# Title Slide

Good afternoon and welcome back to EDirect for PubMed! Today is Part Four: xtract Conditional Arguments.

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

# EDirect for PubMed Agenda

As you know, this is part four out of five; we’re over halfway through!

In the first session, we talked about getting PubMed data. Then, in the second session, we showed you how to extract specific data elements from PubMed XML and arrange them in a custom tabular format, using the xtract command.

Last time, we told you more about xtract, including how to customize separators and group elements together using -block, and also covered some Unix tools for working with files

Today, we’re going to finish our discussion of xtract with the Conditional arguments, which will help us refine our output to get precisely the data we want.

# Today’s Agenda

We’re going to start with a quick recap of Part Three.

Then we’re going to dive into the xtract Conditional arguments, starting with –if.

We’ll then talk about how to use multiple conditions at the same time with –and and –or.

We’ll finish up the Conditional discussion with -position, which lets you limit by an element’s location within the XML record.

And finally, we’ll look at one other xtract argument that lets you define placeholders for blank cells.

# Recap of Part Three

Last session, we talked about different ways to group and format your output, including -tab which lets you customize the separator between columns, and -sep, which lets you customize the separator between multiple values in the same column.

We also talked about -block, which is used to select and group child elements of the same parent. Remember: we used that with author data to group the author last names with the corresponding initials.

# Recap of Part Three (cont’d)

We also talked a little bit about working with files, including using “>” to redirect output from a file, “cat” to read the contents of a file, which we can use to pull in a search string from a text file, and epost, which we used to save a list of PMIDs to the history server so we could retrieve the records with efetch.

# Questions from last class? Homework?

Before we get into today’s class, does anyone have any questions about anything we discussed in the previous sessions, or questions about the homework?

**[PAUSE FOR QUESTIONS]**

# Remember our theme…

Remember the general theme for this class, we keep saying that EDirect is a tool to help you get the data you need, and only the data you need, in the exact format you need it in.

# How EDirect helps you

EDirect can help you do this in a number of ways.

Esearch/efetch get you the data you need.

A lot of what we talked about last time with -tab and -sep get you the data in the format you need.

But in order to filter your results to **just** the data you need, you’ll want to use a special set of xtract arguments, known as Conditional arguments

# -if

The conditional argument you’ll probably use the most is –if. -if lets you limit the data included in the table xtract creates, depending on certain conditions. Data will only be included **if** it matches the condition you specify.

# If-Then (ANIMATED)

For those of you who have done even a little computer programming, you’ll be familiar with if-then statements.The basic idea is that **if** a condition is true, **then** follow a specified set of instructions.

For xtract’s –if argument, we’re following that same structure:

**[CLICK]** If a condition is met for a pattern, **[CLICK]** then, create a new row for the pattern and populate the specified columns.

If the condition isn’t met, we’ll just skip that pattern and move on to the next one.

# -if Example

Let me walk you through an example of when and why you might want to do this. Say we want a list of the authors in our results set with ORCID IDs. We want the names and IDs of each author.

I’ll use this efetch to make sure I have some records that have ORCIDs.

The first thing we need to do is look at some PubMed XML to figure out where to get that ORCID ID.

(SWITCH TO PUBMED IN BROWSER, RETRIEVE RECORD, SHOW XML.)

The ORCID ID is actually in the Identifier element, within the Author element. I also happen to know that the only information in the Identifier element, anywhere in PubMed, is ORCIDs, so if there’s any Identifier element on the record, we know that it has what we want.

Let’s start with a rough draft.

(SWITCH TO CYGWIN)

We want a list of authors, not a list of PubMed records, so we want to make our –pattern Author, not PubmedArticle. Instead of having one record per row, we’ll have one Author per row. If a record has several authors, it will be reflected by several rows.

(DEMO IN CYGWIN)

xtract -pattern Author -sep " " -element LastName,Initials Identifier

We’re using the –sep to keep the LastName and Initials separated by a space, but the Identifier will be separated by a tab as normal.

(EXECUTE)

However, this gives us a list of every author. We just want the authors with ORCID IDs.

(DEMO IN CYGWIN)

xtract -pattern Author -if Identifier -sep " " -element LastName,Initials Identifier

-if tells us to keep any pattern **if** the condition we define is met. In this case, the condition is the existence of an Identifier element inside this pattern, inside this Author element.

(EXECUTE)

# If-Then: -if Identifier

(SWITCH TO SLIDES)

Going back to our If-Then structure:

If the pattern has an Identifier element, then create a new row for the pattern, with two columns of data: LastName and Initials in one column, Identifier in the other.

If the pattern does not have an identifier element, xtract will skip the pattern and move on.

# How –if works (ANIMATED)

To visualize how this works, let’s go back to our triple box from last week. On the left is some mocked-up input XML, on the right will be the output of our command, and the bottom has the actual command.

**[CLICK]** As always, xtract looks for the first occurrence of our –pattern, Author, which is right here. Rather than creating a new row right away, though, **[CLICK]** we check to see whether the Author has an Identifier element, whether the pattern meets our condition. It does, in this case, so **[CLICK]** we create a new row with LastName **[CLICK]**, Initials **[CLICK]**, and Identifier.

**[CLICK]** Then we move on to the next Author, **[CLICK]** and check to see whether **this** one has an Identifier. It doesn’t, so **[CLICK]** we don’t create a new row, and skip right down to the next Author.

**[CLICK]** This one **does** have an Identifier, so we create another row with LastName **[CLICK]**, Initials **[CLICK]**, and Identifier.

Any questions about the basic concept of –if? Because we’re about to do an exercise.

**[PAUSE FOR QUESTIONS]**

# Exercise 1

Let’s do a quick exercise to make sure you get the basics. Write an xtract command that only includes PubMed records if they have MeSH headings attached. We want one row per PubMed record. Each row should have PMID and Citation Status.

You can use the efetch line on this slide (and in your handout) to give you some good sample data. And you may also want to look at the top of your handout, as there are some reminders about what we covered in last week’s class.

# Exercise 1 Solution

(DEMO IN CYGWIN)

efetch -db pubmed -id 26277396,2156457,19649173,21906097,25380814 -format xml | \

xtract -pattern PubmedArticle -if MeshHeading \

-element MedlineCitation/PMID MedlineCitation@Status

(EXECUTE)

You’ll notice that the status of all of these records is MEDLINE. This makes sense, because only MEDLINE records have MeSH headings.

This exercise was one way of limiting your results to only MEDLINE records. There are plenty of other ways to do this: you could include a tagged mesh heading in your esearch query, or you could add an explicit restriction to the MEDLINE subset, by adding MEDLINE tagged with [sb] to your query.

The feature I’m about to show you gives you yet another way to include only MEDLINE records, if that’s what you want to do. If you’re still working on Exercise 1, I’m going to ask you to pause for now. Remember that the answers to all of the exercises are at the bottom of your handout, so you can go over them later if you want, and we’ll make sure to get the recording up as quickly as possible, too.

# -if/-equals

In that exercise, our condition was the presence of an element in a pattern: the presence of the “MeshHeading” element. However, we can also include data based on the value of an element or attribute, as opposed to simple presence.

Like before, we use –if to specify an element. But now we follow that up with an –equals argument to specify what the value of that element should be. We will include the pattern **if** ISOAbbreviation **equals** JAMA.

# If-Then: -if ISOAbbreviation –equals JAMA

Again, following the same structure: If the element ISOAbbreviation equals JAMA, then create a new row for the pattern, with the two columns Volume and Issue.

This also highlights another useful thing about –if: the data in your columns doesn’t have to be from the same element used in your condition. Our –if was ISOAbbreviation, but our output are two completely different elements.

# -if/-equals: Attributes

If we want to use attributes in our conditions, instead of elements, we can do it just like we did before, with the “@” sign.

# If-Then-Else: -if/-equals: Attributes

For this example, **if** the attribute “Status” for the element “MedlineCitation” equals “MEDLINE, **then** create a new row for the pattern, with just the PMID. As I said, this is another way to restrict your results to only MEDLINE records.

# Alternatives to –equals

The -equals argument works great if you know exactly what the element in your -if argument should be equal to, but you also have other options for partial matches. You can use the –contains, -starts-with, -ends-with, and –is-not arguments in place of –equals. Let me show you how that works.

# If-Then: -if/-contains

This example is a little different, because our pattern is now Author, not PubmedArticle.

**If** the Affiliation element in an Author pattern contains the string “Japan”, **then** create a new row for that Author pattern, with two columns: LastName/Initials and Affiliation. As always, if the condition is not met, xtract skips the pattern and move on to the next Author.

-contains is especially useful for the Affiliation element, as not everyone from the same institution always formats their affiliation data the same way, so you can capture variations by using –contains to get a partial match.

Any questions about –equals, -contains, or any of the other stuff we’ve talked about?

**[PAUSE FOR QUESTIONS]**

# Exercise 2

For this exercise, we want an xtract command that only includes PubMed records for articles published in one of the JAMA journals. (This could be JAMA itself, or JAMA cardiology, JAMA oncology, JAMA dermatology, etc.) We know that the journal title abbreviation is going to start with JAMA.

Again we want one row per PubMed record, and we want two columns: PMID and ISOAbbreviation (which is the element name for our Journal Title Abbreviation). You can use the efetch on this slide and in your handout to get some good sample data.

# Exercise 2 Solution

(DEMO IN CYGWIN)

efetch -db pubmed -id 27829097,27829076,19649173,21603067,25380814 -format xml | \

xtract -pattern PubmedArticle -if ISOAbbreviation -starts-with JAMA \

-element MedlineCitation/PMID ISOAbbreviation

(EXECUTE)

Okay, this next part is a little tricky, so I’m going to ask you to stop working on the exercise right now if you’re still trying to get the details right. Remember the answers are at the bottom of your handout for you to review later.

So far, all of our Conditional examples have been in a –pattern, and we’re only including the pattern if the condition is met.

# -if in a –block

(STAY IN CYGWIN)

However, we can also put an –if inside a -block. If you remember back to last time, we used –block very effectively when we were dealing with authors, to keep the corresponding LastName and Initials together. xtract looked for the first occurrence of the –block Author, pulled the LastName and Initials from inside that block, then moved on to the next block, until there were no more blocks in the pattern.

However, if we put an –if inside a –block, we can include only certain blocks, based on **if** they meet the condition. Instead of filtering out patterns that don’t meet our condition, we include every pattern, but only filter out blocks that don’t meet our condition.

So, for example, let’s say we want to see the PMIDs for **all** of the records in our efetch, and we also want to see the DOI (from the ArticleId element), if there is one. However, we have to be careful, because, while the ArticleId **can** include the DOI, it can also contain the PMCID or other identifiers. In fact, many records have multiple ArticleId elements, each with a different kind of identifier. We only want the contents of an ArticleId element if its “IdType” attribute equals doi.

So let’s get started. We’re going to create a new row for every PubMed record, since our pattern is PubmedArticle. We’re going to have the PMID for each record, since there’s no condition before this –element argument.

Then, we have our –block argument of ArticleId. This allows us to focus on the contents of one ArticleId element at time.

Inside the –block, we put an –if and an –equals, and then our –element. xtract will check each ArticleId –block in a pattern, one at a time, to see **if** the IdType attribute **equals** doi. If it does, we will put the ArticleId element in our second column. If it doesn’t, we skip the block, and check the next one. We keep looking through each of a record’s ArticleId’s, but we only add the ArticleId to the output **if** it meets the condition.

(DEMO IN CYGWIN)

efetch -db pubmed -id 16940437,16049336,11972038 -format xml | \

xtract -pattern PubmedArticle -element MedlineCitation/PMID \

-block ArticleId -if ArticleId@IdType -equals doi -element ArticleId

(EXECUTE)

The end result will be one row per PubMed record. Each row will have a PMID in the first column. The second column of each row will either be an ArticleID if the ArticleId is a DOI, or will be blank.

# If-Then: -if in a –block

(STAY IN CYGWIN)

Any questions about using –if and –block together?

**[PAUSE FOR QUESTIONS]**

# Combining multiple conditions

(SWITCH TO SLIDES)

Sometimes, you may want to include multiple conditions in the same xtract, and include only patterns or blocks that meet at least one of the different conditions, or that meet all of the conditions. For that, you’d want to use the –or and –and arguments.

With –or, at least one of the conditions you specify must be true. With –and, all of the conditions that you specify must be true.

# Using –or

The syntax is a little more complicated than what we’ve been doing.

We start our first condition just the same: our –if argument specifies an element or attribute, and we can have an –equals argument that specifies the value it should equal.

For our second condition, we start with –or, rather than –if. Then we can use –equals, -contains or one of those other string-matching arguments, if we want.

This example is just a modified version of our previous example. However, this time, we will be including the ArticleId if it’s a DOI, or if it’s a PMC ID.

# If-Or-Then

**If** the value of the IdType attribute for a given an ArticleId element equals “doi”, **or** the IdType attribute equals “pmc”…

# IF-Or-Then (cont’d)

**…then** put the value of that ArticleID in the second column, and move on to check the next one. On the other hand, if neither of those conditions are true, we skip that ArticleID block and move on to the next ArticleID right away, and check whether that one meets either of these conditions.

Let’s see what that looks like in practice.

(SWITCH TO CYGWIN)

(DEMO IN CYGWIN)

efetch -db pubmed -id 16940437,16049336,11972038 -format xml | \

xtract -pattern PubmedArticle -element MedlineCitation/PMID \

-block ArticleId -if ArticleId@IdType -equals doi \

-or ArticleId@IdType -equals pmc -element ArticleId

(EXECUTE)

We can see, because this process repeats for every ArticleId in the pattern, if a PubMed record has both a DOI and a PMC ID, then both the DOI and the PMC ID will be presented in that row, in the second column.

Any questions about using –or?

**[PAUSE FOR QUESTIONS]**

# Using –and

(BACK TO SLIDES)

The syntax for –and is the same as for –or, but the pattern or block must satisfy **all** conditions to be included.

For this example, our –if is followed by an –equals, but our –and is not.

# If-And-Then

So **if** the pattern has a LastName element with the value “Kamal,” **and** the pattern has any Affiliation element…

# If-And-Then (cont’d)

**…then** create a row for the new pattern with LastName,Initials and Affiliation

# Another –and example

I have one more –and example to show you. For this one, we only want to include records that have any MeSH heading that contains the words Zika Virus, and that have the MeSH heading Microcephaly.

# If-And-Then

For this xtract, if the pattern has a “DescriptorName” element that **contains** the string “Zika Virus,” **and** the pattern has a “DescriptorName” element that **equals** Microcephaly…

# If-And-Then (cont’d)

…**then** create a row for the new pattern with PMID and Article Title. Otherwise, skip the pattern and move on.

# -contains “Zika Virus” –equals Microcephaly

Let’s take a closer look at what’s happening here.

Here’s some mockup XML, showing some of the MeSH headings for a PubMed record. This record would be included in our results, because it has a DescriptorName element that contains the phrase “Zika Virus” **[CLICK]**, and because it has a DescriptorName element that equals “Microcephaly **[CLICK].**

This is why we used –contains for Zika Virus instead of –equals: we wanted to make sure we included records like this one, which have the heading Zika Virus Infection, but don’t have the heading Zika Virus.

Any questions about –and?

 **[PAUSE FOR QUESTIONS]**

# Exercise 3: -if

We want to do a search for BH Smith as an author, and see the different affiliations are listed for that author. We will limit our results to the last five years: 2011 through 2016.

We only want to see the affiliation data for BH Smith, regardless of how many authors are on each citation.

We want our output to be a table of citations with specific data, that includes PMID, author last name/initials (which should always be BH Smith, but including it in our output will help us confirm our conditional arguments are working), and affiliation data.

# Exercise 3: Solution

(DEMO IN CYGWIN)

esearch -db pubmed -query "smith bh[Author]" \

-datetype PDAT -mindate 2011 -maxdate 2016 | \

efetch -format xml | \

xtract -pattern PubmedArticle -element MedlineCitation/PMID \

-block Author -if LastName -equals Smith -and Initials -equals BH -sep " " -element LastName,Initials Affiliation

(EXECUTE)

Okay, if you’re still working on Exercise 3, I’ll ask you to set that aside for now, as we’re going to change gears a little bit.

# Finding the First Author

There’s one more conditional argument that we haven’t talked about yet. It works a bit differently than -if, but it can be pretty useful for isolating specific parts of your data.

Let’s say you want to include author data in your table, but you only want the first author. You can’t do this with –equals or –contains, unless you know the name of the first author.

# -position

However, you can do this using the –position argument.

You use –position in combination with –block. Earlier today, when we talked about –if and –block, we were including a block only **if** it met our condition. With –position, we can include a block only if it is the first occurrence of the block, the second occurrence of the block, etc.

So, if we wanted to include only the first block, we would add a “-position first” argument after it.

If we wanted only the last block, we would add “position last”. This is especially useful, since we can specify that we want the last block, even if we don’t know how many blocks there are.

When using –position, you can also specify a block by number, if you want. So “–position 1” is the same thing as “–position first”.

Let me show you what this looks like:

(DEMO IN CYGWIN)

esearch -db pubmed -query "smith bh[Author]" \

First, I’m going to run a script that you saw last time, when we first introduced -block.

efetch -db pubmed -id 28594955,28594944,28594945,28594943,28594948,28594957 -format xml | \

xtract -pattern PubmedArticle -element MedlineCitation/PMID \

-block Author -sep " " -element LastName,Initials

This script uses -block to go through each Author element for each record, and pair together the corresponding LastName and Initials. It’s also using the -sep/comma trick to put both elements in the same column, and separate them with a space.

[EXECUTE]

If, instead of all the authors, we want only the first author, we can add the -position argument.

efetch -db pubmed -id 28594955,28594944,28594945,28594943,28594948,28594957 -format xml | \

xtract -pattern PubmedArticle -element MedlineCitation/PMID \

-block Author -position first -sep " " -element LastName,Initials

This makes it so that xtract looks author-by-author, but only displays the first Author.

[EXECUTE]

We can easily change this command to show just the last author instead.

efetch -db pubmed -id 28594955,28594944,28594945,28594943,28594948,28594957 -format xml | \

xtract -pattern PubmedArticle -element MedlineCitation/PMID \

-block Author -position last -sep " " -element LastName,Initials

[EXECUTE]

# Dealing with blanks

(STAY IN CYGWIN)

We’re just about done for today, but I want to show you one more argument.

It’s not technically a Conditional argument; it’s actually more closely related to the other formatting arguments like -tab and -sep.

However, it fits thematically with the Conditional arguments: it’s a way of adjusting your output automatically, based on your input.

Let’s go back to our previous example, looking at just the first authors from this group of PubMed records. This time, though, we’re going to add in the Identifier element, so we can also see the ORCID ID for each author.

efetch -db pubmed -id 28594955,28594944,28594945,28594943,28594948,28594957 -format xml | \

xtract -pattern PubmedArticle -element MedlineCitation/PMID \

-block Author -position first -sep " " -element LastName,Initials Identifier

[EXECUTE]

This is all well and good, but if we wanted to add a fourth column after the Identifier, we might get into trouble, because of the blanks. Let me show you:

efetch -db pubmed -id 28594955,28594944,28594945,28594943,28594948,28594957 -format xml | \

xtract -pattern PubmedArticle -element MedlineCitation/PMID \

-block Author -position first -sep " " -element LastName,Initials Identifier LastName

In this example, we still have PMID in the first column, Last Name and Initials in the second column, and ORCID in the third column. We’re adding a fourth column, which should just be the author’s Last Name again.

[EXECUTE]

When a column would be blank (because there’s no data from the input to put there), xtract just skips it. This is causing alignment problems in the tables: where there’s a blank in the third column, xtract puts the data that should go in the fourth column into that blank. Everything gets shifted over, and your table is out of alignment.

The solution to this is the “-def” argument. It lets you specify a “default” value for the cells in your table: a placeholder, essentially. Wherever a blank would be, xtract will put the default instead.

So, following up on our last example, let’s put a default of “N/A”.

efetch -db pubmed -id 28594955,28594944,28594945,28594943,28594948,28594957 -format xml | \

xtract -pattern PubmedArticle -element MedlineCitation/PMID \

-block Author -position first -sep " " -def "N/A" -element LastName,Initials Identifier

Note that the placement is just like -tab and -sep: inside the -block (because -block resets our formatting arguments), and after the conditional arguments.

[EXECUTE]

And now our blanks aren’t blank anymore!

**[PAUSE FOR QUESTIONS]**

# Coming next time…

That’s about all we have time for today, and you’ve done a great job so far of sticking with us.

Next time is our last session. I’ll be back again, and we’ll try to put all of this together and look at some strategies and techniques for building solutions from the ground up.

We’ll also look at a few case studies, including some that were suggested by you!

# Homework

As always, the homework questions are at the bottom of the handout for today’s class. The answers are on the sample code page.

# In the meantime…

You can check out the website for more documentation. Ask us any questions you have with the homework, or about the course content, by going to the Contact page.

# Questions?

Or, if you have any questions right now, I’d be happy to answer them.