# Title Slide

Welcome back to EDirect for PubMed! Today is Part Two, Extracting Data from XML.

My name is Sarah Helson, and I’m a librarian at the National Library of Medicine in Bethesda, Maryland.

# EDirect for PubMed Agenda

This is the second of five sessions in the EDirect for PubMed series. We talked last time about using EDirect commands to search for and retrieve PubMed records in a variety of formats, including XML.

Today, we’re going to start talking about the xtract command, which lets you extract specific data elements from the PubMed XML and arrange them in a custom tabular format

Xtract’s a pretty big topic, so we’ll actually spend most of the next three classes discussing different aspects of it, before finishing up with a broader discussion of EDirect

# Today’s Agenda

We’ll start with a quick recap of last session, and answer any questions about the homework.

We’ll then do a brief refresher course on the basics of XML, so we can actually figure out which data we want to extract from it. Then we’ll talk about creating basic tables with xtract.

And we’ll finish up with a discussion of a few commands that can help you post-process and clean up your output.

# Brief Recap

Remembering back to last week, we talked about a few EDirect commands including esearch, which searches a database and retrieves a list of PMIDs for records that meet your search criteria.

We talked about efetch which lets you retrieve PubMed records in a variety of formats.

We also talked about some Unix commands and operators, like “|”, which pipes the results from one command to the next.

# Tips for Cygwin users

A few reminders for Cygwin users: keyboard shortcuts do not always work the way you are used to. Ctrl + C does not Copy; try Ctrl + Insert instead. Likewise, Ctrl + V does not paste; Shift + Insert is the default.

And again, you can adjust those keyboard shortcuts in the Cygwin options.

# Tips for all users

No matter what terminal you’re using, you will want to remember that Ctrl + C lets you Cancel out of runaway commands.

The up and down arrows let you cycle through your previous commands.

And the “clear” command lets you clear your screen.

# Questions from Previous Class?

Before we get into today’s class, does anyone have any questions about anything we talked about last week?

**[PAUSE FOR QUESTIONS]**

# Remember our theme…

Another thing we talked about last class was the theme for this course: Getting exactly the PubMed data you need, and only the data you need, in the exact format you need it in.

So far, we’ve talked a lot about how to get data from PubMed, by using esearch to search for records and efetch to retrieve them.

But we keep promising that the point of EDirect, and of E-utilities in general, is to be able to customize your output, and select out only the data elements you need from the XML.

In order to do that, we need to make sure everyone’s on the same page, as far as the basic structure and syntax of XML. This should be old hat for most of you, but we figured we would refresh your memory, so just bear with us!

# XML

XML is a markup language used for storing and transporting data. It is designed and structured to be read and used by computers, but can also be read and understood by humans. XML documents are composed of elements.

# XML Basics (ANIMATED)

This is an XML Element. It’s got an open tag and a close tag with the name of the element, in this case “Year”, and it contains some data (in this case “2015”).

**[CLICK]** Some XML Elements also have Attributes, which provide more information about the element. Attributes are inside the open tag, and have a name and a value. In this example case, the attribute name is “Status” and the attribute value is MEDLINE.

# An XML Example (ANIMATED)

Here, we have a fragment of an XML document. This is an example based loosely on a PubMed XML document, but it is not a complete document: it is simplified and missing many required elements.

**[CLICK]** The first thing we see here is the open tag of the element PubmedArticleSet. The close tag for that element is down at the bottom.

**[CLICK]** The rest of document is contained within this element. This demonstrates another important concept in XML: instead of containing text, like our previous example, XML elements can also contain other elements. This creates a hierarchical structure, where child elements are nested inside a parent element.

For example: **[CLICK]** The PubmedArticleSet has a child element, PubmedArticle. **[CLICK]** PubmedArticle has its own children: PMID, DateCreated, Journal, ArticleTitle, and ELocationID.

**[CLICK]** Some of these child elements have text, like ArticleTitle.

**[CLICK]** But others contain children of their own, like DateCreated.

**[CLICK]** And we also have a few elements with attributes. ISSN has an IssnType attribute, with the value “Electronic”. **[CLICK]** The ELocationID element has two attributes.

# Some XML elements repeat

This is another mockup of an XML document, and this one follows a structure you’ll see a lot today, which is the structure you get when you use efetch to get records from PubMed.

Again, the whole document is wrapped in this PubmedArticleSet element, which contains several PubmedArticle child elements. Each PubmedArticle element contains all the information for a single PubMed record.

This also demonstrates another important fact about XML elements: some XML elements can be repeated in the same document. You’ll see this when you fetch multiple PubMed records, but also within a single PubMed record, you’ll see elements like Author and MeSH heading repeat, since an article can have multiple Authors or MeSH headings.

Now, converting this XML structure into a table can be really tricky. We need to make sure we get the right data from the right parts of the hierarchy, and arrange it in a way that makes logical sense. But before we get into that, are there any questions?

**[PAUSE FOR QUESTIONS]**

# xtract

Getting information in an easy to use format is why we want to use xtract.

Xtract is a big part of what’s useful about EDirect. It’s a command that extracts specific elements from XML and arranges them in a customized tabular format.

Like most Unix commands, you control the details of how xtract works by providing arguments. Which arguments you provide are going to determine what the format looks like.

Xtract is part of EDirect, but it’s not actually an E-utilities command. It doesn’t use the API to contact the E-utilities server. It works on XML data that you have already brought down to your computer. It is an important tool for processing data you get from the other EDirect commands, which do contact the server.

# What are we xtract-ing from?

One of the advantages of xtract is that, because it’s not actually contacting the E-utilities server to get its input, it can actually work on any XML.

For most of our examples, our XML will be the output of an efetch that we pipe into xtract, using the -format xml argument.

And you can also use the –input argument to specify an XML file that’s already on the computer.

# PubMed XML Documentation

There are a large number of XML elements in PubMed, and while you will probably be able to easily determine which elements refer to information displayed in the web version, we also have documentation to help with that.

(DEMO SHOWING PUBMED XML DOCUMENTATION IN A WEB BROWSER.)

This documentation lists the different elements you will find in PubMed, as well as their attributes.

This first page you see lists the elements in the order you would see them in the PubMed DTD, but you can also see an alphabetical list. I find this to be the most helpful if I’m looking at an XML record and want to know what a certain element or attribute is referring to.

We also have a new resource for XML element documentation. This is linked to the PubMed DTD and has the same information from the other page.

This page is similar to the alphabetical list of elements that we just saw. Looking at an individual element, such as Abstract, will show you information such as the definition, examples, and where in the XML you will find it.

It also separates out the attributes from the elements, which can come in handy. You can see the allowable values as well as the related elements.

# Before you start xtract-ing

When we’re working with xtract, we’re going to be directing the command to select specific elements and attributes from specific parts of the XML record.

To help us choose the right elements, we have to understand how the XML is structured. The easiest way to do this is to actually just have the XML of some sample PubMed records open in a window that we can consult.

(DEMO RETRIEVING PUBMED XML IN A WEB BROWSER.)

# Get a small sample dataset

You’re going to want to do a lot of testing and refining of your xtract commands, so in addition to looking at example XML to understand the structure, it’s helpful to get a small set of PubMed records in xml that you can use to test our xtract statements.

Ideally, you want a handful of records that are representative of the kind of records you’ll be xtracting from for real. Make sure that the sample records include the same kinds of data as your live data set. For example, if you are trying to extract funding information, make sure you have a few test records that have funding information on them.

If you don’t think your dataset is going to have some of the more exotic fields (comments/corrections, etc.) then avoid them in your test dataset. You can always deal with those issues later.

As we just discussed, there a lot of ways to get XML into an xtract. In order to make sure you can easily recreate my examples, I’m generally going to be using an efetch command today to retrieve sample data, and piping that into xtract.

# Xtract Example 1

Because xtract is so powerful and has so many options, it can sometimes be a little overwhelming, so we’re going to start with a very simple example:

We have a set of PubMed records in XML. We want to have a tabular list of those records with PMID, journal title abbreviation, and article title.

This is pretty basic information. However, even the standard .csv format available from the Send To: menu in the web version of PubMed can’t do this. You’d have to dig the journal title abbreviation out from the Details field.

This is our goal. Now let’s talk about how we do it.

# Questions to ask when making a table (ANIMATED)

We want to make a table. But we need to plan out what our table is going to look like.

**[CLICK]** This is our sample table, and the first thing we want to consider is **[CLICK]** what connects the data in each row? It seems obvious, but if the data in each row isn’t all related, the table doesn’t tell us much. In this example, the connection is that all of the data in a row is from the same PubMed record. That’s the relationship.

**[CLICK]** Then we need to figure out how many rows we need. We want as many rows as we have PubMed records in our XML.

**[CLICK]** How many columns do we need? **[CLICK]** And what data is in each column?

Like with all EDirect commands, we determine these things by carefully choosing our arguments.

# What connects the data in each row? (ANIMATED)

**[CLICK]** We determine what connects our data by using the “-pattern” argument. We specify an XML element for the –pattern argument **[CLICK]** and all of the data in a single row will come from descendants (child elements, children of children, etc.) of a single occurrence of that pattern element.

**[CLICK]** xtract scans the XML document you input until it finds an occurrence of the pattern. **[CLICK]** When it finds an occurrence of the pattern, it creates a new row.

# -pattern PubmedArticle (ANIMATED)

Again, this is a dummy XML document

We’ve specified our –pattern as PubmedArticle, so xtract will scan through until it finds the first occurrence of that element.

**[CLICK]** Once it does, it will create a new row in our output table. **[CLICK]** All of the data in that row will come from within that pattern, within that PubmedArticle.

Once xtract reaches the end of the pattern, **[CLICK]** it looks for the next occurrence of that pattern, and creates another row. **[CLICK]** All of the data in that second row will come from within the second pattern.

And so on **[CLICK]** and so forth **[CLICK]**, until we reach the end of the document, and can find no more occurrences of the pattern. And then we stop.

# How many rows?

This also answers our second question: how many rows?

We will have one row per occurrence of the pattern in the XML input. So, if our –pattern is PubmedArticle, our xtract command will create as many rows as there are records in your XML document.

# Won’t my –pattern always be PubmedArticle? (ANIMATED)

This bring up an interesting question: won’t my pattern always be PubmedArticle, since that will make sure all the data in a single row comes from the same PubMed record?

**[CLICK]** Most of the time, the answer to that is yes. This is what lets you tabulate PubMed records and analyze trends.

**[CLICK]** However, you could change the pattern to see different types of relationships, so if you’re looking at author information or grant data, you might want to use the Author or Grant elements as your pattern. We’ll see an example of this later in class.

# How many columns? (ANIMATED)

So, that covers your rows, now let’s talk about creating columns. You do that using a different argument, -element.

**[CLICK]** You specify one or more XML elements or attributes in the –element argument. **[CLICK]** Each element or attribute you specify creates a new column.

**[CLICK]** How many elements or attributes you specify determines the number of columns…**[CLICK]**….most of the time. There are exceptions to this, which we’ll get into a little bit later, but for now we’re going to assume that these things are true.

# What data is in each column?

Going back to our pattern, remember that all of the data in a row is going to come from within the same pattern.

Xtract looks inside the pattern for the elements and attributes you specified in the –element argument. The value of each occurrence of your first element goes in the first column. Then xtract moves onto the second column, and puts the value of each occurrence of the second element in the second column, etc.

# -element PMID Year ArticleTitle (ANIMATED)

Here’s what that looks like. Again, we have our same dummy XML, and we’re still using a pattern of PubmedArticle.

**[CLICK]** Xtract finds our first pattern, **[CLICK]** then looks within that pattern for the first element in our element argument which is PMID.

**[CLICK]** It finds a PMID element, and puts the contents in the first column.

**[CLICK]** It finds a Year element, and puts the contents in the second column.

**[CLICK]** It finds an ArticleTitle element, and puts the contents in the third column.

There are no more columns, and it’s reached the end of the pattern, so it moves on to the second PubmedArticle, **[CLICK]** creates a new row, and repeats the process **[CLICK]**, looking for **[CLICK]** PMID, **[CLICK]** Year, **[CLICK]** and Article Title.

Again, this process repeats until we run out of patterns. **[CLICK] [CLICK] [CLICK] [CLICK].**

Okay, that’s enough walk through, let’s see what this actually looks like in practice.

# xtract syntax

You’ve already seen a little sneak peek of the syntax xtract uses, but this is a basic xtract command.

Here’s how it works: our xtract command here has a –pattern of PubmedArticle and an –element of ArticleTitle.

(SWITCH TO CYGWIN)

Again, I’m going to use an efetch command to get my XML, then pipe it into my xtract.

(DEMO IN CYGWIN)

efetch -db pubmed -id 24102982,21171099,17150207 -format xml | \

xtract -pattern PubmedArticle -element ArticleTitle

We type in our command, and we should get output that looks like this:

(EXECUTE)

One row per article, with a single column that displays the ArticleTitle.

Let’s say instead of the ArticleTitle, I wanted to see a list of all of the authors from these three PMIDs in one column. Let’s change the xtract to use a different -pattern to do this.

Use -pattern Author to get one author per line.

(DEMO IN CYGWIN)

efetch -db pubmed -id 24102982,21171099,17150207 -format xml | \

xtract -pattern Author -element LastName

This new command tells xtract to look in each occurrence of Author to find that element LastName, and to create a new line each time it does. It outputs more than three lines because we changed our pattern from PubmedArticle to Author. Most of the examples today will use the -pattern PubmedArticle, but I wanted to show you how you can make xtract work for you.

And that’s the basics of xtract. Of course, we keep talking about how xtract gives us tabular output. This is more of a list. What we really want is to get multiple columns.

# Creating multiple columns

(SWITCH TO SLIDES)

To get multiple columns, we can just specify more than one element in our –element argument. We separate the multiple element names with spaces, but we don’t need to repeat the “-element” part.

**[PAUSE FOR QUESTIONS]**

# Exercise 1

Here’s our first exercise:

Write an xtract command that creates a table with one row per PubMed record. Each row should have two columns: Volume and Issue Number.

# Exercise 1 Solution

(DEMO IN CYGWIN:

efetch -db pubmed -id 24102982,21171099,17150207 -format xml | \

xtract -pattern PubmedArticle -element Volume Issue

EXECUTE)

This is a basic solution to a basic question, but depending on what kind of records we’re working with, and what elements we want to extract, we could get into trouble.

If you’re still working on that last exercise, I’m going to ask you to put that aside for the moment, because I want you to see this next example. Remember that all of the solutions to the exercises are at the bottom of your course materials document.

Let’s run a new xtract on one PMID: 27101380.

(DEMO IN CYGWIN)

efetch -db pubmed -id 27101380 -format xml | \

xtract -pattern PubmedArticle -element PMID Year

We would hope for a single PMID and a single Year, but that’s not going to happen.

(EXECUTE)

We have a lot of problems here: even though we’re only xtracting from one PMID, we’ve got way more data than we want.

(SWITCH TO WEB BROWSER)

Let’s look at the XML for this record.

(DEMO IN WEB BROWSER: Search for PMID 27101380, display XML view.)

So far so good, we have a PMID element up here at the top.

But we have 9 <Year> elements in this record, each referring to different types of dates. We also have some other PMIDs down here, to help link this article to other articles. This is common in articles with comments and corrections.

Since you are often using PMID as a unique identifier, or as the first column in your table, this can cause a problem. So, we need a way to isolate the specific elements we need.

(SWITCH TO SLIDES)

# Parent/Child construction

We can use what’s called “Parent/Child construction” to retrieve an element that is the child of a specific parent.

This helps us isolate objects that have the same name, but are in different locations in the hierarchy.

Just use the name of the parent element, a slash, and then the name of the child element. For example, –element MedlineCitation/PMID.

# Back to xtract Example 1…

We now have all the tools we need to solve our Example.

To refresh your memory, our goal was to have a tabular list with PMID, Journal Title Abbreviation, and Article Title.

# Solving xtract Example 1

(DEMO IN WEB BROWSER/CYGWIN.)

We’ll use the same efetch we’ve been using and pipe that into the xtract.

First, we need to identify our –pattern.

Since we know that we want one row per record, we’ll use PubmedArticle again.

Second, we need to find the elements we want to include. We’re looking for PMID, journal title abbreviation and Article Title.

PMID is right here, but to make sure we identify this specific element and not another PMID element, we can use parent/child construction.

Journal title abbreviation is in the ISOAbbreviation element. I don’t think this is a repeated element, but, just to be safe, we could also use parent/child construction here, and identify ISOAbbreviation as a child of Journal.

Finally, we have ArticleTitle for the article title element.

(DEMO IN CYGWIN:

efetch -db pubmed -id 24102982,21171099,17150207 -format xml | \

xtract -pattern PubmedArticle -element MedlineCitation/PMID Journal/ISOAbbreviation ArticleTitle

EXECUTE)

**[PAUSE FOR QUESTIONS]**

# xtract-ing attribute values

(SWITCH TO SLIDES)

As we said earlier, XML elements can also have attributes, and in the PubMed XML, those attributes can carry some important information.

You can get at attributes by using @. Specify the name of the element, followed by @, followed by the name of the attribute.

# Exercise 2

* Write an xtract command that has one row per PubMed record in the data set, and includes the PMID, Journal ISSN and Citation Status
* Put up a green check when you are finished.
* Solutions at the bottom of the handout.

# Exercise 2 Solution

(DEMO IN CYGWIN)

efetch -db pubmed -id 24102982,21171099,17150207 -format xml | \

xtract -pattern PubmedArticle -element MedlineCitation/PMID Journal/ISSN MedlineCitation@Status

Again, we want to keep our –pattern as PubmedArticle. Our first –element will also be the same.

For Journal ISSN, we’ll just need the ISSN element, but we could use Parent/Child construction, just to be safe.

Finally, we’ll get the citation status from the Status attribute of the MedlineCitation element.

(EXECUTE)

# Exercise 3: Putting it all together

We’re going to do a little bit more of this next week, but I want you to take a shot at building a whole EDirect script, from beginning to end.

We want to find out which authors have been writing about traumatic brain injury in athletes. We want only the most current research, so just publications from 2016 and 2017. We only want author names, and we’d like one author name to appear on each line. We want the Last Name and Initials.

Again, we want the whole script, not just the xtract command.

I’ll give you the advice that I always try to use when building EDirect scripts, which is to try and build one part a time, make sure it works, then move on to the next.

# Exercise 3 Solution (ANIMATED)

(DEMO IN CYGWIN)

esearch -db pubmed -query "traumatic brain injury athletes" -datetype PDAT -mindate 2016 -maxdate 2017 | \

efetch -format xml | \

xtract -pattern Author -element LastName Initials

(EXECUTE)

If you haven’t finished working on this, I’m going to ask you to set it aside for now so I can show you two useful commands.

As a reminder – all solutions are at the bottom of the handout.

# sort-uniq-count-rank

(STAY IN CYGWIN)

In our next class, we’re going to talk a little more about different ways of fine-tuning your output format, to make your results as clean and useful as possible. Before we go, though, I want to show you two commands that might help you out in that regard.

Right now, this is just a list of authors, but there are probably a lot of duplicates, especially if authors were publishing multiple articles on the same topic.

If we want to see who are the most prolific authors on a given topic, we can use another handy EDirect command, sort-uniq-count-rank.

I’m going to run the same script again, but I’m going to pipe it into this sort-uniq-count-rank command.

(DEMO IN CYGWIN

esearch -db pubmed -query "traumatic brain injury athletes" -datetype PDAT -mindate 2016 -maxdate 2017 | \

efetch -format xml | \

xtract -pattern Author -element LastName Initials | \

sort-uniq-count-rank

EXECUTE)

What this command does is shows us every unique value from our list of Authors, along with how many times each of those author names appeared, and the sorts it by frequency.

Sort-uniq-count-rank has helped us quantify our results, showing us how frequently each author name occurs in our results, but even after this list was de-duped and sorted, I still have a long list with a long tail of authors that only appear once.

I’m going to take that same series of commands, with the sort-uniq-count-rank at the end, but before I execute it, I’m going to pipe my output into one more Unix command that might be useful here.

If we just want to see the top ten most frequently occurring authors, we could use the Unix command “head”. This limits our output to only the first ten rows of our input.

(DEMO IN CYGWIN

esearch -db pubmed -query "traumatic brain injury athletes" -datetype PDAT -mindate 2016 -maxdate 2017 | \

efetch -format xml | \

xtract -pattern Author -element LastName Initials | \

sort-uniq-count-rank | \

head -n 10

EXECUTE)

# Coming next week…

Next week we’re going to continue our discussion of xtract, and talk about some more advanced arguments that help us limit our output to the exact data we need.

We’re also going cover some ways to put all of what we’ve learned together, and some tips for developing practical solutions to real-world problems.

# In the meantime…

As always, we encourage you to visit our website for more information and documentation on EDirect, as well as the script, slides, and recording, as soon as we can put it up.

Remember to sign up for the utilities-announce mailing list, which you can access from the Contact Us button on our website.

And from that same button, you can send us any questions you have during the week.

# Homework

Like last class, you can find the homework at the bottom of the handout from today.

In the last class, we are going to be looking at case studies and applying what we have learned. But we would like to use your examples and your real-world situations that you think could benefit from EDirect. Please send us your case study suggestions using the Contact Us link on our website.

# Questions?

But with that, I would be happy to take any questions you have now.