FRANK MAYFIELD: Welcome, everybody. Let's get started. It's 1 o'clock on Wednesday at AU, kind of the traditional midway point. And you're in Perfect Plots, Every Time, Every User.

Or more accurately, that is, if they actually do what you tell them to do. Because we always-- we all have those users, right? No matter what you tell them, they're going to go off and do their own thing. And besides, that wouldn't fit all of the proposal form, so I took it off. That's really a more telling example of what we're talking about.

So by now, you should have seen the summary. Today, I'm going to be presenting my method for making each-- making sure that each file in your project has the same page setups that you created for it, and the users won't have to do a thing to make it work. They just go in and pick their page set up and go.

And hopefully, when we're done, you'll understand the basic concept and be able to modify my code for your situations. You'll realize that, without a standard file naming and folder naming convention, you may be a little bit out of luck. So if you don't have that, you might want to get your ducks in a row first.

And I'm going to talk about some other ways you can apply these methods. So more than just learning some new code today, I want you to think outside the box and to think how else you can use these concepts in your office. And in the end, I'll try to present some good examples of that. I hope to. And finally, we're all in the same class-- in this class for the same reason, right? And that's to learn something new that will reduce our picks and clicks, and make you more productive back at the office.

So who am I? My name is Frank Mayfield. I'm a CAD manager for the Oil, Gas, and Chemical group at Benham in Tulsa, Oklahoma. I've been using AutoCAD for a very long time, as you see, most of my professional career. About half that time I've spent as a CAD manager in some regard. And I've spend about-- spent about five years as a programmer using common lsp.

And if you've downloaded a trial of AutoCAD or AutoCAD Lite in the last couple of years, you may have had the opportunity to have a live chat with an All-Star mentor. And that very well may have been me as I have had the good fortune to talk to over 2,000 new users in 50 countries worldwide during that time. It's been a great thing. And this is my 14th AU. First one

was Los Angeles in 1997, and couple of required social media links.

So the focus of this class-- and again, for those of you who weren't here to start with, I've had a cold, so normally I like to stand up and talk and engage, and I have to sit down for a few minutes at some point so I'm going to do it right now. So the focus, it's aimed at CAD managers like you who are dealing with multiple projects, and possibly multiple clients, and therefore multiple page setups. Now if that's not you, don't worry because you'll still get something out of this class I think, but it'll just be a lot easier for you in the long run because you don't have the complexity. And no matter what level of complexity you have, you'll find that the most important aspect is that you have a standardized folder and file naming convention. Because for the method I'm going to show, it won't work without it.

And as you saw in the bio slide, I have been to 14 AUs. So I've been to a lot of classes in that time, which of course means I'm going to a lot of very good ones, and I've been to some bad ones too-- some not so good ones, maybe I should say-- and that includes programming based type classes like this. And from my perspective, most of those that weren't so good that were reprogramming based had something in common. Most of the time they all feature the same thing, which was the dude writing code, right? Well why did some of those turn out not so good?

First of all-- kind of like I'm doing now-- the presenter is sitting behind his laptop, bathed in a aura of light, desperately trying to reproduce the code he remembered to make the class. Inevitably though, something bad happens. He gets a syntax error, or something doesn't compile, something doesn't work right, and now it starts, and I've seen this is million times. He or she-- and everyone in the first few rows-- spends the next 10 or 15 minutes trying to find the missing quote-- closing quote, or the closing paren, or whatever it is that's breaking his code.

And I just didn't want to do that, because that can foul things up. So instead, I'll be showing you plenty of code examples today in the slides, but I'll spend more time talking about the concept instead of writing code. Well I won't be writing any code.

Now you may have seen the handout already. I'll be loosely following that. It goes into much more detail than what I'm going to speak about today. And of course, my code is posted on the web site too, which I expect you to hopefully download. You'll have to change it for your specific situations.

So what exactly is the problem with page setups? First, let me say I'm not bashing page setups at all. I love page setups. And if you go back as far as I do-- back into the late [INAUDIBLE]-- they're are a huge improvement over the good old days, right? The old days were so good. The problem, though-- and this is the main thing-- the problem is page setups are saved in the drawing file itself.

So what that means is you can have a page setup in two files that is named exactly the same, but the settings can be totally different, right? So as you can imagine, this can create huge problems-- excuse me. Because you see, if a user sees a page setup in one drawing file named a certain thing, they're going to expect that page setup to do the exact same thing in another file, but sometimes that doesn't happen because they can be different.

And when it is different, suddenly your users don't trust them anymore. And when they don't trust them, what do they do? They start clicking on things. They start clicking on things that they shouldn't be clicking on, to tell you the truth.

So when I prepared this class, I was thinking about just that. Because I used to go around my office and I'd see people clicking on-- trying to make their plot work right, and it drove me nuts. So I wondered how many plotting controls could they find to click on? Does anybody know? It's in the handout, but how many total controls are there do you think? Just call it out. What was that?

AUDIENCE: 35.

FRANK MAYFIELD: 35, that's a good answer.

AUDIENCE: 32.

FRANK MAYFIELD: 32, well depends on how you want to count them, right? So I found 19 things here that they can click on. And in my office, right up there that guy, that's the only one that need to be clicking on because I've done all that set up for them to begin with.

But not so fast. That guy down here in the corner, he's not really in control is he? No, no, no, no, he opens up the dialog to show us another 16 things that they can click on to screw up your page setups.

So that's a total of 35. You're exactly right, and if that was a guess, congratulations. Maybe

you just want to go ahead and leave and go back to the Casino, right, because you're on a roll. So that's 35 things they can click one, not good.

Let me tell you a little bit about my current job-- as I crunch up my cough drop. You might find it to be similar to you. Normally, at any one time, we're working on a number of different projects, maybe five to ten, a dozen. That'll usually span a number of different client teams-- three, five, something like that.

And most of those times, the clients are absolutely fine with us using our standard title block and our CAD standards, but not always. And no matter how hard I argue against it, it seems like I always have one or two of those that always insist on either using their title block, or modifying ours in some way to suit their needs, or we have to use their layers and colors so that they can print it the way they want to when they get it back. Whatever the case might be, it's not our standard, so it's not my standard setup.

And to add to the confusion-- in my office, anyway-- we have four printers on our floor of people. It's a long, narrow office building, so folks at one end like to use a couple of printers, folks at in end another two. And of course printers always go down right, so you can't rely on all those being up all the time. And we have a single large format plotter, and of course we produce PDFs and DWFs.

So when I started this current job about five years ago, I found that this new group of core users that were there, they used sheet sets, all right? Well, this was new to me. I didn't use a Sheet Set Manager in my previous job, but they seemed to love it. And they embraced it, so I was like, OK, new job, I'll jump on board with you. But it didn't take long to realize why they relied on the Sheet Set Manager so much.

And it wasn't what you think. It's not for all the cool, neat features that you can use it for. It was used mainly, and primarily, for plotting consistency or printing consistency.

So what they did was to use the page setup override file within Sheet Set Manager, and that housed all the page setups for the projects. And this file with a specific template file that was always saved in a certain folder within each project. So it became obvious to me-- excuse me-- obvious to me that this new group of core users had printing problems in the past, and they tried to fix it themselves using sheet set-- SSM and the override file, and they finally got some trust in their printing.

Well, we were pretty busy at the time, so we had a lot of people coming and going. We had some big projects going. And these new folks didn't always get the rules that my core people understood. And the rule, by the way, for them was always to print everything through Sheet Set Manager. Whether it's a single print or you're publishing an entire set, they did it that way.

Well, these new folks, they come in and they want to use the plot command like we were taught to do, and that's when problems started to creep back in. You see, the page setups from that override file-- they should be correct-- they may or may not even be in the drawing that these people are opening. And then the abusers weren't sure if they were right or not, and they might start getting bad plots. So they started trying to fix it themselves, of course, by either trying other settings in that plot dialog like we saw, or worse they changed the page setup, or they create their own. And needless to say, things were a mess, and I needed to get my hands around it really quick.

So given this, what do we do? How did I go to fix this? Well, if I had my own perfect world, page setups wouldn't live in each drawing at all, they'd be part of a group of project settings that's applied to a project and live outside the files, and there'd just be a single spot for it.

But since AutoCAD doesn't do that, I needed to try to create my own little perfect world where the page setups would always be the same no matter what drawing you're in or what method you're using to plot. And finally, I wanted this solution to just happen. I didn't want the users to know what was happening. Wanted it to be magic to them.

And I certainly didn't want to require any interaction on their part. I didn't want them going into a menu to pull down things and get their page setups right. I wanted it to be magic. Let me get a drink here.

So how are we going to do this? Well, I needed to come up with some kind of algorithm in my head for the problem. And in the end, it was really pretty simple in its scope. The first thing that needed to happen is that it needed to fire off every time somebody opens a drawing. Well, the ACADDOC.lsp file is perfect for this. I already use it to manage other things when open, so that piece of the puzzle was already in place for me.

The next thing I needed to know was something about the file that's being opened. Because it can be opening anything, right? And for me, the best way to do this was to analyze either the file's path or its name, possibly both, OK? So once again, it's essential you have a good naming convention in place.

So now you've analyzed the file. You may know what kind it is and where it is, and that should give you enough information to know where you can find a master page setup file for it. In my case, it was that override file in a certain folder. And so once you know what it is, where it is, and where to find it setups, then all you have to do synchronize those with a little lsp code.

And really, that's a very simple two step process. First, delete the existing page setups, because you can't trust them. Somebody might have changed things. And then, just import the good ones after you've flushed the original, once you know that or right from your master file because you made it for that project.

So we're going to go quickly and break down these steps. If you're not familiar with ACADDOC and ACAD.lsp-- ACADDOC.lsp and ACAD.lsp, AutoCAD does look for these files in its support paths, and will load them if it's found. And by default, ACADDOC loads every time a file is opened, and ACAD.lsp only loads with AutoCAD is launched. You can use them for two different things. I use DOC to manage things as a file is being opened.

Now these files don't exist when you install AutoCAD, you have to make them yourselves. Now I keep mine in a custom support folder on our network. And that folder is actually defined in the deployment so that I can control everyone's environment with that file. So when I deploy AutoCAD to a user, that support folder is there. The first thing it sees is ACAD and ACADDOC.lsp, fires off, and I have complete control.

The second part of that little algorithm was we have to analyze the file that we're opening. Now what do I mean by that? What do I mean by analyze it? Well, it's easy enough to get both a file name and its path with lsp, couple little calls there, couple getvars and you have those. Then it's really just a matter of searching for something that you know about the file that constitutes-- that makes you know that it's a valid file to work on.

Now, that could be a particular folder or a naming convention-- which is what we're going to be doing, because I think it's probably the most common scenario-- but it could be something else altogether. You might have metadata embedded in the file through, say, drawing props, or you might have a custom library written into it. You might want to search the block table for a particular block. That might tell you, hey, this is what I want. Or maybe even an entry in the title block, if you find it.

Thing is, everyone's situation in here will be different. This turned into a big hypothetical thing

for me, because I realize soon that nobody in here has the same folder structure or the same set up at their office. But there's got to be something in there that will tell you what the file is and what you can do with it.

So for fun, let's look at some possible paths, and hopefully these look a little bit familiar to you. So here are three possible examples of file path out in the real world. You can see they might start with, oh I don't know, a division or a business unit, maybe a year you do them by.

Further into that, we might finally get to an actual client group, and finally project name or maybe a project number. And then after that, you'll get some more divisions down for administration, engineering, all the things that work in a project. And at some point, you'll find the folders for say BIM or CAD, maybe AutoCAD, whatever you call it.

And then under that, you may have some discipline specific folders, something like that. I imagine most of your folder structures might look similar to this. I would hope so.

Or you maybe a little luckier and have a structure that looks like that, but you have a drive letter that tells you everything you need to know. Got a head nod u here. Knows what I'm talking about. So I was at a-- at my employer before this, the W drive, that was designated solely for a specific client.

It was a particular big box retailer that started with a W, you might have heard of them. No, it wasn't Walgreens. And the G drive, we had a G drive, and that was for government work, and it broke down that way. And it worked-- it was pretty easy for me, because that's all I had to look for.

And hopefully-- I've already said a few times that I hope your naming structure is in place, because there's bad examples in the world too. There's-- the tragically hip David who likes zombies. You never can find anything on his drive.

And worse than that, I've seen people work off their USB drives. Please don't. And then, the worst of all for me-- the hot button for me-- is working from your My Documents on your C drive. We have network shares for a reason. Don't do that.

AUDIENCE: [INAUDIBLE]

FRANK MAYFIELD: They can, yeah. Again, every situation-- everyone's situation here is different. Excuse me. So what are we looking for in these strings that we will get back, the paths, and then what

happens when you find them? Well, you simply might need to know what the client is that you're dealing with.

In this case, you may have a common folder for each one of these clients that contains this master page setup file, a support folder if you will. In which case, the solution becomes pretty easy for you. Figure out the client, know where to go find the master file. Or again, even better, like John here in the front, you may only care about the drive letter, and that might tell you everything you need to know.

So the third step in that little algorithm, looking for-- after you look for the client name, this is where you actually build a path to get to that master page setup file. Glad I got some water. First of all, I start with what I call a global function to get the file's path.

And here, I just used the getvar drawing prefix to return the entire path. And then I define what I call predicate functions that simply return t if it finds what it's looking for, or nil if not. It's a Boolean function. Here we use wcmatch to look for that client name.

And a note for you here, you'll notice that I wrapped the getvar in a string case. The auto list string functions are generally case specific. So if you're looking for anything lowercase with a wcmatch in an uppercase path, it will return nil and you'll wonder why I didn't find my string. So I go ahead and wrap everything in a string case to force it to uppercase. And then whenever I'm forcing a path in or typing a search string in, I make sure it's capitalized also.

Of course, you can do the opposite. If you like lowercase, you can use the optional t option in string case and set it down. Either way, pick one and stick with it or you'll get in trouble when you're searching strings.

All right, so armed with those little helper functions, it's pretty easy to write some conditional logic to decide which page setup file you need to use. In this case, I already know this file lives in the support folder under a client name, and I know what its name, right? I'm the one that was responsible for making it and putting it there, and putting the paid setups in it.

So in the second example with the drive letter, we'll just get some-- make some code to get the drawing's path, and then I'll extract the drive letter using the substring-- just using substring to get the first-- from the first character, I only go one, so I get one letter. I don't wrap that one in a string case, because I don't think you have a map drive that's lowercase, but I might be wrong. This probably would have been good practice to do that. So I return one

character in length, and then some more decision logic.

So in this case, we only have two clients, let's say, so a simple IF statement is going to work for you. If it's a B drive, you know it's a Buy More, so you go ahead and set the path appropriately. Otherwise, in the IF, it has to be your Large Mart client, so you set it to the Large Mart file.

Now both of these are pretty easy examples, and they either search for it or extract a string for evaluation where the client or the project drive is a known quantity. But we can look at another example which is a bit more to our focus today, where the client and project really doesn't matter. And it's a bit more like my office that I describe to where each project has its own named page setup file and it lives in a certain spot.

You might remember that I said, we use an override file in conjunction with the Sheet Set Manager, and we keep the sheets set, and templates, and any of my CAD setup stuff in the 40 CAD folder of each project. That's our name. And this is one of our standard projects, let's say.

So I know where the page set up file lives no matter what the project or client. So I don't have to go looking for the client name and match it against anything, it's always in the same spot. And these could be other examples too. And of course your mileage may vary, but if you know where it's supposed to be and what it's named, you can still build a path to it.

Another side note, if possible, you might consider adding the preceding folder to your search, especially if your folder is simply named like CAD or AutoCAD. There we go, things like that. And the reason why is I've seen too many cases of people creating a CAD folder again on their C drive in My Documents, so they have work CAD maybe, or CAD to differentiate from their stuff they're doing at home in the backyard. It shouldn't be there.

And in the last case here, notice that searching for engineering CAD, that would greatly reduce the likelihood of getting a false positive on your search. That's why, if you can put that leading path on there, it's a good idea. Because CAD can be a lot of things.

So let's look at this scenario now. Again, we'll start with a function to get the drawing's path. And there in the first line, we'll use that-- set our drawn path variable. I call it dpath. We do that.

And up until now, we've used wcmatch to find the key string in the path. But here, we're going

to use a visual lsp function vl-string-search. Now it too will return nil if it doesn't find the string, but it's going to return the position of the first character of the first string instead of t.

Now this is important, because we can use that to help build our full path to our page setup file. Here, you see that that has returned the integer 42, which is the-- where it first found string CAD backslash. Note too that I add the backslash after CAD so it doesn't find catalyst or Cadillac or caddyshack, or something like that.

It stops with that slash. So I set the value to a variable start position, adequately abbreviated of course. And now that we know what-- that we found what we're looking for and where it is in the string, we can continue to build the whole path.

Now since we want to strip off any paths that follow CAD, I use the substring function again to pare it down-- oops, come back here-- to pare it down to just CAD. Excuse me. So I go four character's past where I first find it, because it's C-A-D slash. For a substring, I return that third string here. And finally string string cat the master file name onto the parse path, and now I have the full path to my file again.

And finally, the last step in that four piece algorithm, and while it's the most important step because it does do the synchronization for us, it's really also the easiest and the most straightforward. And this is where we delete the existing page setups and import the good ones. So to delete them, we'll use visual lsp again. And of course if you do, don't ever forget to run vl-load-come to start with.

And here, what we do is step down through the ACAD object to the active document. I'm going to call to get plot configurations, and those are your page setups. And I run a clax-for which is a for loop. And for each one I find, I delete it. Get out of there. I don't know if you're right or not.

Now this code will probably be verbatim for you, unless you need to look for something else in them. And after that, we just import the good setups from that master file we built the path to. And remember, I said I don't want the users even knowing its happening, so I turn off command echo.

Then I run the command line version of PSETUPIN, which is page setup in-- its an import function. And at the end I use an asterisk, right there, to denote it's a wildcard to indicate I want all of them to come in. And it will bring in both layout page setups and model space page

setups, which is fine.

And since, on occasion, we may get a prompt to overwrite depending on your situation, here's a little lsp trick to query whether the command is active or not. And if so, I'll return a no, because no I don't want to overwrite them, I want to just-- because I've already deleted them, right, so I just want to import them back in. But there may be cases where that happens, so I leave it in there.

And that's it. Those two pieces, it's done. And we've talked about this for about 10 or 15 minutes, it seems like a lot of steps and a lot of code to do it. But in reality, the whole thing, each time you open a file, takes milliseconds to run. And, when it's properly conditionalized, it should only work on what you wanted to work on.

That is, if your conditions aren't met, the code just blows through and does nothing, OK? So I have it set to-- it and knows I'm a sheet file. And I only work on those, that's how I get past it. That way, of course, it doesn't throw any errors when it runs on to David's zombie apocalypse drive.

So real quick, a few minutes about organizing the code, or maybe more specifically how I organize mine. So I mentioned earlier, ACADDOC, if found, will run every time a drawing is open. And that gives you a lot of power over your environment. And there is a number of different ways for using it. Again, I prefer a single file in a shared network location. That allows me to make changes in one place, and it'll apply to everybody in my environment.

So if you use ACADDOC already, it's probably no surprise that I do things, like I said, environment variables. And as I go through some of these, they're just examples. I think I counted over 30 different variables in my ACADDOC that I set.

I also ensure that the drawing file type is set to where I want it each time so somebody is not accidentally saving to a wrong file format. And I have some autoload functions, which may be custom functions that are right for a specific purpose. I don't want them loading every time, so they demand loads. When somebody calls it, it loads for them and goes. I have a number of those.

And of course, I may load and execute lsp code to perform any one of a number of tasks. And in this case, this is our class, these are the load and execute commands. Loads and fires.

And finally, although the order of showing you this is a little bit backwards, the first things I do--

excuse me, oops-- is to load vl-load-com right at the beginning, because I know I'm going to use it in other places in my code. So I don't call it twice. There's no need to do that.

Then I make a call to this cool little guy called acad-push-dbmod, and it works in conjunction with acad-pop-dbmod which will appear at the end of the file. Briefly, basically what that does, if you ever set system variables in your ACADDOC, and somebody comes into a drawing and immediately tries to exit, it will say, do you want to save. And the user's like, I didn't do anything.

Well, I did. I set some variables, and I did some things, and they didn't know about. Those two little guys will keep that from happening, and make your users a little happier. They open a file, go, naw that wasn't it, close, and they don't get that, hey you want to save or not.

And most importantly there is this call to load globals.lsp, and I do that right at the top before I continue with the rest of my startup codes so that anything that's defined in there is already loaded. So what's globals? Well, I load it in the beginning in ACADDOC, doc what it does, it sets a lot of global variables and functions based on that drawing that I use in other routines whether they're called on startup or on-demand. If nothing else, it's a good way to not have duplicate code all over the place constantly calling things.

Now I've seen similar systems where people might call it a nit or a start up, or files like that. There's nothing magical about using the name globals. I learned it during my programming days and brought it back over to this system. And the things that are in it-- and again, I put my entire globals file out on the website. So there's functions I've been using to get the drawing info, that's in there, and some predicate functions that return to your nil-- or in some cases a value or a nil.

Here, I'm looking to see if the file is on a different server other than my own. I may not want to deal with something that way. And sometimes, I need to know what version the client's using, so I have some functions to figure that out as well. And I might want to know what the discipline it belongs to, if I'm in a multi-discipline firm. Is it architecture, structural? That can be real handy.

Now we use some third-party software at my firm. And you may be running verticals like Plant 3D. You can have checks for those defined in here too. And finding out what program is running can be a little tricky at times, but there's always something that differentiates the

flavors. Here, the plant content folder variable doesn't exist in regular AutoCAD, it only exists in Plant 3D, so I can look for that, and if so, it returns true.

And we use this little program called CADWorks in my office for now, and it loads a custom arcs routine. So you can look in the arcs list, that little guy, and see if that is a member of it. And you'll know, hey, I'm running CADWorks, I know I need to do certain things. Sometimes finding these keys might take a little digging, but there's always something there.

And on a final note, in my globals file here, you've seen that I wrap them in asterisk where it's returning a variable or a function name, and that's simply a method for me of distinguishing that my global functions so I can visually pick them out of my code easier. Again, I've seen other people do it different ways-- their initials or whatever it might be-- or you might not care at all. It's nothing magic, and you can choose your own method. But if you do, make sure you use legal characters, otherwise your autolsp will barf. It's not good.

So I found that one of the biggest challenges in preparing a class like this-- one that doesn't deal with a straight forward A to Z procedural concept-- is to develop hypothetical scenarios that will hopefully show you enough diversity in what you may encounter when you try to apply it in your company. So in developing the handout, I decided to step through four scenarios, starting with what I thought was the easiest. And with each that follow, it becomes a little bit more complicated.

And while we've been discussing this concept already, I actually managed to hit upon the first three of those. So we're going to recap just a little bit by looking at each, starting with the easiest. You'll remember we talked about this. You're one of the lucky ones who has every drawing in your shop.

They use the same page setups so no matter what. All you need to do is make sure you have a file in a shared location that contains them all. Then it's just a matter of deleting them and importing the good ones every time you open a drawing. You're done.

So remember how we did this? We used some visual lsp code to loop through the configuration object, deleting each as we go. Then some old fashion lsp brings in the good ones. Couldn't be much more simple than that.

Adding a bit of complexity, remember you may have clients or programs at your work that each uses their own set of page setups. And so the master file for this is, say perhaps, in a

support folder for each one of those. And we talked about what this analysis might look like. You may only need the drive letter. It's a B. You've got a-- oops-- if you've got a-- if it's a B you've got a Buy More, if it's L it's Large Mart, et cetera.

Or possibly more common, you need to look for the client name or a number or some identifier in the path. Here, we've defined some predicate functions that tell us if it's a Buy More or A Large Mart, or an Orange Orange project. And of course, you don't want a bunch of nested IF statements.

That's poor programming. So you'll probably be using a conditional. And when you do, make sure you put the most probable condition first and the least probable last, that way the calling won't have near as much to do, good form.

And then the third more complicated scenario, you may have multiple projects spanning a number of clients or program groups, and each may have their own page setups. So I showed you where you may only need to find the key string in the file's path. Excuse me. Remember the key string in our example, it was the engineering CAD folder structure not just CAD, which will allow you both to validate it and build a path to your own setup file.

And it did-- again, we did this by finding the keystream-- engineered CAD in this case-- parsing the original path down to where you find it. We used wcmatch looking for CAD, and then vl-string-search to get in the position. Once found, we substring the rest of the path out and then string CADed on our master page setup file.

One thing to note here, as you're doing this, you can import page setups from both drawing files and DWT template files, but you cannot import them from DWS standards files. I don't know why. It seems like a logical place to keep standard stuff, but you can't do it.

And then finally, I wanted to present a worst case scenario. But you know, there's no way I can cover everything you guys might run into, or probably everything I've run into. In fact, it happened Thursday before I flew in here.

So I picked out three examples, and hopefully they'll be similar to your complications you might run into. And I'll show you how I address those, which hopefully will spur your creative juices when you run into your own. Again, I go into a lot more detail in the handout, and in the code that I posted. It's, I think, pretty well commented.

So these possible complications, well you may need to actually bypass syncing the page

setups for a particular drawing type, or load an alternate set. We'll look at that. Excuse me.

Second, up until now, we've simply done a blast deletion of all the page setups in your open file-- in your file. Remember, we used the asterisk wildcard to say blow them all away. I'll show you a few examples of how I handle changing that.

And lastly, try as you might to keep a standard folder structure in place, there's always going to be that one that just doesn't fit-- the outlier. It's just different for whatever reason, and you'll have to make exceptions for it. We'll look at that too.

So the first one, bypassing, what happens when we need to bypass something? Let's say it's a particular file type that doesn't need to get synced. And by that, I mean, in my office, we have isometric files and I need something different for those, and I don't want to bring in the normal set of page setups.

So here, I've written a small predicate function to find out if it's an ISO or not. I key off the folder, but again it may be the drawing name itself or something inside the drawing for you. Again, it's all different. And then I simply add that condition-- add that check to my AND condition so that if it fails, it returns nil, the whole AND fails, and the code just goes on and does nothing from there because I didn't find what I was looking for.

Or instead of bypassing the ISOs, you may want to load a different set. That's pretty easy too. Simple IF statement may work for that. So here, if it is an ISO, I load one piece-- one bit of page setup, otherwise I load something else. It's pretty easy.

Here's another example of why you might want to skip the synchronization. Sometimes, we might open a file in our office that's over the LAN in another city. We have multiple offices. And I probably don't want to put my page setups in their files, because they point to different printers, and folks in St. Louis just about to come to Tulsa to pick up their prints.

They don't like that. So I take them out. Here I'm checking the path to see if it's in Houston or St. Louis or whatever the case may be, and then simply-- before running the synchronization code-- I make sure it's not from either one of the cities. And if it's not, I continue on.

The next piece of the problem, just blindly deleting all the existing page setups, it's just not going to work for you. For whatever reason, there's some in there you may need to keep. And you may need to blow away the others, though. Here's a few examples.

This is one trick I've used. I just make a list of all my good page setups, the ones that I know they're right. They're the ones you want to delete and reload. Anything else stays. Now those might be a client or a consultant set up-- they expect it to still be there when they get the file back-- or different offices like I just talked about, whatever the case. [INAUDIBLE]

So during that deletion loop, I get the setup name using the visual lsp method get-name. Where did I find that? There's get-name right there. So I get the name and check it against that master list using member. If it's part of the list, I know it's one that I want to delete, so I do if not, it does nothing and just moves on.

And here, we're going to analyze the setup as we're looping through them. So we'll get the-- uh oh, sorry about that. I told you it might happen. So we'll get the name of the setup again, and check it on the fly against possible patterns, again using get-name. Notice here wcmatch. It allows you to separate patterns with commas.

So if there's only a few things, two or three, it might be a very good method for you to check against. And remember, wcmatch is short for wildcard match. So if you're not familiar with all the dozen or so wild card examples that you can use with it, consult the help function, because it has a complete list and good examples of using each. In this case, I've just been using the asterisks.

And finally, one more way to analyze the page setup itself, so in the first two examples we used get-name. But in this case, we might want to check what actual device that setup is directed into. And you can do that with a get-configName method, as we see here. And that's the actual device pulled down in the plot dialog. That's where that will get you.

Now in my office, IT has named each printer and very predictable manner. So I know PTOL is always in the device name somewhere, so I can simply test for it, and delete it if the page set up as a match. That way I know it's mine. Thank you.

And the third worst case scenario from my example is the outlier, the project that just doesn't fit your norm. So to set this up, let's consider what a typical project folder structure might look like. Again, there may be some upfront folders and things like that before you actually get to a client group. And then that might expand into individual client projects. Here, we're looking at to Buy More projects again. And then, each project of course further expands into its typical subfolder structure.

But the outlier is different. This example, this was created from actual real world example we had back home. We were tasked with providing engineering services for a client where there was a single project for their accounting purposes, but in reality there were dozens of subprojects under that. Yeah, if any of you have done government work, this is similar to an IDIQ.

AUDIENCE: On-call services.

FRANK MAYFIELD: There you go, on-call services. Something like that, it's not your normal setup. So we ended up creating a structure similar to what you see here on the right, where there's a bunch of subprojects under the main project or contract. And here's a comparison between the two. So you can see our normal project on the left, and on the right was a special case.

Now you recall, I told you, that in my office, we always keep our templates, et cetera-- and that includes the page setup master file-- in that CAD folder. But in this case, I put them all in this general folder right here, because they're going to apply to everything, and I don't want dozens of the same file. They can be wrong, so one general folder. So everything's going to live in there now. Hope that's making sense by now.

So here was my solution for this. I wrote a simple predicate function to determine if I'm in this goofy project or not. Here I'm keying off of a project number, because I know what it is. And now, we're going to have our normal logic, because of course most projects are going to be normal.

But in this case, I've wrapped the entirety of that code in an IF statement, and it checks to see if the current file is not this oddball project. If not, it goes on about its merry way doing its thing on a normal project, doing its normal checks, blowing out if it needs to. But if it is the ELSE statement of the IF, I just change my arguments here where I'm getting a substring of that path to account for the new general folder, and then string cat-- excuse me-- string cat that master name back onto that new path, whatever it might be, and therefore I have it. A little odd, that way.

So that's kind of how I do that. And remember, I told you one of my goals was to challenge you to think outside the box here today? In really this session, well I've talk page setups, it wasn't solely about that. And to be honest, maintaining a single file full of your master set of page setups, writing this code to synchronize them, it's really not all that hard. It's rather simple. The key to the whole thing is analyzing your file when you open it so you know what to do with it.

Now hopefully, more than a few of you had the same thought during the last 45 minutes or so, something like, yeah, yeah, page setups, whatever. But yeah, man, if I always know what kind of drawing file I'm opening, wow, I can do blank-- whatever it is you're thinking up-- and that's cool, and that's what I hope some of you have thought. And remember, it's your situation back at the office.

It's not mine. I do different things than you do. You're the only one that knows what it is you need to know about the file, and you're the only one that knows what else you can do other than synchronize page setups.

So where else can we go with this? Let's look into this for a few minutes. What about the file type? Well, it's easy enough to get the extension. I won't show you how to do that, but may tell you what you need to know.

Maybe you need to bypass certain types of files. I'm guilty. I know, at one time, I forgot to filter out DWS files and template files, and things would accidentally get updated in those when I didn't want them to. That was-- [AUDIO OUT]

You might keep that in mind. That's a pretty simple example. And up until now, we've just analyzed the path. And again, I've told you, your situation might be different. You might have a very sophisticated file naming structure that's going to tell you everything you need to know about it.

Or there may be something within the drawing that you put there. At a previous job, I wrote a custom dictionary entry-- dictionary entry into my templates. I tracked the version and info about them, where they were used, and I could search for that and know what it was. And like we talked about earlier, there may be something within the drawing-- like a block, or the title block itself, or an entry in the title block-- something that you can search for so you know what it is. And again, only you know what this might be.

So with the things we've already looked at-- disciplines or file types-- let's say you are in a multi-discipline, for and you might decide you need to know what discipline a file belongs to. Of course, that's useful. Of course, I used isometrics earlier, which is more of a type than it is a discipline, but we'll continue on, and you might want to know if it's an architectural file or a structural or a plumbing, civil, whatever it might be. And once you figure that out, you can act accordingly.

This was actually so important to me once that I wrote some code similar to this that I query the discipline, it would return a string that told me what the discipline was and I'd go from there. Well, that's cool. Now we know what kind it is. It's structural, MEP, whatever it is. Now what can you do? Well, here's some examples of some of the things I've done.

Fee is a standards file, you attach a standards DWS file. Well, you can make sure it's the right file in the way in now you know what it is. And if it's not, you can attach the right one just like that.

You can make sure that that discipline's tool palettes are all loaded and active for them so they don't have to go searching for them. You can verify the layer list. Or if it's wrong, you can load their layer list. Same with styles, if each discipline has different styles, any of those styles that we list there.

And you may be like me. You might have a fancy layer import function that's dialogue driven. I go ahead and set the defaults in that dialog to the correct discipline-- architecture, structure, whatever it is-- when it comes up, try to help my user out. I even throw up a warning if a load the wrong set, because I know what they should be loading, right?

And finally, to get you thinking a little further about this, I'm going to share with you a case study on controlling lazy users. I'm sure none of you have those. Well, we had a huge project going on-- again, I'm piping-- so we had dozens and dozens of complex 3D piping models. And they're all [INAUDIBLE] to each other, and all [INAUDIBLE] together, building a big tree. Lots of people working on it. I'm sure you get the picture.

Well, it came to my attention one day that no matter how often we mandated that thou shall turn everything back on when you close the file, they weren't doing it. And it was causing a big headache. It was causing workflow headache, and problems in our Navisworks file that we were trying to get back to clients. Things were turned off. They were isolated.

On this particular job, we were using this third-party program called CADWorks, and it has a proprietary line isolation tool, works kind of neat. And they were leaving things isolated. They'd just work on a particular piece of pipe, everything else was off. Wasn't good. In addition to that, we were-- they were told to make sure it set back to 2D wireframe, and a single model space viewport. This is all for the next person that opens it so it would not dog on the way open.

Well, it wasn't happening. And the bosses came to me and said, Frank, is there any way I can

force it somehow? Can force them to do this, because obviously asking nicely wasn't working, was it? All right, so I thought about it for a while, and I came up with a possible solution. But I knew that before I could apply that solution, I needed to learn some things about the file.

OK, well I kind of already do that already. Of course, I needed to make sure it was the right client and project, and we looked at that a number of times already. I needed to make sure CADWorks was running, because I needed that special tool. I showed you that too.

And lastly, I knew that it needed to be a model file. It couldn't be one of our sheet files. And in this case, I was lucky, because we have all our models in a single folder, and I just had to search for the folder name. So I was feeling pretty good about it.

So what was this solution? Well, I decided I'd just undefine the close command, and I'd write my own. I'd launch it from that ACADDOC file. It would always fire up.

All right, so I'm in sweet so far. Of course, it would fire for every one, and every thing that got opened, but I could pretty simply structure it so that it would only do my custom stuff on my target and just act normal for everybody else. So people in structural, they had no idea that they were using my custom close command.

Let's glance at that code real quick. Obviously, the first thing I did was undefine the close command. Now, if you've never undefined a command and AutoCAD before, yeah it's a great way to mess with people, isn't it?

Remember, after lunch or when they're totally messed up, go back and use the redefine to re-enable it. Anyway, so after that, I decided, OK, I'm going to write my own close command. It would do my bidding for me.

So the first thing I did was make sure I was working on my targeted file. So right off the bat, I use those predicate functions to make sure it's the right client and that the user is indeed running CADWorks, and that it's a model file. And finally, I wanted to make sure it is on my server too. And as long as all those conditions are met, I'm good to go.

And then here, I simply set the viewports to single, right in here. I'll set visual style current to two, which is 2D wireframe, so I'm doing all the things that they were supposed to do for them. And then I found the hard way that I can't access the line isolation command straight to a command called in AutoCAD. And it lives within one of their arcs commands, and I would have

had to run a bunch of reactor's to figure out what that thing was.

So I found that I could run a script and run it from there, so that's what you're seeing here. It was a workaround right here, and all I did was contain that command. Then after that, get the ACAD object, the document, and I send the command close, and it does all that stuff that they were supposed to do, closes the file. And of course, the ELSE to the original IF for the structural people, or whomever it might be, it just closes and they never knew anything happened.

So we're almost to the end. So as we wind down, I hope you'll be, now, able to understand-- as we look back on some of the learning objectives-- hope you'll be able to understand and apply my code to your unique situations. It won't be plug and play. This isn't one of those class where you go copy paste and you're happy. But you should be able to use it now as a starting point.

And again, while synchronizing page setups was the focus of the class today, I hope spurred some thought in you. Hopefully you've gotten to-- gotten you to consider how else you can apply these concepts in your office. So that with them upfront planning co-development, I think you can streamline your printing workflow, you can reduce errors, and you'll save money both on time and materials. And that's always a good thing to tell your boss that you got from AU. Maybe you'll come back next year.

Hey, I want to thank everybody for spending an hour of your valuable AU time with me today. Please don't forget to fill out your surveys both for this in all your AU classes. Of course, you can do so online or through your AU mobile app. If you have any other questions at all, you can always go to the answer bar outside the hub on level two. Thanks again. Everybody have a great Autodesk University.