Training Session_ D3 Platform Overview-20240911_122225-Meeting Recording

September 11, 2024, 6:52AM

43m 23s

Sodanapalli, Praveen 0:07

So we got back on the call.
You just started the call with regarding. I started. You started the recording. OK. I just joined the call. OK. OK, let's do this thing. OK. Share the screen.
Then we'll do the presentation.
Bob.
You guys can see my screen right?
The guys on the call.

Mitrovic, Rastko 0:37

Yep, Yep.

Sodanapalli, Praveen 0:39

Oh, yeah. So let's go back from, I think we work through this slide. So all the tables and all that stuff, let's talk about the data model for the transfers, right.
We were talking about the source and destination, right. So if you look at the M2 transfer entity, we have a source. Maybe I need to sit here so I can talk into the microphone here. Right. OK. So we'll have a source and a destination.
And if you this is the entity definition there and then the source can be either source endpoint provider.
Or or there's an account I think somewhere.
OK, OK.
OK, it's just a source endpoint provider. So sometimes what happens is when we sync with the bill pay payees, we may not be able to find the endpoint for that page. So the payees come through and then payments come through, right. So when we try to link up the payments with the payees, we may not find the pay in that case. So what what we do in that case is we just put the external name in that field instead of linking that to the endpoint provider. So. So only one of them should be null.
Or not null. That's the null when.
Annotations do come for source or for destination. It can be either the link actual link

to the endpoint provider or the just the name of that one.
Or the user account.
Any one of one of them needs to be filled in, right? So if we have an provider link we we have a link to the M to endpoint provider table. If we if we don't have a a external endpoint in our database, we just put them in the name. That's it. Just a string field. So what happens in that case is the UI will show that the endpoint name just name of the P/E, but there's no actual link to that P/E OK or if it is an internal user user, either as a source or destination, we'll have that one. So one of them will be filled in for each one of them.
And you can see the our some of our booleans on the transfer entity saying whether it's an internal source or an external source, that kind of thing come.
So this is for pay multiple which is that worksheet that we're talking about, right are. Yes, we use it only for payments, which is the ACH payments are the bill payments OK.
And then the template is, that is.
Like a we save that template and then you can use that template for as many instances as you want. You can just create a template once and from that point onwards you can start paying through that template OK.
I think we talked about the worksheets and worksheet items, so this is what I was talking about and we can go faster now probably because we talked more about that before, right?
Yep, we talked about attributes transfer providers, right? OK, that's all good.
So these are simple examples of what we do for internal transfers we just create am to transfer row, usually for internal transfers we are the provider, we means we are the processors for that thing. That means we own the recurring model. We also schedule the next transfer. We check all those things when we are ready to send the transfer we send it to the core through core live interface come.
We send it, we put the status as pending and then we set it to processed. If it is today, OK.
There is one more I think Fhn users their own provider. What that means is they own the engine for that. So in that case it just exists like Bill Pay, we send it to them, they will do all of that stuff on their side, OK. The ACH is the A to a where you want the account at a different bank, right? We create the M to end point M to endpoint provider. Also is there.
Status pending. If it is pre note or I have. So for a CHRA to a recipe and we have two

ways of verifying them. Actually we have three ways of them right now so we can do micro deposits which is depositing those sample amounts. The other one is called a prenote. What that means is we we send an empty $0.00 transaction in the ACH file that called a prenote and then we wait for three days for that transaction so that transaction gets processed today for $0.00.
If that transaction does not work.
What that means is if the other bank, because when we run the SH file that other bank will be contacted with the transaction, if they come back and saying account is closed, this account does not exist, they will come back as the NOC snox right notice of action notice of something change. I think ach ones. That's a different process. We can talk about that. So when those come in that pre note we will basically cancel that endpoint. So that's another way of verifying that endpoint.
Other other other banks, they don't. If they don't allow, we know that sometimes no, not zero. It's basically it's a $0.00 for us, but it has a special.
Notation just like so it basically tells you. Just verify this account kind of thing. Yep, status check kind of thing. The other third one. The newer mechanism is called iav instant account verification. OK, which is what I think some accounts use it here to what they do is we contract that account verification to another vendor like mxi think there's another vendor that we were looking at pay where is or somebody.
What that vendor does is they will we will give control to that vendor, he will show the bank account screen there.
You will be entering user ID and password in that screen and then you log in to that bank. That external bank from within our banks interface and then when we log in they will scrape the UI information and then they will basically say your account is verified or not so that is instantly they are verifying the account where you are providing the user ID and password to the of the bank.
Yes, and that.
All of that functionality is provided by his third party vendor. We don't do that, but OK. Yeah, Max, I think he's the only one that does that one. One. There are other vendors, but we have not implemented them. We are looking at other vendors, I think, OK.
So that's the pending verification is for that micro deposits, right ACH transfer M to transfer row is created.
The user account is one of the endpoint provider because it's always the source. The destination is that A to a right? But.

Some efis will allow the bidirectional.
Most, most likely we we prevent it because it's your account on both sides, right? So we can pull the money from that other account, we allow it, but most of them don't enable that one. But most of the time it's going to be one way sourced to the destination.
Come status is pending process, right?
Modifying ACH transfer prior to processing.
Only way you can do that is if the date is future or if the date is not processed or pended some cases in the morning, but usually most of the ACH jobs run in the morning, so by the time you log in it's already gone kind of thing. So once it is in that certain statuses we have rules there. You can't edit the transfer Kemp.
What else? So that's Bill pay pays. So you can create Bill pay pay when you do that, we create the same endpoint endpoint provider rules, right? And pay. We usually set it to status is pending I think. And then we do a pay sync with the provider which is check free rafias and then they'll basically say you're active and then in which case you're active you can start doing your transfers immediately, right and then build pay payment. It's am 2 transfer every way. Every time you do a transfer it's going to be M2 transfer table.
That's that's the thing. And then usually pending.
There is one more status that is in process. I think that is the default status. So as soon as you create a transfer it goes into E in process status.
I no initiate it, I think initiated are impossible. Well, I can go and check. That is the default status for all the transfers and then when we send it to the core, our efis or somebody, and then we map that status. Usually it will go into pending at that time. But if we have not contacted or if the transfer has not transmitted to the vendor, it will be in that initial status itself.
Come there is one more status like approval pending once. So if transfers are going through approvals that's a different flow. So right so before so those things I think will be in a different status than that.
Yeah. We won't talk about approvals because it's a whole different topic. We'll, we'll talk about it next week. Akhm, you can cancel the payment.
If future dated payments, today's payments, you can't do that future dated once you can cancel them in those cases, that's that's what I was talking to you about yesterday. S PR, right? So when you cancel it, we update the status for it. We don't delete the things usually, right.

And then we send the update to the build pair service. Usually they'll mark the transfer as cancelled on their side too.
Alright so.
Our API it's basically we wrote that API as one set of API for all the rails. OK, the reason I was talking about this previously with DI is DI. Each rail is its own API for them. They don't have one centralised API. We did this one for discuss routing. I can talk a little bit more about that. So DI it's Bill pay. They have their own build pay transfers, bill pay endpoints, their own Rep APIs for them.
Internal transfers their own internal transfer endpoints for them, but for us it's the one thing when you want to create a transfer for any rail you just call EMM transfers. It's a crud, right? If you do a post, I think it'll create it, put it's going to modify it, delete day-to-day, get call, get right. Same thing for endpoint CMM endpoint. So you can create the endpoints MMM endpoint groups. It'll do the crowd on that one worksheets. It will do the crowd on that worksheet, right?
So that's what we do. So the least cost routing is we did it this way.
So that we don't want the user to.
Know what rail they're sending the money all they want to do is they want to send money from this point to that point and they shouldn't care how we are sending the money. We can send it through bill pay. We can send it through accounts on us. Any one of them. Right. So we wanted to do that, at least cast routing. So if the bill pay is cheaper to do it, if it between this source and this destination, we wanted to send it to build it. But if the ACH is cheaper, we'll send it to the ACH.
Or if wires is cheaper, we want to send it through that kind of thing. That is the intent of that. That's the reason we basically used one and one API for all of these things. That means we want to figure out we don't want to them to tell us how to send it. We'll figure out how to send it, but usually what happens is in the real practical way. Usually one end .1 rail kind of thing, right? We don't have multiple rails, so it always goes to one endpoint, yeah.
Exactly our code is up. We we have not added that code, but if we want to add that code we we can do that one yeah so.
It has to be the defaulted overnight or yes, so that's it's a different thing. That's a speed, right? Transfer speed. OK. So the only place where we use list cast routing is for honest transfers. So.
Yes. So when you try to send the money to the external account like ASEH and you enter.

Internal routing numbers.
So we will basically instead of creating an ACH payee, we will create an honest payee for you automatically. So the user Enduser does not even know that whether it's a Chr on us, we just create an onus because onas is, it's an internal transfer between the bank accounts, right? So it's faster, it's cheaper, it's free for them, whereas external ACH means the the money has to go to the Aach vendor and then comes back into the same bank, right? It's the both accounts are at the same bank.
So in that case, why do we need to give it to the ACH processor, right? The bank can do the transfer. That's the the thought process behind that starting thing. OK, that's the only.
Example of least cost routing that is in place right now, but we have built it so that we can do that least cost routing if we want to.
So that's the reason this AP is like that. So if we want to add that logic. So when that EMM transfers call comes in, we can analyse these things and then figure out this is cheaper. Let's do that, right. That's the reason we added it like that. But we don't have the code right now. OK.
Alright, so the I think we need to talk about that. Emm metacal. That is really important one. So usually whenever you log in and you are trying to load the UI for money movement related things, anytime you do.
Payment, stab or transfer stab. There is a call to MMM slash meta. OK, so that one. That's one call we it basically goes to the transfer provider, a table. We have a table called transfer provider, right that tells you all the transfer providers that are active for this client, right?
Let's say this guy has check free. This guy has wires, right? All of those will be there. If this guy has the FIS, then that will be there is say transfer provider. So for each transfer provider we ask them what are your metadata values. So metadata is basically one big object pojo object. It will tell you for this provider these are the frequencies. These are the editable fields like when you edit a transfer and these are the capabilities are for that rail. So each rail has its own specific things, right?
So all the specific metadata for each rail is returned and based on what we'll return in that meta call, the UI will react. UI will show if we show recurring transfers are not allowed then UI will not even show that recurring option for you, right? And there are other fields like we could walk through that there's a dead object dto object for that meta and then it has all the flags for each one of those features and.
For example, recently we added how far ahead you can schedule a recurring transfer.

The default is one year, right? If you want to schedule a transfer two years in ahead of time, you you can't do that. So that that's a meta field, right? So some like, let's say A5 server can come back and say you can't schedule something more than three months ahead of time. So for that transfer provider it will be 90 days. OK. So there are other flags like that. So we'd return all of that thing. So the UI will react based on that and then they will show the things for you. OK.
And then EMM enrollment, that one is only active for certain rails. So build pay you do enrollment for internal transfers, you don't enrol, you're automatically there, right. So the ACH is the same way but you do enrol for Bill pays you do enrol for Zelle you. Not sure. I think you do. Enrollment for transfer now, not wires, I think. OK. E bill. See is that endpoint is only for Bill pay. OK. Nobody else has E bills. OK.
Alright, so architectural or we write I I think some of this stuff you already probably know this one and this is where we were talking about why we had to do it this way versus the previous mechanism. So in 2X we had money movement and that was how we had some issues with the performance. We had some issues with the lazy units transaction, hibernate transaction management that kind of thing. So what we did in 3X is.
We.
But we did. I think there are other slides that will tell you more, more about that one, but I'll read through some of this stuff.
The services in money movement are they do very little kind of thing. They do more top level stuff like limits or validations. But most of the stuff is delegated to the adapter. So when you are creating an endpoint we basically say give it to whatever the transfer provider is, let's say build pay right then build pay will do lot of heavy heavy lifting. They will do cheques for that provider right whatever that provider is. So the service doesn't have a lot of stuff, but it does some top level things because we want to verify at the service level whether that endpoint is good or that transfer is good, right? And then there is one more service layer.
This provider SPI service layer, that is.
We had to introduce that recently, not recently 3X release right. The reason being we used to have this those methods in the basic service itself and then we were coming into controller to the service to the adapter and then back to the service, right. So instead what we did is we created a new provider's SBI services.
So there's a new package underneath that service provider interface. Espi means, right? So that's a new different service. So the adapter will call that service adapter

will never call the endpoint service, the top level services, OK. So the hibernate transactions are always at provider SPI levels, OK. The reason being if you initialise a hibernate transaction at the service layer and then it calls the adapter adapter blocks right, because it's making an external network call.
The Hibernet transaction will expire our timeout right? That's the reason we don't. We never do it at the service layer. We come back and do it at the provider espi layer, right? So the adapter after it gets the response back from there, then it will call this provider SPI and then the save it to the database. OK. So the transaction is at this level. So what that go ahead.
Service layer. When we do the calls to the adapter, we handle the calls to adapter mm HMM or something else. We have some of the operations in the service level and the transaction goes away. Let then pull. That cannot be. Yes. Exactly. Yep. Yep. Yep. I was getting to that one. Yes. So that is possible. So what happens is what? What's his question is. So you are at the service layer. You created the endpoint, right? Endpoint is the top layer, right? And then?
You look through adapters. You find the adapter for your build pair, right? And then you call the adapter.
The adapter will create the endpoint provider for you because the provider object right, the endpoint is the top level. Each provider will have its own objects, right? So the provider object fails. Right now it the endpoint is at the top level with no providers underneath because this guy failed underneath right? So that code that service code actually looks at that one at the end and then cleans that up if there is no endpoint provider, it knows that something failed downstream, right? So it it deletes the endpoint. Also at that point.
Makes sense. So at the service layer takes care of these things. We have to manually take care of that one because there's no automatic hibernate transaction rollbacks. OK, so based on the status of the exception, I don't exactly. Exactly. Yeah. So we roll roll back our sometimes the transfer that's the reason. So like let's say the controller is a transfer controller, right? The transfer that service layer, the top service layer will set it in the initiated status I think initiated our initial status I think in process or something like that.
And then the adapter will make the call when the adapter comes back, then it will go into the pending status. If the adapter has some kind of network permission, the transfer will be still in the initial status itself. Then it's gone, right? It's it's not no longer progressing at that point. OK.

Alright, so the other thing that we did is we have dtos between the layers. This is what I was talking trying to see in the next slide.
Previously we used to the give the entities between the service layers and adapters right, so the entities, if we initialise it in service and give it back to the adapters then we are getting this lazy and it's all that stuff. So what we did is exactly So what we did it is only the service services will have access to the entities whenever we cross these boundaries from the service to the controller or service to the adapter we only talk in the DTOS.
So Dtos don't have the entity or we don't have hypernet transactions right? So. Exactly, but the entities will only be accessed in the service layer. That's that's. That's the new thing. OK. And then whenever you're writing the code, you probably want to do that one. So your service definition never sent returns an entity back.
Or it never takes the entity inside OK, always dtos and then what we did is we one went one more player above is controller player has API dtos. Yeah. OK.
So we have two levels of DT OS, the the back end dtos and then the APIDTOSAPIDT 00 S are the DTOS that we send back to the UI and they don't have some sense through data. OK, because we are sending the data back to the UI whereas in the back end between the service and adapter we have to send a lot more data right? So we have more detailed dtos there, OK and then the provider SPI adapter to provider that one is also DTS. So whenever these arrows are there, that's all dtos, right? Exactly. And then this provider SPI to the repository. That's the entities, because we have to save the data there, right. OK.
Good. I don't mapping exactly part of mappings yes and then mappers. Yes, yes we do.
And the last thing is what we have is message that I just. So what we have is.
For faster comparison between the entity so endpoints. So we create a digest of that Endpoint, digest is all the important key value fields for that endpoint like the name, nickname, amount. Those kind of thing are active status, right?
We create a digest of that one and then when we get the external endpoint from buildpay provider, we create a digest there and if the digest matches then we don't have to touch this entity. We don't have to make a database trip right? We just compare that digest and then go on. If the digest is different then we do the object mappers that he was talking about and then we create and we go fetch the entity, we update that entity, we save the entity. We do all all that stuff.
Cap so that digest is going to be our shortcut kind of thing. Just compare it quickly

and then see if it matches OK.
Database calls right? So that way we are not doing tying up the database. That's most expensive network calls, right?
All to repository methods, right? Mm hmm. How would you use this thing?
From the saving point, how code reuse right? Which one are we talking about? Yes. So between the rails.
So build the adapter calls the same service provider interface, calls to save the endpoint and also the internal not internal but ACH transfer also will use the same SBI services. It's a common service, right? That's how we are doing it instead of adapter. Basically writing the code to save it right? So adapter can call the repository directly instead of that we have a service provider layer that way that service provider can be used from all these adapters.
No, APIDTOS is what we are using in the controllers right? But summary DT1DT detailed DT OS are also we use it in controller summary had does not have all the fields. We don't actually go to attributes table and all that stuff summary just gets you enough information to show that small thing on the UI like when whenever you you see the transferred list table.
Right. We just show transfer name, schedule date, status, amount right for fields and summary DT would just have those four fields and when you click on that transfer we expand and we show a lot more stuff, right? That's when we go get the dto detailed dto right?
So we use transfer list is summary because we are giving you back 50 transfers. We don't want to hit lot of different tables right? Just performance kind of thing yeah. OK so.
This one is lot more complex, but we talked about this before, right? So we have endpoints transfers, E bills is special for just E bills, right? And if you talk about that we, although we have all these different rails, it's much more simpler. We have one controller for creating all the endpoints transfers by E bills, right and then the money movement adapter is just an interface and then each each of the providers will implement that money movement adapter.
And then we have different adapters for check free each rail right ACH adapter FIS adapter. All of those guys and each adapter has methods to create the payment, create, update the payment, delete the payment, cancel like, create the provider. All of those things. But we we have not created multiple things for the endpoints or anything. The adapter will take care of all of those functions for you. OK.

In the same signature, yes, yes. Yes, exactly. So you might ask this, this E bill controller and E bill service is only for Bill based, right. So what does the ACH adapter will do those things for them it will not implement those methods. And if you call those methods on that adapter, you'll get a feature not enabled exception.
So and then the provider services, we have these endpoint provider service transfer provider service and then the E bill provider service. They're all there in the money movement based repository, OK. And usually we already have all of those, most of the functions that we want there you, you should see them. That's the code we use. We already have them. You don't have to code new ones usually right. But if we are coding here like let's say tomorrow you have a new build pair after for G1 right pair rails or something.
For them, you just use those existing web methods from that service. You don't have to code them again. Yeah, exactly.
And then we have, I think we have repositories.
At the main service layer and also at the SPI service layer come.
The in future for a specific feature we don't really. We may see that our work and our transfer or endpoint schema might change and what impact does adaptive.
Adapter does not need to care about that because it's a service layer, right? But yeah, we change our entire schema and release of our T3 vdos and OK, if if we are changing the dtos then the adapters also has to change because the DTO will change, right? Yeah.
But the signature of exactly, you'll have to change all of those signatures to. You will change it in the money movement adapter itself, right? Usually the way we code our APIs is we have backwards compatible, so we only add the new fields. We do not delete the fields or we do not.
Yes, yes. So what you do is you do not change the existing adapter methods method methods. You go add your own new method.
Always, always that way you don't break everybody else.
Yeah, whatever. We'll. Yeah, we'll not have a problem if if somebody's still there, then they will still be calling that 1K. Or if you are switching to a newer version, newer API, you'll need to put a change log or something saying that you need to go implement this new function somewhere. All the places, right? It's an interface, right? So we have to provide a concrete. No, you can do a default.
That's what the if you go look at the money movement adapter, go check it out, everything is a default there. It the default is. It'll throw a not feature feature not

enable the exception for you, right? And we have we will exactly and you have to override it. Yeah.
Is the service directly.
Communicates with the repository, so service DT was right. So whenever not directly repository so the service will create the endpoint for you.
Are the transfer also gets created by the transfer service and then it will be sent to the adapter. Adapter will get you a status for you service. Initial service will create the transfer for it right? Or if it is going into the approvals it will just go into the pending before that one. Yeah so.
Do you take the response? We will again convert it to provider service layer. I mean that will come and get with repositories and save not all the time. I think not all the time. So sometimes what happens is the top level service layer will create the entity and then the transfer provider service will just update the entity with the status that we get back. So we get back status and also we get the external reference number or something like that confirmation number right and then we update those guys.
Yeah, they'll have the they don't think kind of thing you will have to we can walk through the code and then we can see which one is creating that right. So the top level services are in the SDK package. These ones are in the espi package. If you see the base money moment base, you'll see two packages there. Yes, ticket package and yes. Yeah, espi package, right. So that's the two layers of the services. OK.
Alright, so this is a typical call stack. How the request comes through the controller and then the service will do the schedule transfer inside the schedule transfer we do some validations, limits, cheques, whether you have money or not. All of that stuff right, or whether the endpoints are good. Whether you can do the transfer between this and this, all all kinds of cheques and then.
So that's the validation part, right? So there's a validation of the get endpoints for user. We check whether the you can do that, whether you can do the scheduled date, all that stuff right when everything is good to go.
We do call that process transfer request on that after itself and that will do its own validate cheques kind of thing.
Basically that one also can go out and create the transfer at the third party and in internal transfer case we just created ourselves, right? And then.
Transfer providers will create it and then we it can do a sync to get the latest status and all that stuff. Yep.
Yes, there is an approval interceptor. There is a worksheet interceptors for certain

things, yes.
That's the AOP doing its things right approvals. We implemented it as a AOP. So that approval interceptor basically intercepts the transfer and cheques whether you need approval for that, if it it needs approval then it does not go into the other thing. So if it needs approval, we don't even send it to the bill pay provider at that point it it is set up set up to be an approval transfer only when it is approved that's when we send the transfer to build pair.
Secondary user do any of like yes, yes, yes, yes and then the worksheet approval interceptor also does the same thing for the worksheet level. Yep.
I thought this is roughly last one I think so. This lists all the repos we have.
For money movement, I know this is a lot more big ones, but most of these repos are not microservices by themselves, they're all broken down by the rails, so those top ones are the adapter repos. They're all.
Produce artefacts that are rolled into money movement D3 money movement somewhere. Where is the D3 money movement?
Actually this one is we got this week. Yeah, this is from. This is from a week in D3 money movement. So all these adapters, these are all the current adapters we have and then they all roll into D3 money movement. We have AMM jobs where we put all the jobs, the scheduled jobs clean up jobs, all kinds of jobs. MMM clean up jobs is where we can do the clean up things.
We have the same repeated here.
MMM, base is where we put that SDK stuff. The SPI interface all the.
Service interfaces are there right? I think there is some core livestock also there for internal transfers. D3 ACH is its own service that processes the ACH files. The ACH records all of that stuff, right approvals is approvals.
When something needs an approval then it goes there. That approval also has.
Calls the endpoint rest endpoints to provide the status and an approval kind of thing so the UI can do a pull trying to figure out whether something is approved or not. D3 fed wire repo has the D3 wire.
No, that one should be gone. I think there should be a wire adapter reporter there that has all the information. This one, this one D3 fed wire just has the schema dtos swagger dot that will provide to Zions. Other than that there's no code there, they it does not have a artefacts or anything there.
And then money movement export is what he's talking about yesterday, right? So that one will do. The PDF exports for you PDF of a transfers.

Endpoints, I think. Then that's a newer functionality D3 Money movement Web hook is mainly used for incoming web hooks for wires, and recently we were doing a realtime web hook to so wires is a. Basically we send the wire to a fi.
And then they will. Basically when the virus processed, when they get the confirmation, they'll send the updated status through the web hook for us. So that's an incoming call to us, right. And the real time transfers also follows the same thing.
And then the last one, the one that has the build failing thing, that's the builder direct, that is a new feature for FIS, so.
Through Efis, you can pay some billers directly. Basically, you're paying your loan payments. Those kind of things.
But that fys is what do you call their sunsetting that product? I think that's that's another way to say it. They're killing that product in a way. So we don't have A use for it and I think they're going to call that they're they have a new product they have. So we're waiting for that new product spec to come along. So we'll rename that and then we'll repurpose that repo.
But that is also part of our current thing. Money movement stuff, OK.
Because that is not an adapter. That's not a rail.
We decided that that is.
Not at the rail level kind of thing. It's just an architectural decision. I think we were, we had discussions about this, right. So a rail is a much more expensive process if you want to do a rail, you have to provide transfer provider table, right? You have to have a metadata, you have to have an adapter, you have to have a lot of lot more stuff, right? This one, what we decided is it's not actually providing a transfer, but it's just doing the last minute. What do you call?
Last mile payment processing.
So the transfer actually lives on the ACH trial for till the end. Only when we are about to process the transfer that that's when we send it to biller direct. So that's why we did not design it as a rail.

41:13
A.

Sodanapalli, Praveen 41:14
If we want, we could do it that way, but then it's a lot more work for us to create a range. It's that one is a custom code. So yeah, Windows paid for it. Similar

experiment that we created. Adaptive yes realtime. We figured out that's a new. Theoretical it's not a full rail, it's.
Not Sav, it's it's 3/4 kind of thing. The reason being that rail cannot exist without Esch. It depends on Esch. For most of the information. So the ACH is always needs to be there for the real time transfer provider to be there. But it is its own rail so the money can be sent through real time, right? It has its own frequencies, it has its own scheduled date, it's its own things. It's not like it is separate from ACH but at the same time.You need to create the ACH 1st and then create the real time.It cannot exist outside of the ISCH provider. That's the reason I'm saying it's not a full rail, but it's mostly rail.Famous with somebody?No, no, no, no, no, no, PPDCCD still ach, real time is not ach. We don't write ACH files real time. We send it to a real time API. It's a different API, right, the the processor is a different person. That's the difference, yeah.
OK, I think this is the end of it. Yeah. So what time is it? 1:00, right on time. See.
Alright. So any questions before we break? We'll probably break for lunch and then we can come back, talk about the alerts. Right.
No questions. We can stop the recording, right? Whoever is recording and then we'll come back 2:00, right?
OK. Yeah. So Disan and Rasko Brasco will come back in an hour, roughly 2:00, OK.

BD **Batinica, Dusan** 43:21
Yeah, sure.

MR **Mitrovic, Rastko** 43:22
OK, OK.

Anagandula, Venkatesh stopped transcription