

# Entrez Programming Utilities Help


Last Updated: November 13, 2018



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-.

-  [E-utilities 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

<b>E-utilities Quick Start</b> .....	1
Release Notes .....	1
Announcement .....	1
Introduction .....	1
Searching a Database .....	1
Uploading UIDs to Entrez .....	4
Downloading Document Summaries .....	5
Downloading Full Records .....	8
Finding Related Data Through Entrez Links .....	9
Getting Database Statistics and Search Fields .....	10
Performing a Global Entrez Search .....	12
Retrieving Spelling Suggestions .....	13
Demonstration Programs .....	14
For More Information .....	18
<b>A General Introduction to the E-utilities</b> .....	19
Introduction .....	19
Usage Guidelines and Requirements .....	19
The Nine E-utilities in Brief .....	22
Understanding the E-utilities Within Entrez .....	23
Combining E-utility Calls to Create Entrez Applications .....	27
Demonstration Programs .....	29
For More Information .....	29
<b>Sample Applications of the E-utilities</b> .....	31
Introduction .....	31
Basic Pipelines .....	31
ESearch – ESummary/EFetch .....	31
EPost – ESummary/EFetch .....	32
ELink – ESummary/Efetch .....	33

ESearch – ELink – ESummary/EFetch.....	34
EPost – ELink – ESummary/EFetch.....	35
EPost – ESearch.....	37
ELink – ESearch.....	38
Application 1: Converting GI numbers to accession numbers.....	39
Application 2: Converting accession numbers to data.....	39
Application 3: Retrieving large datasets.....	40
Application 4: Finding unique sets of linked records for each member of a large dataset.....	41
Demonstration Programs.....	42
For More Information.....	42
<b>The E-utilities In-Depth: Parameters, Syntax and More.....</b>	<b>43</b>
Introduction.....	43
General Usage Guidelines.....	43
E-utilities DTDs.....	44
EInfo.....	44
ESearch.....	45
EPost.....	49
ESummary.....	50
EFetch.....	52
ELink.....	59
EGQuery.....	64
ESpell.....	65
ECitMatch.....	65
Release Notes.....	66
Demonstration Programs.....	67
For More Information.....	67
<b>The E-utility Web Service (SOAP).....</b>	<b>69</b>
Termination Announcement.....	69
For More Information.....	69

<b>Entrez Direct: E-utilities on the UNIX Command Line</b> .....	71
Getting Started.....	71
Searching and Filtering.....	75
Structured Data.....	79
Sequence Records.....	90
Sequence Coordinates.....	96
Complex Objects.....	97
Advanced Topics.....	100
Automation.....	105
Local Data Cache.....	108
XML Processing.....	111
Examples.....	114
Appendices.....	134
Release Notes.....	149
For More Information.....	161

# E-utilities Quick Start

Eric Sayers, PhD<sup>1</sup>

Created: December 12, 2008; Updated: November 1, 2017.

## Release Notes

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

## Announcement

On May 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\[pdat\]`**

---

<sup>1</sup> NCBI; Email: [sayers@ncbi.nlm.nih.gov](mailto:sayers@ncbi.nlm.nih.gov).

<sup>✉</sup> Corresponding author.

## 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*

```
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>0193yIkBjmM60UBXuvBvPfBIq8-9nIsldXuMP0hhuMH-
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 citations:*

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](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?](https://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=nuccore&id=34577062,24475906&rettype=fasta&retmode=text)  
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 nucleotide GIs 34577062 and 24475906*

**[https://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?  
dbfrom=nucleotide&db=gene&id=34577062,24475906](https://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?dbfrom=nucleotide&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 nucleotide GIs 34577062 and 24475906*

**[https://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?  
dbfrom=nucleotide&db=gene&id=34577062&id=24475906](https://eutils.ncbi.nlm.nih.gov/entrez/eutils/elink.fcgi?dbfrom=nucleotide&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](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"
"http://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\[orgn\]](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 contains the PATH & parameters for the ESearch call
# $esearch_result contains the result of the ESearch call
# the results are displayed & 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 $utils_root  = "https://www.ncbi.nlm.nih.gov/entrez/efetch";
my $ePost_url   = "$utils_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 - first 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 "Response status message: [" . $response->message . "]\n";
print "Response 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](mailto:eutilities@ncbi.nlm.nih.gov).



# A General Introduction to the E-utilities

Eric Sayers, PhD<sup>1</sup>

## 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.

---

<sup>1</sup> NCBI; Email: [sayers@ncbi.nlm.nih.gov](mailto:sayers@ncbi.nlm.nih.gov).

 Corresponding author.

## 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 policy changes affecting the E-utilities. To register **tool** and **email** values, simply send an e-mail to [utilities@ncbi.nlm.nih.gov](mailto:utilities@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.

## Coming in December 2018: API Keys

On December 1, 2018, NCBI will begin enforcing the use of API keys that will offer enhanced levels of supported access to the E-utilities. After that date, any site (IP address) posting more than 3 requests per second to the E-utilities without an API key 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 ([utilities@ncbi.nlm.nih.gov](mailto:utilities@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 new *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.

We encourage regular E-utility users to obtain an API key as soon as possible and begin the process of incorporating it into code. We also encourage users to monitor their request rates to determine if they will require rates higher than 10 per second. As stated above, we can potentially have higher rates negotiated prior to the beginning of enforcement on December 1, 2018.

## 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 enter into a licensing agreement and lease the data for free from NLM. For more information on this please see <http://www.nlm.nih.gov/databases/leased.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)

*utils.ncbi.nlm.nih.gov/entrez/efetch/egquery.fcgi*

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

## ESpell (spelling suggestions)

*utils.ncbi.nlm.nih.gov/entrez/efetch/espell.fcgi*

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

## ECitMatch (batch citation searching in PubMed)

*utils.ncbi.nlm.nih.gov/entrez/efetch/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
Biosystems	BSID	biosystems
Books	Book ID	books

*Table 1 continues on next page...*

*Table 1 continued from previous page.*

Entrez Database	UID common name	E-utility Database Name
Conserved Domains	PSSM-ID	cdd
dbGaP	dbGaP ID	gap
dbVar	dbVar ID	dbvar
Epigenomics	Epigenomics ID	epigenomics
EST	GI number	nucest
Gene	Gene ID	gene
Genome	Genome ID	genome
GEO Datasets	GDS ID	gds
GEO Profiles	GEO ID	geoprofiles
GSS	GI number	nucgss
HomoloGene	HomoloGene ID	homologene
MeSH	MeSH ID	mesh
NCBI C++ Toolkit	Toolkit ID	toolkit
NCBI Web Site	Web Site ID	ncbisearch
NLM Catalog	NLM Catalog ID	nlmcatalog
Nucleotide	GI number	nucore
OMIA	OMIA ID	omia
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
UniGene	UniGene Cluster ID	unigene
UniSTS	STS ID	unists

## 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_databasedb=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<sup>1</sup>

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]';
```

---

<sup>1</sup> NCBI; Email: sayers@ncbi.nlm.nih.gov.

<sup>✉</sup> Corresponding author.

```
#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>/);

### 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 UUIDs (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 &rettype=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<sup>1</sup>

Created: May 29, 2009; Updated: October 3, 2018.

## 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.

## api\_key – enforced in December 2018

In December 2018, NCBI will begin enforcing the practice of using an API key for sites that post more than 3 requests per second. Please see Chapter 2 for a full discussion of this new policy.

---

<sup>1</sup> NCBI; Email: [sayers@ncbi.nlm.nih.gov](mailto:sayers@ncbi.nlm.nih.gov).

 Corresponding author.

## 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](#)

## 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

## 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`

## 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 100,000 records. To retrieve more than 100,000 UIDs, submit multiple esearch requests while incrementing the value of **retstart** (see Application 3).

### 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 and 'first+author' and 'pub+date' for PubMed. 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.

### 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]
```

## 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/efetch.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/efetch.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/efetch.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/efetch.fcgi?db=pmc&term=stem+cells  
+AND+free+fulltext\[filter\]
```

Search in Nucleotide for all tRNAs:

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

Search in Protein for a molecular weight range:

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

```
epost.fcgi?db=protein&id=15718680,157427902,119703751
```

### 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](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.

```
esummary.fcgi?db=protein&id=15718680,157427902,119703751
```

### 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](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](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.

`efetch.fcgi?db=protein&id=15718680,157427902,119703751`

### 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 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, HTML 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
<b>All Databases</b>		
Document summary	docsum	xml, <i>default</i>
List of UIDs in XML	uilest	xml
List of UIDs in plain text	uilest	text
<b>db = bioproject</b>		
Full record XML	xml, <i>default</i>	xml, <i>default</i>
<b>db = biosample</b>		
Full record XML	full, <i>default</i>	xml, <i>default</i>
Full record text	full, <i>default</i>	text
<b>db = biosystems</b>		
Full record XML	xml, <i>default</i>	xml, <i>default</i>
<b>db = gds</b>		
Summary	summary, <i>default</i>	text, <i>default</i>
<b>db = gene</b>		
text ASN.1	null	asn.1, <i>default</i>
XML	null	xml
Gene table	gene_table	text
<b>db = homogene</b>		
text ASN.1	null	asn.1, <i>default</i>

Table 1 continues on next page...

Table 1 continued from previous page.

XML	<i>null</i>	xml
Alignment scores	alignmentscores	text
FASTA	fasta	text
HomoloGene	homologene	text
<b>db = mesh</b>		
Full record	full, <i>default</i>	text, <i>default</i>
<b>db = nlmcatalog</b>		
Full record	<i>null</i>	text, <i>default</i>
XML	<i>null</i>	xml
<b>db = nuccore, nucest, nucgss, protein or popset</b>		
text ASN.1	<i>null</i>	text, <i>default</i>
binary ASN.1	<i>null</i>	asn.1
Full record in XML	native	xml
Accession number(s)	acc	text
FASTA	fasta	text
TinySeq XML	fasta	xml
SeqID string	seqid	text
<b>Additional options for db = nuccore, nucest, nucgss or popset</b>		
GenBank flat file	gb	text
GBSeq XML	gb	xml
INSDSeq XML	gbc	xml
<b>Additional option for db = nuccore and protein</b>		
Feature table	ft	text
<b>Additional option for db = nuccore</b>		
GenBank flat file with full sequence (contigs)	gbwithparts	text
CDS nucleotide FASTA	fasta_cds_na	text
CDS protein FASTA	fasta_cds_aa	text
<b>Additional option for db = nucest</b>		
EST report	est	text
<b>Additional option for db = nucgss</b>		
GSS report	gss	text
<b>Additional options for db = protein</b>		
GenPept flat file	gp	text

Table 1 continues on next page...

Table 1 continued from previous page.

GBSeq XML	gp	xml
INSDSeq XML	gpc	xml
Identical Protein XML	ipg	xml
<b>db = pmc</b>		
XML	<i>null</i>	xml, <i>default</i>
MEDLINE	medline	text
<b>db = pubmed</b>		
text ASN.1	<i>null</i>	asn.1, <i>default</i>
XML	<i>null</i>	xml
MEDLINE	medline	text
PMID list	uilst	text
Abstract	abstract	text
<b>db = sequences</b>		
text ASN.1	<i>null</i>	text, <i>default</i>
Accession number(s)	acc	text
FASTA	fasta	text
SeqID string	seqid	text
<b>db = snp</b>		
text ASN.1	<i>null</i>	asn.1, <i>default</i>
XML	<i>null</i>	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	uilst	text or xml
<b>db = sra</b>		
XML	full, <i>default</i>	xml, <i>default</i>
<b>db = taxonomy</b>		
XML	<i>null</i>	xml, <i>default</i>
TaxID list	uilst	text or xml
<b>db = clinvar</b>		

Table 1 continues on next page...

Table 1 continued from previous page.

ClinVar Set	clinvarset	xml, <i>default</i>
UID list	uilst	text or xml
<b>db = gtr</b>		
GTR Test Report	gtracc	xml, <i>default</i>

### 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 sequence.

### 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 sequence.

### 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/Nucline

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=nucleotide&id=21614549&strand=1&seq\\_start=1&seq\\_stop=100&rettype=fasta&retmode=text](https://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=nucleotide&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=nucleotide&id=21614549&strand=2&seq\\_start=1&seq\\_stop=100&rettype=fasta&retmode=text](https://eutils.ncbi.nlm.nih.gov/entrez/eutils/efetch.fcgi?db=nucleotide&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=nucleotide&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](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 nucore sequences have "related sequences" links.*

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

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

### **datatype**

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 **datatype**.

### **reldate**

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

### **mindate, maxdate**

Date range used to limit a link operation by the date specified by **datatype**. 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+allergies>

## 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 '|'.  
[1]

<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 output: 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](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<sup>1</sup> and Vadim Miller<sup>2</sup>

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](mailto: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.

---

<sup>1</sup> NCBI; Email: [sayers@ncbi.nlm.nih.gov](mailto:sayers@ncbi.nlm.nih.gov). <sup>2</sup> NCBI; Email: [miller@ncbi.nlm.nih.gov](mailto:miller@ncbi.nlm.nih.gov).

<sup>✉</sup> Corresponding author.



# Entrez Direct: E-utilities on the UNIX Command Line

Jonathan Kans, PhD<sup>1</sup>

Created: April 23, 2013; Updated: November 13, 2018.

## 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. Functions take search terms from command-line arguments. Individual operations are combined to build multi-step queries. Record retrieval and formatting normally complete the process.

EDirect also includes an argument-driven function that simplifies the extraction of data from document summaries or other results that are returned in structured XML format. This can eliminate the need for writing custom software to answer ad hoc questions. Queries can move seamlessly between EDirect commands and UNIX utilities or scripts to perform actions that cannot be accomplished entirely within Entrez.

### Installation

EDirect will run on UNIX and Macintosh computers that have the Perl language installed, and under the Cygwin UNIX-emulation environment on Windows PCs. To install the EDirect software, copy the following commands and paste them into a terminal window:

```
cd ~
/bin/bash
perl -MNet::FTP -e \
  '$ftp = new Net::FTP("ftp.ncbi.nlm.nih.gov", Passive => 1);
  $ftp->login; $ftp->binary;
  $ftp->get("/entrez/entrezdirect/edirect.tar.gz");'
gunzip -c edirect.tar.gz | tar xf -
rm edirect.tar.gz
builtin exit
export PATH=${PATH}:${HOME}/edirect >& /dev/null || setenv PATH "${PATH}:${HOME}/edirect"
./edirect/setup.sh
```

---

<sup>1</sup> NCBI; Email: kans@ncbi.nlm.nih.gov.

<sup>✉</sup> Corresponding author.

This downloads several scripts into an "edirect" folder in the user's home directory. The setup.sh script then downloads any missing Perl modules, and may print an additional command for updating the PATH environment variable in the user's configuration file. Copy that command, if present, and paste it into the terminal window to complete the installation process. The editing instructions will look something like:

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

## Entrez Direct Functions

Navigation functions support exploration within the Entrez databases:

- **esearch** performs a new Entrez search using terms in indexed fields.
- **elink** looks up neighbors (within a database) or links (between databases).
- **efilter** filters or restricts the results of a previous query.

Records can be retrieved in specified formats or as document summaries:

- **efetch** downloads records or reports in a designated format.

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

- **xtract** converts EDirect XML output into a table of data values.

Several additional functions are also provided:

- **info** obtains information on indexed fields in an Entrez database.
- **epost** uploads unique identifiers (UIDs) or sequence accession numbers.
- **nquire** sends a URL request to a web page or CGI service.

## Entering Query Commands

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.

In order to begin an Entrez search, the user types "esearch" and then enters the required -db (database) and -query arguments. A query on unqualified search terms:

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

constructs the appropriate Entrez Utilities (E-utilities) URL from the query terms and executes the search. EDirect handles many technical details behind the scenes (avoiding the learning curve normally required for E-utilities programming), and saves the results on the Entrez history server.

## 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. Piping `esearch` to `elink`:

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

will look up related articles (precomputed PubMed neighbors) of the initial results.

## 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. Continuing the query links to all protein sequences published in the neighbor articles:

```
esearch -db pubmed -query "opsin gene conversion" | \
elink -related | \
elink -target protein
```

The vertical bar pipe symbol also allows the query to continue on the next line.

## 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:

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

Levels of lycopene  $\beta$ -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  $\beta$ -cyclase (LCYB) is a key enzyme
involved directly in the synthesis of  $\alpha$ -carotene and  $\beta$ -carotene through
...
```

Using `efetch -format "medline"` instead produces a report that 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]
MDTLLKTPNKLDFIPQFHGFERLCSNNPYPSRVRLGVKKRAIKIVSSVSGSAALLDLPETKKENLDF
ELPLYDTSKSQVVDLAIVGGGPAGLAVAQQVSEAGLSVCSIDPSPKLIWPNNYGVWVDEFEAMDLLDCLD
TTWSGAVVYVDEGVKKDLSPYGRVNRKQLKSKMLQKCITNGVKFHQSKVTNVVHEEANSTVVCSDGVKI
QASVVLDTATGFSRCLVQYDKPYNPGYQVAYGIIAEVDGHPFDVDKMFMDWRDKHLDSYPELKERNSKIP
TFLYAMPFSSNRIFLEETSLVARPGLRMEDIQERMAARLKHLGINVKRIEEDERCVIPMGGPLPVLPQRV
VGIGGTAGMVHPSTGYMVARTLAAPIVANAIVRYLGSPSSNSLRGDQLSAEVWRDLWPIERRQREFFC
FGMDILLKLDLDTARRFFDAFFDLQPHYWHGFLSSRLFLPELLVFGLSLFSHASNTSRLEIMTKGTVPLA
KMINNLVQDRD
...

```

Additional FASTA -format variants are `fasta_cds_na`, `fasta_cds_aa`, and `gene_fasta`.

Sequence records can also be obtained as GenBank (-format gb) or GenPept (-format gp) flatfiles, 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; Camelinae; 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 ferlcsnnp py psrvrlgvkk raikivssvv sgsaalldlv
    61 petkkenldf elplydtsks qvvdlaivgg gpaglavaqq vseaglsvcs idpspkliwp
    121 nnygvwvdef eamdllldcld ttwsgavvyv degvkkdlr pygrvnrkql kskmlqkcit
    181 ngvkfhqskv tnvvheean tvvcsdgvki qasvldatg fsrclvqydk pynpgyqvay
    241 giiaevdghp fdvdkmvfmd wrdkhldsy elkerkskip tflyampfss nrifleetsl
    301 varpglrmed iqermaarlk hlginvkrie edercvipmg gplpvlpqr vvgiggtagmv
    361 hpstgymvar tlaaapivan aivrylgsp ssnlrgdqls aevwrdlwp errrqrfffc
    421 fgmdillklld ldatrrffda ffdlqphywh gflssrlflp ellvfglslf shasntsrl
    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"

```

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

```

efilter -days 60 -datetype PDAT
efilter -mindate 1990 -maxdate 1999 -datetype PDAT

```

restrict results to articles published in the previous two months or in the 1990s, respectively.

## 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
[ALL]	All Fields
[AUTH]	Author
[FAUT]	Author - First
[LAUT]	Author - Last
[PDAT]	Date - Publication
[FILT]	Filter
[JOUR]	Journal
[LANG]	Language
[MAJR]	MeSH Major Topic
[SUBH]	MeSH Subheading
[MESH]	MeSH Terms
[PTYP]	Publication Type
[WORD]	Text Word
[TITL]	Title
[TIAB]	Title/Abstract
[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
[ALL]	All Fields
[AUTH]	Author
[GPRJ]	BioProject
[ECNO]	EC/RN Number
[FKEY]	Feature key
[FILT]	Filter
[GENE]	Gene Name

[JOUR]	Journal
[KYWD]	Keyword
[MLWT]	Molecular Weight
[ORGN]	Organism
[PACC]	Primary Accession
[PROP]	Properties
[PROT]	Protein Name
[SQID]	SeqID String
[SLEN]	Sequence Length
[SUBS]	Substance Name
[WORD]	Text Word
[TITL]	Title
[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]
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.)

## 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 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 "NP_567004 [ACCN]" |
elink -related |
efilter -query "28000:30000 [MLWT]" |
elink -target structure |
efilter -query "0:2 [RESO]"

```

produces:

```

<ENTREZ_DIRECT>
  <Db>structure</Db>
  <WebEnv>NCID_1_545606712_172.16.22.25_5555_1348089299_358182861</WebEnv>
  <QueryKey>7</QueryKey>
  <Count>39</Count>
  <Step>5</Step>
</ENTREZ_DIRECT>

```

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

(The QueryKey value is 7 instead of 5 because each elink command obtains the record count by running a separate ESearch query immediately after the ELink operation.)

## 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.) For example, the query:

```

esearch -db protein -query "amyloid* [PROT]" |
elink -target pubmed |
esearch -db gene -query "apo* [GENE]" |
elink -target pubmed |
esearch -query "(#3) AND (#6)" |
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 ...
...

```

The use of (#3) AND (#6) instead of (#2) AND (#4) above reflects the need for each elink command to execute a separate ESearch query, which increments the QueryKey, in order to obtain the record count. The -label argument can be used to get around this artifact.

The label value is prefixed by a "#" symbol and placed in parentheses in the final search. Thus:

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

will return:

```
15299880
9235985
9141131
8421693
...
```

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

## Structured Data

### Advantages of XML Format

The ability to obtain Entrez records in structured 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 many pieces of information are 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 Data into Tabular Form

The xtract function uses command-line arguments to direct the selective conversion of XML data into a tab-delimited table. The -pattern argument divides the results into rows, while placement of data into columns is controlled by -element. A trivial example:

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

will print the number of records in the current query.

Xtract provides control over data conversion with a divide-and-conquer strategy using separate arguments for element selection, path exploration, conditional processing, and report formatting.

Element selection finds every occurrence of each indicated item, printing values as they are encountered. Exploration control limits selection by context, presenting specified objects one at a time. Conditional processing filters by content, requiring presence (or absence) of a particular data value in order to continue. Finally, custom formatting can override the normal tabular layout of the default output.

The details and ramifications of this flexible approach are discussed in the remainder of this section.

## Extraction Arguments

Selection arguments (-element, -first, and -last) extract and print data values from the indicated element names:

```
-element Id -first Name Title
```

Exploration arguments (-pattern, -group, -block, and -subset) limit data extraction to specified regions of the XML, visiting all relevant objects one at a time. This sets a context for data collection, eliminates the need to provide the full path to a data element, and uncouples the concept of "what to look for" from "where to find it":

```
-pattern DocumentSummary
-block Author
```

Each pattern can have multiple groups, each group can have multiple blocks, and each block can have multiple subsets. This design allows nested exploration of complex, hierarchical data to be controlled by a linear chain of command-line argument statements.

Conditional processing arguments restrict exploration statements by object name and value (-if and -unless) or item location (-position):

```
-if Source -equals "J Bacteriol"
-position first
```

These commands are issued immediately after an exploration argument.

Formatting arguments (-ret, -tab, -sep, -pfx, -sfx, and -def) allow extensive customization of the default row/column table presentation:

```
-pfx "\n[" -sfx "]" \t" -sep " " -tab "" -ret "\n\n" -def "--"
```

and apply to subsequent selection statements.

(The "\n" escape sequence indicates a line break, while "\t" specifies a tab character.)

## XML Document Summaries

Entrez provides a document summary in structured XML format for every record. Piping a query to "efetch -format docsum":

```
esearch -db pubmed -query "Garber ED [AUTH] AND PNAS [JOUR]" |
elink -related |
efilter -query "mouse" |
efetch -format docsum
```

will generate an XML document summary set:

```
<DocumentSummarySet status="OK">
  <DbBuild>Build150407-2207m.3</DbBuild>
  <DocumentSummary>
    <Id>19650888</Id>
    <PubDate>2009 Aug 3</PubDate>
    <EPubDate>2009 Aug 3</EPubDate>
    <Source>BMC Microbiol</Source>
    <Authors>
      <Author>
        <Name>Cano V</Name>
        <AuthType>Author</AuthType>
        <ClusterID></ClusterID>
      </Author>
      <Author>
        <Name>Moranta D</Name>
        <AuthType>Author</AuthType>
      </Author>
    </Authors>
  </DocumentSummary>
  ...
</DocumentSummarySet>
```

Piping the document summary output to:

```
xtract -outline
```

will give an indented overview of the XML structure hierarchy:

```
DbBuild
DocumentSummary
  Id
  PubDate
  EPubDate
  Source
  Authors
    Author
      Name
      AuthType
      ClusterID
    Author
      Name
    ...
```

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

```
esearch -db pubmed -query "Garber ED [AUTH] AND PNAS [JOUR]" |
elink -related |
efilter -query "mouse" |
efetch -format docsum |
xtract -pattern DocumentSummary -element Id SortFirstAuthor Title
```

returns the PubMed identifier (PMID), first author name, and article title:

```
19650888      Cano V      Klebsiella pneumoniae triggers a cytotoxic ...
19262028      Suto J      Metabolic consequence of congenital asplenia ...
19248821      Fukumoto N  Hypoalgesic behaviors of P/Q-type voltage- ...
18822497      Trishin AV   [Protective activity of secreted proteins of ...
18582214      Singh A      Generation and characterization of monoclonal ...
...
```

Using xtract "-synopsis" instead of -outline show the full path to each element. Piping those results to "sort-uniq-count" (see below) produces a table of unique path counts.

## Processing Results with UNIX Utilities

A tab-delimited table can be processed by many UNIX utilities. For example:

```
esearch -db pubmed -query "Garber ED [AUTH] AND PNAS [JOUR]" |
elink -related |
efilter -query "mouse" |
efetch -format docsum |
xtract -pattern DocumentSummary -element Id SortFirstAuthor Title |
sort -t '$\t' -k 2,2f -k 3,3f
```

sorts the results of the previous example by author name and then (if there are multiple publications by the same author) alphabetically by title:

```
17474906      Benghezal M   Inhibitors of bacterial virulence ...
19650888      Cano V      Klebsiella pneumoniae triggers a cytotoxic ...
17102561      Chatterjee S  How reliable are models for malaria vaccine ...
17371870      Clements A   Secondary acylation of Klebsiella ...
17142396      Fresno S    A second galacturonic acid transferase is ...
16735743      Fresno S    The ionic interaction of Klebsiella ...
...
```

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. (The following two functions are now included as scripts with the EDirect software.) For example:

```
WordAtATime() {
  sed 's/[^a-zA-Z0-9]/ /g; s/^ *// ' |
  tr 'A-Z' 'a-z' |
  fmt -w 1
```



```

}
alias word-at-a-time='WordAtATime'

SortUniqCountRank() {
    sort -f |
    uniq -i -c |
    perl -pe 's/\s*(\d+)\s(.+)/$1\t$2/' |
    sort -t $'\t' -k 1,1nr -k 2f
}
alias sort-uniq-count-rank='SortUniqCountRank'

```

Titles can be passed to a pair of these UNIX alias commands:

```

esearch -db pubmed -query "Casadaban transposition immunity" |
elink -related |
efetch -format docsum |
xtract -pattern DocumentSummary -element Title |
word-at-a-time |
sort-uniq-count-rank

```

to generate a table of word occurrence counts, sorted by frequency:

```

296    of
175    the
114    transposition
102    and
94     in
93     mu
83     a
61     dna
61     tn3
55     transposon
...

```

## Output Format Customization

The line break between `-pattern` objects can be overridden with `-ret`, and the tab character between fields can be replaced by `-tab`.

The `-sep` argument is used to distinguish multiple elements of the same type and control their separation independently of the `-tab` argument. For example:

```

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

```

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

```

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

```

The `-sep` value also applies to unrelated `-element` items that are grouped with commas. Otherwise the `-tab` value delineates individual fields.

Groups or fields are preceded by the `-pfx` value and followed by the `-sfx` value, both of which are initially empty.

Missing data values can be marked by the `-def` argument. For example:

```
esearch -db pubmed -query "deuteranopia" |
efetch -format xml |
xtract -pattern PubmedArticle -def "-" \
-first MedlineCitation/PMID Author/LastName Keyword
```

inserts a dash in a column where the specified element is missing.

## Pubmed Article XML Records

The `PubmedArticle` object (for `-db pubmed`) has a more detailed structure than the `DocumentSummary`:

```
esearch -db pubmed -query "tetrachromacy" |
efetch -format xml |
xtract -outline
```

More information is fielded, including author names, dates, and the abstract:

```
PubmedArticle
  MedlineCitation
    PMID
    DateCompleted
      Year
      Month
      Day
    DateRevised
      Year
      Month
      Day
    Article
      Journal
        ISSN
      JournalIssue
        Volume
        Issue
        PubDate
          Year
          Month
          Day
      Title
      ISOAbbreviation
    ArticleTitle
    Pagination
      MedlinePgn
    Abstract
```

```

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

```

Using this information to craft a new xtract statement:

```

esearch -db pubmed -query "tetrachromacy" |
efetch -format xml |
xtract -pattern PubmedArticle -element MedlineCitation/PMID LastName

```

results in a table of all authors for each record:

```

23393278    Sabbah      Troje      Gray      Hawryshyn
20884587    Jordan      Deeb       Bosten    Mollon
18230593    Koshitaka   Kinoshita  Vorobyev  Arikawa
17685813    Wachtler    Doi        Lee       Sejnowski
...

```

(Note that "-element MedlineCitation/PMID" uses the "Parent/Child" construct to prevent the display of additional PMID items that may occur later in CommentsCorrections objects.)

The -first or -last arguments can be used instead of -element, if appropriate.

## Exploration of XML Sets

Individual PubmedArticle objects can be retrieved directly by efetch:

```
efetch -db pubmed -id 20643751 -format xml
```

The resulting XML has authors with separate fields for last name and initials:

```

...
<AuthorList>
  <Author>
    <LastName>Inamdar</LastName>
    <ForeName>Arati A</ForeName>
    <Initials>AA</Initials>
  </Author>
  <Author>
    <LastName>Masurekar</LastName>
    <ForeName>Prakash</ForeName>
    <Initials>P</Initials>
  </Author>
  <Author>

```

```

    <LastName>Bennett</LastName>
    <ForeName>Joan Wennstrom</ForeName>
    <Initials>JW</Initials>
  </Author>
</AuthorList>
...

```

Without being given any guidance about context, an `-element` statement with "Initials" and "LastName" arguments:

```

efetch -db pubmed -id 1413997,6301692,781293 -format xml |
xtract -pattern PubmedArticle -element MedlineCitation/PMID \
  -element Initials LastName

```

will explore the current record for each argument separately, and thus print all author initials followed by all author last names:

```

1413997  RK    CR    JS    Mortimer    Contopoulou    King
6301692  MA    NR    Krasnow    Cozzarelli
781293   MJ    Casadaban

```

Inserting a `-block` statement redirects data exploration to consider each author one at a time. Subsequent `-element` statements only see the current author's values:

```

efetch -db pubmed -id 1413997,6301692,781293 -format xml |
xtract -pattern PubmedArticle -element MedlineCitation/PMID \
  -block Author -element Initials LastName

```

which restores the correct association of initials and last name:

```

1413997  RK    Mortimer    CR    Contopoulou    JS    King
6301692  MA    Krasnow    NR    Cozzarelli
781293   MJ    Casadaban

```

Adding a `-sep` statement to replace the normal tab between group members, and using a comma to combine the two arguments ("Initials,LastName") into a group:

```

efetch -db pubmed -id 1413997,6301692,781293 -format xml |
xtract -pattern PubmedArticle -element MedlineCitation/PMID \
  -block Author -sep " " -element Initials,LastName

```

results in more desirable formatting of author names:

```

1413997  RK Mortimer    CR Contopoulou    JS King
6301692  MA Krasnow    NR Cozzarelli
781293   MJ Casadaban

```

The first or last structured object can be selected by the `-position` statement:

```

efetch -db pubmed -id 1413997,6301692,781293 -format xml |
xtract -pattern PubmedArticle -element MedlineCitation/PMID \
  -block Author -position first -sep " " -element Initials,LastName

```

## Exploring Separate XML Regions

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.

Combining independent fields with commas allows them to be treated as sets. The tab that normally separates these can be replaced with a `-sep` argument:

```
efetch -db pubmed -id 6092233,4640931,4296474 -format xml |
xtract -pattern PubmedArticle -element MedlineCitation/PMID \
  -block AuthorList -sep "/" -element LastName "#Author" \
  -block PubDate -sep " " -element Year,Month MedlineDate |
sort -t $'\t' -k 3,3n -k 2,2f
```

This generates a table that allows easy parsing of author last names, counts the number of authors present, and prints the date each record was published, sorting the results by author count:

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

(Note 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 of Subsets Within XML Sets

Medical Subject Headings (MeSH terms) in a record may be assigned subheadings:

```
...
<MeshHeading>
  <DescriptorName>RNA, Messenger</DescriptorName>
  <QualifierName>genetics</QualifierName>
</MeshHeading>
<MeshHeading>
  <DescriptorName>Transcription, Genetic</DescriptorName>
```

```

</MeshHeading>
<MeshHeading>
  <DescriptorName>beta-Galactosidase</DescriptorName>
  <QualifierName>genetics</QualifierName>
  <QualifierName>metabolism</QualifierName>
</MeshHeading>
</MeshHeadingList>
...

```

Visiting each MeSH term with a `-block` statement, and adding a `-subset` statement within the `-block`, allows nested exploration of the subheadings for the current MeSH term:

```

efetch -db pubmed -id 6162838 -format xml |
xtract -pattern PubmedArticle -tab "" -element MedlineCitation/PMID \
  -block MeshHeading -pfx "\n" -tab "" -element DescriptorName \
  -subset QualifierName -pfx " / " -tab "" -element QualifierName

```

and creates a list of MeSH terms with associated subheadings:

```

6162838
Base Sequence
DNA, Recombinant
Escherichia coli / genetics
...
RNA, Messenger / genetics
Transcription, Genetic
beta-Galactosidase / genetics / metabolism

```

## Selection of Attributes

The MeSH term and subheading fields actually have major topic attributes:

```

...
<MeshHeading>
  <DescriptorName MajorTopicYN="N">beta-Galactosidase</DescriptorName>
  <QualifierName MajorTopicYN="Y">genetics</QualifierName>
  <QualifierName MajorTopicYN="N">metabolism</QualifierName>
</MeshHeading>
...

```

that can be selected as "DescriptorName@MajorTopicYN" or "@MajorTopicYN":

```

efetch -db pubmed -id 6162838 -format xml |
xtract -pattern PubmedArticle -tab "" -element MedlineCitation/PMID \
  -block MeshHeading -pfx "\n" -sep "|" -tab "" \
  -element DescriptorName@MajorTopicYN,DescriptorName \
  -subset QualifierName -pfx " / " -sep "|" -tab "" \
  -element "@MajorTopicYN,QualifierName"

```

The major topic value is placed before each MeSH term or subheading:

```

6162838
N|Base Sequence
Y|DNA, Recombinant

```

```
N|Escherichia coli / N|genetics
...
N|RNA, Messenger / Y|genetics
N|Transcription, Genetic
N|beta-Galactosidase / Y|genetics / N|metabolism
```

The results can be processed by the UNIX stream editor "sed":

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

to display an asterisk for major ("starred" MeSH term) concepts:

```
6162838
Base Sequence
*DNA, Recombinant
Escherichia coli / genetics
...
RNA, Messenger / *genetics
Transcription, Genetic
beta-Galactosidase / *genetics / metabolism
```

## Recording Values in Variables

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

```
efetch -db pubmed -id 1413997,6301692,781293 -format xml |
xtract -pattern PubmedArticle -PMID MedlineCitation/PMID \
-block Author -element "&PMID" \
-sep " " -tab "\n" -element Initials,LastName
```

produces a list of authors, with the PMID in the first column of each row:

```
1413997    RK Mortimer
1413997    CR Contopoulou
1413997    JS King
6301692    MA Krasnow
6301692    NR Cozzarelli
781293     MJ Casadaban
```

## Variable Initialization

Variables can be initialized with a literal value in parentheses:

```
efetch -db pubmed -id 1413997,6301692,781293 -format xml |
xtract -pattern PubmedArticle -element MedlineCitation/PMID \
-block Author -sep " " -tab "" \
-element "&COM" Initials,LastName -COM "(, )"
```

This can be used as a placeholder to prevent missing data from shifting columns in a table, or to have additional control over output formatting:

```
1413997    RK Mortimer, CR Contopoulou, JS King
6301692    MA Krasnow, NR Cozzarelli
781293     MJ Casadaban
```

All variables are reset when the next record is processed.

## Conditional Processing

Xtract provides `-if` and `-unless` arguments that filter by element name or name plus data value. For example:

```
esearch -db pubmed -query "Cozzarelli NR [AUTH]" |
efetch -format xml |
xtract -pattern PubmedArticle -if "#Author" -eq 3 \
  -block Author -if LastName -is-not Cozzarelli \
    -sep ", " -tab "\n" -element LastName,Initials |
sort | uniq
```

will select papers with exactly 3 authors and print the coauthor names:

```
Ackerman, RS
Adams, DE
Alexandrov, AI
Arimondo, PB
Bauer, WR
...
```

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 "\-"
```

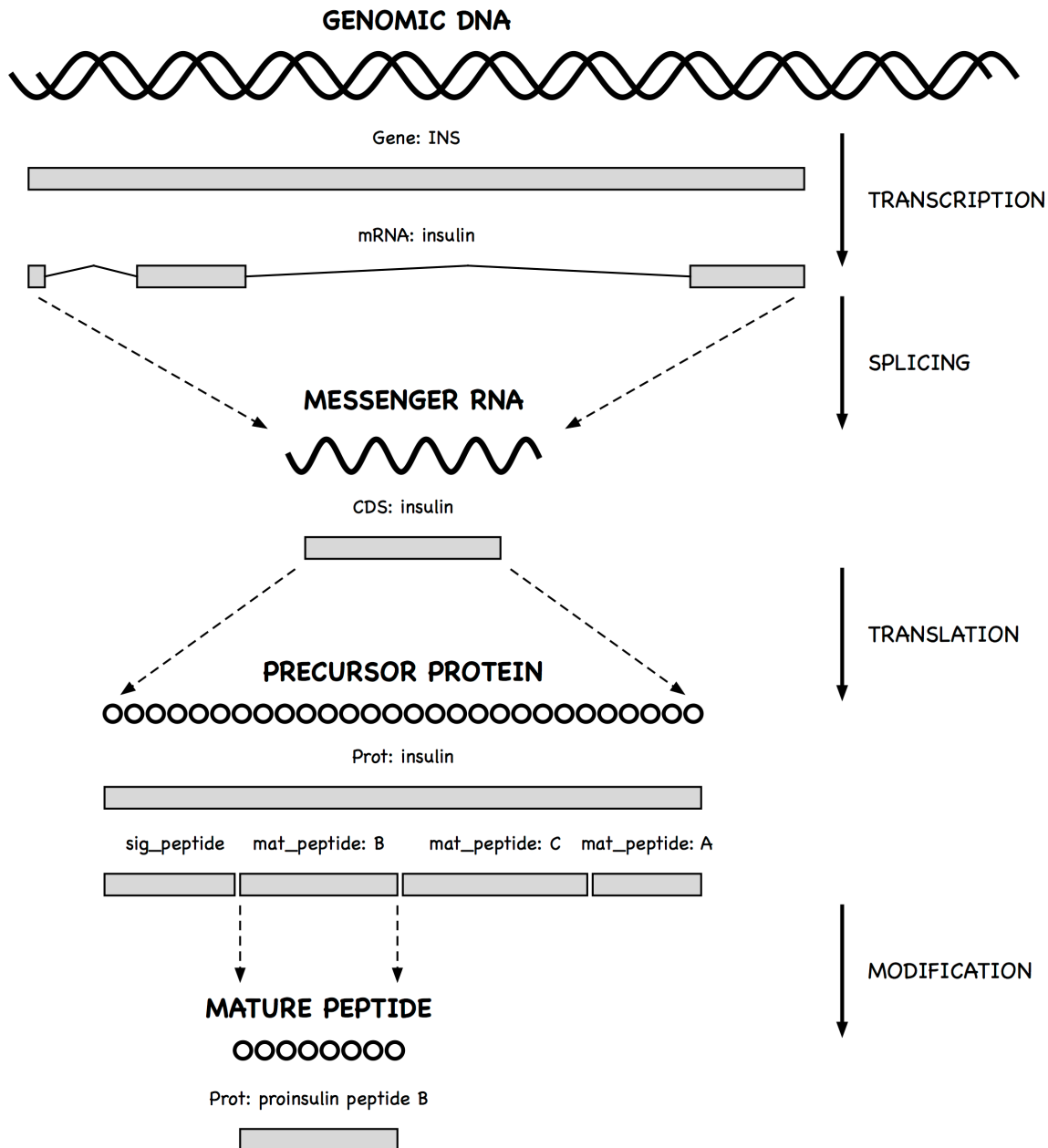
Parallel `-if` and `-unless` statements can be used to provide a more complex response to alternative conditions that includes nested exploration.

## Sequence Records

### NCBI Data Model for Sequence Records

The NCBI represents sequence records in a data model that 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. Features carry information about the biology of a given region, with a location that refers to specific intervals on a particular sequence. Some features may also point to the product sequence of a particular transformation.





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.

Features display specific biological annotation in qualifiers. For example, the name of a gene is shown in the /gene qualifier. 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>
        ...
```

INSDSeq XML presents 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"

```

CDS          /calculated_mol_wt=1357
             /peptide="GCCSDPRCAWRC"
             1..17
             /coded_by="AY159318.1:<1..54"
             /note="nAChR antagonist"

```

in a structured feature table:

```

...
<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_qual>
    <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_qual>
</INSDFeature>
...

```

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. The xtract -insd argument simplifies this process, as shown below.

## Generating Qualifier Extraction Commands

Because obtaining specific qualifier values from INSDSeq XML is somewhat more complex than previous cases, the xtract -insd argument can be used to generate extraction instructions.

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 constructed query.

Providing an optional (complete/partial) location indication, a feature key, and then one or more qualifier names:

```
xtract -insd complete mat_peptide "%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 \
-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 peptide \
-element INSDQualifier_value
```

Incorporating the `xtract -insd` command in a query for marine snail venom peptides:

```
esearch -db pubmed -query "conotoxin" |
elink -target protein |
efilter -query "mat_peptide [FKEY]" |
efetch -format gpc |
xtract -insd complete mat_peptide "%peptide" product peptide
```

produces a table with columns for accession number, calculated peptide length, product name, and peptide sequence:

AG059814.1	32	dell13b conotoxin	DCPTSCPTTCANGWECCKGYPCVRQHCSGCNH
AA033169.1	16	alpha-conotoxin GIC	GCCSH PACAGNNQHIC
ADB65788.1	20	conotoxin Cal 16	LEMQGCVCNANAKFCCGEGR
AAN78128.1	12	alpha-conotoxin ImI	GCCSDPRCAWRC
AAF23167.1	31	BeTX toxin	CRAEGTYCENDSQCLNECCWGGCGHPCHHP
ADB65789.1	20	conotoxin Cal 16	LEMQGCVCNANAKFCCGEGR
AAN78279.1	21	conotoxin Vx-II	WIDPSHYCCCGGGCTDDCVNC
ABW16858.1	15	marmophin	DWEYHAHPKPNSFWT
...			

Piping the results to a series of UNIX commands:

```
grep -i conotoxin |
awk -F '\t' -v 'OFS=\t' '{if ( 10 <= $2 && $2 <= 30 ) print}' |
sort -t $'\t' -u -k 3,4 |
sort -t $'\t' -k 2,2n -k 3,3f |
cut -f 1,3- |
column -s $'\t' -t
```

filters by product name, limits the results to a specified range of peptide lengths, removes redundant accessions, sorts the table by peptide length, deletes the length column, and aligns the columns for cleaner printing:

AAN78128.1	alpha-conotoxin ImI	GCCSDPRCAWRC
AAN78127.1	alpha-conotoxin ImII	ACCSDRRRCRWRC
ADB43130.1	conotoxin Cal 1a	KCCKRHHGCHPCGRK
ADB43131.1	conotoxin Cal 1b	LCCKRHHGCHPCGRT
AAO33169.1	alpha-conotoxin GIC	GCCSH PACAGNNQHIC
ADB43128.1	conotoxin Cal 5.1	DPAPCCQHPIETCCRR
AAD31913.1	alpha A conotoxin Tx2	PECCSH PACNVDHPEICR
ADB43129.1	conotoxin Cal 5.2	MIQRSQCCAVKKNCHVG
ADD97803.1	conotoxin Cal 1.2	AGCCPTIMYKTGACRTNRCR
ADB65789.1	conotoxin Cal 16	LEMQGCVCNANAKFCCGEGR
AAD31912.1	alpha A conotoxin Tx1	PECCSDPRCNSSHP ELCGRR
AAN78279.1	conotoxin Vx-II	WIDPSHYCCCGGGCTDDCVNC
ADB43125.1	conotoxin Cal 14.2	GCPADCPNTCDSSNKCS PGFPG
ADD97802.1	conotoxin Cal 6.4	GCWLCLGPNACCRG SVCHDYCPR
CAH64846.1	four-loop conotoxin	CRPSGSPCGVTSICCGRCSR GKCT
AAD31915.1	O-superfamily conotoxin TxO2	CYDSGTSCNTGNQCCSGWCIFVCL
AAD31916.1	O-superfamily conotoxin TxO3	CYDGGTSCDSGIQCCSGWCIFVCF
AAD31920.1	omega conotoxin SVIA mutant 1	CRPSGSPCGVTSICCGRCSR GKCT
AAD31921.1	omega conotoxin SVIA mutant 2	CRPSGSPCGVTSICCGRCSR GKCT
ABE27010.1	conotoxin fel4.1	SPGSTICKMACRTGN GHKYPFCNCR
ABE27011.1	conotoxin fel4.2	SSGSTVCKMMCR LGYGHLYPSCGCR
ABE27007.1	conotoxin p114.1	GPGSAICNMACRLG QGHMYPFCNCN
ABE27008.1	conotoxin p114.2	GPGSAICNMACRLEH GHLYPFCHCR
ABE27009.1	conotoxin p114.3	GPGSAICNMACRLEH GHLYPFCNCD
...		

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 -t $'\t' -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
XP_006499075.1	Mus musculus	C57BL/6J
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 tcccagggg
181 ccccaactcc ctggtaacct ctgaccttcc gtgtcctatc ctcccttctt 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 0-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:

```
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`, allow direct use of the 0-based coordinates:

```
efetch -db nuccore -format gb -id NC_000013.11 \
  -chr_start 32315479 -chr_stop 32399671
```

and eliminate the need for writing a UNIX shell command to increment the two values.

Xtract 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 ChrStart and ChrStop fields are 0-based, sees that the desired output is 1-based, and translates the command to convert coordinates using the `-inc` argument. 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
```

## Complex Objects

### 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/\*" construct is used to visit the individual components, even though they may have different names. Piping the XML 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 `TaxaSet` (-db taxonomy) and `Gene-commentary` in `Entrezgene` (-db gene). Thus, they can have 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:

```

1      <TaxaSet>
      <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 actually closed by the `</Taxon>` stop tag on line 6, and not by the first `</Taxon>` encountered on line 3.

When a recursive object is given to an exploration command, selection of data using the `-element` command:

```
efetch -db taxonomy -id 9606,7227,10090 -format xml |
xtract -pattern Taxon \
  -element TaxId ScientificName GenbankCommonName Division
```

does not examine fields in the internal objects, and returns information only for the main entries:

9606	Homo sapiens	human	Primates
7227	Drosophila melanogaster	fruit fly	Invertebrates
10090	Mus musculus	house mouse	Rodents

The `"*/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-slash prefix:

```
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
...
```

## Advanced Topics

### Storing Common Phrases in Alias Files

Long or complicated search phrases can be saved in a file to avoid having to retype (or copy and paste) the full text for each query. Each line of the file has a shortcut keyword, a tab character, and the expanded search term. Shortcuts are referenced by placing them in parentheses after prefixing with a pound ("**#**") sign.

For example, given a file named `"q_aliases"` containing:

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

the `esearch` line in:

```
esearch -alias q_aliases -db nlmcatalog -query "Science (#jour_filt)" |
efetch -format docsum |
```

```
xtract -pattern DocumentSummary -element ISOAbbreviation \
  -subset ISSNInfo -sep "|" -element issn,issntype
```

will be expanded to:

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

with the query producing:

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

An alias file can also be read in a separate instruction at the beginning of a pipeline or script:

```
eproxy -alias q_aliases
```

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

## Additional EDirect Options

ESearch and EFilter can be given a -sort argument to specify the order of results when the records are retrieved:

```
esearch -db pubmed -query "opsin gene conversion" -sort "last author" |
efetch -format docsum |
xtract -pattern DocumentSummary -element Id LastAuthor PubDate Title
```

ELink can return links to the citation list using "-name pubmed\_pubmed\_citedin", but only for publications with full text deposited in PubMed Central (PMC). For example, the query:

```
esearch -db pubmed -query "Beadle GW [AUTH]" |
elink -related -name pubmed_pubmed_citedin |
efetch -format docsum |
xtract -pattern Author -element Name |
sort-uniq-count-rank |
head -n 10
```

produces a ranked list of the ten most cited authors:

```
13      Beadle GW
8        Ephrussi B
8        Glass NL
7        Hawley RS
7        Mitchell MB
7        PERKINS DD
7        Tatum EL
6        Mitchell HK
6        YANOFSKY C
5        Langley CH
```

Similarly, "-name pubmed\_pubmed\_refs" returns an article's reference list, again for publications deposited in PMC.

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

```
esearch -db pubmed -query "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    ...
...
```

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

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

The elink -batch flag will bypass the Entrez history mechanism for large queries.

## Xtract Special Topics

Self-closing tags of the standard form:

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

or alternative form:

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

have no text content and thus cannot be selected with an -element command. If the tag contains an attribute:

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

it can be selected by matching on the specified value:

```
-group Seq-interval_strand \
  -block Seq-interval_strand -if Na-strand@value -equals plus -lbl "+" \
  -block Seq-interval_strand -if Na-strand@value -equals minus -lbl "-"
```

The `-pattern`, `-group`, `-block`, and `-subset` commands provide a nested hierarchy of loop organizers for exploration of XML objects. Each pattern can contain multiple groups, each group can encompass multiple blocks, and each block can have multiple subsets.

Use of different argument names allows a linear representation of loop nesting, and provides sufficient flexibility to identify and extract arbitrary data from XML records in Entrez.

Sketching in 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
```

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

```
for each Pubmed record {
  for each Author {
    print Initials LastName
  }
  for each MeSH term {
    if Subheadings are present {
      print Term Name
      for each Subheading {
        print Subheading Name
      }
    }
  }
}
```

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

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

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

## Querying External Web Services

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

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

reads the URL and then tag/value pairs to generate an E-utilities query:

```
https://eutils. . . .gov/entrez/eutils/esearch.fcgi?db=pubmed&term=insulin
```

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

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

is converted to:

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

Multiple values between tags are combined with commas. Thus:

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

is transformed into:

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

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

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

will be sent as:

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

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

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

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

so the command:

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

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

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

```
xtract -format
```

## Automation

### Entrez Direct Commands Within Scripts

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

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

For example, executing a script file containing:

```
#!/bin/bash -norc

parse_fields() {
    echo "$1" |
    xtract -pattern Field \
        -pfx "[" -sfx "]" -element Name \
        -pfx " " -sfx " " -element FullName Description |
    sort -t $'\t' -k 2,2f | column -s $'\t' -t
}

dbs=`einfo -dbs | sort`

for db in $dbs
do
    eix=`einfo -db $db`
    flds=`parse_fields "$eix"`

    echo "$db"
    echo " "
    echo "$flds"
    echo " "

    sleep 1
done
```

will obtain the list of Entrez databases:

```
annotinfo
assembly
bioproject
...
```

and then return the abbreviations, names, and descriptions of indexed search fields, for each individual database:

```
...
mesh

[ALL]   All Fields           All terms from all searchable fields
[FILT]   Filter              Limits the records
[MESH]   MeSH Terms          MeSH Terms
[MHUI]   MeSH Unique ID      NLM MeSH Browser Unique ID
[MULT]   Multi               Multi
[PREV]   Previous Indexing   Previous Indexing
[TYPE]   Record Type         Record type
[REG]    Registry Number     Registry Number
[NOTE]   Scope Note          Scope Note
[ALSO]   See Also            See Also
[SUBS]   Substance Name      Substance Name
[WORD]   Text Word           Free text
[TN]     Tree Number         Tree Number
[UID]    UID                 Unique number assigned to publication
...
```

The shell script command:

```
sleep 1
```

adds a one second delay between steps in a loop, and can be used to help prevent overuse of the Entrez 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 "$org [LNCE] AND family [RANK]" < /dev/null |
  efetch -format docsum |
  xtract -pattern DocumentSummary -lbl "$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 "$org [LNCE] AND family [RANK]" |
  efetch -format docsum |
  xtract -pattern DocumentSummary -lbl "$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

Because of technical limits in the Entrez link server, it may be necessary to perform an elink operation on a large set of records by using a function that splits unique identifiers or sequence accession numbers into smaller groups:

```
JoinIntoGroupsOf() {
    xargs -n "$@" echo |
    sed 's/ /,/g'
}
alias join-into-group-of='JoinIntoGroupsOf'
```

The following example will process 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'
```

## Local Data Cache

Entrez Direct users normally obtain selected data records with efetch, which makes calls to the efetch.fcgi network server. This solution 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.

## Random Access Archive

As an alternative, the entire set of PubMed records can be obtained by file transfer protocol, and can be kept current by retrieving daily update files. The release files can be

decompressed and piped through xtract for bulk analysis, but they do not provide random access to individual records.

Recent advances in computer file system technology now allow all of these published journal article citations to be repackaged as individual files on an inexpensive 500 gigabyte external solid state drive, using a hierarchy of folders to organize the 28 million PubMed records. This approach uses pairs of digits in the PMID as nodes in the directory structure, providing direct and rapid access to any record. For example, PMID 12345678 would be stored (as a compressed XML file) at /Archive/12/34/56/12345678.xml.gz.

EDirect now includes an archive-pubmed script to fully automate the download, update, extraction, and storage process. The path to the dedicated solid state drive is passed to the script on the command-line:

```
archive-pubmed -path /Volumes/alexandria
```

It can be rerun on a daily or weekly basis to keep the archive up to date. Once the archive is populated, PMIDs can be piped to the fetch-pubmed script in order to retrieve PubMedArticle XML records from the local data cache. fetch-pubmed requires the path to the Archive subdirectory in order to find the data. For example:

```
esearch -db pubmed -query "cancer AND regulation" |  
efetch -format uid |  
fetch-pubmed -path /Volumes/alexandria/Archive
```

The -path argument is not needed by either script if an environment variable is set up in the user's .bash\_profile configuration file:

```
export EDIRECT_PUBMED_MASTER=/Volumes/alexandria
```

and EDIRECT\_PUBMED\_MASTER is assumed to be set in subsequent examples.

## Local Query Index

A similar divide-and-conquer strategy can be used to create a local information retrieval index suitable for large data mining queries. A second new script, index-pubmed, automates this activity. (It is a superset of archive-pubmed, and can also be run at regular intervals.)

For selected PubMed text fields (the title and primary abstract), 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" would be located at /Postings/c/a/n/c/canc.trm. A query on cancer thus only needs to load a very small subset of the total index.

This design supports efficient Boolean expression evaluation (e.g., "(literacy AND numeracy) NOT (adolescent OR child)"), unrestricted wildcard truncation (e.g.,

"therap\*"), phrase queries (e.g., "selective serotonin reuptake inhibitor"), and proximity searches (e.g., "vitamin c ~ ~ common cold").

## Local Search Commands

The phrase-search script controls access to the local indices. The actual underlying work is done by the rchive program. Term counts are returned by -count (combining wildcard expansions into a single number) or by -counts (which expands wildcards and displays the individual terms and counts). Using -query returns the PMIDs that match the search expression.

For example, examining the term list with a truncated phrase:

```
phrase-search -count "catabolite repress*"
```

translates to:

```
rchive -path /Volumes/alexandria/Postings -count "catabolite repress*"
```

and returns the total counts of the individual words:

```
4325      catabolite
104977    repress*
```

Using -counts instead of -count returns the expanded terms and the individual postings counts:

```
4325      catabolite
12002     repress
1         repressa
5         repressable
1         repressae
1         repressant
2         repressants
1         repressc
1         represses
26322     repressed
1         repressedm
...
```

Searching with a phrase:

```
phrase-search -query "selective serotonin reuptake inhibitor*" |
fetch-pubmed
```

generates a list of PMIDs for records that contain the phrase, and then retrieves the relevant PubmedArticle XML from the local archive. Using -search instead of -query evaluates the same expression using words that have been processed through the Porter2 stemming algorithm.

## XML Processing

In addition to `-first` and `-last` commands, `xtract` has other `-element` variants that return transformed versions of the selected data values. It also has methods that make it easy to wrap extracted values in XML tags for further processing by `xtract`.

## Data Transformation

Numeric evaluation commands work on a set of data with the same tag names. For example, `-num` will count the number of named objects (with the `-element` `"#Author"` shortcut translated to `-num Author`). Integer values in XML objects can be added with `-sum`, and their arithmetic mean can be calculated with `-avg`. The number of characters in selected contents is returned by `-len` (with `-element` `"%Title"` implemented as `-len Title`).

Text and string commands work on individual string values. Sentences are split at punctuation marks with `-words`, or converted to upper case with `-upper`. The `-year` command returns the first four-digit token encountered in the data, so it works correctly with an integer `Year` or the semi-structured `MedlineDate` string. Applying `-year` to `"PubDate/*"` will examine the values of all internal objects, regardless of tag name, and is a general solution for obtaining the PubMed publication year.

The full set of `-element` variants can be seen by running `xtract -help`.

## Sequential Processing

Data analysis frequently involves several sequential steps of text or numeric processing. Examining the growth of PubMed abstracts over time, for example, requires isolating individual words per abstract, counting the words per article, filtering the results by year, and then computing the average word count per year. The individual operations are all supported by `xtract` functions, so wrapping intermediate values in XML can eliminate the need for a separate script to perform the calculations.

Running an `esearch` query on articles published in a chosen journal, limiting the results to articles with abstracts, obtaining the individual PMIDs, and fetching the records from the local data cache:

```
esearch -db pubmed -query "PNAS [JOUR]" -pub abstract |
efetch -format uid |
fetch-pubmed
```

returns an XML `PubmedArticleSet` containing just over 115,000 `PubmedArticle` records. The next step will be to extract the year of publication and the individual words from the article's abstract. The desired fields in the original XML are:

```
...
<PubDate>
  <Year>2018</Year>
  ...
</PubDate>
```

```
...
<Abstract>
  <AbstractText>Ammonia plays a key role in the ....</AbstractText>
</Abstract>
...
```

Piping the PubmedArticleSet to an initial version of the extraction commands:

```
xtract -stops -pattern PubmedArticle \
  -year "PubDate/*" \
  -pfc "\n" -sep "\n" -words Abstract/AbstractText
```

prints the year and each individual abstract word on a separate line:

```
2018
ammonia
plays
a
key
role
in
the
neutralization
of
atmospheric
acids
...
```

## Wrapping Results in XML

Customization arguments make it possible to wrap intermediate values (e.g., individual words per abstract, and word counts per article) back into XML form, so they can be sent to another xtract command for the next processing step. Piping the PubmedArticleSet to:

```
xtract -stops -head "<Set>" -tail "</Set>" -hd "<Rec>" -tl "</Rec>" \
  -pattern PubmedArticle \
  -pfx "<Year>" -sfx "</Year>" -year "PubDate/*" \
  -pfx "<Abst>" -sfx "</Abst>" -sep "</Abst><Abst>" \
  -words Abstract/AbstractText
```

allows extracted values to be written as structured XML fragments (shown reformatted for easier reading):

```
<Set>
  <Rec>
    <Year>2018</Year>
    <Abst>ammonia</Abst>
    <Abst>plays</Abst>
    <Abst>a</Abst>
    <Abst>key</Abst>
    <Abst>role</Abst>
    <Abst>in</Abst>
    <Abst>the</Abst>
    <Abst>neutralization</Abst>
```

```

<Abst>of</Abst>
<Abst>atmospheric</Abst>
<Abst>acids</Abst>
...

```

The `-wrp` argument is a convenience that sets all of the appropriate customization arguments, shown above, given just the XML object names:

```

xtract -stops -wrp Set,Rec -pattern PubmedArticle \
  -wrp "Year" -year "PubDate/*" \
  -wrp "Abst" -words Abstract/AbstractText

```

making it easy to wrap the intermediate values.

The results are then piped to the next step, which uses `-num` to count the number of words that were in the abstract of each article:

```

xtract -wrp Set,Pub -pattern Rec \
  -wrp "Year" -element Year \
  -wrp "Num" -num Abst

```

again wrapping the results as structured XML:

```

<Set>
  <Pub><Year>2018</Year><Num>198</Num></Pub>
  <Pub><Year>2018</Year><Num>167</Num></Pub>
  <Pub><Year>2018</Year><Num>242</Num></Pub>
...

```

Redirecting the output with `> countsByYear.xml` saves the results in a file, to be used for all subsequent processing.

## Selection of XML Subsets

The `xtract -select` argument allows record subsetting by data value, such as the year of publication. It acts as an `"-if"` statement, and is followed by conditionals, but with no `-element`, since its purpose is to pass along the entirety of all records that satisfy the condition.

This can be more convenient, and is significantly faster, than running a separate search query for each year, and it can look at data elements that are not indexed by Entrez. It is used in the loop below to limit the processed query results to one year at a time, passing the relevant subset to a second `xtract` command:

```

for yr in {1960..2018}
do
  cat countsByYear.xml |
  xtract -wrp 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      121
1970      119
1971      126
...
2016      207
2017      206
2018      205

```

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.

## Examples

Additional examples of using EDirect to answer ad hoc questions are shown in this section.

### 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

```

This search produces:

```

74      Lomonte B
73      Gutiérrez JM
49      Soares AM
48      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
```

reports the number of selected papers per year:

```
173    1979
102    1980
96     1978
92     1981
66     1983
...
```

## 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
```

returns the number of articles ranked by country (or region) of study:

```
567    United States
207    Spain
176    Great Britain
156    Germany
123    India
118    Europe
113    France
100    Taiwan
89     Japan
83     Thailand
75     Italy
74     England
...
```

## Research History

What is the historic pattern of publication on diphtheria, pertussis, and tetanus?

```
#!/bin/bash
result=""
for disease in diphtheria pertussis tetanus
do
  current=`for (( yr = 2010; yr >= 1900; yr -= 10 ))
  do
    esearch -db pubmed -query "$disease [TITL] AND $yr:${(yr+9)} [PDAT]" |
    xtract -pattern ENTREZ_DIRECT -lbl "${yr}s" -element Count
  done`
  heading=`echo -e "${disease:0:4}" | tr [a-z] [A-Z]`
```

```

current=`echo -e "Years\t$heading\n-----\t----\n$current"`
if [ -n "$result" ]
then
    result=`join -t $'\t' <(echo "$result") <(echo "$current")`
else
    result=$current
fi
done
echo "$result"

```

gives per-decade counts of relevant papers for each disease:

Years	DIPH	PERT	TETA
-----	----	----	----
2010s	577	1708	914
2000s	892	1966	1344
1990s	1150	2661	1615
1980s	780	1746	1485
1970s	749	698	1524
1960s	1152	635	2086
1950s	1226	491	1540
1940s	452	173	239
1930s	157	26	46
1920s	128	5	21
1910s	83	7	41
1900s	93	3	28

## Protein Homolog

Is there a mammalian equivalent of lycopene cyclase?

```

esearch -db protein -query \
    "lycopene beta cyclase [PROT] AND tomato [ORGN]" |
elink -related |
efetch -format gpc |
xtract -pattern INSDSeq -element INSDSeq_division |
sort-uniq-count-rank

```

In the resulting list of GenBank division codes:

905	BCT
856	ENV
609	PLN
197	CON
127	PAT
2	SYN

there are no similar sequences (protein neighbors) in the HUM, PRI, ROD, MAM, VRT, or INV divisions, so lycopene cyclase is not present in animals.

## 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 and saves the GenPept results to individual files named by their sequence accessions:

```

EFN61235.gpf
EFN80340.gpf
EGW08477.gpf
EKC18433.gpf
ELK28555.gpf

```

using the right angle bracket (">") UNIX output redirection character.

## Archaea Enzyme

Which archaebacteria have chloramphenicol acetyltransferase?

```

esearch -db protein -query \
  "chloramphenicol acetyltransferase [PROT] AND archaea [ORGN]" |
efetch -format gpc |
xtract -pattern INSDSeq -element INSDSeq_organism INSDSeq_definition |
grep -i chloramphenicol | cut -f 1 | sort -f | uniq

```

produces a list of organism names:

```

Methanobrevibacter ruminantium
Methanobrevibacter smithii
Methanosarcina acetivorans
...

```

## 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

```

This query uses geometric comparison (structure neighboring) 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 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
```

## Chromosome Locations

Where are mammalian calmodulin genes located?

```
esearch -db gene -query "calmodulin [PFN] AND mammalia [ORGN]" |
efetch -format docsum |
xtract -pattern DocumentSummary -MAP "(-)" -MAP MapLocation \
    -element Id Name "&MAP" ScientificName
```

The MAP variable is initialized with a literal 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
326597	CALM	-	Bos taurus
50663	Calm2	6q12	Rattus norvegicus
24244	Calm3	1q21	Rattus norvegicus
12315	Calm3	7 9.15 cM	Mus musculus
12314	Calm2	17 E4	Mus musculus
617095	CALM1	-	Bos taurus
396838	CALM3	6	Sus scrofa
...			

The -else command can also be used to insert placeholders for missing data:

```
esearch -db gene -query "calmodulin [PFN] AND mammalia [ORGN]" |
efetch -format docsum |
xtract -pattern DocumentSummary \
```

```
-if MapLocation -element Id Name MapLocation ScientificName \
  -else -element Id Name -lbl "\-" -element ScientificName
```

The `-def` command can achieve the same result for missing elements:

```
esearch -db gene -query "calmodulin [PFN] AND mammalia [ORGN]" |
efetch -format docsum |
xtract -pattern DocumentSummary \
  -def "-" -element Id Name MapLocation ScientificName
```

## 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 |
```

This search returns the chromosome accession and the (0-based) gene start and stop positions:

```
NC_000023.11      33339608      31119221
```

These are then passed to `efetch` in (0-based) `-chr_start` and `-chr_stop` arguments:

```
xargs -n 3 sh -c 'efetch -db nuccore -format gbc \
  -id "$0" -chr_start "$1" -chr_stop "$2"' |
```

which converts them to (1-based) `-seq_start` and `-seq_stop` arguments and retrieves an INSDSeq XML subset record for the indicated region. That contains a number of alternatively-spliced dystrophin mRNA and CDS features.

Data extraction computes the number of intervals for each mRNA location (corresponding to individual exons or UTRs), and obtains the transcript sequence accession, transcript length, and product name from qualifiers:

```
xtract -insd complete mRNA "#INSDInterval" \
  transcript_id "%transcription" product |
```

Final processing sorts by number of exons:

```
grep -i dystrophin |
sed 's/dystrophin, transcript variant //g' |
sort -k 2,2nr -k 4,4nr
```

resulting in a table of exon counts and transcript lengths:

NC_000023.11	79	NM_004010.3	14083	Dp427p2
NC_000023.11	79	NM_000109.3	14069	Dp427c
NC_000023.11	79	NM_004009.3	14000	Dp427p1
NC_000023.11	79	NM_004006.2	13993	Dp427m
NC_000023.11	78	XM_006724468.1	13920	X1
NC_000023.11	78	XM_006724469.1	13802	X2

```
NC_000023.11      77      XM_006724470.1      13881      X3
...
```

## Genome Range

What genes are in a given range on the human Y chromosome?

```
esearch -db gene -query "Homo sapiens [ORGN] AND Y [CHR]" |
efilter -status alive | efetch -format docsum |
xtract -pattern DocumentSummary -NAME Name -DESC Description \
  -block GenomicInfoType -if ChrLoc -equals Y \
  -min ChrStart,ChrStop -element "&NAME" "&DESC" |
sort -k 1,1n | cut -f 2- |
between-two-genes ASMT IL3RA
```

This query returns a table of gene names and descriptions, for the human "Y" chromosome, in the region between the ASMT and IL3RA genes:

IL3RA	interleukin 3 receptor subunit alpha
LOC101928032	uncharacterized LOC101928032
LOC101928055	uncharacterized LOC101928055
SLC25A6	solute carrier family 25 member 6
LOC105373102	uncharacterized LOC105373102
LINC00106	long intergenic non-protein coding RNA 106
ASMTL-AS1	ASMTL antisense RNA 1
ASMTL	acetylserotonin O-methyltransferase-like
P2RY8	purinergic receptor P2Y8
AKAP17A	A-kinase anchoring protein 17A
ASMT	acetylserotonin O-methyltransferase

(The "-if ChrLoc -equals Y" test is necessary because certain genes (e.g., IL9R) are present in the pseudoautosomal regions common to both X and Y chromosomes:

```
...
<GenomicInfo>
  <GenomicInfoType>
    <ChrLoc>Y</ChrLoc>
    <ChrAccVer>NC_000024.10</ChrAccVer>
    <ChrStart>57184100</ChrStart>
    <ChrStop>57197336</ChrStop>
    <ExonCount>10</ExonCount>
  </GenomicInfoType>
  <GenomicInfoType>
    <ChrLoc>X</ChrLoc>
    <ChrAccVer>NC_000023.11</ChrAccVer>
    <ChrStart>155997580</ChrStart>
    <ChrStop>156010816</ChrStop>
    <ExonCount>10</ExonCount>
  </GenomicInfoType>
</GenomicInfo>
...
```

with each gene copy annotated in its own GenomicInfoType block.)

## 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 $chr [CHR]" |
  efilter -query "alive [PROP] AND genotype protein coding [PROP]" |
  efetch -format docsum |
  xtract -pattern DocumentSummary -NAME Name \
    -block GenomicInfoType -if ChrLoc -equals "$chr" \
    -tab "\n" -element ChrLoc,"&NAME" |
  sort | uniq | cut -f 1 | sort-uniq-count-rank
done
```

returns a count of unique protein-coding genes per chromosome:

```
2067    1
1268    2
1071    3
755     4
873     5
1034    6
935     7
690     8
801     9
739    10
1288   11
1027   12
335    13
607    14
608    15
862    16
1181   17
277    18
1402   19
545    20
248    21
445    22
844    X
71     Y
13     MT
```

The range construct cannot be used for Roman numerals, so the equivalent query on *Saccharomyces cerevisiae* would need to explicitly list all chromosomes:

```
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]".

## 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 -t $'\t' -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_002695.1      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_017659.1      147060       Escherichia coli O83:H1 str. NRG 857C plasmid ...
...

```

The sed command removes extraneous text in the title (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 | 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 -t $'\t' -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

```

## Amino Acid Composition

What is the amino acid composition of human titin?

```

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 )
efetch -db protein -id "Q8WZ42.4" -format gpc |
xtract -pattern INSDSeq -element INSDSeq_sequence |
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

```

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

## Amino Acid Substitutions

What are the missense products of green-sensitive opsin?

```

ApplySNPs() {
    seq=" "
    last=" "

    while read rsid accn res pos
    do
        if [ "$accn" != "$last" ]
        then
            insd=$(efetch -db protein -id "$accn" -format gbc < /dev/null)
            seq=$(echo $insd | xtract -pattern INSDSeq -element INSDSeq_sequence)
            last=$accn
        fi

        pfx=" "
        sfx=" "
    done
}

```

```

echo ">$rsid [$accn $res@$pos]"
if [ $pos -gt 1 ]
then
  pfx=$(echo ${seq:0:$pos-1})
fi
if [ $pos -lt ${#seq} ]
then
  sfx=$(echo ${seq:$pos})
fi
echo "$pfx$res$sfx" | fold -w 50
done
}

esearch -db gene -query "CBD [GENE] AND human [ORGN]" |
elink -target snp |
efetch -format xml |
xtract -pattern Rs -pfx "rs" -RSID Rs@rsId \
  -block FxnSet -if @fxnClass -equals missense \
  -sep "." -element "&RSID" @protAcc,@protVer \
  @residue -tab "\n" -l-based @aaPosition |
sort -t '$'\t' -k 2,2 -k 4,4n -k 3,3f -k 1.3n | uniq |
ApplySNPs

```

The query returns an intermediate table of non-synonymous amino acid substitutions (with 0-based location coordinates) derived from single nucleotide polymorphisms:

```

rs1238141906      NP_000504.1      K      41
rs1189783086      NP_000504.1      L      43
rs1284438666      NP_000504.1      I      64
rs1223726997      NP_000504.1      T      65
...

```

The rows are then processed to produce protein sequences with the individual residue substitutions in upper case:

```

>rs1238141906 [NP_000504.1 K@41]
maqqswwslqlragrhpdqsyedstqssiftytntsnstrgpfKgpnyhiapr
wvyhltsvwmifvviavsvftnglvlaatkfkklrhplnwilvnlavadl
aetviastisvvnqvygyfvlghpmcvlegytvslcgitglwslaiiswe
...

```

### 3'UTR Sequences

What are the 3' UTR sequences for lycopene cyclase?

```

ThreePrimeUTRs() {
  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 \
    -last INSDInterval_to -element "&SEQ" "&PRD" |
  while read acc pos seq prd

```

```

do
  if [ $pos -lt ${#seq} ]
  then
    echo -e ">$acc 3'UTR: $((pos+1))..${#seq} $prd"
    echo "${seq:$pos}" | fold -w 50
  elif [ $pos -ge ${#seq} ]
  then
    echo -e ">$acc NO 3'UTR"
  fi
done
}

esearch -db nuccore -query "5.5.1.19 [ECNO]" |
efilter -molecule mrna -source refseq |
efetch -format gbc | ThreePrimeUTRs

```

prints the sequences immediately following the CDS stop codon:

```

>NM_001328461.1 3'UTR: 1737..1871 lycopene beta cyclase, chloroplastic
gatgaatatagagttactgtgttgtaagctaatacatcatactgatgcaag
tgcattatcacatttacttctgctgatgattgttcataagattatgagtt
agccatttatcaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa
>NM_001316759.1 3'UTR: 1628..1690 lycopene beta cyclase, chloroplastic
atccgagtaattcggaatcttgtccaattttatatagcctatattaatac
...

```

## 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 ChrStart ChrStop |
awk -F '\t' -v 'OFS=\t' '{print $1, $2, $3+1, $4+1}'

```

obtains a series of homologous genes, converting the gene coordinates to 1-based positions suitable for retrieving sequence regions:

```

5053      NC_000012.12      102917603      102838326
18478     NC_000076.6      87521795      87584137
38871     NT_037436.4      7760453       7763166
24616     NC_005106.4      28066639      28129772
378962    NC_007115.6      17420391      17402704
...

```

Given a shell script named "upstream.sh":

```

#!/bin/bash -norc

bases=1500

```

```

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

while read id accn start stop
do
    if [[ $start -eq 0 || $stop -eq 0 || $start -eq $stop ]]
    then
        echo "Skipping $id due to ambiguous coordinates"
        continue
    fi
    if [ $start -gt $stop ]
    then
        stop=$(( start + bases ))
        start=$(( start + 1 ))
        strand=2
    else
        stop=$(( start - 1 ))
        start=$(( start - bases ))
        strand=1
    fi
    rslt=`efetch -db nuccore -id $accn -format fasta \
        -seq_start $start -seq_stop $stop -strand $strand < /dev/null`
    echo "$rslt"
done

```

the data lines can be piped through:

```
upstream.sh 500
```

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

## Author Combinations

What are the authorship patterns among selected individuals?

The "coauthors.sh" script takes author name arguments to construct a custom data extraction command for analyzing research collaboration patterns:

```

#!/bin/bash -norc

if [ "$#" -lt 2 ]
then
    echo "Must supply at least two author names"
    exit 1
fi

query="xtract -pattern PubmedArticle -element MedlineCitation/PMID"

# append a -block statement for each author argument
for auth in "$@"

```

```

do
    query=`echo "$query -block Author -if LastName -equals \"\$auth\" \" \" \
        \"-sep \" \" -element LastName,Initials\"`
done

query=`echo "$query | sort -t \"'\t'\" -k 2f -k 1,1n\"`

if [ -t 0 ]
then
    # stand-alone command, print constructed query for later use
    echo "$query"
else
    # dynamically execute query on XML data piped to script
    res=`eval "$query"`
    echo "$res"
fi

```

If XML publication data are piped to the script, it will read the data and immediately execute the generated xtract query. Otherwise, if called as a stand-alone command, it will print the custom query instructions for later use.

Running the following command:

```

esearch -db pubmed -query "Casadaban MJ [AUTH] OR Berg CM [AUTH]" |
efetch -format xml |
./coauthors.sh Casadaban Groisman Berg Garber |
./extract-fuse.pl pubmed > author_patterns.htm

```

first produces an internal result table of PMIDs grouped by author combination:

```

...
7635839    Casadaban MJ
9634770    Casadaban MJ
1827084    Casadaban MJ    Groisman EA
2954879    Casadaban MJ    Groisman EA
3020001    Casadaban MJ    Groisman EA
3525518    Casadaban MJ    Groisman EA
3542967    Casadaban MJ    Groisman EA
6324195    Casadaban MJ    Groisman EA
3301525    Casadaban MJ    Groisman EA    Berg CM

```

The sorted lines are then piped to the "extract-fuse.pl" script:

```

#!/usr/bin/perl

my $max = scalar @ARGV;
if ( $max < 1 ) {
    die "Need argument for database\n";
}
my $db = $ARGV[0];

my $thisline = "";
my $laststr = "";
my $str = "";

```

```

my $uid = "";
my $uidlist = "";
my $count = 0;
my $base = "https://www.ncbi.nlm.nih.gov";

my $pfx = "";
while ($thisline = <STDIN>) {
    $thisline =~ s/\r//;
    $thisline =~ s/\n//;

    if ($thisline =~ /^([^\t+)\t(.+)\$/ ) {
        $uid = $1;
        $str = $2;

        if ( lc ($str) ne lc ($laststr) and $laststr ne "" ) {
            $laststr =~ s/\t/, /g;
            print "<p>( <a href=\"$base/$db/$uidlist\">";
            print " $count </a> ) - $laststr</p>\n";
            $pfx = "";
            $count = 0;
            $uidlist = "";
        }
        $laststr = $str;

        $uidlist .= "$pfx$uid";
        $pfx = ", ";
        $count++;
    }
}

if ( $laststr ne "" ) {
    $laststr =~ s/\t/, /g;
    print "<p>( <a href=\"$base/$db/$uidlist\">";
    print " $count </a> ) - $laststr</p>\n";
}

```

which combines them into PubMed query URLs, one for each author pattern:

```
https://www.ncbi.nlm.nih.gov/pubmed/1827084,2954879,3020001,...
```

Those are then wrapped, along with a record count, in the appropriate HTML tags for web display. If the resulting file is opened with a browser, it presents an argument-order-dependent view of author collaboration:

```

( 55 ) - Berg CM

( 10 ) - Berg CM, Berg DE

( 1 ) - BERG CM, GARBER ED

( 6 ) - Berg DE, Berg CM

( 39 ) - Casadaban MJ

```

```
( 6 ) - Casadaban MJ, Groisman EA
```

```
( 1 ) - Casadaban MJ, Groisman EA, Berg CM
```

Clicking on a hyperlinked record count number opens the document summary or individual article page, so the actual publications can be examined.

## Indexed 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
```

This produces a list of field abbreviations and names filtered by index type:

```
[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 '\n' -pfx '<p><a href="http://dx.doi.org/' \
  -sep '>' -sfx '</a></p>' -encode ArticleId,"&PMID"
```

extracts the DOIs and constructs the appropriate URL references:

```
<html><body>
<p><a href="http://dx.doi.org/10.1038/leu.2013.340">24496283</a></p>
<p><a href="http://dx.doi.org/10.1073/pnas.1310656110">23818607</a></p>
<p><a href="http://dx.doi.org/10.1126/science.1241318">23788787</a></p>
...
```

These intermediate lines are then piped through:

```
xtract -format
```

to produce a minimal HTML document with clickable links:



```
<?xml version="1.0"?>
<!DOCTYPE html>
<html>
  <body>
    <p>
      <a href="http://dx.doi.org/10.1038/leu.2013.340">24496283</a>
    </p>
    <p>
      <a href="http://dx.doi.org/10.1073/pnas.1310656110">23818607</a>
    </p>
    ...
  </body>
</html>
```

## Phrase Searching

Can phrase searching be simulated in Entrez?

The "entrez-phrase-search" script included with EDirect takes advantage of the fact that some short phrases are indexed in certain Entrez fields. Given an input phrase, the script generates overlapping pairs of adjacent words, separately queries on each pair to determine which are present in the pubmed title or abstract index, and keeps those that appear in at least 10 articles. Independent phrases are separated by a plus ("+") sign.

For example, running the following command:

```
entrez-phrase-search -db pubmed -field WORD \
  selective serotonin reuptake inhibitor + monoamine oxidase inhibitor
```

will generate word pairs from each phrase and run a query on each pair. The individual term counts are:

```
11343    selective serotonin
11892    serotonin reuptake
 6714    reuptake inhibitor
21722    monoamine oxidase
 3680    oxidase inhibitor
```

The combined query will return a search result with 36 articles, and these can then be retrieved by piping to efetch. The script in its current form will not match phrases with plurals (e.g., serotonin reuptake inhibitors) or hyphens (e.g., monoamine-oxidase inhibitor).

## Gene-Protein Links

What proteins are produced by a given gene?

Given a query in the gene database, the following commands:

```
esearch -db gene -query "beta galactosidase [PFN]" |
elink -target protein -name gene_protein_refseq -cmd neighbor |
xtract -pattern LinkSet -element Id
```

will show the gene ID in the first column and linked RefSeq protein UIDs in subsequent columns.

Piping the results to a Perl script named "gene-protein-links.pl" will read the identifiers and run separate efetch queries on the gene and protein databases:

```
#!/usr/bin/perl

while ($line = <STDIN>) {

    chomp ($line);
    @uids = split( /\t/, $line);
    $gene = $uids [0];
    $proteins = join (' ', @uids [1..$#uids]);

    $symbol = $data = '';

    $cmd = "efetch -format docsum -db gene -id $gene | ";
    $cmd .= "xtract -pattern DocumentSummary -element Name CommonName";
    open (CMD, "$cmd|");
    while (<CMD>) {
        $symbol .= $_;
    }
    close CMD;

    if ($proteins ne "") {
        $cmd = "efetch -format docsum -db protein -id $proteins | ";
        $cmd .= "xtract -pattern DocumentSummary -element Caption Slen Title";
        open (CMD, "$cmd|");
        while (<CMD>) {
            $data .= $_;
        }
        close CMD;
    }

    print "$symbol$data\n";
}
```

printing the gene symbol and organism common name, followed by the protein accessions, lengths, and titles:

```
GLB1          human
NP_001129074  546    beta-galactosidase isoform c preproprotein ...
NP_001073279  647    beta-galactosidase isoform b [Homo sapiens]
NP_000395     677    beta-galactosidase isoform a preproprotein ...

Glb1          house mouse
NP_033882     647    beta-galactosidase preproprotein [Mus musculus]

Glb1          Norway rat
NP_001101662  647    beta-galactosidase precursor [Rattus norvegicus]
...
```

## Bulk Downloads

How can the entire set of GenBank records for mammals be obtained?

```
ftp-ls ftp.ncbi.nlm.nih.gov genbank |
grep ".seq.gz" |
grep -e gbmam -e gbpri -e gbrod |
while read file
do
  ftp-cp ftp.ncbi.nlm.nih.gov genbank "$file"
  gunzip "$file"
  rm "$file"
done
```

will use the `ftp-ls` and `ftp-cp` scripts (included with the EDirect software) to retrieve and print GenBank flatfiles for human, primate, rodent, and other mammals:

```
GBMAM1.SEQ          Genetic Sequence Data Bank
                    February 15 2015

                    NCBI-GenBank Flat File Release 206.0

                    Other Mammalian Sequences (Part 1)

20709 loci, 155323216 bases, from 20709 reported sequences

LOCUS      AB000170          2732 bp    mRNA    linear    MAM ...
DEFINITION Sus scrofa mRNA for endopeptidase 24.16, complete cds.
ACCESSION  AB000170
VERSION    AB000170.1  GI:1783121
KEYWORDS   endopeptidase 24.16 type M3; endopeptidase 24.16 type M1.
SOURCE     Sus scrofa (pig)
  ORGANISM Sus scrofa
            Eukaryota; Metazoa; Chordata; Craniata; Vertebrata;
            Euteleostomi; Mammalia; Eutheria; Laurasiatheria;
            Cetartiodactyla; Suina; Suidae; Sus.
REFERENCE  1
  AUTHORS  Kato,A., Sugiura,N., Saruta,Y., Hosoiri,T., Yasue,H. and
            Hirose,S.
  TITLE    Targeting of endopeptidase 24.16 to different subcellular
            compartments by alternative promoter usage
  JOURNAL  J. Biol. Chem. 272 (24), 15313-15322 (1997)
  PUBMED   9182559
REFERENCE  2 (bases 1 to 2732)
  AUTHORS  Hirose,S.
  TITLE    Direct Submission
  JOURNAL  Submitted (27-DEC-1996) Shigehisa Hirose, Tokyo Institute of
            Technology, Department of Biological Sciences; 4259
            Nagatsuta-cho, Midori-ku, Yokohama, Kanagawa 226-8501, Japan
FEATURES             Location/Qualifiers
     source           1..2732
                     /organism="Sus scrofa"
```

```

                                /mol_type="mRNA"
                                /db_xref="taxon:9823"
                                /tissue_type="Liver"
                                /dev_stage="Adult"
                                /note="porcine"
mRNA                          1..2732
                                /note="corresponding to exon1,5-16 of this gene;
                                endopeptidase 24.16 type 1"
CDS                            175..2289
                                /standard_name="endopeptidase 24.16"
                                /note="oligopeptidase M :neurolysin :sBAP(soluble
                                angiotensin-binding protein) :MEP(microsomal
                                metalloendopeptidase)"
                                /codon_start=1
                                /product="endopeptidase 24.16 type M1"
                                /protein_id="BAA19060.1"
                                /db_xref="GI:1783122"
                                ...

```

For systems with Aspera Connect installed, the `asp-ls` and `asp-cp` scripts can be used for faster retrieval:

```

asp-ls genbank |
grep ".seq.gz" |
grep -e gbman -e gbprl -e gbrd |
while read file
do
    asp-cp genbank "$file"
    gzcac "$file"
    rm "$file"
done

```

## Appendices

### Setting Contact Address and Script Name

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

```
econtact -email author_email_address -tool name_of_script
```

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

### Command-Line Arguments

Arguments for the EDirect functions are listed below:

Use `esearch` to start a new Entrez search on indexed terms:

## Query Specification

-db	Database name
-query	Query string

## Document Order

-sort	Result presentation order
-------	---------------------------

## Date Constraint

-days	Number of days in the past
-datatype	Date field abbreviation
-mindate	Start of date range
-maxdate	End of date range

## Limit by Field

-field	Query words individually in field
-pairs	Query overlapping word pairs

## Spell Check

-spell	Correct misspellings in query
--------	-------------------------------

## Miscellaneous Arguments

-label	Alias for query step
--------	----------------------

The elink function looks up related articles or associated records:

## Destination Database

-related	Neighbors in same database
-target	Links in different database
-name	Link name (e.g., pubmed_protein_refseq)

## Direct Record Selection

-db	Database name
-id	Unique identifier(s)

## Advanced Control

-cmd	Command type (returns eLinkResult XML)
-mode	"ref" uses LinkOut provider's web site
-holding	Name of LinkOut provider

## Batch Processing

-batch	Bypass Entrez history mechanism
--------	---------------------------------

## Miscellaneous Arguments

-label            Alias for query step

Use efilter to restrict search or link results by indexed terms:

#### Query Specification

-query            Query string

#### Document Order

-sort            Result presentation order

#### Date Constraint

-days            Number of days in the past

-datatype        Date field abbreviation

-mindate         Start of date range

-maxdate         End of date range

#### Limit by Field

-field            Query words individually in field

-pairs            Query overlapping word pairs

#### Spell Check

-spell            Correct misspellings in query

#### Publication Filters

-pub            abstract, clinical, english, free, historical,  
journal, last\_week, last\_month, last\_year,  
medline, preprint, review, structured

#### Sequence Filters

-country         usa:minnesota, united\_kingdom, "pacific ocean", ...

-feature         gene, mrna, cds, mat\_peptide, ...

-location         mitochondrion, chloroplast, plasmid, plastid

-molecule       genomic, mrna, trna, rrna, ncrna

-organism         animals, archaea, bacteria, eukaryotes, fungi,  
human, insects, mammals, plants, prokaryotes,  
protists, rodents, viruses

-source           genbank, insd, pdb, pir, refseq, swissprot, tpa

#### Gene Filters

-status           alive

-type            coding, pseudo

#### Miscellaneous Arguments

-label            Alias for query step

The record retrieval function is `efetch`:

#### Format Selection

<code>-format</code>	Format of record or report
<code>-mode</code>	text, xml, asn.1, json
<code>-style</code>	withparts, conwithfeat

#### Direct Record Selection

<code>-db</code>	Database name
<code>-id</code>	Unique identifier or accession number

#### Sequence Range

<code>-seq_start</code>	First sequence position to retrieve
<code>-seq_stop</code>	Last sequence position to retrieve
<code>-strand</code>	Strand of DNA to retrieve

#### Gene Range

<code>-chr_start</code>	Sequence range from 0-based coordinates
<code>-chr_stop</code>	in gene docsum GenomicInfoType object

#### Sequence Flags

<code>-complexity</code>	0 = default, 1 = bioseq, 3 = nuc-prot set
<code>-extend</code>	Extend sequence retrieval in both directions
<code>-extrafeat</code>	Bit flag specifying extra features

#### Miscellaneous

<code>-raw</code>	Skip database-specific XML modifications
<code>-json</code>	Convert adjusted XML output to JSON

The `xtract` function is used for processing XML data:

#### Processing Flags

<code>-strict</code>	Remove HTML and MathML tags
<code>-mixed</code>	Allow mixed content XML
<code>-accent</code>	Excise Unicode accents and diacritical marks
<code>-ascii</code>	Unicode to numeric HTML character entities
<code>-compress</code>	Compress runs of spaces
<code>-stops</code>	Retain stop words in selected phrases

#### Data Source

<code>-input</code>	Read XML from file instead of stdin
<code>-transform</code>	File of substitutions for <code>-translate</code>

## Exploration Argument Hierarchy

-pattern	Name of record within set
-group	Use of different argument
-block	names allows command-line
-subset	control of nested looping

## Exploration Constructs

Object	DateRevised
Parent/Child	Book/AuthorList
Heterogeneous	"PubMedArticleSet/*"
Exhaustive	"History/**"
Nested	"*/Taxon"
Recursive	"***/Gene-commentary"

## Conditional Execution

-if	Element [@attribute] required
-unless	Skip if element matches
-and	All tests must pass
-or	Any passing test suffices
-else	Execute if conditional test failed
-position	[first last outer inner even odd all]
-select	Select record subset by conditions

## String Constraints

-equals	String must match exactly
-contains	Substring must be present
-starts-with	Substring must be at beginning
-ends-with	Substring must be at end
-is-not	String must not match

## Numeric Constraints

-gt	Greater than
-ge	Greater than or equal to
-lt	Less than
-le	Less than or equal to
-eq	Equal to
-ne	Not equal to

## Format Customization

-ret	Override line break between patterns
-tab	Replace tab character between fields
-sep	Separator between group members
-pfx	Prefix to print before group
-sfx	Suffix to print after group
-plg	Prologue to print once before elements
-elg	Epilogue to print once after elements
-rst	Reset -sep through -elg



-clr	Clear queued tab separator
-pfc	Preface combines -clr and -pfx
-deq	Delete and replace queued tab separator
-wrp	Wrap elements in XML object
-def	Default placeholder for missing fields
-lbl	Insert arbitrary text

#### Element Selection

-element	Print all items that match tag name
-first	Only print value of first item
-last	Only print value of last item
-NAME	Record value in named variable

#### -element Constructs

Tag	Caption
Group	Initials,LastName
Parent/Child	MedlineCitation/PMID
Recursive	"**/Gene-commentary_accession"
Unrestricted	"PubDate/*"
Attribute	DescriptorName@MajorTopicYN
Range	MedlineDate[1:4]
Substring	"Title[phospholipase   rattlesnake]"
Object Count	"#Author"
Item Length	"%Title"
Element Depth	"^PMID"
Variable	"&NAME"

#### Special -element Operations

Parent Index	"+"
Object Name	"?"
XML Subtree	"**"
Children	"\$"
Attributes	"@"

#### Numeric Processing

-num	Count
-len	Length
-sum	Sum
-min	Minimum
-max	Maximum
-inc	Increment
-dec	Decrement
-sub	Difference
-avg	Average
-dev	Deviation
-med	Median
-bin	Binary
-bit	Bit Count

## String Processing

-encode	URL-encode <, >, &, ", and ' characters
-upper	Convert text to upper-case
-lower	Convert text to lower-case
-title	Capitalize initial letters of words
-year	Extract first 4-digit year from string
-translate	Substitute values with -transform table

## Text Processing

-terms	Partition text at spaces
-words	Split at punctuation marks
-pairs	Adjacent informative words
-letters	Separate individual letters
-clauses	Break at phrase separators
-indices	Index normalized words

## Sequence Processing

-revcomp	Reverse-complement nucleotide sequence
----------	--

## Sequence Coordinates

-0-based	Zero-Based
-1-based	One-Based
-ucsc-based	Half-Open

## Command Generator

-insd	Generate INSDSeq extraction commands
-------	--------------------------------------

## -insd Argument Order

Descriptors	INSDSeq_sequence INSDSeq_definition INSDSeq_division
Flags	[complete partial]
Feature(s)	CDS,mRNA
Qualifiers	INSDFeature_key "#INSDInterval" gene product

## Miscellaneous

-head	Print before everything else
-tail	Print after everything else
-hd	Print before each record
-tl	Print after each record

## Reformatting

-format	[copy compact flush indent expand]
---------	------------------------------------

## Modification

-filter	Object
---------	--------

```
[retain|remove|encode|decode|shrink|expand|accent]
[content|cdata|comment|object|attributes|container]
```

#### Validation

```
-verify          Report XML data integrity problems
```

#### Summary

```
-outline         Display outline of XML structure
-synopsis         Display count of unique XML paths
```

#### Documentation

```
-examples        Examples of EDirect and xtract usage
```

The `efinfo` function returns information on Entrez indexed fields:

#### Database Selection

```
-db              Database name
-dbs             Get all database names
```

#### Data Summaries

```
-fields         Print field names
-links          Print link names
```

Several additional functions are provided by EDirect:

#### epost

```
-db              Database name
-id             Unique identifier(s) or accession number(s)
-format         uid or acc
-input          Read from file instead of stdin
-label         Alias for query step
```

#### eproxy

```
-alias          File of aliases
-pipe           Read aliases from stdin
```

#### econtact

```
-email          Contact person's address
-tool           Name of script or program
```

#### nquire

```
-get            Uses HTTP GET instead of POST
-url           Base URL for external search
```

In addition, -email and -tool are available in all E-utilities-calling functions to override default values, -http get will force the use of GET instead of POST, -alias will specify a file of shortcut keywords and query strings or URL sections, and -help will print the list of arguments for each function.

For debugging, -silent will suppress link failure retry messages, -verbose will display the <ENTREZ\_DIRECT> field values at each step, -debug will print the internal URL query and XML results of each step, and -base will specify a particular server for quality assurance testing.

## EFetch Formats

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

-db	-format	-mode	Report Type
(all)	docsum		DocumentSummarySet XML
	docsum	json	DocumentSummarySet JSON
	full		Same as native except for mesh
	uid		Unique Identifier List
	url		Entrez URL
	xml		Same as -format full -mode xml
bioproject	native		BioProject Report
	native	xml	RecordSet XML
biosample	native		BioSample Report
	native	xml	BioSampleSet XML
biosystems	native	xml	Sys-set XML
gds	native	xml	RecordSet XML
	summary		Summary
gene	gene_table		Gene Table
	native		Gene Report
	native	asn.1	Entrezgene ASN.1
	native	xml	Entrezgene-Set XML
	tabular		Tabular Report
homologene	alignmentscores		Alignment Scores
	fasta		FASTA
	homologene		Homologene Report

	native		Homologene List
	native	asn.1	HG-Entry ASN.1
	native	xml	Entrez-Homologene-Set XML
mesh			
	full		Full Record
	native		MeSH Report
	native	xml	RecordSet XML
nlmcatalog			
	native		Full Record
	native	xml	NLMCatalogRecordSet XML
pmc			
	medline		MEDLINE
	native	xml	pmc-articleset XML
pubmed			
	abstract		Abstract
	medline		MEDLINE
	native	asn.1	Pubmed-entry ASN.1
	native	xml	PubmedArticleSet XML
(sequences)			
	acc		Accession Number
	est		EST Report
	fasta		FASTA
	fasta	xml	TinySeq XML
	fasta_cds_aa		FASTA of CDS Products
	fasta_cds_na		FASTA of Coding Regions
	ft		Feature Table
	gb		GenBank Flatfile
	gb	xml	GBSet XML
	gbc	xml	INSDSet XML
	gene_fasta		FASTA of Gene
	gp		GenPept Flatfile
	gp	xml	GBSet XML
	gpc	xml	INSDSet XML
	gss		GSS Report
	ipg		Identical Protein Report
	ipg	xml	IPGReportSet XML
	native	text	Seq-entry ASN.1
	native	xml	Bioseq-set XML
	seqid		Seq-id ASN.1
snp			
	chr		Chromosome Report
	docset		Summary
	fasta		FASTA
	flt		Flat File
	native	asn.1	Rs ASN.1
	native	xml	ExchangeSet XML
	rsr		RS Cluster Report

	ssexemplar		SS Exemplar List
sra	native	xml	EXPERIMENT_PACKAGE_SET XML
	runinfo	xml	SraRunInfo XML
structure	mmdb		Ncbi-mime-asn1 strucseq ASN.1
	native		MMDB Report
	native	xml	RecordSet XML
taxonomy	native		Taxonomy List
	native	xml	TaxaSet XML

## ESearch Sort Order

ESearch -sort values for several databases are listed below:

-db	-sort
—	—
gene	Chromosome Gene Weight Name Relevance
geoprofiles	Default Order Deviation Mean Value Outliers Subgroup Effect
pubmed	First Author Journal Last Author Pub Date Recently Added Relevance Title
(sequences)	Accession Date Modified Date Released Default Order Organism Name Taxonomy ID
snp	

```

Chromosome Base Position
Default Order
Heterozygosity
Organism
SNP_ID
Success Rate

```

## ELink Commands

ELink -cmd options produce results as LinkSet XML:

-cmd	Result
neighbor	Neighbors or links
neighbor_score	Neighbors with computed similarity scores
acheck	All links available
ncheck	Existence of neighbors
lcheck	Existence of external links (LinkOuts)
llinks	Non-library LinkOut providers
llinkslib	All LinkOut providers
prlinks	Primary LinkOut provider, or URL for single UID with -mode ref

## 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>138982028</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

## 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

## Modify Contents:

sed      Replaces text strings  
-e      Specify individual expression  
tr      Translates characters  
-d      Delete character  
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

## 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

### 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

`-l` 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

`pwd` Prints working directory path

Additional documentation with detailed explanations and examples can be obtained by typing "man" followed by a command name.

## Terminal Keyboard Shortcuts

Control and escape sequences can be used within a terminal session to navigate through the command history and to move the cursor for editing the command currently being entered:

Command history:

Ctrl-n	Next command
Ctrl-p	Previous command

Move cursor forward:

Ctrl-e	To end of line
Ctrl-f	By one character
Esc-f	By one argument

Move cursor backward:

Ctrl-a	To beginning of line
Ctrl-b	By one character
Esc-b	By one argument

Delete:

Del	Previous character
Ctrl-d	Next character
Ctrl-k	To end of line
Ctrl-u	Entire line
Ctrl-w	Previous word
Esc-Del	Previous argument
Esc-d	Next argument

Autocomplete:

Tab	Completes directory or file names
-----	-----------------------------------

Program control:

Ctrl-c	Quit running program
^x^y	Run last command replacing x with y

(Note that Control sequences are typed by holding down Control, hitting the other key, and releasing Control, while Escape sequences are typed by hitting Escape and then hitting the other key.)

## Release Notes

### EDirect 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.

### EDirect 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.

### EDirect 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.

### EDirect Version 10.1: October 9, 2018

- Xtract -mixed improves support for mixed-content XML.

### EDirect 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).

### EDirect Version 9.90: September 17, 2018

- Normalized archive path for low-value PMIDs in preparation for incremental indexing.

### EDirect 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.

### EDirect 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.

### EDirect Version 9.60: August 9, 2018

- Local archive script removes newlines inside PubMed text fields.
- Efetch adds missing newline at end of PubmedArticleSet XML.

### EDirect 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.

### EDirect Version 9.40: July 18, 2018

- Xtract handles misplaced spaces in attributes.
- Xtract -format repairs misplaced spaces in attributes.

### EDirect 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.

### EDirect Version 9.20: June 26, 2018

- Portability and efficiency improvements to local data cache scripts.
- Xtract handles misplaced spaces in self-closing tags.

### EDirect Version 9.10: June 18, 2018

- Added Parent/\* element exploration construct to xtract.
- Xtract -year reliably obtains the year from "PubDate/\*".

### EDirect 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.

### EDirect Version 8.90: June 4, 2018

- Xtract -transform and -translate allow data value substitution.
- Xtract -wrp simplifies wrapping of extracted values in XML tags.

### EDirect Version 8.80: May 29, 2018

- Efetch removes MathML tags from PubmedArticle XML contents, unless the -raw flag is used.

### EDirect 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.

### EDirect 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.

### EDirect Version 8.50: April 13, 2018

- Efetch markup tag removal modified after change in server.
- Xtract -phrase filter split into -require and -exclude commands.

### EDirect 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.

### EDirect Version 8.30: April 4, 2018

- Xtract is now available for ARM processors.

### EDirect Version 8.20: March 12, 2018

- Minor changes to local record archiving scripts.

### EDirect Version 8.10: March 2, 2018

- Xtract -strict and -mixed support MathML element tags in PubmedArticle XML.

### EDirect 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.

### EDirect Version 7.90: February 1, 2018

- Minor change to installation commands for tcsh.

### EDirect Version 7.80: January 12, 2018

- Updated setup.sh script with additional error checking.

### EDirect Version 7.70: December 27, 2017

- Added archive-pubmed script to automate local record archiving.

### EDirect 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.

### EDirect 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.

### EDirect Version 7.40: September 27, 2017

- Xtract -hash reports checksums for local record archiving.
- Initial support for API keys.

### EDirect 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.

### EDirect 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.

### EDirect 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.

### EDirect 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.

### EDirect 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.

### EDirect 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.

### EDirect 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/>).

### EDirect 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.

### EDirect Version 6.50: March 6, 2017

- Xtract -degloss replaces -html to remove HTML <i>, <b>, <u>, <sup> and <sub> tags.

### EDirect 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.

### EDirect Version 6.30: February 13, 2017

- Efetch -format docsum skips GI-less sequences without summaries.
- Xtract local indexing commands moved to -extras documentation.



### EDirect Version 6.20: January 30, 2017

- Xtract -limit and -index allow extraction of selected records from XML file.

### EDirect 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.

### EDirect Version 6.00: December 27, 2016

- Efetch -format docsum removes eSummaryResult wrapper.
- Fixed content truncation bug when Xtract encounters very long sequences.

### EDirect 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.

### EDirect 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).

### EDirect 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.

### EDirect Version 5.60: November 22, 2016

- Fixed bug in -datatype argument for Esearch and Efilter.
- Added optional argument to filter-stop-words script to indicate replacement.

### EDirect 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.

### EDirect Version 5.40: November 7, 2016

- Added filter-stop-words and xy-plot scripts.

### EDirect 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).

### EDirect 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.

### EDirect 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.

### EDirect 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.

### EDirect 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.

### EDirect Version 4.80: August 9, 2016

- Xtract -insd controlled vocabularies updated.

### EDirect 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.

### EDirect 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.

### EDirect 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.

### EDirect 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].

### EDirect 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.

### EDirect Version 4.20: May 24, 2016

- Xtract command-line argument parsing improvements.
- Nquire -get supersedes -http get.

### EDirect Version 4.10: May 3, 2016

- Xtract -format removes multi-line XML comments and CDATA blocks.

### EDirect 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.

### EDirect Version 3.90: March 21, 2016

- Code optimizations for increased Xtract speed.

### EDirect Version 3.80: February 29, 2016

- Xtract can distribute its work among available processor cores for additional speed.

### EDirect Version 3.70: February 8, 2016

- Xtract performance improvements.

### EDirect Version 3.60: January 11, 2016

- The setup.sh configuration script now downloads a precompiled Xtract executable for selected platforms.

### EDirect Version 3.50: December 27, 2015

- Xtract reports error for element:value construct outside of -match or -avoid arguments.

### EDirect Version 3.40: December 20, 2015

- Xtract -insd supports extraction from multiple features (e.g., CDS,mRNA).

### EDirect Version 3.30: December 3, 2015

- Efetch -format docsum can accept a single sequence accession number in the -id argument.

### EDirect Version 3.20: November 30, 2015

- Xtract supports -match conditional execution on values recorded in variables.

### EDirect 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.

### EDirect 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.

### EDirect 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.

### EDirect Version 2.80: September 9, 2015

- Xtract uses the "\*/Child" construct for nested exploration into recursive structures, replacing the -trim argument.

### EDirect Version 2.70: July 14, 2015

- Added entrez-phrase-search script to query on adjacent word pairs indexed in specific fields.

### EDirect Version 2.60: June 23, 2015

- Xtract -match and -avoid support "Parent/Child" construct for BLAST XML.

### EDirect Version 2.50: April 9, 2015

- Xtract capitalized -Pattern handles recursively-defined top-level objects.

### EDirect Version 2.40: March 25, 2015

- EDirect programs use the http\_proxy environment variable to work behind firewalls.

### EDirect Version 2.30: March 11, 2015

- Cleaned up logic in setup.sh configuration script.
- EPost -format acc works properly on protein accessions.

### EDirect Version 2.20: March 4, 2015

- Xtract -match and -avoid recognize "@attribute" without element or value.

### EDirect Version 2.10: February 3, 2015

- Added ftp-ls and ftp-cp scripts for convenient access to the NCBI anonymous ftp server.

### EDirect Version 2.00: August 28, 2014

- Introduced copy-and-paste installation commands with setup.sh configuration script.

### EDirect 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.

### EDirect Version 1.80: July 15, 2014

- EPost -format acc accepts accessions in an -id argument on the command line.

### EDirect Version 1.70: April 23, 2014

- EFetch -format docsum decodes HTML entity numbers embedded in the text.

### EDirect Version 1.60: April 3, 2014

- Minor enhancements to xtract -insd.

### EDirect 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.

### EDirect 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.

### EDirect 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".

### EDirect 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.

### EDirect 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.

### EDirect Version 1.00: February 6, 2014

- Initial public release.

## 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 [utilities@ncbi.nlm.nih.gov](mailto:utilities@ncbi.nlm.nih.gov).