acen | cut -f 1,2,6,7 | awk '/^X\t/'
```

returns data with the (estimated) sequence coordinates of the human X chromosome centromere (here showing where the p and q arms meet):

```
X    p    58100001    61000000
X    q    61000001    63800000
```

Nquire can also produce a list of files in an FTP server directory:

```
nquire -lst ftp://nlmpubs.nlm.nih.gov online/mesh/MESH_FILES/xmlmesh
```

or a list of FTP file names preceded by a column with the file sizes:

```
nquire -dir ftp.ncbi.nlm.nih.gov gene/DATA
```

Finally, nquire can download FTP files to the local disk:

```
nquire -dwn ftp.nlm.nih.gov online/mesh/MESH_FILES/xmlmesh desc2021.zip
```

If Aspera Connect is installed, the nquire −asp command will provide faster retrieval from NCBI servers:

```
nquire -asp ftp.ncbi.nlm.nih.gov pubmed baseline pubmed22n0001.xml.gz
```

Without Aspera Connect, nquire −asp defaults to using the −dwn logic.

XML Namespaces

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

```
nquire -url http://webservice.wikipathways.org getPathway -pwId WP455 |
xtract -pattern "ns1:getPathwayResponse" -element ":gpml" |
transmute -decode64 |
```

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
```

Automatic Xtract Format Conversion

Xtract can now detect and convert input data in JSON, text ASN.1, and GenBank/GenPept flatfile formats. The transmute commands or shortcut scripts, described below, are only needed if you want to inspect the intermediate XML, or to override default conversion settings.

JSON Arrays

Consolidated gene information for human β-globin retrieved from a curated biological database service developed at the Scripps Research Institute:

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

contains a multi-dimensional array of exon coordinates in JavaScript Object Notation (JSON) format:

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

This can be converted to XML with transmute **−j2x** (or the **json2xml** shortcut script):

```
transmute -j2x |
```

with the default"**−nest** element" argument assigning distinct tag names to each level:

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

JSON Mixtures

A query for the human green-sensitive opsin gene:

```
nquire -get http://mygene.info/v3/gene/2652 |
transmute -j2x |
```

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:

```
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
```

Xtract **-path** can explore using multi-level object addresses, delimited by periods or slashes:

```
xtract -pattern opt -path pathway.wikipathways.id -tab "\n" -element id
```

Conversion of ASN.1

Similarly to -j2x, transmute **-a2x** (or **asn2xml**) will convert Abstract Syntax Notation 1 (ASN.1) text files to XML.

Tables to XML

Tab-delimited files are easily converted to XML with transmute **-t2x** (or **tbl2xml**):

```
nquire -ftp ftp.ncbi.nlm.nih.gov gene/DATA gene_info.gz |
gunzip -c | grep -v NEWENTRY | cut -f 2,3 |
transmute -t2x -set Set -rec Rec -skip 1 Code Name
```

This takes a series of command-line arguments with tag names for wrapping the individual columns, and skips the first line of input, which contains header information, to generate a new XML file:

```
...
<Rec>
  <Code>1246500</Code>
  <Name>repA1</Name>
</Rec>
```

```
<Rec>
  <Code>1246501</Code>
  <Name>repA2</Name>
</Rec>
...
```

The transmute −t2x **−header** argument will obtain tag names from the first line of the file:

```
nquire -ftp ftp.ncbi.nlm.nih.gov gene/DATA gene_info.gz |
gunzip -c | grep -v NEWENTRY | cut -f 2,3 |
transmute -t2x -set Set -rec Rec -header
```

CSV to XML

Similarly to −t2x, transmute **−c2x** (or **csv2xml**) will convert comma-separated values (CSV) files to XML.

GenBank Download

The entire set of GenBank format release files be downloaded with:

```
fls=$( nquire -lst ftp.ncbi.nlm.nih.gov genbank )
for div in \
  bct con env est gss htc htg inv mam pat \
  phg pln pri rod sts syn tsa una vrl vrt
do
  echo "$fls" |
  grep ".seq.gz" | grep "gb${div}" |
  sort -V | skip-if-file-exists |
  nquire -asp ftp.ncbi.nlm.nih.gov genbank
done
```

Unwanted divisions can be removed from the "for" loop to limit retrieval to specific sequencing classes or taxonomic regions.

GenBank to XML

Recent GenBank virus data can be downloaded with:

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

GenBank flatfiles can be parsed into INSDSeq XML with transmute **−g2x** (or **gbf2xml**):

```
gunzip -c *.seq.gz | transmute -g2x |
```

They can then be filtered by organism name or taxon identifier with xtract **−select**:

```
xtract -pattern INSDSeq -select INSDQualifier_value -equals "taxon:11292" |
```

and used to obtain feature location intervals and underlying sequences of individual coding regions:

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

GenPept to XML

The latest GenPept daily incremental update file can be downloaded:

```
nquire -ftp ftp.ncbi.nlm.nih.gov genbank daily-nc Last.File |
sed "s/flat/gnp/g" |
```

```
nquire -ftp ftp.ncbi.nlm.nih.gov genbank daily-nc |
gunzip -c | transmute -g2x |
```

and the extracted INSDSeq XML can be processed in a similar manner:

```
xtract -pattern INSDSeq -select INSDQualifier_value -equals "taxon:2697049" |
xtract -insd mat_peptide product sub_sequence
```

Local PubMed Cache

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.

Recent advances in technology provide an affordable and practical alternative. High-performance NVMe solid-state drives (which eliminate rotational delays for file access and bookkeeping operations) are readily available for purchase. Modern high-capacity file systems, such as APFS (which uses 64-bit inodes) or Ext4 (which can be configured for 100 million inodes), are now ubiquitous on contemporary computers. A judicious arrangement of multi-level nested directories (each containing no more than 100 subfolders or record files) ensures maximally-efficient use of these enhanced capabilities.

This combination of features allows local record storage (populated in advance from the PubMed FTP release files) to be an effective replacement for on-demand network retrieval, while avoiding the need to install and support a legacy database product on your computer.

Random Access Archive

EDirect can now preload over 30 million live PubMed records onto an inexpensive external 500 GB (gigabyte) solid state drive as individual files for rapid retrieval. For example, PMID 12345678 would be stored at:

```
/Archive/12/34/56/12345678.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 system, with no need for the user to download, configure, and maintain complicated third-party database software.

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

```
export EDIRECT_PUBMED_MASTER=/Volumes/external_drive_name
```

or set separate environment variables to keep the intermediate steps on the external SSD but leave the resulting archive in a designated area of the computer's internal storage:

```
export EDIRECT_PUBMED_MASTER=$HOME/internal_directory_name
export EDIRECT_PUBMED_WORKING=/Volumes/external_drive_name
```

In the latter case it will store around 180 GB on the internal drive for the local archive, or up to 250 GB if the local search index (see below) is also built.

Then run **archive-pubmed** to download the PubMed release files and distribute each record on the drive. This process will take several hours to complete, but subsequent updates are incremental, and should finish in minutes.

Retrieving over 125,000 compressed PubMed records from the local archive:

```
esearch -db pubmed -query "PNAS [JOUR]" -pub abstract |
efetch -format uid | stream-pubmed | gunzip -c |
```

takes about 20 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:

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

requires 13 seconds to match 7854 subsequent articles. Retrieving them from the local archive:

```
efetch -format uid | fetch-pubmed |
```

takes less than one second. Printing the names of all authors in those records:

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

allows creation of a frequency table:

```
sort-uniq-count-rank
```

that lists the authors who most often cited the original papers:

```
117    Cozzarelli NR
84     Maxwell A
61     Wang JC
59     Osheroff N
...
```

Fetching from the network service would extend the 13 second running time to over 2 minutes.

Local Search Index

A similar divide-and-conquer strategy is 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:

```
/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 underlying software supports efficient expression evaluation, unrestricted wildcard truncation, phrase queries, and proximity searches.

The **phrase-search** script provides access to the local search system.

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

```
phrase-search -fields

phrase-search -terms TITL

phrase-search -totals PROP
```

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

```
phrase-search -count "catabolite repress*"

phrase-search -counts "catabolite repress*"
```

Query evaluation includes Boolean operations and parenthetical expressions:

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

Adjacent words in the query are treated as a contiguous phrase:

```
phrase-search -query "selective serotonin reuptake inhibitor"
```

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

```
phrase-search -query "vitamin c + + common cold"

phrase-search -query "vitamin c ~ ~ common cold"
```

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):

```
phrase-search -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:

```
phrase-search -query "C14.907.617.812* [TREE] AND 2015:2019 [YEAR]"

phrase-search -query "Raynaud Disease [MESH]"
```

The phrase-search **−filter** command allows PMIDs to be generated by an EDirect search and then incorporated as a component in a local query:

Natural Language Processing

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

Running **archive-pubmed −extras** periodically (monthly) will automatically refresh any out-of-date support files and then index the connections in CHEM, DISZ, GENE, and several gene subfields (GRIF, GSYN, and PREF):

```
phrase-search -terms DISZ | grep -i Raynaud

phrase-search -counts "Raynaud* [DISZ]"

phrase-search -query "Raynaud Disease [DISZ]"
```

Data Analysis and Visualization

All query commands return a list of PMIDs, which can be piped directly to **fetch-pubmed** to retrieve the uncompressed records. For example:

```
phrase-search -query "selective serotonin ~ ~ ~ reuptake inhibit*" |
fetch-pubmed |
xtract -pattern PubmedArticle -num AuthorList/Author |
sort-uniq-count -n |
```

```
reorder-columns 2 1 |
head -n 25 |
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 12,966 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:

```
0      51
1    1382
2    1897
3    1906
...
```

The phrase-search and fetch-pubmed scripts are front-ends to the **rchive** program, which is used to build and search the inverted retrieval system. Rchive is multi-threaded for speed, retrieving records from the local archive in parallel, and fetching the positional indices for all terms in parallel before evaluating the title words as a contiguous phrase.

The cumulative size of PubMed can be calculated with a running sum of the annual record counts:

```
phrase-search -totals YEAR |
print-columns '$2, $1, total += $1' |
```

Exponential growth over time will appear as a roughly linear curve on a semi-logarithmic graph:

```
print-columns '$1, log($2)/log(10), log($3)/log(10)' |
xy-plot annual-and-cumulative.png
```

Rapidly Scanning PubMed

If the **expand-current** script is run, an ad hoc scan can be performed on the nonredundant set of live PubMed records:

```
cat $EDIRECT_PUBMED_WORKING/Scratch/Current/*.xml |
xtract -timer -turbo -pattern PubmedArticle -PMID MedlineCitation/PMID \
  -group AuthorList -if "#LastName" -eq 7 -element "&PMID" LastName
```

finding 1,700,652 articles with seven authors. (This query excludes consortia and additional named investigators. Author count is now indexed in the ANUM field.)

Xtract uses the Boyer-Moore-Horspool algorithm to partition an XML stream into individual records, distributing them among multiple instances of the data exploration and extraction function for concurrent execution. A multi-core computer with a solid state drive can process all of PubMed in under 4 minutes.

The expand-current script now calls xtract -index to place an XML size object immediately before each PubMed record:

```
...
</PubmedArticle>
<NEXT_RECORD_SIZE>6374</NEXT_RECORD_SIZE>
<PubmedArticle>
...
```

The xtract **-turbo** flag reads this precomputed information to approximately double the speed of record partitioning, which is the rate-limiting step when many CPU cores are available. With proper cooling, it should allow up to a dozen cores to contribute to batch data extraction throughput.

User-Specified Term Index

Running **custom-index** with a PubMed indexer script and the names of the fields it populates:

```
custom-index $( which idx-stemmed ) STEM
```

integrates user-specified indices into the local search system. The **idx-stemmed** script:

```
xtract -accent -set IdxDocumentSet -rec IdxDocument -pattern PubmedArticle \
  -wrp IdxUid -element MedlineCitation/PMID -clr -rst -tab "" \
  -group PubmedArticle -pkg IdxSearchFields \
    -block PubmedArticle -stemmed ArticleTitle,Abstract/AbstractText
```

has reusable boilerplate in its first three lines, and indexes title and abstract words stemmed by the Porter2 algorithm:

```
...
<IdxDocument>
  <IdxUid>2539356</IdxUid>
  <IdxSearchFields>
    <STEM pos="126">act</STEM>
    <STEM pos="188">addit</STEM>
    <STEM pos="146">base</STEM>
    ...
  </IdxSearchFields>
</IdxDocument>
...
```

Once the final inversion:

```
...
<InvDocument>
  <InvKey>act</InvKey>
  <InvIDs>
    <STEM pos="126">2539356</STEM>
  </InvIDs>
</InvDocument>
<InvDocument>
  <InvKey>addit</InvKey>
  <InvIDs>
    <STEM pos="188">2539356</STEM>
  </InvIDs>
</InvDocument>
...
```

and posting steps are completed, the new fields are ready to be searched:

```
phrase-search -query "monoamine oxidase inhibitor [STEM]"
```

Processing by XML Subset

A query on articles with abstracts published in a chosen journal, retrieved from the local cache, and followed by a multi-step transformation:

```
esearch -db pubmed -query "PNAS [JOUR]" -pub abstract |
efetch -format uid | fetch-pubmed |
xtract -stops -rec Rec -pattern PubmedArticle \
  -wrp Year -year "PubDate/*" -wrp Abst -words Abstract/AbstractText |
xtract -rec Pub -pattern Rec \
  -wrp Year -element Year -wrp Num -num Abst > countsByYear.xml
```

returns structured data with the year of publication and number of words in the abstract for each record:

```
<Pub><Year>2018</Year><Num>198</Num></Pub>
<Pub><Year>2018</Year><Num>167</Num></Pub>
<Pub><Year>2018</Year><Num>242</Num></Pub>
```

(The ">" redirect saves the results to a file.)

The following "for" loop limits the processed query results to one year at a time with xtract −select, passing the relevant subset to a second xtract command:

```
for yr in {1960..2021}
do
  cat countsByYear.xml |
  xtract -set Raw -pattern Pub -select Year -eq "$yr" |
  xtract -pattern Raw -lbl "$yr" -avg Num
done |
```

that applies −avg to the word counts in order to compute the average number of abstract words per article for the current year:

```
1969    122
1970    120
1971    127
...
2018    207
2019    207
2020    208
```

This result can be saved by redirecting to a file, or it can be piped to:

```
tee /dev/tty |
xy-plot pnas.png
```

to print the data to the terminal and then display the results in graphical format. The last step should be:

```
rm countsByYear.xml
```

to remove the intermediate file.

Identifier Conversion

The archive-pubmed script also downloads MeSH descriptor information from the NLM FTP server and generates a conversion file:

```
...
<Rec>
  <Code>D064007</Code>
  <Name>Ataxia Telangiectasia Mutated Proteins</Name>
  ...
  <Tree>D12.776.157.687.125</Tree>
  <Tree>D12.776.660.720.125</Tree>
</Rec>
...
```

that can be used for mapping MeSH codes to and from chemical or disease names. For example:

```
cat $EDIRECT_PUBMED_MASTER/Data/meshconv.xml |
xtract -pattern Rec \
  -if Name -starts-with "ataxia telangiectasia" \
    -element Code
```

will return:

```
C565779
C576887
D001260
D064007
```

More information on a MeSH term could be obtained by running:

```
efetch -db mesh -id D064007 -format docsum
```

External NLP Data

Additional NLP annotation on PubMed can be downloaded and indexed by running **index-extras**.

Recent research at Stanford University defined biological themes, supported by dependency paths, which are indexed in THME and CONV fields. Theme keys in the Global Network of Biomedical Relationships are taken from a table in the paper:

A+	Agonism, activation	**N**	Inhibits
A–	Antagonism, blocking	**O**	Transport, channels
B	Binding, ligand	**Pa**	Alleviates, reduces
C	Inhibits cell growth	**Pr**	Prevents, suppresses
D	Drug targets	**Q**	Production by cell population
E	Affects expression/production	**Rg**	Regulation
E+	Increases expression/production	**Sa**	Side effect/adverse event
E–	Decreases expression/production	**T**	Treatment/therapy
G	Promotes progression	**Te**	Possible therapeutic effect
H	Same protein or complex	**U**	Causal mutations
I	Signaling pathway	**Ud**	Mutations affecting disease course
J	Role in disease pathogenesis	**V+**	Activates, stimulates
K	Metabolism, pharmacokinetics	**W**	Enhances response
L	Improper regulation linked to disease	**X**	Overexpression in disease
Md	Biomarkers (diagnostic)	**Y**	Polymorphisms alter risk
Mp	Biomarkers (progression)	**Z**	Enzyme activity

Themes common to multiple chemical-disease-gene relationships are disambiguated so they can be queried individually. The expanded list, along with MeSH category codes, can be seen with:

```
phrase-search -extras
```

Integration with Entrez

Use phrase-search -filter to combine an EDirect search result with a local query:

```
esearch -db pubmed -query "complement system proteins [MESH]" |
efetch -format uid |
phrase-search -filter "L [THME] AND D10* [TREE]"
```

This finds PubMed papers about complement proteins and limits them by the "improper regulation linked to disease" theme and the "lipids" MeSH chemical category:

```
448084
1292783
1379443
...
```

Intermediate lists of PMIDs can be saved to a file and piped (with "cat") into a subsequent phrase-search –filter query. They can also be uploaded to the Entrez history server by piping to **epost**:

```
epost -db pubmed
```

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 "${EDIRECT_PUBMED_MASTER}"
sudo mdutil -E "${EDIRECT_PUBMED_MASTER}"
```

and disable FSEvents logging with:

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

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

Automation

Unix Shell Scripting

A shell script can be used to repeat the same sequence of operations on a number of input values. The Unix shell is a command interpreter that supports user-defined variables, conditional statements, and repetitive execution loops. Scripts are usually saved in a file, and referenced by file name.

Comments start with a pound sign ("#") and are ignored. Quotation marks within quoted strings are entered by "escaping" with a backslash ("\"). Subroutines (functions) can be used to collect common code or simplify the organization of the script.

Combining Data from Adjacent Lines

Given a tab-delimited file of feature keys and values, where each gene is followed by its coding regions:

```
gene    matK
CDS     maturase K
gene    ATP2B1
CDS     ATPase 1 isoform 2
CDS     ATPase 1 isoform 7
gene    ps2
CDS     peptide synthetase
```

the **cat** command can pipe the file contents to a shell script that reads the data one line at a time:

```
#!/bin/bash

gene=""
while IFS=$'\t' read feature product
do
  if [ "$feature" = "gene" ]
  then
    gene="$product"
  else
    echo "$gene\t$product"
  fi
done
```

The resulting output lines, printed by the **echo** command, have the gene name and subsequent CDS product names in separate columns on individual rows:

```
matK      maturase K
ATP2B1    ATPase 1 isoform 2
ATP2B1    ATPase 1 isoform 7
ps2       peptide synthetase
```

Dissecting the script, the first line selects the Bash shell on the user's machine:

```
#!/bin/bash
```

The latest gene name is stored in the "gene" variable, which is first initialized to an empty string:

```
gene=""
```

The **while** command sequentially reads each line of the input file, **IFS** indicates tab-delimited fields, and **read** saves the first field in the "feature" variable and the remaining text in the "product" variable:

```
while IFS=$'\t' read feature product
```

The statements between the **do** and **done** commands are executed once for each input line. The **if** statement retrieves the current value stored in the feature variable (indicated by placing a dollar sign (**$**) in front of the variable name) and compares it to the word "gene":

```
if [ "$feature" = "gene" ]
```

If the feature key was "gene", it runs the **then** section, which copies the contents of the current line's "product" value into the persistent "gene" variable:

```
then
  gene="$product"
```

Otherwise the **else** section prints the saved gene name and the current coding region product name:

```
else
  echo "$gene\t$product"
```

separated by a tab character. The conditional block is terminated with a **fi** instruction ("if" in reverse):

```
fi
```

In addition to else, the **elif** command can allow a series of mutually-exclusive conditional tests:

```
if [ "$feature" = "gene" ]
then
  ...
elif [ "$feature" = "mRNA" ]
then
  ...
elif [ "$feature" = "CDS" ]
then
  ...
else
  ...
fi
```

A variable can be set to the result of commands that are enclosed between "**$(**" and "**)**" symbols:

```
mrna=$( echo "$product" | grep 'transcript variant' |
        sed 's/^.*transcript \(variant .*\).*$/\1/' )
```

Entrez Direct Commands Within Scripts

EDirect commands can also be run inside scripts. Saving the following text:

```
#!/bin/bash

printf "Years"
for disease in "$@"
do
  frst=$( echo -e "${disease:0:1}" | tr [a-z] [A-Z] )
  printf "\t${frst}${disease:1:3}"
done
printf "\n"

for (( yr = 2020; yr >= 1900; yr -= 10 ))
do
  printf "${yr}s"
  for disease in "$@"
  do
    val=$(
      esearch -db pubmed -query "$disease [TITL]" |
      efilter -mindate "${yr}" -maxdate "$((yr+9))" |
      xtract -pattern ENTREZ_DIRECT -element Count
    )
    printf "\t${val}"
  done
  printf "\n"
done
```

to a file named "scan_for_diseases.sh" and executing:

```
chmod +x scan_for_diseases.sh
```

allows the script to be called by name. Passing several disease names in command-line arguments:

```
scan_for_diseases.sh diphtheria pertussis tetanus |
```

returns the counts of papers on each disease, by decade, for over a century:

```
Years    Diph    Pert    Teta
2020s    104     281     154
2010s    860     2558    1296
2000s    892     1968    1345
1990s    1150    2662    1617
1980s    780     1747    1488
...
```

A graph of papers per decade for each disease is generated by piping the table to:

```
xy-plot diseases.png
```

Passing the data instead to:

```
align-columns -h 2 -g 4 -a ln
```

right-justifies numeric data columns for easier reading or for publication:

```
Years    Diph    Pert    Teta
2020s     104     281     154
2010s     860    2558    1296
2000s     892    1968    1345
```

```
1990s    1150    2662    1617
1980s     780    1747    1488
...
```

while piping to:

```
transmute -t2x -set Set -rec Rec -header
```

produces a custom XML structure for further comparative analysis by xtract.

Time Delay

The shell script command:

```
sleep 1
```

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

Xargs/Sh Loop

Writing a script to loop through data can sometimes be avoided by creative use of the Unix xargs and sh commands. Within the "sh −c" command string, the last name and initials arguments (passed in pairs by "xargs −n 2") are substituted at the "$0" and "$1" variables. All of the commands in the sh string are run separately on each name:

```
echo "Garber ED Casadaban MJ Mortimer RK" |
xargs -n 2 sh -c 'esearch -db pubmed -query "$0 $1 [AUTH]" |
xtract -pattern ENTREZ_DIRECT -lbl "$1 $0" -element Count'
```

This produces PubMed article counts for each author:

```
ED Garber       35
MJ Casadaban    46
RK Mortimer     85
```

While Loop

A "while" loop can also be used to independently process lines of data. Given a file "organisms.txt" containing genus-species names, the Unix "cat" command:

```
cat organisms.txt |
```

writes the contents of the file:

```
Arabidopsis thaliana
Caenorhabditis elegans
Danio rerio
Drosophila melanogaster
Escherichia coli
Homo sapiens
Mus musculus
Saccharomyces cerevisiae
```

This can be piped to a loop that reads one line at a time:

```
while read org
do
  esearch -db taxonomy -query "$org [LNGE] AND family [RANK]" < /dev/null |
  efetch -format docsum |
  xtract -pattern DocumentSummary -lbl "$org" \
```

```
    -element ScientificName Division
done
```

looking up the taxonomic family name and BLAST division for each organism:

```
Arabidopsis thaliana        Brassicaceae          eudicots
Caenorhabditis elegans      Rhabditidae           nematodes
Danio rerio                 Cyprinidae            bony fishes
Drosophila melanogaster     Drosophilidae         flies
Escherichia coli            Enterobacteriaceae    enterobacteria
Homo sapiens                Hominidae             primates
Mus musculus                Muridae               rodents
Saccharomyces cerevisiae    Saccharomycetaceae    ascomycetes
```

(The "< **/dev/null**" input redirection construct prevents esearch from "draining" the remaining lines from stdin.)

For Loop

The same results can be obtained with organism names embedded in a "for" loop:

```
for org in \
  "Arabidopsis thaliana" \
  "Caenorhabditis elegans" \
  "Danio rerio" \
  "Drosophila melanogaster" \
  "Escherichia coli" \
  "Homo sapiens" \
  "Mus musculus" \
  "Saccharomyces cerevisiae"
do
  esearch -db taxonomy -query "$org [LNGE] AND family [RANK]" |
  efetch -format docsum |
  xtract -pattern DocumentSummary -lbl "$org" \
    -element ScientificName Division
done
```

File Exploration

A for loop can also be used to explore the computer's file system:

```
for i in *
do
  if [ -f "$i" ]
  then
    echo $(basename "$i")
  fi
done
```

visiting each file within the current directory. The asterisk ("*") character indicates all files, and can be replaced by any pattern (e.g., "*.txt") to limit the file search. The if statement "−f" operator can be changed to "−d" to find directories instead of files, and "−s" selects files with size greater than zero.

Processing in Groups

EDirect supplies a **join-into-groups-of** script that combines lines of unique identifiers or sequence accession numbers into comma-separated groups:

```
#!/bin/sh
xargs -n "$@" echo |
sed 's/ /,/g
```

The following example demonstrates processing sequence records in groups of 200 accessions at a time:

```
...
efetch -format acc |
join-into-groups-of 200 |
xargs -n 1 sh -c 'epost -db nuccore -format acc -id "$0" |
elink -target pubmed |
efetch -format abstract'
```

Programming in Go

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, at the cost of added complexity during development.

Google's Go language (also known as "golang") is "an open source programming language that makes it easy to build simple, reliable, and efficient software". Go eliminates the need for maintaining complicated "make" files. The build system assumes full responsibility for downloading external library packages. Automated dependency management tracks module release numbers to prevent version skew.

As of 2020, the Go development process has been streamlined to the point that it is now easier to use than some popular scripting languages.

To build Go programs, the latest Go compiler must be installed on your computer. A link to the installation URL is in the Documentation section at the end of this web page.

basecount.go Program

Piping FASTA data to the basecount binary executable (compiled from the basecount.go source code file shown 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
```

The full (uncommented) source code for basecount.go is shown here, and is discussed below:

```
package main

import (
    "eutils"
    "fmt"
    "os"
    "sort"
)

func main() {

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

    countLetters := func(id, seq string) {

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

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

        var keys []rune
        for ky := range counts {
            keys = append(keys, ky)
        }
        sort.Slice(keys, func(i, j int) bool { return keys[i] < keys[j] })

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

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

Performance can be measured with the Unix "time" command:

```
time basecount < NC_000014.fsa
```

The program reads and counts the 107,043,718 bases of human chromosome 14, from an existing FASTA file, in under 2.5 seconds:

```
NC_000014.9    A 26673415    C 18423758    G 18559033    N 16475569    T 26911943
2.287
```

basecount.go Code Review

Go programs 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 (
    "eutils"
    "fmt"
    "os"
    "sort"
)
```

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]++
        }
```

(String iteration by range returns position and letter pairs. Since the code does not use the position, its value is absorbed by an underscore ("_") character.)

Maps are not returned in a defined order, so map keys are loaded to a keys **array**, which is then sorted:

```
        var keys []rune
        for ky := range counts {
            keys = append(keys, ky)
        }
        sort.Slice(keys, func(i, j int) bool { return keys[i] < keys[j] })
```

(The second argument passed to sort.Slice is an **anonymous** function literal used to control the sort order. It is also a **closure**, implicitly inheriting the keys array from the enclosing function.)

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)
    }
}
```

Note that the sequence of human chromosome 14, processed above, is stored in its entirety as a single contiguous Go string. No special coding considerations are needed for input, access, or memory management, even though it is over 107 million characters long.

Go Dependency Management

EDirect includes source code for the **eutils** helper library, which consolidates common functions used by xtract, transmute, and rchive, including the FASTA parser/streamer used by the basecount program shown above.

In addition to around two dozen eutils "*.go" files, the distribution contains "go.mod" and "go.sum" module files for the eutils package. They were created by running "./build.sh" in the eutils directory prior to release on the FTP site.

Modules provide a mechanism for automatically managing external dependencies. They record version numbers and checksums for the packages imported by eutils source files during development. Go will then retrieve the same versions of those packages, along with all of their internal support packages, if the eutils library is later incorporated into other software development projects.

Use of modules allows external Go packages to evolve independently, publishing newer versions with incompatible function argument signatures on their own schedules, while ensuring that this natural software development cycle does not break a working library or application build at some inopportune time in the future.

Compiling a Go Project

Each project typically resides in its own directory. The source code can be split into multiple files, and the build process will normally compile all of the "*.go" files together.

Create a new directory named "basecount" with:

```
cd ~
mkdir basecount
```

and copy the **basecount.go** source code file into that directory.

The program can then be compiled by running:

```
cd basecount
go mod init basecount
echo "replace eutils => $HOME/edirect/eutils" >> go.mod
go mod tidy
go build
```

but for convenience these commands are usually incorporated into a build script. To do this, save the following script to a file named **build.sh** in the same directory:

```
#!/bin/bash

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

if [ ! -f "go.sum" ]
then
  go mod tidy
fi

go build
```

To compile the executable, enter the basecount directory, set the Unix execution permission bit, and run the build script:

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

The build script runs "**go mod init**" to generate "go.mod", and "**go mod tidy**" to generate " go.sum", if either module file is not already present.

(The "$(basename $PWD)" construct sets the executable's default name to match the parent directory, without needing to manually customize the "go mod init" line for each project.)

(The "replace eutils => $HOME/edirect/eutils" construct computes the path for finding the local eutils source code directory in the standard EDirect installation location.)

The "**go build**" instruction compiles the source file(s) for the application and all dependent libraries (caching the compiled object files for faster use later). It will then link these into a binary executable file that can run on the development machine.

You can select specific input files, change the executable program's name, and cross-compile for a different platform, with additional arguments to "go build":

```
env GOOS=darwin GOARCH=arm64 go build -o basecount.Silicon basecount.go
```

Separate projects in a single directory could be built by changing the "go build" line to:

```
for fl in *.go
do
  go build -o "${fl%.go}" "$fl"
done
```

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:

```
import subprocess
import shlex

def execute(cmmd, data=""):
    if isinstance(cmmd, str):
        cmmd = shlex.split(cmmd)
    res = subprocess.run(cmmd, input=data,
                         capture_output=True,
                         encoding='UTF-8')
    return res.stdout.strip()

def pipeline(cmmds, data=""):
    def flatten(cmmd):
        if isinstance(cmmd, str):
            return cmmd
        else:
            return shlex.join(cmmd)
    if not isinstance(cmmds, str):
        cmmds = ' | '.join(map(flatten, cmmds))
    res = subprocess.run(cmmds, input=data, shell=True,
                         capture_output=True,
                         encoding='UTF-8')
    return res.stdout.strip()

def efetch(*, db, id, format, mode=""):
    cmmd = ('efetch', '-db', db, '-id', str(id), '-format', format)
    if mode:
        cmmd = cmmd + ('-mode', mode)
    return execute(cmmd))
```

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
```

(Note that the import command uses "edirect", without the ".py" extension.)

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

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

or as a sequence of strings:

```
edirect.execute(('efetch', '-db', 'nuccore', '-id', 'NM_000518.5', '-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)
```

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

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

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), "")
```

NCBI C++ Toolkit Access

EDirect scripts can be called from the NCBI C++ toolkit using the **ncbi::edirect::Execute** function:

```
#include <misc/eutils_client/eutils_client.hpp>
```

The function signature has separate parameters for the script name and its command-line arguments, followed by an optional string to be passed via stdin:

```
string Execute (
  const string& cmmd,
  const vector<string>& args,
  const string& data = kEmptyStr
);
```

Multiple steps are chained together by using the previous result as the data argument in the next command:

```
string seq = ncbi::edirect::Execute
  ( "efetch", { "-db", "nuccore", "-id", "NM_000518.5", "-format", "fasta" } );
```

```
string sub = ncbi::edirect::Execute
  ( "transmute", { "-extract", "-1-based", "-loc", "51..494" }, seq );
string prt = ncbi::edirect::Execute
  ( "transmute", { "-cds2prot", "-every", "-trim" }, sub );
```

The argument vector can also be generated dynamically, under program control:

```
vector<string> args;

args.push_back("-db");
args.push_back("pubmed");
args.push_back("-format");
args.push_back("abstract");
args.push_back("-id");
args.push_back(uid);
```

Citation matching can be performed on a CPub object reference with the -asn argument:

```
string uid = ncbi::edirect::Execute
  ( "cit2pmid", { "-asn", FORMAT(MSerial_FlatAsnText << pub) } );
```

or you can use -title, -author, -journal, -volume, -issue, -pages, and -year arguments.

If a matching PMID is found, it can be retrieved as PubmedArticle XML and transformed into Pubmed-entry ASN.1:

```
string xml = ncbi::edirect::Execute
  ( "efetch", { "-db", "pubmed", "-format", "xml" }, uid );
string asn = ncbi::edirect::Execute
  ( "pma2pme", { "-std" }, xml );
```

The ASN.1 string can then be read into memory with an object loader for further processing:

```
#include <objects/pubmed/Pubmed_entry.hpp>

unique_ptr<CObjectIStream> stm;
stm.reset ( CObjectIStream::CreateFromBuffer
  ( eSerial_AsnText, asn.data(), asn.length() ) );

CRef<CPubmed_entry> pme ( new CPubmed_entry );
stm->Read ( ObjectInfo ( *pme ) );
```

Additional Examples

EDirect examples demonstrate how to answer ad hoc questions in several Entrez databases. The detailed examples have been moved to a separate document, which can be viewed by clicking on the ADDITIONAL EXAMPLES link.

Appendices

Command-Line Arguments

Each EDirect program has a **–help** command that prints detailed information about available arguments. These include –**sort** values for esearch, **–format** and **–mode** choices for efetch, and **–cmd** options for elink.

Einfo Data

Einfo field data contains status flags for several term list index properties:

```
<Field>
  <Name>ALL</Name>
  <FullName>All Fields</FullName>
  <Description>All terms from all searchable fields</Description>
  <TermCount>280005319</TermCount>
  <IsDate>N</IsDate>
  <IsNumerical>N</IsNumerical>
  <SingleToken>N</SingleToken>
  <Hierarchy>N</Hierarchy>
  <IsHidden>N</IsHidden>
  <IsTruncatable>Y</IsTruncatable>
  <IsRangable>N</IsRangable>
</Field>
```

Unix Utilities

Several useful classes of Unix text processing filters, with selected arguments, are presented below:

Process by Contents:

```
sort    Sorts lines of text

  -f    Ignore case
  -n    Numeric comparison
  -r    Reverse result order

  -k    Field key (start,stop or first)
  -u    Unique lines with identical keys

  -b    Ignore leading blanks
  -s    Stable sort
  -t    Specify field separator

uniq    Removes repeated lines

  -c    Count occurrences
  -i    Ignore case

  -f    Ignore first n fields
  -s    Ignore first n characters

  -d    Only output repeated lines
  -u    Only output non-repeated lines

grep    Matches patterns using regular expressions

  -i    Ignore case
  -v    Invert search
  -w    Search expression as a word
  -x    Search expression as whole line

  -e    Specify individual pattern

  -c    Only count number of matches
  -n    Print line numbers
  -A    Number of lines after match
  -B    Number of lines before match
```

Regular Expressions:

```
  Characters

    .     Any single character (except newline)
    \w    Alphabetic [A-Za-z], numeric [0-9], or underscore (_)
    \s    Whitespace (space or tab)
    \     Escapes special characters
    []    Matches any enclosed characters

  Positions

    ^     Beginning of line
    $     End of line
    \b    Word boundary

  Repeat Matches

    ?     0 or 1
    *     0 or more
    +     1 or more
    {n}   Exactly n

  Escape Sequences

    \n    Line break
    \t    Tab character
```

Modify Contents:

```
  sed         Replaces text strings

    -e        Specify individual expression
    s///      Substitute
        /g    Global
        /I    Case-insensitive
        /p    Print

  tr          Translates characters

    -d        Delete character
    -s        Squeeze runs of characters

  rev         Reverses characters on line
```

Format Contents:

```
  column  Aligns columns by content width

    -s    Specify field separator
    -t    Create table

  expand  Aligns columns to specified positions

    -t    Tab positions

  fold    Wraps lines at a specific width

    -w    Line width
    -s    Fold at spaces
```

Filter by Position:

```
cut     Removes parts of lines

  -c    Characters to keep
  -f    Fields to keep
  -d    Specify field separator

  -s    Suppress lines with no delimiters

head    Prints first lines

  -n    Number of lines

tail    Prints last lines

  -n    Number of lines
```

Miscellaneous:

```
wc        Counts words, lines, or characters

  -c      Characters
  -l      Lines
  -w      Words

xargs     Constructs arguments

  -n      Number of words per batch

mktemp    Make temporary file

join      Join columns in files by common field

paste     Merge columns in files by line number
```

File Compression:

```
tar     Archive files

  -c    Create archive
  -f    Name of output file
  -z    Compress archive with gzip

gzip    Compress file

  -k    Keep original file
  -9    Best compression

unzip   Decompress .zip archive

  -p    Pipe to stdout

gzcat   Decompress .gz archive and pipe to stdout
```

Directory and File Navigation:

```
cd      Changes directory

  /     Root
  ~     Home
  .     Current
```

```
  ..    Parent
  -     Previous

ls      Lists file names

  -1    One entry per line
  -a    Show files beginning with dot (.)
  -l    List in long format
  -R    Recursively explore subdirectories
  -S    Sort files by size
  -t    Sort by most recently modified
  .*    Current and parent directory

pwd     Prints working directory path
```

File Redirection:

```
<         Read stdin from file
>         Redirect stdout to file
>>        Append to file
2>        Redirect stderr
2>&1      Merge stderr into stdout
|         Pipe between programs
<(cmd)    Execute command, read results as file
```

Shell Script Variables:

```
$0      Name of script
$n      Nth argument
$#      Number of arguments
"$*"    Argument list as one argument
"$@"    Argument list as separate arguments
$?      Exit status of previous command
```

Shell Script Tests:

```
-d      Directory exists
-f      File exists
-s      File is not empty
-n      Length of string is non-zero
-x      File is executable
-z      Variable is empty or not set
```

Shell Script Options:

```
set       Set optional behaviors

  -e      Exit immediately upon error
  -u      Treat unset variables as error
  -x      Trace commands and argument
```

File and Directory Extraction:

```
        BAS=$(printf pubmed%03d $n)
        DIR=$(dirname "$0")
        FIL=$(basename "$0")
```

Remove Prefix:

```
          FILE="example.tar.gz"
  #       ${FILE#.*}  -> tar.gz
  ##      ${FILE##.*} -> gz
```

Remove Suffix:

```
          FILE="example.tar.gz"
          TYPE="http://identifiers.org/uniprot_enzymes/"
  %       ${FILE%.*}  -> example.tar
          ${TYPE%/}   -> http://identifiers.org/uniprot_enzymes
  %%      ${FILE%%.*} -> example
```

Loop Constructs:

```
          while IFS=$'\t' read ...
          for sym in HBB BRCA2 CFTR RAG1
          for col in "$@"
          for yr in {1960..2020}
          for i in $(seq $first $incr $last)
          for fl in *.xml.gz
```

Additional documentation with detailed explanations and examples can be obtained by typing "man" followed by a command name.

Release Notes

EDirect release notes describe the history of incremental development and refactoring, from the original implementation in Perl to the redesign in Go and shell script. The detailed notes have been moved to a separate document, which can be viewed by clicking on the RELEASE NOTES link.

For More Information

Announcement Mailing List

NCBI posts general announcements regarding the E-utilities to the utilities-announce announcement mailing list. This mailing list is an announcement list only; individual subscribers may **not** send mail to the list. Also, the list of subscribers is private and is not shared or used in any other way except for providing announcements to list members. The list receives about one posting per month. Please subscribe at the above link.

References

The Smithsonian Online Collections Databases are provided by the National Museum of Natural History, Smithsonian Institution, 10th and Constitution Ave. N.W., Washington, DC 20560-0193. https://collections.nmnh.si.edu/.

den Dunnen JT, Dalgleish R, Maglott DR, Hart RK, Greenblatt MS, McGowan-Jordan J, Roux AF, Smith T, Antonarakis SE, Taschner PE. HGVS Recommendations for the Description of Sequence Variants: 2016 Update. Hum Mutat. 2016. https://doi.org/10.1002/humu.22981. (PMID 26931183.)

Holmes JB, Moyer E, Phan L, Maglott D, Kattman B. SPDI: data model for variants and applications at NCBI. Bioinformatics. 2020. https://doi.org/10.1093/bioinformatics/btz856. (PMID 31738401.)

Hutchins BI, Baker KL, Davis MT, Diwersy MA, Haque E, Harriman RM, Hoppe TA, Leicht SA, Meyer P, Santangelo GM. The NIH Open Citation Collection: A public access, broad coverage resource. PLoS Biol. 2019. https://doi.org/10.1371/journal.pbio.3000385. (PMID 31600197.)

Kim S, Thiessen PA, Cheng T, Yu B, Bolton EE. An update on PUG-REST: RESTful interface for programmatic access to PubChem. Nucleic Acids Res. 2018. https://doi.org/10.1093/nar/gky294. (PMID 29718389.)

Mitchell JA, Aronson AR, Mork JG, Folk LC, Humphrey SM, Ward JM. Gene indexing: characterization and analysis of NLM's GeneRIFs. AMIA Annu Symp Proc. 2003:460-4. (PMID 14728215.)

Percha B, Altman RB. A global network of biomedical relationships derived from text. Bioinformatics. 2018. https://doi.org/10.1093/bioinformatics/bty114. (PMID 29490008.)

Schuler GD, Epstein JA, Ohkawa H, Kans JA. Entrez: molecular biology database and retrieval system. Methods Enzymol. 1996. https://doi.org/10.1016/s0076-6879(96)66012-1. (PMID 8743683.)

Wei C-H, Allot A, Leaman R, Lu Z. PubTator central: automated concept annotation for biomedical full text articles. Nucleic Acids Res. 2019. https://doi.org/10.1093/nar/gkz389. (PMID 31114887.)

Wu C, Macleod I, Su AI. BioGPS and MyGene.info: organizing online, gene-centric information. Nucleic Acids Res. 2013. https://doi.org/10.1093/nar/gks1114. (PMID 23175613.)

Documentation

EDirect navigation functions call the URL-based Entrez Programming Utilities:

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

NCBI database resources are described by:

https://www.ncbi.nlm.nih.gov/pubmed/34850941

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

An introduction to shell scripting for non-programmers is at:

https://missing.csail.mit.edu/2020/shell-tools/

Instructions for downloading and installing the Go compiler are at:

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

Additional NCBI website and data usage policy and disclaimer information is located at:

https://www.ncbi.nlm.nih.gov/home/about/policies/

Public Domain Notice

A copy of the NCBI Public Domain Notice, which applies to EDirect, is shown below:

```
Government do not and cannot warrant the performance or results that
may be obtained by using this software or data. The NLM and the U.S.
Government disclaim all warranties, express or implied, including
warranties of performance, merchantability or fitness for any particular
purpose.

Please cite the author in any work or product based on this material.
```

Getting Help

Please refer to the PubMed and Entrez help documents for more information about search queries, database indexing, field limitations and database content.

Suggestions, comments, and questions specifically relating to the EUtility programs may be sent to eutilities@ncbi.nlm.nih.gov.

Entrez Direct Release Notes

Jonathan Kans, PhD[1]

Created: April 23, 2013; Updated: November 23, 2022.

EDirect was conceived in 2012, prototyped in Perl, and released to the public in 2014.

Xtract was subsequently rewritten in the compiled Go programming language, for a hundredfold speed improvement on modern multi-processor computers. Transmute and rchive are also provided as platform-specific Go executables.

A major refactoring, for ease of maintenance and long-term stability, was completed in 2020. EDirect now exists as a set of Unix shell scripts plus the trio of utilities implemented in Go. The original Perl code has been retired.

2022

Version 18.2: November 23, 2022

- Code modified to support newly-deployed SOLR backend for PubMed database.
- Added -quick flag to esearch, elink, and efetch to override truncation limit workaround.
- Added -chunk argument to elink and efetch to control internal batch size.
- Nquire -len returns Content-Length of HTTP file without downloading.
- Xtract -prose removes HTML decorations and converts newlines to spaces in PMC paragraphs.
- Xtract -sort now handles unsigned real numbers with a decimal point.
- Archive-pubmed -nihocc adds CITED and CITES link fields, from NIH Open Citation Collection data.
- Phrase-search -link CITED or CITES looks up links for a set of PMIDs sent through stdin.
- Archive-pubmed -scrub now removes all -extras and -nihocc postings.

Version 18.1: October 21, 2022

- Archive-pubmed -extras GENE indexing adds PREF (preferred gene name), GSYN (gene synonym), and GRIF (gene reference-into-function) fields.
- Ref2pmid -options test uses citmatch service without requiring local archive.

Version 18.0: October 12, 2022

- Efetch -chunk overrides number of records to retrieve in each network request.
- Gbf2ref script can read EMBL/UniProt format in addition to GenBank/GenPept.
- Gbf2ref populates ATHR field with string of all authors in citation.
- Ref2pmid -options "verify" first determines if existing PMID in citation is correct.
- Ref2pmid -options "remote" calls citmatch service if local algorithm failed to find an unambiguous match.
- Use ref2pmid -options "strict,remote,verify" for best citation matching accuracy and performance.
- Edict server preload -file reads precomputed ref2pmid results, for use by nquire -edict match -citation.
- Xtract -jour calls CleanJournal to handle encoded ampersands and apostrophes.
- Download-ncbi-data -journals uses xtract -jour for better journal title cleanup.

Author Affiliation: 1 NCBI; Email: kans@ncbi.nlm.nih.gov.

Corresponding author.

Version 17.9: September 20, 2022

- Consolidated local archive cache maintenance functions.
- Moved first-level inverted index cache files out of Archive directory.
- Moved PubMed sentinel files to a new Sentinels directory.
- Above changes will make it feasible to have separate "live" and "next" copies of local archive files, eliminating nightly down-time if a 2 TB solid-state drive is available.

Version 17.8: September 6, 2022

- Refactored XML parser improves speed of "parent / star" processing.
- Restored proper archiving order for adjacent versions of the same publication.
- Adjusted code for downloading desc2022.xml MeSH descriptor file.

Version 17.7: August 29, 2022

- Citation matcher maps journal to standard name, checks for unresolvable ambiguity.
- Xtract -wrp delays reencoding if -replace, allowing -rep " & " instead of -rep " & ".

Version 17.6: August 15, 2022

- Recompiled with Go 1.19 for faster execution on all processors.
- Download-ncbi-data "journals" choice generates journal title lookup files with and without a leading article, allowing both "Journal of Immunology" and "The Journal of Immunology" to match the expected "J Immunol" record regardless of original cataloging.
- Similar ampersand expansion (of encoded " & " to " and ") is done for journal title lookup keys, but not for the local archive JOUR index, so that it matches the current term list convention in PubMed.
- Precomputed journal tables attempt to resolve ambiguous name or alias collisions by favoring journals that are currently indexed in MEDLINE. This removes two competing entries with "Journal of Immunology" aliases.
- Journal sets file includes multiple choices for 2671 abbreviations that cannot be resolved automatically, separated by vertical bars, so "dmj" maps to "Danish medical journal | Diabetes & metabolism journal".
- Edict.go server adds support for remote journal title queries.
- Local archive citation matcher caches most recent unique query results to avoid repeatedly recalculating the same reference on all components of a pop/phy/mut/eco set.
- Xtract -reg and -exp, regular expression patterns for -replace, now protected by mutex.
- Xtract -aliases reads mapping file to support -classify argument, which uses multiple whole word or phrase substring matching to populate custom Entrez indices.
- Deprecated index-pubmed script now just calls archive-pubmed -index.
- Custom-index runs collect and expand steps, previously performed by index-pubmed.
- Added idx-affil custom indexing helper script for affiliation experiments.
- Added idx-journals to index Journal/Title in JRNL field, skipping the ISOAbbreviation, ISSN, ISSNLinking, MedlineTA, and NlmUniqueID elements also indexed in JOUR field.
- Transmute -gbf uses file of accessions for filtering a GenBank flatfile stream.
- Speed of the (rate-limiting) -gbf flatfile partitioning step (using an uncompressed GenBank release file on a dedicated machine) was just under 40,000 records per second. This is sufficient to support the possibility of applying the local PubMed archive approach to the orders-of-magnitude larger set of sequence records.
- Fixed bug in RepairUnicodeMarkup that produced unbalanced symbols when runs of Unicode superscripts or subscripts were immediately followed by another type of non-ASCII character, such as a Greek letter.

Version 17.5: August 2, 2022

- Added edict.go remote server for local archive (RESTful equivalent of phrase-search commands), with support for search, fetch, (compressed) stream, and (citation) match operations. The -host and -port arguments override the default "localhost:8080" address used for testing.
- Nquire -edict is a shortcut for access to a running edict server, defaulting to "localhost:8080".
- Set the NQUIRE_EDICT_SERVER environment variable to override the nquire -edict address.
- Nquire -preview is a shortcut for the upcoming PubMed SOLR server.
- Nquire -get and -url send explicit curl -X GET and -X POST arguments.
- Archive-pubmed -extras flag indexes chemical, disease, and gene references (extracted from article contents by NCBI text mining and NLM indexing groups) as CHEM, DISZ, and GENE fields. This supersedes the index-extras script, and only runs if any relevant data file is out of date or not yet downloaded.
- Archive-pubmed writes two independently-compressed blocks (xml+DOCTYPE header and PubmedArticle XML record data) to each file. When streaming sets of compressed files for efficient network transfer, the headers before each record are skipped by advancing a fixed number of bytes.
- New asn2ref script helps with citation matching from Seq-entry ASN.1 records.
- Elink internal chunk size lowered to 400 to avoid server timeouts.
- Test-eutils -preview runs the basic -alive tests on the SOLR server.

Version 17.4: July 18, 2022

- Efilter handles <Query> or <Id> items in the ENTREZ_DIRECT message.
- Archive-pubmed second-level inverted index cache moved to Increment folder.
- Downloading each PubMed ftp release file validates contents, retries on failure.
- Normalization removes combining accents in author affiliation field.
- Remaining functionality needed for full EDirect support of PubMed SOLR server includes combining queries (in esearch) and specifying a range of output records (in efetch).

Version 17.3: June 29, 2022

- Esearch -title queries individual words with [TITL] to bypass automatic term mapping.
- Elink supports <Query> or <Id> items in the ENTREZ_DIRECT message.
- Local index restores PAIR field to accelerate inexact citation matching.
- Removed TLEN and TNUM fields from local index.
- Local archive adds xml and DOCTYPE lines to every PubmedArticle record. This will allow biopython's Bio.Entrez subpackage to use the archive files directly.
- Fetch-pubmed removes the xml and DOCTYPE prefix from individual PubmedArticle records. A single pair precedes the <PubmedArticleSet> wrapper.
- Fetch-pubmed -turbo places <NEXT_RECORD_SIZE> objects in front of each XML record, to allow faster XML data extraction with xtract -turbo.
- Xtract -mirror reverses the characters in a string.

Version 17.2: June 13, 2022

- Esearch passes a <Query> field in the ENTREZ_DIRECT message when using the upcoming PubMed SOLR backend. This allows efetch to circumvent the 10,000 PMID-per-query limit by repetitive searching with a sliding window of dynamically-resized create date ranges.
- Efetch temporarily calls "transmute -mixed -normalize pubmed" twice, as a quick fix to compensate for the SOLR server's unexpected encoding of non-ASCII characters with the "&#x...;" construct.
- Internal esearch -count and -uids added to access PubMed SOLR without using history.

- Esearch -translate and -components now handle SOLR output variant.
- Local indexing adds bad date, future date, medline date, has abstract, and versioned to the PROP field.
- Local JOUR field now indexes MedlineTA, NlmUniqueID, and ISSNLinking values.
- Local YEAR field standardizes on 4 directory levels for all dates (e.g., /1/8/9/2/), which includes 104,824 citations from the eighteenth and nineteenth centuries.
- Run "archive-pubmed -clear -index" to refresh the inverted index cache after changes to indexing code.
- Phrase-search -totals prints the term list with document counts for the given field.
- Improved local citation matcher parsing of "in press" journal references.

Version 17.1: May 16, 2022

- Esearch -translate and -components, instead of -query, return the full query translation or the individual translation components, respectively. For -db pubmed they show the automatic term mapping expansions.
- Archive-pubmed adds a second level of caching for faster search index rebuilding. Use -clean to remove Inverted folder intermediate files, or -clear to also remove Archive inverted index cache files.
- Improved author name normalization and indexing of apostrophes and combining accents.
- New cit2pmid script calls "https://pubmed.ncbi.nlm.nih.gov/api/citmatch" service, or with "-local" flag performs citation matching using EDirect local archive and search system.
- Added ncbi::edirect:Execute function to control EDirect from NCBI C++ toolkit programs.

Version 17.0: April 18, 2022

- Removed STEM field (Porter2 algorithm) from standard local indexing.
- Run rchive -e2delete "${EDIRECT_PUBMED_MASTER}/Archive" to clear the incremental cache. This is needed to remove residual STEM postings.
- Next execution of archive-pubmed -daily will reinitialize the pre-indexed cache in under 90 minutes. Future daily updates should take around 3 minutes.
- Archive-pubmed -index can subsequently rebuild local search indices on demand in under 2 hours.
- Xtract adds -stemmed argument to index Porter2-processed sentences.
- Use custom-index $(which idx-stemmed) STEM to manually restore stemmed index.

Version 16.9: April 7, 2022

- Recompiled with Go 1.18, which should execute faster on ARM and Apple Silicon processors.
- Nquire -pubchem, previously a legacy alias for -pugrest, is now repurposed for more convenient access to PubChem Pathways, which replaces the retired Entrez BioSystems database.
- Xtract automatic detection of JSON input accidentally conflated the default record names for top-level objects and arrays. Now it uses the same policy as transmute -j2x ("opt" and "anon", respectively).
- Local search index adds PROP field for publication types.
- Using parallel pgzip (de)compression library for faster search index construction.
- Archive-pubmed -daily keeps an incremental cache of pre-indexed files up to date.
- Archive-pubmed -index uses the cache to repopulate all local retrieval system files in under 3 hours, superseding the slower index-pubmed process.

Version 16.8: March 17, 2022

- Phrase-search supports MESH queries by mapping to indexed TREE or CODE fields.
- Phrase-search reads an expandable file of JOUR field aliases (e.g., PNAS).
- Xtract -pairx extends -pairs to include isolated single words.
- Idx-pairs helper for custom-index reintroduces modified non-positional title word PAIR index.
- EDIRECT_PREVIEW environment variable allows testing of the upcoming PubMed API.

- Elink stores identifiers in ENTREZ_DIRECT message instead of history with new PubMed server.
- Efilter accepts ENTREZ_DIRECT instantiated identifiers when using the new PubMed API.

Version 16.7: March 1, 2022

- Added ds2pme script to convert PubMed DocumentSummary XML to Pubmed-entry ASN.1.
- Phrase-search adds -words (replacing -partial) and -pairs, for local index citation matching test.
- Filter-stop-words script adds optional -plus argument to support phrase-search -pairs.
- Index-pubmed script adds TLEN and TNUM indices, phrase-search can query those fields by range.
- Xtract -trim removes leading and trailing spaces, and leading zeros.
- Xtract -wct counts the number of words in a string, obeying -stops and -stems modifiers.
- Transmute -g2r (gbf2ref) tracks previously-encountered titles and authors, suppressing duplicate citations, which allows xtract -select STAT to create non-redundant inpress or unpub subsets.
- Transmute -r2p (and ref2pmid shortcut) reads gbf2ref subsets for local citation matching.

Version 16.6: February 14, 2022

- Nquire -pugwait polls asynchronous PubChem PUG-REST searches, returning ENTREZ_DIRECT structure with instantiated compound identifiers. (Support for this form was included in the 2020 EDirect redesign.)
- Nquire -timer prints milliseconds between initial service request and completion of network data transfer.
- Xtract warns that capitalized exploration arguments (e.g., -Block), which were needed for recursive data in the original (retired) Perl implementation, are now deprecated, and will be removed in the near future.
- Added pma2pme script to convert PubmedArticle XML to Pubmed-entry ASN.1.
- Archive-pubmed script takes optional -asn argument to save Pubmed-entry ASN.1 files in the local archive, coexisting with the PubmedArticle XML records. Use fetch-pubmed -asn to retrieve.
- Index-pubmed script now creates indices for author (AUTH, ANUM, FAUT, LAUT, CSRT, INVR) and citation (JOUR, VOL, ISS, PAGE, LANG) fields, eliminating the need for a separate custom-index step. To be used with TITL, TIAB, and YEAR fields for local citation matching experiments with phrase-search script.
- Xtract modifies -plain (removal of mixed-content sections) and adds -simple (normalization of accented letters), both derived from -basic (cleanup of superscripts and subscripts).
- Xtract repurposes -author and adds -prose, to correct commonly misused lookalike characters (sharp S and lower-case beta) and remove accents and markup, for use in generating search indices and ASN.1.
- Xtract -auth replaces commas, periods, and hyphens to allow queries with GenBank reference authors.
- Xtract -month "PubDate/*" finds first month name or abbreviation, returning a number from 1 to 12.
- Xtract -page extracts first page (digits and letters) from a page range.
- Xtract adds -hex (hexadecimal), -oct (octal), and -acc (running total accumulator) numeric arguments.
- Xtract -element "." (period) generates text ASN.1 from customized XML records. Underscores in XML tag names show unlabeled braces around SEQUENCE OF components (single), unquoted INTEGER or ENUMERATED values (trailing), or alternative CHOICE selections (internal).
- Xtract -element "%" (percent) generates JSON from customized XML records. Underscores in XML tag names show unlabeled brackets (single), named arrays in brackets (leading), or unquoted values (trailing).
- Transmute -g2r (and gbf2ref shortcut) extract reference fields from sequence flatfiles for citation matching.
- Phrase-search -partial queries title words individually and combines results for ranked matches.
- EDirect now uses a custom internal Unicode-to-ASCII conversion map.

Version 16.5: January 3, 2022

- Using efetch.fcgi instead of esearch.fcgi to retrieve UID lists, in preparation for new PubMed API.
- Nquire adds initial implementation of -puglist and -pugwait helper functions for PubChem.
- Xtract -verify raises maxDepth limit to 30 to avoid warning about depth of PMC XML records.
- Rchive -invert uses a separate map for each initial character, now runs in 1/3 less time.
- Added indexing of INVR investigators to idx-authors helper script.

2021

Version 16.4: December 20, 2021

- Nquire -pugrest -inchi silently adds "InChI=" prefix if it is missing in the argument value.
- Xtract -year "PubDate/*" construct, broken in recent refactoring, again returns a single 4-digit year.
- Separate EDIRECT_PUBMED_MASTER and EDIRECT_PUBMED_WORKING environment variables also apply to index-extras and custom-index scripts.

Version 16.3: December 15, 2021

- Esummary -format docsum has a clearer "redundant argument" message.
- Efilter -db assembly adds -status shortcut.
- Nquire adds -pugrest and -pugview shortcuts for PubChem Power User Gateway services.
- Nquire allows Windows version of curl to recognize Cygwin paths.
- Xtract supports multiple -insd feature clauses for output on a single line.
- Xtract -insd source taxid qualifier extracts integer from "taxon:###" db_xref.
- Xtract -accent no longer needs -strict or -mixed processing flags.
- Transmute -plain removes accents and diacritical marks from text.
- Local archive scripts use 2022 PubMed release files and 2022 MeSH data files.
- Local archive Extras directory added to store original MeSH data files and NLP downloads. Separate environment variables (EDIRECT_PUBMED_MASTER and EDIRECT_PUBMED_WORKING) can place Archive, Data, and Postings folders on the computer's internal drive, while keeping release files and indexing intermediates on the external SSD.
- Renamed alternative edirect-install.sh script to update-edirect.sh.

Version 16.2: October 28, 2021

- Xtract automatically detects and processes JSON, text ASN.1, and GenBank/GenPept formats. An explicit transmute command is only needed if you wish to inspect the intermediate XML, or to override the default conversion arguments.
- Transmute -j2x -nest "element" choice adds "_E" suffixes to multi-dimensional array components. This is now the default for both transmute and the xtract JSON converter, assigning a distinct tag name to each level.
- Transmute -g2x parses the DBLINK section to capture BioProject and BioSample identifiers.
- Using printf instead of echo to generate configuration commands in the install-edirect.sh script.
- Efetch -db pubmed -format uid retrieves PMIDs in chunks of 10,000 in preparation for new PubMed API.
- Added idx-metadata script for indexing JOUR, LANG, MAJR, MESH, and SUBH fields in the local archive.
- Added edirect.py module file for import by Python programs. Use edirect.execute to run individual EDirect commands, Unix tools, or shell scripts, controlling multiple steps by passing the previous result to

the next command. Or use edirect.pipeline to launch a chain of several commands contained in a single argument. An edirect.efetch shortcut that takes named arguments is also provided.

Version 16.1: October 13, 2021

- Efetch -strand argument can accept "\+" or "\-". The leading backslash is required.
- Efetch -express flag works like -immediate, but on several sequence records at a time.
- Nucleotide accession lookup is only needed for master sequences.
- Epost looks up all nucleotide accessions to silently skip replaced records.
- Xtract -self applies a default value to allow detection of empty self-closing tags.
- Xtract uses both time interval and record count to trigger an output buffer flush.
- Minor refactoring of xtract argument parsing code.
- Removed obsolete setup.sh script.
- Updated cacert.pem file.

Version 16.0: October 4, 2021

- Xtract -insd feat_intervals is a version of feat_location that generates 0-based positions.
- Transmute -extract accepts -0-based and -1-based modifiers, defaulting to the 1-based feat_location form.
- The eutils.SequenceExtract library function has a new isOneBased boolean argument.
- Added fuse-segments to the set of scripts for post-processing magicblast -outfmt asn.
- Renamed uniq-columns script to uniq-table.

Version 15.9: September 27, 2021

- Redesigned install-edirect.sh script, which no longer calls setup.sh.
- Alternative edirect-install.sh script does not offer to edit configuration files.
- Installation prints PATH command to use for current terminal session.
- Successfully installed and ran test-eutils and test-edirect on Cloud.
- Minor refactoring of xtract -element variant code.
- Modified snp2hgvs script to remove records where the Id and SNP_ID do not match.

Version 15.8: September 16, 2021

- Efetch -db clinvar leaves VCV prefix for -format vcv.
- Added gene column to spdi2tbl component of snp2tbl.
- Renamed spdi2prod script to tbl2prod, now reads output of snp2tbl.
- Xtract -hgvs allows R and Y nucleotide ambiguity characters.
- Xtract -author replaces commas and periods in GenBank reference authors.
- Refactored eutils.FASTAConverter into tokenizer and streamer components.
- Added uniq-columns script.

Version 15.7: September 9, 2021

- Efetch -immediate flag avoids memory overflow on sets of extremely large sequences.
- Efetch -format fasta chunk size reduced to 50 records at a time.
- Xtract -backward presents elements in reverse order.
- Modified spdi2prod script to print "wild-type" protein and CDS translation.
- Renamed filter-table script to filter-columns.
- Moved all help text to a new help subfolder.
- Recompiled with Go 1.17, which generates smaller binaries that execute faster.

Version 15.6: September 1, 2021

- Added datasets and sra-toolkit choices to download-ncbi-software script.
- Added efilter -keyword and -purpose shortcuts for "purposeofsampling" keywords.
- Restored missing efilter -pub and -released shortcuts.
- Xtract -ncbi2na and -ncbi4na functionality available in common eutils library.
- Added vendor option to Go build.sh scripts to cache source code for all external library dependencies.

Version 15.5: August 16, 2021

- Documentation introduces Go compiled language, dependency management with modules, and EDirect's local eutils library package.
- Now including go.mod and go.sum module files for eutils and cmd project directories.
- Transmute -counts implements Go base count example.
- Added xml2fsa script for converting INSDSeq XML to FASTA.
- Xtract -fasta, used by xml2fsa script, now generates conventional 70 uppercase characters per line.
- Xtract -element "~" for object contents joins "?" for object name.
- Added download-ncbi-software script, initially for magic-blast on linux, macosx, and win64 platforms.
- Added blst2tkns, split-at-intron, fuse-ranges, and find-in-gene scripts for merging overlapping reference coordinate matches from magicblast -outfmt asn.
- Nucleotide accession lookups may use [PACC] or [ACCN] fields.
- Nquire supports NQUIRE_IPV4 environment variable for diagnosing suspected IPv6 problem.

Version 15.4: July 16, 2021

- Added efilter -source select as shortcut for RefSeq Select dataset.
- Efetch -id removes PMC prefix in front of PubMed Central identifiers.
- Transmute -align -a argument adds m choice for using commas to separate integers into groups of 3 digits.
- Added transmute -align -w argument to specify minimum width for all columns.
- Improved scripting introduction in automation section of documentation.

Version 15.3: June 21, 2021

- Transmute -separate replaced by -combine, default is not to remove internal top-level objects, which result from retrieving large sets of data in smaller chunks.
- Added transmute -search for finding positions of patterns in sequence data (e.g., restriction sites), -circular flag allows match to pattern spanning origin of circular molecule.
- Added disambiguate-nucleotides script to expand degenerate base letters for restriction enzymes like AccI.
- Added transmute -find for searching non-sequence text that includes digits, spaces, and punctuation, -relaxed flag ignores spacing differences, punctuation.
- Added idx-words helper script to split words at punctuation for indexing [WORD] field.
- Local indexing speed improvements.
- Removed deprecated rchive -phrase function, not needed with positional indices.

Version 15.2: June 3, 2021

- Expand-current prepares decompressed nonredundant PubMed archives with xtract -index, which places <NEXT_RECORD_SIZE> objects in front of each XML record.
- Xtract -turbo uses precomputed <NEXT_RECORD_SIZE> information to double the speed of the (rate-limiting) partitioning step, allowing additional CPU cores to participate in XML data extraction.

- Added custom-index, which calls a user-supplied helper script to build an initial PMID-term index, and then completes the inversion and posting steps to integrate the new field into the local search system.
- Added idx-author sample helper script, with boilerplate xtract instructions for generating an IdxDocument set, and specific commands for populating a novel [ANUM] index.
- Rchive -invert converts input strings to lower-case if that was not already done in the indexing step.
- Added xtract -includes, like -contains but requiring the substring match to align on word boundaries.
- Index-pubmed adds a new [TITL] field, which indexes only the article title.
- Phrase-search -title performs an exact search on the local [TITL] field.
- Index-pubmed replaces the [NORM] field with [TIAB], the conventional code for title and abstract. Legacy queries using [NORM] will be internally redirected to [TIAB].
- README and readme.pdf files are more concise, with several sections now residing only in the more detailed web documentation.

Version 15.1: May 20, 2021

- Xtract and transmit now run as native executables on Apple Silicon machines.
- Setup.sh script can add edirect folder to PATH in both .bash_profile and .zshrc for MacOS.
- Documented efetch and elink reading large lists of identifiers from stdin or file, bypassing need for epost.
- Nquire supports -ncbi, -eutils, and -pubchem URL shortcuts.
- Nquire uses --cacert instead of --capath in curl command.
- Efetch -db snp uses -format -self flag to keep self-closing attribute-free PAIRED or SINGLE items in XML.
- SNP processing adds Gene, Hgvs, and Spdi fields, and (when different from Id) adds OldId field.
- Added transmute -codons to display nucleotide codons above protein residues.
- Transmute -a2x, -c2x, -g2x, -j2x, and -t2x have shortcut scripts asn2xml, csv2xml, gbf2xml, json2xml, and tbl2xml, respectively.

Version 15.0: April 29, 2021

- Removed edirect.pl and setup-deps.pl scripts containing original Perl implementation.
- Deprecated -oldmode and -newmode arguments, and USE_NEW_EDIRECT environment variable.
- Simplified -hgvs output structure, and converted to 0-based positions for SPDI processing.
- Added snp2hgvs script as shortcut for -hgvs extraction from SNP docsum.
- Added hgvs2spdi script to adjust CDS-relative -hgvs offsets into sequence-relative positions suitable for use by transmute -replace.
- Added spdi2prod script to pair modified NM translation with modified NP sequence.
- Added snp2tbl script to produce tab-delimited table of adjusted SNP values in one step.
- Transmute -replace and -extract use -lower to specify lower-case output.
- Transmute -g2x writes INSDAltSeqItem_first-accn and INSDAltSeqItem_last-accn objects for WGS master.
- Xtract -pkg and -enc XML generators accept multiple slash-delimited object names.

Version 14.9: April 15, 2021

- EDirect runtime errors display an "ERROR:" tag in inverse bold red text on the terminal.
- Efetch -id now works on WGS master accessions in nucleotide databases.
- Lookup of rs/ss numbers supported in -id argument for -db snp.
- Epost uses chunks of 1000 to avoid server truncation.
- Transmute -hgvs parsing now supports ".g" genomic and ".c" coding sequences, in addition to ".p" proteins. The ".c" item location uses <Offset> rather than <Position>, since it is relative to the first base of

the CDS initiation codon, not the sequence. Additional operations will be required to adjust this value for SPDI processing.
- The next release will retire the edirect.pl and setup-deps.pl scripts, and will ignore the old/new mode arguments and environment variable.

Version 14.8: March 24, 2021

- Esearch has improved logic to recognize [FILT], [PROP], and [ORGN] controlled-vocabulary phrases without the need for extra quotation marks to protect the query.
- Elink runs -cmd acheck to confirm empty history result, can detect stale links to deleted records.
- Elink writes structured message on empty history input to avoid breaking pipeline of multiple link operations.
- Nquire adds -dir command to show ftp directory listing with column of file sizes.
- Blank lines are ignored by sort-uniq-count, sort-uniq-count-rank, and sort-table scripts.
- Removed eblast script after URL-based BLAST service was disabled.
- Use of edirect.pl script (through -oldmode command-line argument or USE_NEW_EDIRECT=false environment variable setting) prints DEPRECATED warning message.

Version 14.7: March 8, 2021

- Xtract -insd handles qualifiers without values (e.g., pseudo, transgenic).
- Xtract -insd adds feat_location qualifier to print feature intervals.
- Transmute -extract reads xtract -insd feat_location interval format.
- Transmute -remove -first and -last arguments can use sequence letters instead of count.
- Efetch -format ipg processes potentially large records one at a time.
- Error message printed if non-integer -id argument is used with -db taxonomy.
- Nquire -dwn and -asp file download failure report improved.
- Refactored common.go and transmute.go functions into reusable local "eutils" package.
- Go compiler uses "replace eutils => ../eutils" line in go.mod file to import from local eutils package.
- Transmute -degenerate > gdata.go recreates library file with updated genetic code mapping tables.
- Removed obsolete and deprecated scripts.
- Added sort-table, filter-table, and print-columns scripts.
- Revised eblast script as prelude for running on cloud.

Version 14.6: February 3, 2021

- Transmute -cds2prot added to translate coding regions with nucleotide substitutions.
- Transmute -revcomp and -molwt added, sharing code with xtract functions.
- Transmute -remove, -retain, and -replace allow script-driven editing of sequences and insertion of SNP bases, with argument values obtained by -hgvs parsing of HGVS data.
- Transmute -diff simplifies visualization of sequence point mutations.
- Esearch and efilter enforce restriction of -country shortcut to sequence databases.

Version 14.5: January 19, 2021

- Added -hgvs support for genomic single nucleotide substitutions.
- Sort by RefSeq accession numbers within categories of -hgvs output.
- Updated efilter -db snp -class frameshift mapping to FXN entry.
- Updated test-eutils upon retirement of Entrez sparcle database.
- Updated Amino Acid Substitutions example.
- Added Reference Formatting example.

- Xtract -replace uses -reg and -exp values for regular expression substitution.

Version 14.4: January 13, 2021

- HGVS format parsed into XML by xtract and transmute -hgvs commands. Initial implementation supports amino acid missense and nonsense variations. More coding to follow.
- Transmute -align -a argument adds z choice for padding numbers with leading zeros.
- Fixed bug in efetch -start and -stop subrange arguments.
- Restored trailing newline lost in switch from echo to printf.

Version 14.3: January 7, 2021

- Major overhaul of EDirect documentation completed.
- Release notes and additional examples moved to separate web pages.
- Xtract -doi cleans DOI data and generates complete URL.
- Transmute -align -a argument adds n and N choices for aligning to decimal point.
- Added align-columns script as preferred front-end to transmute -align.
- Transmute -j2p works on a concatenated stream of JSON records.
- Transmute -aa1to3 and -aa3to1 amino acid abbreviation converters added for HGVS processing.
- Improved lookup of accessions in -id argument, now includes 21 Entrez databases with 10 fields.
- Improved code that maps PDB protein accessions with case-sensitive chain letters.
- Efetch -db bioproject no longer converts formats in order to handle accession input.
- Efetch -format fasta adds newline if missing before angle bracket.
- Uses printf "%s" instead of echo to avoid misinterpreting backslash in retrieved records.
- Nquire keeps error messages from leaking into output file.
- Confirmed that EDirect will run on Apple Silicon under Rosetta translation environment.

2020

Version 14.2: December 14, 2020

- Transmute -normalize now handles unexpected DocumentSummary attributes.
- Transmute -t2x -heading takes tags from columns in first row of file.
- Transmute -align converts a tab-delimited table to columns padded with spaces.
- Nquire -raw does minimal URL encoding for GeneOntology query.
- Efetch -db bioproject maps -format docsum to -format xml.
- Download-ncbi-data script updated to use desc2021.xml and supp20201.xml for mesh lookup table.

Version 14.1: November 30, 2020

- Nquire always uses new implementation.
- Scripts updated to use new nquire features.
- Efetch and esummary warn about -db mismatches and collisions between -format choices.
- Efetch supports PDB accessions with case-sensitive chain letters in the -id argument for -db protein.
- Efetch supports GCA and GCF accessions in the -id argument for -db assembly.
- Esearch -mindate and -maxdate can be used alone, with defaults filling in the missing argument.
- Xtract -set, -rec, -pkg, and -enc shortcuts join -wrp for creating XML objects at appropriate levels during XML generation.
- Xtract -head can read separate arguments for individual column headings.
- Xtract uses double-hyphen to append value into variable.

Version 14.0: November 12, 2020

- Redesigned EDirect active by default, deselected by running "export USE_NEW_EDIRECT=false" to set environment variable.
- Old implementation also chosen by adding -oldmode as the first argument to individual efetch, efilter, einfo, elink, epost, esearch, esummary, and nquire commands.
- Automatic retry supported on empty result for all formats, not just structured data.
- Esearch protects [PROP], [FILT], and [ORGN] queries in biological databases from parsing artifact on controlled vocabulary entries with embedded "or" and "not".
- Efetch adds -showgaps flag.
- Xtract -wrp causes angle brackets, ampersands, quotation marks, and apostrophes to be reencoded in the new XML, without the need for explicitly using -encode.
- Xtract formatting, modification, normalization, and conversion functions moved to transmute, with old calls automatically redirected to avoid breaking user scripts.
- Transmute -format takes optional -comment and -cdata flags to keep XML comments and CDATA blocks.
- Transmute -a2x converts text ASN.1 data to XML.
- Xtract -ncbi2na and -ncbi4na decompress nucleotide sequence converted from text ASN.1 hex representation.

Version 13.9: September 14, 2020

- Xtract -fasta splits long sequences into groups of 50 letters.
- Xtract -select restores -in to indicate name of identifier file.
- Redesigned EDirect selected by running "export USE_NEW_EDIRECT=true" to set environment variable.
- New implementation also chosen by adding -newmode as the first argument to individual efetch, efilter, einfo, elink, epost, esearch, esummary, and nquire commands.

Version 13.8: August 27, 2020

- Deprecated econtact and eproxy utilities.
- Deprecated -alias argument.
- Added accn-at-a-time and skip-if-file-exists scripts.
- Xtract -g2x converts GenBank and GenPept to INSDSeq XML.
- Xtract -c2x is a CSV (comma-separated values) variant of -t2x.

Version 13.7: May 28, 2020

- Removed xtract -repair shortcut for -unicode conversion.
- Xtract -decode supports direct Base64 decoding.
- Xtract -normalize added for Efetch post-processing.

Version 13.6: April 17, 2020

- Efilter -db snp -class shortcuts mappings updated to reflect changes to FXN index.
- Xtract adds -matches and -resembles string conditional tests, -order string processing command.
- Minor change to term granularity for MeSH [TREE] index creation.

Version 13.5: February 21, 2020

- Xtract -select -streaming uses case-insensitive test, concurrent processing.
- Minor change to term granularity for reenabled [CODE] index creation.

Version 13.4: February 4, 2020

- Local [CONV] term list indexes GNBR theme plus relationship plus identifier pair.
- Theme indexing splits genes at semicolons, adds M prefix to OMIM, H to ChEBI identifiers.
- Run "source theme-aliases" to load commands for navigating theme identifier connections.

Version 13.3: January 28, 2020

- Index-extras downloads newest version of Global Network of Biomedical Relationships theme data.
- Xtract -contour replaces -synopsis -leaf variant.

Version 13.2: January 7, 2020

- Expand-current can be run after archive-pubmed as well as index-pubmed.

2019

Version 13.1: December 30, 2019

- Local index only builds TREE fields for A, C, D, E, F, G, and Z categories.
- Stop word list now includes "studies" and "study" in addition to "studied".
- Xtract -select driver file command names changed to -retaining, -excluding, and -appending.
- New efetch -format types documented for -db clinvar.
- Updated MeSH tree category descriptions in phrase-search -help.

Version 13.0: December 16, 2019

- Xtract -histogram collects data for sort-uniq-count on entire set of records.
- Xtract -wrp with empty string or dash resets -sep, -pfx, -sfx, -plg, and -elg customization values.
- Xtract -fwd and -awd print once before and once after a set of object instances.
- Xtract -t2x uses asterisk before column name to indicate XML contents, will not escape angle brackets in string.

Version 12.9: December 9, 2019

- Xtract -plain, an -element variant that removes embedded mixed-content markup, now also converts Unicode subscripts/superscripts, cleans bad spaces.
- Xtract -select -in changed to -select -using.
- Xtract -select -adding matches by identifier and appends XML metadata.
- Xtract -select -merging requires original records and identifier-metadata file to be in same order.
- Moved xtract -examples text to separate hlp-xtract.txt file.
- Efetch -format asn maps to -format asn.1.
- Disabled building of [CODE] field in local index.
- Minor changes to term granularity for faster local index creation.
- Index-pubmed takes optional [-collect | -index | -invert | -merge | -promote] argument to resume processing in an intermediate step.

Version 12.8: December 3, 2019

- Xtract -molwt sequence processing command added.
- Xtract -len triggers -def instead of returning 0 for missing object.
- Xtract -pairs does not drop the first component of hyphenated terms (e.g., site-specific).

- Xtract -synopsis with optional -leaf argument only reports content nodes.
- Adjusted test-eutils -esummary dbvar test.
- Local archive asks user to check TRIM status on slow update.
- Local archive checks that volume is actually a solid-state drive.

Version 12.7: November 21, 2019

- Local archive detects slow performance, reminds user about antivirus scanning and content indexing.
- Local archive warns if Mac file system is not APFS, prints instructions on how to reformat drive.
- Xtract -examples adds namespace prefix and Base64 decoding example.
- Efetch -db nucleotide/nuccore -style withparts/conwithfeat processes -id accession list one record at a time.

Version 12.6: November 15, 2019

- Added support for automatic fallback to IPv4.
- Xtract -insd adds mol_wt as a synonym for calculated_mol_wt.
- Changed granularity of local index for [YEAR] field.
- If archiving is slow, ask user to ensure that antivirus scanning and content indexing are disabled.

Version 12.5: November 6, 2019

- Efetch -start and -stop subset retrieval arguments are now documented.
- Efetch -format docsum rescues uid attribute if no existing <Id> tag, now using case-sensitive test to prevent <id> from blocking.
- Added efetch -revcomp flag, sets -strand 2.
- Einfo sorts -fields and -links results by tag name.
- Fixed stdin-vs-argument bug in ftp-cp introduced when moving code into edirect.pl for Anaconda issue.
- Local search system [THME] field also indexes disambiguated themes - Jc (chemical-disease) and Jg (gene-disease) - under original code J (role in disease pathogenesis).

Version 12.4: October 28, 2019

- Expanded README file to cover more advanced features of potential interest to codeathon participants.
- Elink -released also accepts four-digit year.
- Local search system builds separate [CODE] and [TREE] indices for MeSH code and hierarchy values.

Version 12.3: October 23, 2019

- Added single-line copy-and-execute commands, in curl and wget flavors, as convenient options for EDirect installation.
- Elink -db pubmed supports -cited and -cites to follow reference connections from the NIH Open Citation Collection dataset.
- Added missing retry code to esearch and esummary.
- Xtract -t2x converts tab-delimited table to XML.
- Xtract -sort rearranges records by designated identifier.
- Xtract -split breaks up a large XML stream into multiple files.
- Xtract -chain changes_spaces_to_underscores.
- Efetch -db gtr -format docsum -mode json downloads in smaller groups to avoid timeouts.
- Nquire -get does not need -url command if followed immediately by URL argument.
- Consolidated code for perl-based commands (e.g., nquire, transmute) into master edirect.pl script.
- Wrappers to all perl-based commands now handle conflict with Anaconda installation.

- Moved external resource indexing code from xtract to rchive.
- Added phrase-search -filter command to pipe efetch -format uid results into local query.
- Download-ncbi-data script updated to use desc2020.xml and supp2020.xml for mesh lookup table.
- Experimental index-extras script loads natural language processing results into local retrieval system.
- Experimental fetch-extras scripts retrieves indexed NLP fields per PMID saved in local archive.

Version 12.2: September 27, 2019

- Added install-edirect.sh script, with download link in web documentation, for easier installation.
- Updated setup.sh script to ask permission to edit the PATH setting in the user's .bash_profile file.
- Xtract -is-before and -is-after tests compare order of strings.
- Xtract -mul, -div, and -mod numeric processing commands added.

Version 12.1: August 29, 2019

- Local query index now creates field-specific subdirectories immediately below Postings folder.
- Term position index files, needed for phrase and proximity searching in [NORM] and [STEM] fields, not made for [CODE] and [YEAR].
- Phrase-search -terms [field] prints complete term list for given field.
- Rchive -help gives example of how to sequentially ascend the MeSH [CODE] hierarchy index.
- Added expand-current script, to be run after index-pubmed, to prepare for fast scanning of all PubMed records.
- Added -repeat option to test-eutils monitoring script.

Version 12.0: August 14, 2019

- Efetch adds -format bioc for -db pubmed and -db pmc, retrieving annotated records from PubTator Central.
- Added download-ncbi-data script to consolidate and replace special-case scripts.

Version 11.9: August 5, 2019

- Updated test-eutils driver files due to retirement of unigene.
- Test-eutils progress line prints a period for success, x for failure.
- Test-eutils -timer prints response times in milliseconds for each query.
- Added exclude-uid-lists script.
- Added download-pmc-bioc script.
- Nquire supports -bioc-pubmed and -bioc-pmc shortcuts.
- Fetch-pubmed -fresh generates uncompressed PubMed files from local archive for fast data extraction.
- Index-pubmed retains compressed PubMed intermediate files for batch scanning.
- Xtract supports -select parent/element@attribute^version -in file_of_identifiers.

Version 11.8: July 23, 2019

- Xtract -j2x converts newline to space, compresses runs of spaces.
- Xtract -j2x supports JSON null tags.

Version 11.7: July 3, 2019

- Efetch -format docsum -mode json reads 500 records at a time, in conformance with the server limit.
- Fetch-pubmed -all sequentially streams all live records from the local cache.
- Xtract -plain removes embedded mixed-content markup tags.

- Added download-pmc-oa to fetch the open-access subset of PubMed Central.

Version 11.6: June 11, 2019

- Xtract -path implementation was simplified.
- Einfo -db all returns a combined eInfoResult XML, containing field and link names, and record and term counts, for all Entrez databases in one operation.

Version 11.5: June 7, 2019

- Xtract -is-equal-to and -differs-from conditional arguments compare values in two named elements.

Version 11.4: May 23, 2019

- Efetch -db snp -format json reads 10 records at a time for the initial server deployment.
- Updated SNP examples to use efetch -format json and xtract -j2x.
- Added Genes in Pathways example.
- Added xml2tbl script.

Version 11.3: May 3, 2019

- Xtract -j2x converts JSON stream to XML suitable for -path navigation.
- Xtract -format -self retains self-closing tabs with no attributes.
- Esample replaces xtract -samples.

Version 11.2: April 15, 2019

- Efetch -db snp only supports -format docsum and -format json.
- Efilter -db biosystems has -kind and -pathway shortcuts.

Version 11.1: April 3, 2019

- Xtract optimizes performance for 6 CPUs with hyperthreading.

Version 11.0: March 11, 2019

- Xtract -path generates exploration commands from dotted object path.
- Xtract -format -separate retains internal </parent><parent>.

Version 10.9: February 1, 2019

- Xtract -insd supports a sub_sequence qualifier that uses -nucleic and produces upper-case sequence.
- Xtract now has an -is-within string conditional test.

Version 10.8: January 20, 2019

- EDIRECT_DO_AUTO_ABBREV environment variable restores relaxed matching of command-line arguments.
- Efilter shortcut -journal added for -db pubmed.
- Efilter -pub last_* shortcuts moved to -released.
- Efilter -pub and -feature can take comma-separate list of choices.
- Transmute -docsum command added.
- Transmute -decode and -encode commands renamed to -unescape and -escape.
- Transmute -decode64 and -encode64 commands added.

Version 10.7: January 14, 2019

- Xtract -nucleic uses bracketed range direction to determine whether to reverse complement the sequence.

2018

Version 10.6: December 13, 2018

- Local archive script creates command for saving data path in configuration file.
- Xtract -reverse returns -words output in reverse order.
- Efilter shortcut added for -db snp (e.g., -class missense).
- Efetch -format gbc (INSDSeq XML) supports -style withparts and -style conwithfeat.

Version 10.5: December 4, 2018

- EDirect commands and pipelines support faster access with API keys.
- Xtract attributes can be delimited by quotation marks or apostrophes.
- Transmute -encode and -decode commands added.
- Simplified processing of local inverted index intermediate files.

Version 10.4: November 13, 2018

- Rchive local indexing code refactored for faster performance.
- Xtract -deq deletes and replaces queued tab separator after the fact.
- Efilter -organism queries in [ORGN] field if argument is not in shortcut list.

Version 10.3: November 1, 2018

- Rchive -invert, -merge, -promote, and -query steps make better use of multiple processor cores.
- New phrase-search script replaces local-phrase-search.

Version 10.2: October 15, 2018

- Transmute -x2j joins -j2x to simplify the use of JSON-based services.
- Efetch -json converts adjusted XML output to JSON as a convenience.
- Xtract tag alphabet expanded to accommodate converted JSON data.
- Nquire -ftp takes server, directory, and filename arguments, sends data to stdout.

Version 10.1: October 9, 2018

- Xtract -mixed improves support for mixed-content XML.

Version 10.0: September 27, 2018

- Efilter can search for sequence records by sample collection location (e.g., -country "canada new brunswick").
- Xtract parsing code was refactored in preparation for improvements in handling mixed-content XML data.
- Added transmute script for format conversions (e.g., -j2x for JSON to XML).

Version 9.90: September 17, 2018

- Normalized archive path for low-value PMIDs in preparation for incremental indexing.

Version 9.80: September 4, 2018

- Xtract XML block reader can run on separate thread for improved performance on computers with surplus processor cores.
- Fixed bug in string cleanup when text starts with a non-ASCII Unicode character.
- Efetch regular expression pattern for detecting mixed-content tags was adjusted.

Version 9.70: August 22, 2018

- Local archive builds parallel stemmed and non-stemmed indices of terms in the title and abstract.
- Rchive and local-phrase-search use -query for evaluation of non-stemmed terms, -search for evaluation using the stemmed index.

Version 9.60: August 9, 2018

- Local archive script removes newlines inside PubMed text fields.
- Efetch adds missing newline at end of PubmedArticleSet XML.

Version 9.50: July 30, 2018

- Local indexing scripts adjusted to accommodate projected range of PMID values.
- Fixed inconsistency in positional indexing of terms with embedded non-alphanumeric characters.
- EDIRECT_PUBMED_WORKING environment variable keeps local archive intermediate files on a separate volume.
- Rchive and local-phrase-search use -exact to round-trip ArticleTitle contents without interpretation as a query formula.

Version 9.40: July 18, 2018

- Xtract handles misplaced spaces in attributes.
- Xtract -format repairs misplaced spaces in attributes.

Version 9.30: July 9, 2018

- Local data indexing retains intermediate products, allows rapid streaming of non-redundant current records.
- Index preparation removes apostrophe in trailing 's possessives.
- Wildcard minimum varies with prefix-driven posting character depth.

Version 9.20: June 26, 2018

- Portability and efficiency improvements to local data cache scripts.
- Xtract handles misplaced spaces in self-closing tags.

Version 9.10: June 18, 2018

- Added Parent/* element exploration construct to xtract.
- Xtract -year reliably obtains the year from "PubDate/*".

Version 9.00: June 6, 2018

- Fetch-pubmed -path supplies missing Archive directory if root path is given.
- Efetch cleanup of MathML markup properly handles parentheses.

Version 8.90: June 4, 2018

- Xtract -transform and -translate allow data value substitution.
- Xtract -wrp simplifies wrapping of extracted values in XML tags.

Version 8.80: May 29, 2018

- Efetch removes MathML tags from PubmedArticle XML contents, unless the -raw flag is used.

Version 8.70: May 14, 2018

- Local phrase indexing now uses positional indices instead of adjacent overlapping word pairs.
- Xtract -select uses conditional expressions to filter records.

Version 8.60: April 26, 2018

- Efetch -format uid pauses between groups, retries on failure.
- Fetch delay drops from 1/3 to 1/10 second if API key is used.
- Local phrase indexing uses smaller files to avoid memory contention.
- Phrase index removes hyphens from selected prefixes.

Version 8.50: April 13, 2018

- Efetch markup tag removal modified after change in server.
- Xtract -phrase filter split into -require and -exclude commands.

Version 8.40: April 9, 2018

- Efetch removes markup tags in all PubMed XML.
- Xtract without -strict prints warnings if markup tags are encountered.
- Xtract proximity search moved from -matches to -phrase.

Version 8.30: April 4, 2018

- Xtract is now available for ARM processors.

Version 8.20: March 12, 2018

- Minor changes to local record archiving scripts.

Version 8.10: March 2, 2018

- Xtract -strict and -mixed support MathML element tags in PubmedArticle XML.

Version 8.00: February 26, 2018

- Efetch -raw skips database-specific XML modifications.
- Added local-phrase-search script.
- Xtract -strict, -mixed, and -repair flag speed improvements.

Version 7.90: February 1, 2018

- Minor change to installation commands for tcsh.

Version 7.80: January 12, 2018

- Updated setup.sh script with additional error checking.

2017

Version 7.70: December 27, 2017

- Added archive-pubmed script to automate local record archiving.

Version 7.60: November 15, 2017

- Epost -id numeric argument bug fixed.
- Xtract conditional tests can now use subrange specifiers.
- Xtract -strict and -mixed use separate -repair flag to normalize Unicode superscripts and subscripts.

Version 7.50: October 31, 2017

- Setup instructions now work with the tcsh shell.
- API key value is taken from the NCBI_API_KEY environment variable.
- Efetch -format gb supports -style withparts and -style conwithfeat.
- Xtract supports optional element [min:max] substring extraction.
- Xtract -position supports [first|last|outer|inner|all] argument values.
- Added prepare-stash script for local record archive.

Version 7.40: September 27, 2017

- Xtract -hash reports checksums for local record archiving.
- Initial support for API keys.

Version 7.30: September 6, 2017

- Modified stash-pubmed script to work around Cygwin artifact.
- Removed unpack-pubmed script.
- Xtract -archive replaces -stash for local record archiving.
- Xtract -gzip allows compression of archived XML records.

Version 7.20: August 28, 2017

- Added download-pubmed, download-sequence, unpack-pubmed, stash-pubmed, and fetch-pubmed scripts, for experimental local record storage.
- Xtract -flags [strict|mixed] added to support new local storage scripts.
- Removed obsolete, original Perl implementation of xtract.pl.

Version 7.10: August 10, 2017

- Xtract -ascii converts non-ASCII Unicode to hexadecimal numeric character references.
- Setup script recognizes Cygwin running under the MinGW emulator.

Version 7.00: July 10, 2017

- Xtract -mixed and -strict handle multiply-escaped HTML tags.
- Efetch removes normal and escaped HTML tags from PubMed fields.

- Esearch -field processes individual query terms using the designated field, also removing stop words.
- Esearch -pairs splits the query phrase into adjacent overlapping word pairs.

Version 6.90: July 5, 2017

- Xtract -mixed replaces -relaxed, and -accent replaces -plain.
- Efetch uses larger chunks for -format uid, url, and acc.
- Esearch -log shows constructed URL and QueryTranslation result.

Version 6.80: June 8, 2017

- Modified download instructions to use edirect.tar.gz archive.
- The ftp-cp script can now read from stdin without the need for xargs.
- Rerunning ftp-cp or asp-cp only attempts to download missing files.

Version 6.70: May 8, 2017

- Added asp-cp script for faster download of NCBI ftp files using Aspera Connect.
- Xtract -strict and -relaxed handle empty HTML tag variants (e.g., <b/> and <sup/>).

Version 6.60: April 25, 2017

- Xtract -strict replaces -degloss to remove HTML <i>, <b>, <u>, <sup> and <sub> tags from XML contents.
- Xtract -relaxed allows HTML tags in XML contents, to support current PubMed ftp release files.
- Xtract -plain removes Unicode accents.
- The setup.sh script prints an error message if it cannot fetch missing Perl modules.

Version 6.50: March 6, 2017

- Xtract -degloss replaces -html to remove HTML <i>, <b>, <u>, <sup> and <sub> tags.

Version 6.40: March 1, 2017

- Epost detects accession.version input for sequence databases and sets -format acc.
- Xtract -html [remove|encode] converts <i> and <b> tags embedded in XML contents.

Version 6.30: February 13, 2017

- Efetch -format docsum skips GI-less sequences without summaries.
- Xtract local indexing commands moved to -extras documentation.

Version 6.20: January 30, 2017

- Xtract -limit and -index allow extraction of selected records from XML file.

Version 6.10: January 19, 2017

- Added run-ncbi-converter script for processing ASN.1 release files.
- Xtract -format flush option added.
- Removed obsolete accession-dot-version conversion code.

2016

Version 6.00: December 27, 2016

- Efetch -format docsum removes eSummaryResult wrapper.
- Fixed content truncation bug when Xtract encounters very long sequences.

Version 5.90: December 21, 2016

- Efetch and Elink readied for switch to accession-dot-version sequence identifier.
- Xtract -insd recognizes INSDInterval_iscomp@value and other boolean attributes.
- Xtract adds experimental phrase processing commands for word index preparation.

Version 5.80: December 12, 2016

- Efilter adds shortcuts for -db gene (e.g., -status alive, -type coding).
- Xtract numeric conditional tests can use an element name for the second argument (e.g., -if ChrStop -lt ChrStart finds minus strand genes).

Version 5.70: November 30, 2016

- Xtract -format takes an optional [compact|indent|expand] argument. Processing compact XML is about 15% faster than indent form. Using expand places each attribute on a separate line for ease of reading.

Version 5.60: November 22, 2016

- Fixed bug in -datetype argument for Esearch and Efilter.
- Added optional argument to filter-stop-words script to indicate replacement.

Version 5.50: November 16, 2016

- Efetch -id allows non-numeric accessions only for sequence databases.
- Xtract element selection no longer considers fields in recursive sub-objects.
- Xtract introduces a double-star "**/Object" construct to flatten recursive child objects for linear exploration.
- Xtract conditional tests ignore empty self-closing tags.
- Xtract -else simplifies insertion of a placeholder to indicate missing data.

Version 5.40: November 7, 2016

- Added filter-stop-words and xy-plot scripts.

Version 5.30: October 31, 2016

- Added support for ecitmatch utility.
- Added amino-acid-composition and between-two-genes scripts.
- The sort-uniq-count and sort-uniq-count-rank scripts take an optional argument (e.g., -n for numeric comparisons, -r to reverse order).

Version 5.20: October 26, 2016

- Setup script no longer modifies the user's configuration file to update the PATH variable. Instead, it now prints customized instructions for the user to execute. The user may choose to run these commands, but is free to edit the .bash_profile file manually.
- Xtract deprecates -match and -avoid functions and the Element:Value conditional shortcut.
- Xtract -if and -unless commands use compound statements for conditional execution (e.g., -if Element -equals Value).
- Colon now separates namespace prefix from element name in xtract arguments (e.g., -block jats:abstract). Colon at start of element name matches any namespace prefix.
- Xtract -insd uses a dash as placeholder for missing field. Experimental -insdx command is deprecated.
- Precompiled versions of xtract are now provided for Darwin, Linux, and CYGWIN_NT platforms. The appropriate executable is downloaded by the setup script.

Version 5.10: October 13, 2016

- Xtract adds -0-based, -1-based, and -ucsc numeric extraction/conversion commands for sequence positions from several Entrez databases.

Version 5.00: September 26, 2016

- Efetch -format fasta removes blank lines between records.
- Xtract -insdx uses a dash to indicate a missing field.
- Xtract -insd no longer has blank lines between records.
- Xtract -input allows reading XML data from a file.

Version 4.90: September 14, 2016

- Epost -input allows reading from an input file instead of using data piped through stdin.
- Efilter now supports the -sort argument.
- Xtract -filter can recover information in XML comments and CDATA blocks.

Version 4.80: August 9, 2016

- Xtract -insd controlled vocabularies updated.

Version 4.70: August 4, 2016

- Einfo -db request can also display -fields and -links data summaries.
- Einfo -dbs prints database names instead of eInfoResult XML.

Version 4.60: July 18, 2016

- Elink -cmd acheck returns information on all available links for a record.
- Efilter -pub structured limits to articles with structured abstracts.

Version 4.50: July 1, 2016

- Esearch and Efilter detect and report -query phrase quotation errors.
- Efilter -pub shortcut adds last_week, last_month, and last_year choices.
- Efetch sets -strand 2 for minus strand if -seq_start > -seq_stop or if -chr_start > -chr_stop.

Version 4.40: June 21, 2016

- Transitioning to use of https for access to NCBI services.
- Epost -db assembly -format acc uses [ASAC] field instead of [ACCN].

Version 4.30: June 13, 2016

- Efilter -pub preprint limits results to ahead-of-print articles.
- Xtract -pattern Parent/* construct can now process catenated XML files.

Version 4.20: May 24, 2016

- Xtract command-line argument parsing improvements.
- Nquire -get supersedes -http get.

Version 4.10: May 3, 2016

- Xtract -format removes multi-line XML comments and CDATA blocks.

Version 4.00: April 4, 2016

- Esearch adds -spell to correct known misspellings of biological terms in the query string.
- Efilter adds -spell to correct query misspellings, and -pub, -feature, -location, -molecule, -organism, and -source shortcuts. Run efilter -help to see the choices available for each argument.

Version 3.90: March 21, 2016

- Code optimizations for increased Xtract speed.

Version 3.80: February 29, 2016

- Xtract can distribute its work among available processor cores for additional speed.

Version 3.70: February 8, 2016

- Xtract performance improvements.

Version 3.60: January 11, 2016

- The setup.sh configuration script now downloads a precompiled Xtract executable for selected platforms.

2015

Version 3.50: December 27, 2015

- Xtract reports error for element:value construct outside of -match or -avoid arguments.

Version 3.40: December 20, 2015

- Xtract -insd supports extraction from multiple features (e.g., CDS,mRNA).

Version 3.30: December 3, 2015

- Efetch -format docsum can accept a single sequence accession number in the -id argument.

Version 3.20: November 30, 2015

- Xtract supports -match conditional execution on values recorded in variables.

Version 3.10: November 18, 2015

- Efetch adds -chr_start and -chr_stop arguments to specify sequence range from 0-based coordinates in gene docsum GenomicInfoType object.

Version 3.00: October 30, 2015

- Xtract rewritten in the Go programming language for speed. The setup.sh configuration script installs an older Perl version (2.99) if a local Go compiler is unavailable.
- Efetch -format docsum only decodes HTML entity numbers in select situations.

Version 2.90: October 15, 2015

- Xtract warns on use of deprecated arguments -present, -absent, and -trim, in preparation for release of much faster version.

Version 2.80: September 9, 2015

- Xtract uses the "*/Child" construct for nested exploration into recursive structures, replacing the -trim argument.

Version 2.70: July 14, 2015

- Added entrez-phrase-search script to query on adjacent word pairs indexed in specific fields.

Version 2.60: June 23, 2015

- Xtract -match and -avoid support "Parent/Child" construct for BLAST XML.

Version 2.50: April 9, 2015

- Xtract capitalized -Pattern handles recursively-defined top-level objects.

Version 2.40: March 25, 2015

- EDirect programs use the http_proxy environment variable to work behind firewalls.

Version 2.30: March 11, 2015

- Cleaned up logic in setup.sh configuration script.
- EPost -format acc works properly on protein accessions.

Version 2.20: March 4, 2015

- Xtract -match and -avoid recognize "@attribute" without element or value.

Version 2.10: February 3, 2015

- Added ftp-ls and ftp-cp scripts for convenient access to the NCBI anonymous ftp server.

2014

Version 2.00: August 28, 2014

- Introduced copy-and-paste installation commands with setup.sh configuration script.

Version 1.90: August 8, 2014

- Xtract -format combines multiple XML results into a single valid object.
- Improved suppression of 0-count failure messages with -silent flag in scripts.

Version 1.80: July 15, 2014

- EPost -format acc accepts accessions in an -id argument on the command line.

Version 1.70: April 23, 2014

- EFetch -format docsum decodes HTML entity numbers embedded in the text.

Version 1.60: April 3, 2014

- Minor enhancements to xtract -insd.

Version 1.50: March 29, 2014

- Esearch -sort specifies the order of results when records are retrieved.
- Xtract exploration arguments (e.g., -block) now work on self-closing tags with data in attributes.

Version 1.40: March 17, 2014

- Xtract -format repairs XML line-wrapping and indentation.
- Implemented -help flag to display the list of command-line arguments for each function.

Version 1.30: March 3, 2014

- Xtract -insd partial logic was corrected to examine both 5' and 3' partial flags, and the location indicator recognizes "+" or "complete" and "-" or "partial".

Version 1.20: February 26, 2014

- Xtract -insd detects if it is part of an EDirect sequence record query, and dynamically executes the extraction request for specific qualifier values. When run in isolation it generates extraction instructions that can be incorporated (with modifications, if necessary) into other queries.

Version 1.10: February 10, 2014

- ESummary was replaced by "efetch -format docsum" to provide a single command for all document retrieval. The esummary command will continue to work for those who prefer it, and to avoid breaking existing scripts.
- Xtract processes each -pattern object immediately upon receipt, eliminating the need for using xargs and sh to split document retrieval into smaller units.

Version 1.00: February 6, 2014

- Initial public release.

2013

Version 0.00: April 23, 2013

- Initial check-in of web documentation page.

Version 0.00: March 20, 2013

- Initial check-in of edirect.pl script source code.

Entrez Direct Examples

Jonathan Kans, PhD[1]

Created: April 23, 2013; Updated: November 23, 2022.

Some early EDirect examples required post-processing with shell script:

```
start=$(( start + 1 ))
stop=$(( stop + 1 ))
```

or awk commands:

```
awk -F '\t' -v 'OFS=\t' '{print $1, $2+1, $3+1}'
```

to increment 0-based sequence coordinates so that the proper region was retrieved:

```
efetch -db nuccore -id $accn -format fasta -seq_start $start -seq_stop $stop
```

This led to -element derivatives that could do simple adjustments within an xtract command:

```
-element ChrAccVer -inc ChrStart -1-based ChrStop
```

and to efetch arguments that could take 0-based coordinates directly:

```
efetch -db nuccore -id $accn -format fasta -chr_start $start -chr_stop $stop
```

These helped eliminate the need for scripts to perform otherwise trivial modifications on extracted data.

More recent work allows easier after-the-fact numeric manipulation using the filter-columns:

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

and print-columns:

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

scripts, which accept column designators in an argument and pass them to internal awk commands. The NF (number of fields) and NR (current record number) built-in variables can also be used.

PubMed

Author Frequency

Who are the most prolific authors on rattlesnake phospholipase?

```
esearch -db pubmed -query \
  "crotalid venoms [MAJR] AND phospholipase [TIAB]" |
efetch -format xml |
xtract -pattern PubmedArticle \
  -block Author -sep " " -tab "\n" -element LastName,Initials |
sort-uniq-count-rank
```

Placing author names on separate lines allows sort-uniq-count-rank to produce a frequency table:

```
87    Lomonte B
77    Gutiérrez JM
64    Soares AM
```

Author Affiliation: 1 NCBI; Email: kans@ncbi.nlm.nih.gov.

Corresponding author.

```
53    Marangoni S
43    Giglio JR
39    Bon C
...
```

Publication Distribution

When were the most papers about Legionnaires disease published?

```
esearch -db pubmed -query "legionnaires disease [TITL]" |
efetch -format docsum |
xtract -pattern DocumentSummary -element PubDate |
cut -c 1-4 |
sort-uniq-count-rank
```

In this case sort-uniq-count-rank reports the number of selected papers per year:

```
173    1979
102    1980
96     1978
92     1981
66     1983
...
```

Using −year "PubDate/*" on a PubmedArticle record takes the first four-digit number it encounters, and the result does not need to be trimmed:

```
esearch -db pubmed -query "legionnaires disease [TITL]" |
efetch -format xml |
xtract -pattern PubmedArticle -year "PubDate/*" |
sort-uniq-count-rank
```

Treatment Locations

What is the geographic distribution of sepsis treatment studies?

```
esearch -db pubmed -query \
  "sepsis/therapy [MESH] AND geographic locations [MESH]" |
efetch -format xml |
xtract -pattern PubmedArticle \
  -block MeshHeading -if DescriptorName@Type -equals Geographic \
    -tab "\n" -element DescriptorName |
sort-uniq-count-rank
```

This returns the number of articles ranked by country or region:

```
660    United States
250    Spain
182    Germany
168    India
155    Taiwan
139    Japan
126    China
124    France
123    Europe
...
```

Note that England and United Kingdom will appear as two separate entries.

Indexed Date Fields

What date fields are indexed for PubMed?

```
einfo -db pubmed |
xtract -pattern Field \
  -if IsDate -equals Y -and IsHidden -equals N \
    -pfx "[" -sfx "]" -element Name \
    -pfx "" -sfx "" -element FullName |
sort -k 2f | expand
```

Indexed dates are shown with field abbreviations and descriptive names:

```
[CDAT]  Date - Completion
[CRDT]  Date - Create
[EDAT]  Date - Entrez
[MHDA]  Date - MeSH
[MDAT]  Date - Modification
[PDAT]  Date - Publication
```

Digital Object Identifiers

How are digital object identifiers obtained from PubMed articles?

```
esearch -db pubmed -query "Rowley JD [AUTH]" |
efetch -format xml |
xtract -head '<html><body>' -tail '</body></html>' \
  -pattern PubmedArticle -PMID MedlineCitation/PMID \
    -block ArticleId -if @IdType -equals doi \
      -tab "" -pfx '<p><a href="' -sfx '">' -doi ArticleId \
      -tab "\n" -pfx '' -sfx '</a></p>' -element "&PMID" |
transmute -format
```

The −doi command extracts the DOIs and constructs URL references. The −pfx and −sfx arguments used here enclose each PMID with a clickable link to its DOI:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE html>
<html>
  <body>
    <p>
      <a href="https://doi.org/10.1038%2Fleu.2013.340">24496283</a>
    </p>
    <p>
      <a href="https://doi.org/10.1073%2Fpnas.1310656110">23818607</a>
    </p>
    ...
  </body>
</html>
```

Reference Formatting

How were references in the main EDirect documentation page formatted?

```
efetch -db pubmed -format docsum \
  -id 26931183 31738401 31600197 29718389 14728215 29490008 8743683 31114887 23175613 |
```

Relevant document summary fields were collected with:

```
xtract -set Set -rec Rec -pattern DocumentSummary \
  -sep ", " -sfx "." -NM Name -clr \
  -sfx "." -SRC Source -clr \
  -pfx ":" -PG Pages -clr \
  -pfx "(PMID " -sfx ".)" -ID Id -clr \
  -group ArticleId -if IdType -equals doi \
    -pfx "https://doi.org/" -sfx "." -DOI Value -PG "(.)" -clr \
  -group DocumentSummary \
    -wrp Name -element "&NM" \
    -wrp Title -element Title \
    -wrp Source -element "&SRC" \
    -wrp Year -year PubDate \
    -wrp Pages -element "&PG" \
    -wrp DOI -element "&DOI" \
    -wrp Id -element "&ID" |
```

This generated intermediate XML with partial formatting:

```
...
<Rec>
  <Name>Wei CH, Allot A, Leaman R, Lu Z.</Name>
  <Title>PubTator central: automated concept ... full text articles.</Title>
  <Source>Nucleic Acids Res.</Source>
  <Year>2019</Year>
  <Pages>.</Pages>
  <DOI>https://doi.org/10.1093/nar/gkz389.</DOI>
  <Id>(PMID 31114887.)</Id>
</Rec>
...
```

References were printed with:

```
xtract -pattern Rec -deq "\n\n" -tab " " -sep "" \
  -element Name Title Source Year,Pages DOI Id
```

followed by manual editing of hyphenated initials:

```
...
Wei C-H, Allot A, Leaman R, Lu Z. PubTator central: automated concept
annotation for biomedical full text articles. Nucleic Acids Res. 2019.
https://doi.org/10.1093/nar/gkz389. (PMID 31114887.)
...
```

Using xtract **-reg** and **-exp** regular expressions to guide xtract **-replace**:

```
efetch -db pubmed -id 31114887 -format xml |
xtract -pattern PubmedArticle -block Author -deq "\n" \
  -element LastName -reg "[a-z .]" -exp "" -replace ForeName
```

can generate hyphenated initials directly from the ForeName field:

```
<Author>
  <LastName>Wei</LastName>
  <ForeName>Chih-Hsuan</ForeName>
  <Initials>CH</Initials>
</Author>
```

by removing lower-case letters, spaces, and periods.

Nucleotide

Coding Sequences

What are the coding sequences in the *Escherichia coli* lac operon?

```
efetch -db nuccore -id J01636.1 -format gbc |
xtract -insd CDS gene sub_sequence
```

Sequence under a feature location is obtained with the −insd "sub_sequence" argument:

```
J01636.1    lacI    GTGAAACCAGTAACGTTATACGATGTCGCAGAGTATGCCG...
J01636.1    lacZ    ATGACCATGATTACGGATTCACTGGCCGTCGTTTTACAAC...
J01636.1    lacY    ATGTACTATTTAAAAAACACAAACTTTTGGATGTTCGGTT...
J01636.1    lacA    TTGAACATGCCAATGACCGAAAGAATAAGAGCAGGCAAGC...
```

Untranslated Region Sequences

What are the 5' and 3' UTR sequences for lycopene cyclase mRNAs?

```
SubSeq() {

  xtract -pattern INSDSeq -ACC INSDSeq_accession-version -SEQ INSDSeq_sequence \
    -group INSDFeature -if INSDFeature_key -equals CDS -PRD "(-)" \
      -block INSDQualifier -if INSDQualifier_name -equals product \
        -PRD INSDQualifier_value \
      -block INSDFeature -pfc "\n" -element "&ACC" -rst \
        -first INSDInterval_from -last INSDInterval_to -element "&PRD" "&SEQ"
}

UTRs() {

  while IFS=$'\t' read acc fst lst prd seq
  do
    if [ $fst -gt 1 ]
    then
      echo -e ">$acc 5'UTR: 1..$((fst-1)) $prd"
      echo "${seq:1:$((fst-2))}" | fold -w 50
    else
      echo -e ">$acc NO 5'UTR"
    fi
    if [ $lst -lt ${#seq} ]
    then
      echo -e ">$acc 3'UTR: $((lst+1))..${#seq} $prd"
      echo "${seq:$lst}" | fold -w 50
    else
      echo -e ">$acc NO 3'UTR"
    fi
  done
}

esearch -db nuccore -query "5.5.1.19 [ECNO]" |
efilter -molecule mrna -feature cds -source refseq |
efetch -format gbc |
SubSeq | UTRs
```

Sequences before the start codon and after the stop codon are obtained with Unix substring commands:

```
>NM_001324787.1 NO 5'UTR
>NM_001324787.1 NO 3'UTR
>NM_001324979.1 5'UTR: 1..262 lycopene beta cyclase, chloroplastic/chromoplastic
attatagaaatacttaagatatatcattgccctttaatcatttattttta
actcttttaagtgtttaaagattgattctttgtacatgttctgcttcatt
tgtgttgaaaattgagttgttttcttgaattttgcaagaatataggggac
cccatttgtgttgaaaattgagcagctttctttgtgttttgttcgatttt
tcaagaatataggaccccatttttctgttttcttgagataaattgcacctt
gttgggaaaat
>NM_001324979.1 3'UTR: 1760..1782 lycopene beta cyclase, chloroplastic/chromoplastic
attcgacttatctgggatcttgt
...
```

5-Column Feature Table

How can you generate a 5-column feature table from GenBank format?

```
XtoT() {

  xtract -pattern INSDSeq -pfx ">Feature " \
      -first INSDSeqid,INSDSeq_accession-version \
    -group INSDFeature -FKEY INSDFeature_key \
      -block INSDInterval -deq "\n" \
        -element INSDInterval_from INSDInterval_to \
          INSDInterval_point INSDInterval_point \
          "&FKEY" -FKEY "()" \
      -block INSDQualifier -deq "\n\t\t\t" \
        -element INSDQualifier_name INSDQualifier_value
}

efetch -db nuccore -id U54469 -format gb |
transmute -g2x |
XtoT
```

Exploring the INSDSeq XML hierarchy with a −pattern {sequence} −group {feature} −block {qualifier} construct, and adding proper indentation, can produce feature table submission format:

```
>Feature U54469.1
1       2881    source
                        organism        Drosophila melanogaster
                        mol_type        genomic DNA
                        db_xref         taxon:7227
                        chromosome      3
                        map             67A8-B2
80      2881    gene
                        gene            eIF4E
80      224     mRNA
892     1458
1550    1920
1986    2085
2317    2404
2466    2881
                        gene            eIF4E
                        product         eukaryotic initiation factor 4E-I
                        ...
```

An **xml2tbl** script containing this xtract command is now included with EDirect.

WGS Components

How can you get FASTA format for all components of a WGS project?

```
GenerateWGSAccessions() {

  gb=$( efetch -db nuccore -id "$1" -format gbc < /dev/null )
  ln=$( echo "$gb" | xtract -pattern INSDSeq -element INSDSeq_length )
  pf=$(
    echo "$gb" |
    xtract -pattern INSDSeq -element INSDSeq_locus |
    sed -e 's/010*/01/g'
  )
  seq -f "${pf}%07.0f" 1 "$ln"
}

GenerateWGSAccessions "JABRPF000000000" |
```

returns the expanded list of accessions from the WGS master:

```
JABRPF010000001
JABRPF010000002
JABRPF010000003
...
JABRPF010000061
JABRPF010000062
JABRPF010000063
```

Piping those accessions to:

```
efetch -db nuccore -format fasta
```

retrieves the individual components in FASTA format:

```
>JABRPF010000001.1 Enterococcus faecalis strain G109-2 contig00001, ...
ACACTAATATGTGTCTTTTTAGACACTAGCTCACTAAAAAAAATAGTCATAATTTCTTCATTATTAAAAT
CCAACAATTGTGAAATCAATTTAATATCCGATGCTTTGAAAACAACTTCTCCTTTTAATTTTTTGTAAAT
CGTTGAAGCGGATATTGGTTGCCCGTATGCAGTCATTTCATTTGCCAACCACTCAACATTTTTTCCTTTC
...
```

Protein

Amino Acid Composition

What is the amino acid composition of human titin?

```
#!/bin/bash -norc

AAComp() {

  abbrev=( Ala Asx Cys Asp Glu Phe Gly His Ile \
           Xle Lys Leu Met Asn Pyl Pro Gln Arg \
           Ser Thr Sec Val Trp Xxx Tyr Glx )

  tr A-Z a-z |
  sed 's/[^a-z]//g' |
  fold -w 1 |
  sort-uniq-count |
  while read num lttr
```

```
  do
    idx=$(printf %i "'$lttr'")
    ofs=$((idx-97))
    echo -e "${abbrev[$ofs]}\t$num"
  done |
  sort
}

efetch -db protein -id Q8WZ42 -format gpc |
xtract -pattern INSDSeq -element INSDSeq_sequence |
AAComp
```

This produces a table of residue counts using the three-letter amino acid abbreviations:

```
Ala    2084
Arg    1640
Asn    1111
Asp    1720
Cys    513
Gln    942
Glu    3193
Gly    2066
His    478
Ile    2062
Leu    2117
Lys    2943
Met    398
Phe    908
Pro    2517
Ser    2463
Thr    2546
Trp    466
Tyr    999
Val    3184
```

Longest Sequences

What are the longest known insulin precursor molecules?

```
esearch -db protein -query "insulin [PROT]" |
efetch -format docsum |
xtract -pattern DocumentSummary -element Caption Slen Title |
grep -v receptor | sort -k 2,2nr | head -n 5 | cut -f 1 |
xargs -n 1 sh -c 'efetch -db protein -id "$0" -format gp > "$0".gpf'
```

Post-processing excludes the longer "insulin-like receptor" sequences, sorts by sequence length, and saves the GenPept results to individual files named by their sequence accessions, using the right angle bracket (">") Unix output redirection character:

```
EFN61235.gpf
EFN80340.gpf
EGW08477.gpf
EKC18433.gpf
ELK28555.gpf
```

Archaea Enzyme

Which archaebacteria have chloramphenicol acetyltransferase?

```
esearch -db protein -organism archaea \
  -query "chloramphenicol acetyltransferase [PROT]" |
efetch -format gpc |
xtract -pattern INSDSeq -element INSDSeq_organism INSDSeq_definition |
grep -i chloramphenicol | grep -v MULTISPECIES |
cut -f 1 | sort -f | uniq -i
```

Filtering on the definition line produces a list of archaeal organism names:

```
Euryarchaeota archaeon
Methanobrevibacter arboriphilus
Methanobrevibacter gottschalkii
Methanobrevibacter millerae
Methanobrevibacter oralis
...
```

Gene

Gene Counts

How many genes are on each human chromosome?

```
for chr in {1..22} X Y MT
do
  esearch -db gene -query "Homo sapiens [ORGN] AND $chr [CHR]" |
  efilter -status alive -type coding |
  efetch -format docsum |
  xtract -pattern DocumentSummary -NAME Name \
    -block GenomicInfoType -if ChrLoc -equals "$chr" \
      -tab "\n" -element ChrLoc,"&NAME" |
  sort | uniq | cut -f 1 |
  sort-uniq-count-rank |
  reorder-columns 2 1
done
```

This returns a count of unique protein-coding genes per chromosome:

```
1     2011
2     1224
3     1046
4     742
5     854
6     1015
7     911
8     666
9     763
10    718
11    1279
12    1012
13    327
14    601
15    585
16    835
17    1147
18    266
19    1391
20    528
21    232
22    429
```

```
X       839
Y       63
MT      13
```

The range construct cannot be used for Roman numerals, so the equivalent query on *Saccharomyces cerevisiae* would need to explicitly list all chromosomes, including the mitochondrion:

```
for chr in I II III IV V VI VII VIII IX X XI XII XIII XIV XV XVI MT
```

Plastid genes can be selected with "source plastid [PROP]" or −location plastid.

Chromosome Locations

Where are mammalian calmodulin genes located?

```
esearch -db gene -query "calmodulin * [PFN] AND mammalia [ORGN]" |
efetch -format docsum |
xtract -pattern DocumentSummary \
  -def "-" -element Id Name MapLocation ScientificName
```

The −def command adds a dash to prevent missing data from shifting columns in the table:

```
801         CALM1       14q32.11      Homo sapiens
808         CALM3       19q13.32      Homo sapiens
805         CALM2       2p21          Homo sapiens
24242       Calm1       6q32          Rattus norvegicus
12313       Calm1       12 E          Mus musculus
50663       Calm2       6q12          Rattus norvegicus
24244       Calm3       1q21          Rattus norvegicus
12315       Calm3       7 9.15 cM     Mus musculus
12314       Calm2       17 E4         Mus musculus
80796       Calm4       13 A1         Mus musculus
617095      CALM1       -             Bos taurus
396838      CALM3       -             Sus scrofa
520277      CALM3       -             Bos taurus
364774      Calml5      17q12.2       Rattus norvegicus
...
```

Exon Counts

How many exons are in each dystrophin transcript variant?

```
esearch -db gene -query "DMD [GENE] AND human [ORGN]" |
efetch -format docsum |
xtract -pattern DocumentSummary -block GenomicInfoType \
  -tab "\n" -element ChrAccVer,ChrStart,ChrStop |
xargs -n 3 sh -c 'efetch -db nuccore -format gbc \
  -id "$0" -chr_start "$1" -chr_stop "$2"' |
```

This retrieves an INSDSeq XML subset record for the indicated gene region, which contains a number of alternatively-spliced dystrophin mRNA and CDS features.

Data extraction computes the number of intervals for each mRNA location (individual exons plus non-adjacent UTRs), and obtains the transcript sequence accession, transcript length, and product name from qualifiers:

```
xtract -insd complete mRNA "#INSDInterval" \
  transcript_id "%transcription" product |
```

Final filtering and sorting:

```
grep -i dystrophin |
sed 's/dystrophin, transcript variant //g' |
sort -k 2,2nr -k 4,4nr
```

results in a table of exon counts and transcript lengths:

```
NC_000023.11    79    NM_004010.3       14083    Dp427p2
NC_000023.11    79    NM_004009.3       14000    Dp427p1
NC_000023.11    79    NM_004006.3       13992    Dp427m
NC_000023.11    79    NM_000109.4       13854    Dp427c
NC_000023.11    78    XM_006724468.2    13923    X1
NC_000023.11    78    XM_017029328.1    13916    X4
NC_000023.11    78    XM_006724469.3    13897    X2
NC_000023.11    77    XM_006724470.3    13875    X3
...
```

Upstream Sequences

What sequences are upstream of phenylalanine hydroxylase genes?

```
esearch -db nuccore -query "U49897 [ACCN]" |
elink -target gene |
elink -target homologene |
elink -target gene |
efetch -format docsum |
xtract -pattern DocumentSummary -if GenomicInfoType -element Id \
  -block GenomicInfoType -element ChrAccVer -1-based ChrStart ChrStop |
```

This produces a table with 1-based sequence coordinates:

```
5053       NC_000012.12    102958441    102836889
18478      NC_000076.7     87357657     87419999
38871      NT_037436.4     7760453      7763166
24616      NC_005106.4     28066639     28129772
378962     NC_007115.7     17409367     17391680
...
```

Then, 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:

```
>NC_000012.12:c102958941-102958442 Homo sapiens chromosome 12, GRCh38.p13 ...
TGAAGTCGAGAAGCTCCTGCTCCTCGGGGCTGAGCGGGTCGTAAGAGCCCTCGTCCGACGAGTAGGATGA
GACCGGCGAGCCGGCCATGGAGTTCAAGTCGTTGGAGTAGTTGGGGGAGATGGTGGGCGACAGGACGCCT
GCCTGGAAGGCGGCGCTCACCGCGTCATGCTCGTCCAGCAGCTGCTGCAGCGCGCGGATGTACTCGACCG
...
```

Assembly

Complete Genomes

What complete genomes are available for *Escherichia coli*?

```
esearch -db assembly -query \
  "Escherichia coli [ORGN] AND representative [PROP]" |
elink -target nuccore -name assembly_nuccore_refseq |
efetch -format docsum |
xtract -pattern DocumentSummary -element AccessionVersion Slen Title |
sed 's/,.*//' |
sort-table -k 2,2nr
```

This search finds genomic assemblies and sorts the results by sequence length, allowing complete genomes to be easily distinguished from smaller plasmids:

```
NC_002695.2    5498578    Escherichia coli O157:H7 str. Sakai DNA
NC_000913.3    4641652    Escherichia coli str. K-12 substr. MG1655
NC_002128.1    92721      Escherichia coli O157:H7 str. Sakai plasmid pO157
NC_002127.1    3306       Escherichia coli O157:H7 str. Sakai plasmid pOSAK1
```

The sed command removes text (e.g., complete genome, complete sequence, primary assembly) after a comma.

A similar query for humans, additionally filtering out scaffolds, contigs, and plasmids:

```
esearch -db assembly -query "Homo sapiens [ORGN] AND representative [PROP]" |
elink -target nuccore -name assembly_nuccore_refseq |
efetch -format docsum |
xtract -pattern DocumentSummary -element AccessionVersion Slen Title |
sed 's/,.*//' |
grep -v scaffold | grep -v contig | grep -v plasmid | grep -v patch |
sort
```

returns the assembled chromosome and mitochondrial sequence records:

```
NC_000001.11    248956422    Homo sapiens chromosome 1
NC_000002.12    242193529    Homo sapiens chromosome 2
NC_000003.12    198295559    Homo sapiens chromosome 3
```

```
NC_000004.12    190214555    Homo sapiens chromosome 4
NC_000005.10    181538259    Homo sapiens chromosome 5
NC_000006.12    170805979    Homo sapiens chromosome 6
NC_000007.14    159345973    Homo sapiens chromosome 7
NC_000008.11    145138636    Homo sapiens chromosome 8
NC_000009.12    138394717    Homo sapiens chromosome 9
NC_000010.11    133797422    Homo sapiens chromosome 10
NC_000011.10    135086622    Homo sapiens chromosome 11
NC_000012.12    133275309    Homo sapiens chromosome 12
NC_000013.11    114364328    Homo sapiens chromosome 13
NC_000014.9     107043718    Homo sapiens chromosome 14
NC_000015.10    101991189    Homo sapiens chromosome 15
NC_000016.10    90338345     Homo sapiens chromosome 16
NC_000017.11    83257441     Homo sapiens chromosome 17
NC_000018.10    80373285     Homo sapiens chromosome 18
NC_000019.10    58617616     Homo sapiens chromosome 19
NC_000020.11    64444167     Homo sapiens chromosome 20
NC_000021.9     46709983     Homo sapiens chromosome 21
NC_000022.11    50818468     Homo sapiens chromosome 22
NC_000023.11    156040895    Homo sapiens chromosome X
NC_000024.10    57227415     Homo sapiens chromosome Y
NC_012920.1     16569        Homo sapiens mitochondrion
```

This process can be automated to loop through a list of specified organisms:

```
for org in \
  "Agrobacterium tumefaciens" \
  "Bacillus anthracis" \
  "Escherichia coli" \
  "Neisseria gonorrhoeae" \
  "Pseudomonas aeruginosa" \
  "Shigella flexneri" \
  "Streptococcus pneumoniae"
do
  esearch -db assembly -query "$org [ORGN]" |
  efilter -query "representative [PROP]" |
  elink -target nuccore -name assembly_nuccore_refseq |
  efetch -format docsum |
  xtract -pattern DocumentSummary -element AccessionVersion Slen Title |
  sed 's/,.*//' |
  grep -v -i -e scaffold -e contig -e plasmid -e sequence -e patch |
  sort-table -k 2,2nr
done
```

which generates:

```
NC_011985.1    4005130    Agrobacterium radiobacter K84 chromosome 1
NC_011983.1    2650913    Agrobacterium radiobacter K84 chromosome 2
NC_005945.1    5228663    Bacillus anthracis str. Sterne chromosome
NC_003997.3    5227293    Bacillus anthracis str. Ames chromosome
NC_002695.1    5498450    Escherichia coli O157:H7 str. Sakai chromosome
NC_018658.1    5273097    Escherichia coli O104:H4 str. 2011C-3493 ...
NC_011751.1    5202090    Escherichia coli UMN026 chromosome
NC_011750.1    5132068    Escherichia coli IAI39 chromosome
NC_017634.1    4747819    Escherichia coli O83:H1 str. NRG 857C chromosome
NC_000913.3    4641652    Escherichia coli str. K-12 substr. MG1655
NC_002946.2    2153922    Neisseria gonorrhoeae FA 1090 chromosome
NC_002516.2    6264404    Pseudomonas aeruginosa PAO1 chromosome
NC_004337.2    4607202    Shigella flexneri 2a str. 301 chromosome
```

```
NC_003028.3    2160842    Streptococcus pneumoniae TIGR4 chromosome
NC_003098.1    2038615    Streptococcus pneumoniae R6 chromosome
```

SNP

SNP Data Table

How can you obtain a tab-delimited table of SNP attributes from a document summary?

```
efetch -db snp -id 11549407 -format docsum |
snp2tbl
```

The **snp2tbl** script extracts fields from HGVS data and prints the individual values for further processing:

```
rs11549407    NC_000011.10    5226773    G    A    Genomic    Substitution    HBB
rs11549407    NC_000011.10    5226773    G    C    Genomic    Substitution    HBB
rs11549407    NC_000011.10    5226773    G    T    Genomic    Substitution    HBB
rs11549407    NG_000007.3     70841      C    A    Genomic    Substitution    HBB
rs11549407    NG_000007.3     70841      C    G    Genomic    Substitution    HBB
rs11549407    NG_000007.3     70841      C    T    Genomic    Substitution    HBB
...
rs11549407    NM_000518.5     167        C    A    Coding     Substitution    HBB
rs11549407    NM_000518.5     167        C    G    Coding     Substitution    HBB
rs11549407    NM_000518.5     167        C    T    Coding     Substitution    HBB
rs11549407    NP_000509.1     39         Q    *    Protein    Termination     HBB
rs11549407    NP_000509.1     39         Q    E    Protein    Missense        HBB
rs11549407    NP_000509.1     39         Q    K    Protein    Missense        HBB
```

Columns are SNP identifier, accession.version, offset from start of sequence, letters to delete, letters to insert, sequence class, variant type, and gene name. The internal steps (**snp2hgvs**, **hgvs2spdi**, and **spdi2tbl**) are described below.

Amino Acid Substitutions

What are the missense products of green-sensitive opsin?

```
esearch -db gene -query "OPN1MW [PREF] AND human [ORGN]" |
elink -target snp | efilter -class missense |
efetch -format docsum |
```

SNP document summaries contain HGVS data of the form:

```
NC_000023.11:g.154193517C>A, ... ,NP_000504.1:p.Ala285Val
```

This can be parsed by an xtract **-hgvs** command within the **snp2hgvs** script:

```
snp2hgvs |
```

into a structured representation of nucleotide and amino acid replacements:

```
...
<HGVS>
  <Id>782327292</Id>
  <Gene>OPN1MW</Gene>
  <Variant>
    <Class>Genomic</Class>
    <Type>Substitution</Type>
    <Accession>NC_000023.11</Accession>
    <Position>154193516</Position>
    <Deleted>C</Deleted>
```

```
      <Inserted>A</Inserted>
      <Hgvs>NC_000023.11:g.154193517C&gt;A</Hgvs>
    </Variant>
    ...
    <Variant>
      <Class>Coding</Class>
      <Type>Substitution</Type>
      <Accession>NM_000513.2</Accession>
      <Offset>853</Offset>
      <Deleted>C</Deleted>
      <Inserted>T</Inserted>
      <Hgvs>NM_000513.2:c.854C&gt;T</Hgvs>
    </Variant>
    <Variant>
      <Class>Protein</Class>
      <Type>Missense</Type>
      <Accession>NP_000504.1</Accession>
      <Position>284</Position>
      <Deleted>A</Deleted>
      <Inserted>D</Inserted>
      <Hgvs>NP_000504.1:p.Ala285Asp</Hgvs>
    </Variant>
    <Variant>
      <Class>Protein</Class>
      <Type>Missense</Type>
      <Accession>NP_000504.1</Accession>
      <Position>284</Position>
      <Deleted>A</Deleted>
      <Inserted>V</Inserted>
      <Hgvs>NP_000504.1:p.Ala285Val</Hgvs>
    </Variant>
  </HGVS>
  ...
```

where the original 1-based HGVS positions are converted to 0-based in the XML.

Passing those results through the **hgvs2spdi** script:

```
  hgvs2spdi |
```

converts CDS-relative offsets to sequence-relative positions:

```
  ...
  <SPDI>
    <Id>782327292</Id>
    <Gene>OPN1MW</Gene>
    ...
    <Variant>
      <Class>Coding</Class>
      <Type>Substitution</Type>
      <Accession>NM_000513.2</Accession>
      <Position>935</Position>
      <Deleted>C</Deleted>
      <Inserted>A</Inserted>
      <Hgvs>NM_000513.2:c.854C&gt;A</Hgvs>
      <Spdi>NM_000513.2:935:C:A</Spdi>
    </Variant>
    <Variant>
      <Class>Coding</Class>
      <Type>Substitution</Type>
```

```
      <Accession>NM_000513.2</Accession>
      <Position>935</Position>
      <Deleted>C</Deleted>
      <Inserted>T</Inserted>
      <Hgvs>NM_000513.2:c.854C&gt;T</Hgvs>
      <Spdi>NM_000513.2:935:C:T</Spdi>
    </Variant>
    ...
  </SPDI>
  ...
```

Piping this through the **spdi2tbl** script:

```
spdi2tbl |
```

completes the final step of the **snp2tbl** process to generate a SNP Data Table. Filtering that through:

```
grep Protein | grep Missense | cut -f 1-5
```

results in a table of amino acid substitutions sorted by accession, position, and residue:

```
rs1238141906    NP_000504.1    40    E    K
rs1189783086    NP_000504.1    42    P    L
rs1257135801    NP_000504.1    45    H    Y
rs1284438666    NP_000504.1    63    V    I
rs1223726997    NP_000504.1    64    I    T
...
```

Those rows are processed in a while loop that caches the current sequence data:

```
while read rsid accn ofs del ins
do
  if [ "$accn" != "$last" ]
  then
    seq=$( efetch -db protein -id "$accn" -format gpc < /dev/null |
           xtract -pattern INSDSeq -lower INSDSeq_sequence )
    last="$accn"
  fi
  pos=$((ofs + 1))
  echo ">$rsid [$accn $ins@$pos]"
  echo "$seq" |
  transmute -replace -offset "$ofs" -delete "$del" -insert "$ins" -lower |
  fold -w 50
done
```

and uses transmute **-replace** to generate modified FASTA with substituted residues in upper case:

```
>rs1238141906 [NP_000504.1 K@41]
maqqwslqrlagrhpqdsyedstqssiftytnsnstrgpfKgpnyhiapr
wvyhltsvwmifvviasvftnglvlaatmkfkklrhplnwilvnlavadl
aetviastisvvnqvygyfvlghpmcvlegytvslcgitglwslaiiswe
...
```

SNP-Modified Product Pairs

How can you match codon modifications with amino acid substitutions?

```
efetch -db snp -id 11549407 -format docsum |
snp2tbl |
tbl2prod
```

For SNPs that have different substitutions at the same position, the **tbl2prod** script translates coding sequences (after nucleotide modification), 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...
rs11549407    NM_000518.5:167:C:+    MVHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWTQR...
rs11549407    NP_000509.1:39:Q:+     MVHLTPEEKSAVTALWGKVNVDEVGGEALGRLLVVYPWTQR...
```

The "+" sign indicates the unmodified "wild-type" nucleotide or amino acid.

Structure

Structural Similarity

What archaea structures are similar to snake venom phospholipase?

```
esearch -db structure -query "crotalus [ORGN] AND phospholipase A2" |
elink -related |
efilter -query "archaea [ORGN]" |
efetch -format docsum |
xtract -pattern DocumentSummary \
  -if PdbClass -equals Hydrolase \
    -element PdbDescr |
sort -f | uniq -i
```

Structure neighbors use geometric comparison to find proteins that are too divergent to be detected by sequence similarity with a BLAST search:

```
Crystal Structure Of Autoprocessed Form Of Tk-Subtilisin
Crystal Structure Of Ca2 Site Mutant Of Pro-S324a
Crystal Structure Of Ca3 Site Mutant Of Pro-S324a
...
```

Taxonomy

Taxonomic Lineage

What are the major taxonomic lineage nodes for humans?

```
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 -is-not clade \
      -tab "\n" -element Rank,ScientificName
```

This uses the double star / child construct to recursively explore the data hierarchy:

```
9606            Homo sapiens
superkingdom    Eukaryota
kingdom         Metazoa
phylum          Chordata
subphylum       Craniata
superclass      Sarcopterygii
```

```
class           Mammalia
superorder      Euarchontoglires
order           Primates
...
```

Taxonomy Search

Which organisms contain an annotated RefSeq genome MatK gene?

```
esearch -db nuccore -query "MatK [GENE] AND NC_0:NC_999999999 [PACC]" |
efetch -format docsum |
xtract -pattern DocumentSummary -element TaxId |
sort -n | uniq |
epost -db taxonomy |
efetch -format docsum |
xtract -pattern DocumentSummary -element ScientificName |
sort
```

The first query obtains taxonomy UIDs from nucleotide document summaries and uploads them for separate retrieval from the taxonomy database:

```
Acidosasa purpurea
Acorus americanus
...
Zingiber spectabile
Zygnema circumcarinatum
```

SRA

Installation of EDirect on Cloud

To install the EDirect software, open a terminal window and execute the following command:

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

At the end of installation, answer "y" to have the script run the PATH update command to edit your configuration file, so that EDirect programs can be run in subsequent terminal sessions.

One installation is complete, run:

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

to set the PATH for the current terminal session.

Downloading BLAST Software

Obtain the Magic-BLAST software if it is not already installed:

```
download-ncbi-software magic-blast
```

Preparing Chromosome Files

Retrieve the sequence of human chromosome 7 with:

```
efetch -db nuccore -id NC_000007 -format fasta -immediate > NC_000007.fsa
```

Download protein-coding genes on human chromosome 7 in document summary format, and extract the gene ranges into a tab-delimited table:

```
esearch -db gene -query "Homo sapiens [ORGN] AND 7 [CHR]" |
efilter -status alive -type coding | efetch -format docsum |
gene2range "7" > NC_000007.gen
```

SRA Gene Analysis

Run a **magicblast** search of SRR6314034 against chromosome 7:

```
magicblast -sra SRR6314034 -subject NC_000007.fsa -outfmt asn |
asn2xml > SRR6314034.xml
```

Passing the alignment details through the **blst2tkns** script:

```
cat SRR6314034.xml |
blst2tkns |
```

tokenizes the alignment parts:

```
index          1
score          6268
start          33088037
stop           33167397
strand         plus
match          12
mismatch       1
genomic-ins    1
...
```

Piping these to the **split-at-intron** script:

```
split-at-intron |
```

detects large genomic insertions:

```
6268    1    plus    33088037..33091003,33163798..33167397
4910    2    plus    33089462..33091003,33163799..33167397
1814    3    plus    33164965..33167397
...
```

These are filtered by minimum score with:

```
filter-columns '$1 > 1000' |
```

The **fuse-ranges** script then merges overlapping alignments:

```
fuse-ranges |
```

into a minimal set of extended ranges:

```
plus     33088037    33091003    2967
plus     33163798    33167398    3601
plus     33272248    33272278    31
plus     33394238    33394551    314
minus    33088037    33091004    2968
minus    33163798    33167398    3601
minus    33255886    33256012    127
minus    33394326    33394551    226
```

Running each range through the **find-in-gene** script:

```
while read std min max len
do
```

```
  cat NC_000007.gen |
  find-in-gene "$std" "$min" "$max"
done |
sort -f | uniq -i
```

returns the names of gene(s) that overlap the aligned segments:

```
BBS9
```

PubChem

PubChem in Entrez

Entrez can search on the complete synonym of a compound:

```
esearch -db pccompound -query "catechol [CSYN]" |
efetch -format uid
```

to return a small number of closely matching compound identifiers (CIDs):

```
73160
9064
289
```

Entrez document summaries for a compound:

```
efetch -db pccompound -id 289 -format docsum |
xtract -pattern DocumentSummary -element MolecularFormula IsomericSmiles
```

contains general descriptive information fields for the compound:

```
C6H6O2    C1=CC=C(C(=C1)O)O
```

Power User Gateway (PUG) REST Query Form

PubChem also supports a RESTful service for more advanced queries. Nquire provides a -pugrest URL shortcut. The base form of a search request is is:

```
nquire -pugrest [compound|substance|assay] [input] [operation] [output]
```

Searches can use the name of a compound:

```
nquire -pugrest compound name catechol cids TXT
```

to obtain the best matching compound identifier:

```
289
```

The CID can be used as the input key to obtain a title and description:

```
nquire -pugrest compound cid 289 description XML
```

or to retrieve a much more detailed record:

```
nquire -pugrest compound cid 289 record XML |
```

that includes the canonical or isomeric SMILES codes:

```
xtract -pattern PC-InfoData \
  -if PC-Urn_label -equals SMILES -and PC-Urn_name -equals Isomeric \
    -element PC-InfoData_value_sval
```

PubChem Chemical Identifiers

Certain identifier types require POST arguments to encode special symbols:

```
nquire -pugrest compound smiles description XML \
  -smiles "C1=CC=C(C(=C1)O)O" |
xtract -pattern InformationList -element Title Description
```

This returns the chemical name and description:

```
Catechol    Catechol is a benzenediol comprising...
```

Other identifier key types that are encoded in separate arguments are shown below:

```
nquire -pugrest compound inchi synonyms TXT \
  -inchi "1S/C6H6O2/c7-5-3-1-2-4-6(5)8/h1-4,7-8H"

nquire -pugrest compound inchikey cids JSON \
  -inchikey "YCIMNLLNPGFGHC-UHFFFAOYSA-N"

nquire -pugrest compound/fastsubstructure/smarts/cids/XML \
  -smarts "[#7]-[#6]-1=[#6]-[#6](C#C)=[#6](-[#6]-[#8])-[#6]=[#6]-1"
```

(Nquire -inchi will supply the expected "InChI=" prefix if it is missing in the argument string.)

PUG-REST Asynchronous Queries

Some PUG-REST queries are computationally intensive and run asynchronously:

```
nquire -pugrest compound/superstructure/cid/2244/XML |
```

The returned <ListKey> token is piped to an nquire -pugwait command, which polls the server until the results are available:

```
nquire -pugwait
```

Identifiers are then downloaded and placed directly in an ENTREZ_DIRECT message:

```
<ENTREZ_DIRECT>
  <Db>pccompound</Db>
  <Count>3750</Count>
  <Id>87</Id>
  <Id>175</Id>
  <Id>176</Id>
  ...
  <Id>162400221</Id>
  <Id>162416056</Id>
  <Id>162417911</Id>
</ENTREZ_DIRECT>
```

Support for this form was added when EDirect was redesigned in 2020.