CAS WEBINARS

EQUINOX TRACK 2: WORKING WITH MARC RECORDS IN PERL

JUNE 11, 2020

CAPTIONING PROVIDED BY:

CAPTIONACCESS

contact@captionaccess.com

www.captionaccess.com

>> Good afternoon, everyone. We are going to get started with this program. This track is sponsored by NC Cardinal with captioning made possible by Equinox Open Library Initiative. And we would like to thank our captioner.
We would also like to think the other conference sponsors for making this part of the puzzle. Mobius and Evergreen in Indiana. This is being recorded and will be available on YouTube following the closing of the conference. We would like to encourage Evan to use the Chapman door to those questions. So please leave your mics muted unless you are unless Rogan is asking for questions. The commentators will be collecting your questions along the way and posing into the presenter at the end of the session. So feel free to put your questions in.
We would like to introduce the present for the session, Rogan Hamby from equinox, will be talking about Working With MARC Records in Perl.
>> Hi. Good afternoon. I am Rogan Hamby. This was originally proposed as a preconference session where I was going to cover in the 1st half of the preconference sort of command line, working with Perl and Marco.MARC records. With the shift to an online method of doing the conference, and my not really being sure how that had a presentation would work in this format, we scale it back in a bit. So I'm going to focus on what would have been the second half of the preconference and looking at PL/PERL or Perl inside functions. Which is when to be a pretty common use case.
Now, I encourage people to use the chat. Ask questions. I will try to keep an eye on chat. I normally use looking at people, using the will to list to see if I am going to quickly or need to back up a little bit. So I will have to rely on nonvisual cues this time. So feel free to make comments on all of that stuff.
This presentation is for everybody. But it is focused towards beginners. If you are an experienced developer, someone that is a lot of data work and has worked with Perl, you are probably not going to learn a whole lot here. This is to help people learn what they can do if they have not done this kind of thing and to hopefully empower them a little bit. But if you already know all the stuff in this presentation and you just want to hang out, you are welcome.
If you do not know Perl yet, and you want to learn it, I have good news for you my opinion is Perl is actually pretty easy to learn. But there are a few caveats. If you have not started working with Perl yet, and you go out onto the greater web and you look for how do I do this, you will probably find

3-5 ways to do it. And most of them will probably be valid. It is not a language where there is one way to do anything.
So I do encourage you to use all the resources out there in the world and learn. Somebody in the chat said there might be 40 ways. That might be true in fact, I know it's true sometimes.
But also look at if you want to eventually contribute to the Perl code of Evergreen, when how that is written. There are certain conventions it follows. But if you are writing just for yourself, there are certain best practices that you should follow, but you can kind of have your own style just for your own functions. If you are looking to interview back to the community, you should learn the conventions of the community.
What you will need to do this kind of stuff. You'll need access to your database, course. And you will need the appropriate levels of access. There are things you can do with PL/PERL that are interesting in MARC records with just read access, but the most interesting things involved writing access. So you can update and manipulate records. That's kind of common sense. That needs to be said.
And the question comes up of Wendy want to use Perl rather than just standard SQL? And the answer is when you either can't do it with pure SQL doing so is it ordinarily difficult for some reason. I made a reference in the slide about Rube Goldberg. If you're not familiar with Rube Goldberg machines, feel free to go to YouTube. There is a fascinating culture of people making extremely collocated machines to do simple tasks. If you find any good ones, send them to me. I love those. It consultants feel like that when you're using pure SQL. You're trying to do this thing that you know is very simple something like Perl, which is very strong at manipulations, and SQL is necking it very difficult. So to sometimes easier just to drop in A PL/PERL function.
And aisles provide the caveat, test servers are your friends. Don't test on live data. Always and forever.
The question that inevitably comes up, is what you used to edit? My answer is whatever makes sense for you. I prefer the command line. There are GUI things that work very well. I use psql, but if you prefer others, just use what works for you. It really doesn't matter so long as it pushes the job at the end of the day.
So what are PL/PERL functions? Well, the simple way of saying it is that a PL/PERL function is a bunch of code in the middle of a SQL function that is called out to the Perl interpreter. And it will pass parameters into it just like it would any normal SQL function and to return value from it.

Chris Sharp mentioned to chat that PG Admin is a good tool. It's probably the most popular one out there, but I know that there are others as well, and I don't really keep track of them.
So here are some of the things you're going to want to work with. In order to benefit MARC records Doc MARC Records inside Perl. These are Perl libraries. MARC character set, MARC Lent, MARC Field. Our users periodically but not with great frequency. The two at the bottom I have put in red, MARC Record and MARC File::XML, those are the meat and potatoes. Those are things you are going to use all the time if you are manipulating MARC records and Perl.
And you can use other useful libraries so long as they are available on your system. Business ISBN is particularly useful for looking at those darn 020 that are A and Z. And running the 10 digits to digits so you don't have to reinvent the wheel. And there is a text CSV if you have any kind of deliberative data you want to break up and manipulate.
Sure, you can split it up manually and do stuff with it, but why we reinvent the wheel?
So fundamentally, what does MARC Perl do for you? It just takes care of the grunt work. Abstracts manipulating all the little MARC level staff so that you can just focus on the task without doing all of the tiny level staff yourself.
Why? As most people probably know, MARC is treated in Evergreen like MARC XML. And because it is XML that also text, and you could manipulate it like text. But your pup -- you're probably going to be building one of those Rube Goldberg machines, and it's probably dangerous. Something that may seem like it's going to be a simple search and replace, you could pretty easily end up with unintended consequences and really start screwing some stuff up. So use tools appropriate to the job, in this case, MARC medication tools.
Now, those who don't know, we store MARC in a few places. These are what I think of as the most important places MARC is in the database. And they are the record entries for the bibliotable, the serial table, and the authority table.
Andy when you are going to use the most probably is the bibliotable. That is where your standard serial, monographs, DVD player, CD player, materials, are going to live. And you may use cereal and authority as well, just depending on the needs of your institution.
I mentioned this. MARC is stored as MARC XML. It is important to note because it pops up occasionally in terms of moving things around, that the actual field data type that is stored in is text, not XML. But that's okay. We

have easy ways to work with it. Unfortunately, only pass it through to a function, it is very easy to tell it is XML and create MARC Record object with it.
Now, I'm going to point you towards equinox migration tools. This is a git positive. Anyone is able to use the tools in it. And there is a seat SQL directory that includes a number of examples of PL/PERL functions, as well as standard SQL functions.
And the past, going from memory here, but I believe it's SQL base to get to the various SQL files.
So let's get into the actual mechanics. We are 11 minutes in and we haven't had a code slide yet? First, I want to take a second to see if there are any questions at this point that people have that we should address. I don't think I can see the Q and A window from here. I am sharing my screen.
>> There is not a Q&A. There is just a chat. There is a lot of chat, but I haven't seen any questions so far.
>> Okay. I am trying to watch chat while I do it. And I see there is a discussion about PGM.
If you are interested in a graphical interface, catch up with the discussion between Blake and Chris about that. I think Chris said that he uses psql plus vim now. For people trying to decide what they want to use, if you’re coming from a graphical user environment, I can understand that using the command line tools line is intimidating, but for pure mobility and being able to jump on and create a connection to somewhere and use it, it's really useful.
Bradley mentions Navicat. I'm not familiar with that personally. These are commercial tools that are very good as well. Do what works for you. I'm not a purist in that regard. Despite my giving some -- some grief, because it amuses me.
But if there are not questions at this point, we will return to the code slide. The boilerplate.
This is the basic layout of a PL/PERL function. And if it looks a whole lot like any SQL function, then it should. Because it is pretty similar. At the top, create or replace function, technically you don't even need the word replace. Give the name of it, see what you're going to pass as parameters, one or more, say how you're going to return it, do your Perl declarations. This is going to be different from a standard SQL function. You're going to have to use a MARC Record or something else. But you will do whatever you do and return from the function back to the environment that called it.

You're going to have a function declaration at the end to say that the function is over and declare the language as PL/PERL or PL/ PERLU.
I'm not going to get into much of a discussion between with PL/PERL versus PL/PERLU, I don't think that's critical. Suffice it to say there is a functional difference if you get into some of the ways that Perl was allowed to interact with the wider database environment.
Interview are using PL/PERL but not PL/PERLU, and you run into something you can't do, you should have a discussion with your database administrator about that. If you really need to do things that way, and if you need it.
But I will leave that between you and your database administrator and back into a whole lot of detail here. Suffice it to say, I use PERLU here, but PL/PERL does a lot by itself.
And here I am pointing, for those who don't know, the elephant is the -- I don't know if it's unofficial anymore, at one time the elephant was an unofficial mascot of the development project. I think it may have been officially adopted at some point.
But this is really in boilerplate the parts. Declaring the function and declaring the end of the function. And then the Perl part, as illustrated by the camel walking towards it is just the stuff in between.
So dead simple. Nothing to intimidate people. Is anybody feeling -- this is normally an in person presentation I would look out and look for people nodding when I ask if they are feeling overwhelmed by it. How are people in chat feeling about it? Good? Okay.
So this is probably the simplest actual PL/PERL function in the world that actually does something. All it does is it says it's going to be a function called print title. It's going to take text in as its input. Because the MARC XML is text. It is going to return text. And use MARC::Record MARC File::XML MARC Field. Feel specifically because I'm going to be looking at subfields in the record. I'm going to declare that I'm taking it as my first parameter the text box and declaring XML for it. Don't have to do it this way, but it's a normal Perl convention for taking in a parameter. You can use it here just like you could in a command Pro the command line program. And then I'm going to call MARC::Record and say new from XML, to that text blob and give me a record object.
As indicated by that record variable. And once I have that, I can use the field indicators to say give me the 245a and so that into a title variable and return it. That's all it does. Just print the title. By itself, not a terrible dock terribly useful function. But this demonstrates the basic principals involved.

And combined with other French analogy, you can build MARC reading programs that do quite a bit.
I'm checking something real quick. So can MARC past the parse? Did you get a slide? Yes.
So we are going to look at another function now to illustrate something that is functionally useful. This is from migration tools. I just copied and pasted from it, which I mentioned earlier. And this is just one step up from print title. Mike Fisher asked, what is shift? I will step back was a bit. This gets into Perl basics. I'm not going to go over Perl a whole lot here. But in Perl, the term shift is a common convention for take something that is a parameter and taking it off the array or list of parameters that have passed in assigning it to a variable. Exactly what Chris just said in chat.
And part of the reason I pointed it out here is it shows that delineation. That this region of text really is a Perl in here, not quasi-Perl.
So can MARC parse? This is from the actual migration tools. It's one step up from print title, but it does something that is actually useful. Just like the previous one, we have taken a blob of marked text Dr. MARC text, and we are going to declare some calling for the Mark record library sent XML and character set. And UTF 8, but I don't think I need that here. The big difference is this is an evaluation statement. We are going to try to make the record from XML. We are going to call MARC Record and assign it to a variable just like we did before. In this case we are not assuming we are going to work. We are going to evaluate it. And if the evaluation fails, we return 0. If the evaluation succeeds, we return one copy it's all it really does it say, MARC::Record successfully pass whatever your sending to us and making MARC Record out of it. MARC Record object because it can't, then is not useful to try to do other things with it.
This could be useful on its own to pass over say a collection of bibs, and say are there any bad bids in here? Other anything that can't actually be successfully noted into biblio.record entry or something like that.
We want to look at an example now of something where this can be useful. And I'm going to use an example of fixed fields. I forgot who it was that was asking, someone was asking during the herding bibliographic cats presentation yesterday, about the tools we use for fixed fields. Jeremiah. And I said I would show them off today, and I will, maybe a live station.
Let's say you need to change position 7 of characters from m to s. And I'm not going to go over how fixed fields work a whole bunch here. Even talked about it and is making the most neat MARC presentation, we talked

about it in the cat presentation yesterday. Feel free to go back to those for more detail once we have all these loaded on YouTube.
But suffice it to say if I changed the m to an s, that is in position 7, that this would be a serial record instead of a monograph record. Could you do this with pure SQL? You could. You could use a search replaced. You could use red X, you could use some positioning tricks with SQL. But what if there is some condition somewhere in the record you didn't anticipate? It just things like when there are better tools available, you should use them. Part of this is a very simple example. What if it is a little more complicated replacement? Say, things with repeating fields? Then you are laying the work for landmines. And if you need to learn the tools to deal with the more complicated situations, then they are the right tools to use and you already need to learn them, you might as will use them for the simple situations, too. At least that's my opinion.
So we are meant to come back to that when you do practical. Look at the fixed fields tools. We can go ahead and do that now.
Can everyone see the terminal screen okay? I need to make text a little bigger?
>> I can see it. You could make it a bit larger.
>> How is that?
>> That's better.
>> I'm going to warn you, I'm using some Bluetooth earbuds for listening, and I think they are about to die on me, though they should have three hours of charge on them. So I may stop being able to hear you in a little bit.
>> Okay. So we can type. (laughing).
>> So let's do this. The tools I talked about earlier are largely in migration tools schema. And of course at this size -- let me change this display. At this size is not going to be easy to see them. But you can see their names and there is a bunch of stuff in here for different functions.
So let's look -- this is using the -- data set, by the way. And let's look at actually changing a record and we will look at the Perl functions and associated code that works with it to accomplish what we need as we go.
So I'm going to pick on record 248. For those who do not spend in excess of another time in test databases, this is Ernest Cline’s ready player one, it is a monograph book record. And if you want to confirm that Evergreen knows it as a book, here is a little query that will get you that information for the search format. It reads from the metabib attribute vector list and the coded value map and config schema. And you type what the search format is. It says it's a book.

Now, let's change it and after I change it I will show you the tools that made that happen. So select * from migration tools.modify bibliofixed fields.
If you are wondering, why do I reference bibliofixed fields, these tools are in our migration toolset. We sometimes are doing this with records that we haven't loaded into the actual production tables yet. So there are variant functions for referencing stage tables that are not actually part of Evergreen yet. This one actually references biblio.fixed. The change from a book to -- what do people think? E-book? Blu-ray? Maybe a piece of music how is that? It's going to be wrong, but for demonstration processes it's fine.
Oh. It did not like something there. So we may have a bug to look at. Interesting. This is always the fun of doing live demos, right? Let's see what happens if we try something else. There we go. Likes that more. Perl was wondering how I also copied that area, so I will look at it later. I may have a patch to apply later today. But that is the nature of software, right?
So let's record that and see what it says now. And now it says it's a Blu-ray. Which is exactly what we wanted. And if we had the error pack for this year, we could go to NCD Blu-ray icon and all that kind of fun stuff.
So let's look at what makes that possible. First, I'm going to show you the contents of the table. It helps if I spell migration correctly.
It's going to be difficult to see all this at once, so I will go through a little bit of it. But this is a table in the migration tools schema that lists a lot of the fixed field information needed to set search and icon formats for some of the common types that Evergreen uses.
Andrea is known to -- as saying take 100 screenshots as interns. I could have done that, but why not live on the edge of? Andrea is probably cringing at me saying that right now.
For those not familiar with all the fixed fields in and outs, I'm not going to go over all of them right now because we would be here for a long time doing it. But some of them are very simple and straightforward. Some of them get convoluted. For instance, a lot of people don't realize that books, it is an item type of letter a and a bib level of A, but there are certain item forms and other fields than the reader that cannot have certain values and it would still be considered a book.
In check, fixed fields are the fun part. We need to talk sometime about your definition of fun, Mike. Just saying.
That is what this table is meant to represent. The things that can't be reporting forms, the item forms, it needs a bib level, but that is, but these

different physical this gift is how to become all that kind of stuff. And these things, depending on what they are, the 007's, and 008, and the leader. Can these transitive bib templates? Yes. Bib templates are basically bare-bones frameworks of these things.
Blake asked about music. There is music, and there are also specific subforms of music. So there is multiple variations accounted for in here. I'm not going to go through every line, but you can pull the migration tools and I didn't say this upfront, these tools were built to handle challenges of a specific project. They are a toolbox, but they don't cover every eventuality. There are things I would like to have done in them that Justin fit within the scope of those projects. So if anybody else uses these tools and build on top of them, because projects allow them to extend them, feel free to send me patters. I would be glad to add them.
So that is one critical part of this, and as the table of different search formats. Icon formats, and the fixed field data necessary to represent them.
Now, next up is I am going to actually pull up the text of it here. The function I actually called is what I call a quality-of-life function. And that was this.
At this size, it's not super convenient to read, but basically I send it the web ID, and the text of what I want the fixed field thing to be. And then it comes through here inquiries that search format table to gather all the information. As a little bit of analysis to say what needs to be included or not, and then calls another function called modify fixed fields to actually do the work. Just so I don't have to each time, query the table manually to send all that row by row information. It's a pure SQL function, quality of life. The actual function that does the work is really a little more fun – that is a little more fun than typing in my demo.
I guess I should have pointed out -- it doesn't matter. What that function calls this one called modify fixed fields. Which doesn't make any assumption about where the record lives. Stage table or anywhere else. It lets the function calling it worry about that.
And this is APL Perl function. Just like we showed before with the simpler examples, you create or replace, give the name, declare the parameters being passed in, make your Perl declarations, use strict, use warnings, I set a dated number in here probably from when I was doing testing elsewhere, maybe I was outputting to -- it's from my debugging period. Dated number is used in debugging a lot just to provide output. Like I want to know what ending of this array. Did I declare the array reference

correctly or is it not what I think it is? So you can give yourself the output look at it. That kind of stuff.
They're useful if you're passing around references rather than full objects, which is the good thing to do all the time.
If that sounds like gibberish and you are early in your Perl career, I do recommend reading up on references and pointers and all that stuff. If you do it early in your life, it will make your Perl coding much easier and it will save you a lot of time.
So the rest of this is just stuff. I don't want to go through it line by line. But I am pulling in those different parameters, looking at them, seeing what I need to do to recognize the market reader leader. Do I have to put in the 008? The placeholder, is the correct link for that type of record, which is going to vary. And in all the way down towards the bottom, I actually start putting all that in and turn out as a mark record -- MARC Record and close out the function.
And if we actually look at that record, if we look at the mark market, I am used to my text being very small here. Looking at it you just I type is kind of weird to me out.
If you look at it now, we can see that nothing script also does is put in a 919 this is the record was modified but automated fixed field changes. Which just leaves me a little historical trail to say this record was touched. And in other data projects like deduplications and things like that, I can make use of that.
Actually I did not mean to quit there.
So any questions at that point? Are not keeping up the chat very well. Any questions at this point? Okay. Where did my presentation slides go to?
>> So if you had questions about the syntax, that might be more than you can do.
>> Can everyone see the presentation slides again?
>> Yes. Can you hear me Rogan?
>> Okay. For Mike and Jeremiah, lots of questions about syntax. I wasn't planning on going over the Perl and a lot of detail here. Ideally for folks, that is unfortunate my speakers have gone out. Can everyone still hear me okay? I am going to try to get them running while I talk again.
And I'm really upset because these things are supposed to have a four hour charge and they were complete recharged before we started at 12:00. So I'm kind of irked about that. I am running my microphone off of much older simple analog microphone. Of course, the more advanced technology is what is failing.

Getting back to syntax, if you have courses about the syntax, feel free to send them to me later. I will be glad to answer what I can. I'm probably nothing best person in the world to teach anybody Perl, but I managed to teach a fair bit of it to myself, so I at least know what I do. But I am not a Perl group by any stretch of the imagination. But I can excellent my own code, for whatever that's worth.
Jeremiah missed most of it. Don't worry. This video will be up on YouTube, so we will put it up there. Mike is pointing to the O'Riley Perl programming book. I recommended for anybody learning. I am very fond of that book.
It has remained a staple of people learning Perl for 20 plus years. I don't remember when it was first published. Over 20 years ago, and for good reason.
So the next step after that little side trip into the realm of live demo, I was going to do another practical example. And this is adding 856 subfield 9s. 856 subfield 9s for those who don't know, are -- let me back up a little bit. 856s are parts of MARC records MARC records that say here's a record. In Evergreen, when you add a subfield 9 on them, they basically say only use this link when displaying at these org units.
So in the background, they create basically a call number to help control viewing and searching. And it's a convenient way of addressing the need of having electronic resource records like books from overdrive or axis 360 or hoopla or something like that available in the catalog.
And is not an uncommon thing for you to get a batch of records perhaps from a vendor, and they have the 856s, but you need to add those subfield 9s. Make sure -- is especially helpful in the consortium environment? Absolutely.
So what this function is going to do is add a subfield 9. Just like we showed before, and I feel like answering the same things over and over again, but that is the nature of a fundamental peer you want to go over it and over it again. Got the function it exists, create or replace function, I try to name things as confusingly as possible, so I just called add subfield 9. I'm going to send it to MARC, send a couple of pieces of text. A -- U and a new 9. What those are, is sometimes a record will have multiple it 56s in it. And you don't necessarily want to add your subfield 9 to every 856.
So this allows sending part of the text that would be in the subfield U, to say only add what is in this new 9 variable for the U and that it 56 matters the second part.
So say a record has several links, and it has one to overdrive, you can just put overdrive in here so it only adds the subfield 9 to the overdrive.

And then have the stock Perl stuff after we declare the function stuff, use strict, use warnings, used record, use MARC XML.
Once again see that shift convention. We had some questions and chat about shift. Last time we use shift once because we are only passing one variable. This time I'm calling it three times. Because I'm popping three times. There are multiple ways to do this. You can do all on one line, and just pointed at the array or there are other ways to pull it. This is how I tend to do it unless I have a whole bunch of the whole bunch of them, in which case I might do it a different way.
Here I am validating the record and the reader, and all that fun stuff just to make sure that anything is kosher. And going to process correctly.
Then at the bottom, I had my @uris. That is a decoration for an array or a list but basically a series of values. And I'm saying to MARC XML record and grabbed me the 856s. And then return unless at the @, that is a Perl convention for something being null.
So long as I managed to put nothing in there, there actually are 856s, it will continue. If it's empty, there is no need to continue doing something for the record that doesn't have any of them. So just return out.
And then I'm doing the work. For each of the fields inside the 856, a list of uris. And it's important to note that using MARC Record, by using Mark Maxfield, these are not going to be the actual XML nodes. These are MARC Record and MARC keel objects that you will be manipulating by those rules.
So you can go to -- and look at the documentation on them. Look at the tools for examples. It's pretty straightforward.
Gravity field, and grabbed the subfield U. I'm storing it in the SFU subfield. In Evergreen we care what the indicators are. We test for the indicators. We test the subfield U. Matching or having a partial match to our matching text. In other words, is it actually an overdrive record or something else we want to ignore? If it is, add the subfield 9 to the 8 to 6. And then pop out of there. And at the very end, the mark having been manipulated, return as an XML record. Remember, that MARC XML is not an actual XML record, per se, despite my naming convention here. I probably should have just called it MARC Record or records or something. But it is a record object. So we want to render it back as XML to return it back to post-arrest to save it and the record entry table.
And then I tend to be lazy. I don't want to actually worry about pulling the MARC out of the record and passing it. So I tend to create a simple SQL function to get the ID and the partial and mine text. And that handles it for

me. Because why introduce human error into something you're going to do over and over again?
And that is all this does. Pulling the MARC XML, sending it to the PL/PERL function, getting XML back and entering the bibliotable with it.
And this is an example. Select add subfield line, past the bid ID. So I want to test for LAV.overdrive .com, and fits there, And the subfield 9. And that's it. And I should be able to hear folks again, by the way.
So that is the end of the presentation. We have 5-10 minutes to chat before I have to surrender the space within next presenters. Which I believe are Galen and Mike, who will be talking about Making Use of Perl and Evergreen. So they will be talking about more intermediate to advanced stuff than I have. I suspect it will be pretty interesting. So I hope people who are interested stick around for their hesitation.
I'm not sure while Pearl Jam is continuing.-- I get it. I do like Pearl Jam. In fact, I was listening to alive the other day.
I do remember the 1990s. And actually I still listen to Pearl Jam quite a bit. I'm not sure which album it's from, but along with alive I was us into the three-part dance of the clairvoyants recently which I really have enjoyed. But probably not fair to listen to my musical tastes. And I apologize for having to captioner had to type clairvoyants. But I will be glad to hang on a few minutes for any questions or discussion. And when the host I really to take it back over, feel free to do so.
>> Thanks. Can you hear me? While we were talking, I actually charged my headphones to last at least 20 minutes. Second guess we will see if anybody has any questions, and in case you need to look at your slides, I won’t take the slides back yet.
The slides will go up on the website, and I will do some editing to add in a few slides with the live demo stuff we did.
>> Great job, kudos, thanks very much.
>> All right. I think there any questions at this point. I think Chris said earlier people were welcome to post a mailing list. Absolute or feel free to send me a message if you have a specific question or you don't feel comfortable living on a public list. But the Evergreen community is pretty friendly. I will mute myself and allow the transition to begin. Thank you all for coming. Have a good day.
>> Thanks, Rogan.