BRIAN EKINS: So I'm Brian Ekins. I work for Autodesk. I've worked for Autodesk 18 years now. And I started in the CAD industry in the mid-'80s and was more an end user. But I got involved in the CAD industry with the vendors pretty early on.

I worked for Intergraph initially, and I was doing training teaching people how to use CAD. So end user training. And then was an application engineer for quite a few years, and doing really more advanced modeling stuff. And as part of that, I enjoyed programming. And so when I was developing-- putting together material to show customers how our software worked better than everybody else's, I would write little programs to help with that.

And then went Intergraph started developing Solid Edge they said, hey, why don't you help us design the API? I said, I don't know anything about that. And they said, well you have this-- you use the API, so you have a user's perspective. So they talked me into it. So I did solid edge API.

And then came to Autodesk in the late 90s, and worked on the inventor API. And now I'm working on the fusion API, but still heavily involved with within inventor.

So I have had this blog that I haven't had a lot of time to work on lately-- and so there's another guy that's also participating in it-- but it has a lot of Inventor material here and were posting some Fusion stuff but I want to add some more Inventor stuff too that I think important but

And so my expertise is the inventor API. So I'm not a big iLogic expert, but I'll kind of explain that as we go through here. So why teach this class? And a lot of it is based on my experience looking at the questions coming in on the forum and answering a lot of times. And so just yesterday, I guess, I went on there and snapshotted three pages and then grouped them together. But all of these have iLogic in the name. Right? So people are asking all these questions supposedly related to iLogic.

But then as I would go in and look at the questions, a lot of them had nothing to do with iLogic other than they were using iLogic to have the code, and we'll talk about that a little bit. But that's what I want to do, is really talk about what is iLogic really-- because I think there's a misunderstanding.

So what is iLogic, and what isn't it too? And then talk about what is the inventor API, and how does that fit in with iLogic? And a little bit about how to use it too. And then what tools are there available to use the API? And then learn how there's lots of samples available out there that aren't iLogic specific. Right? And that's what I would notice on the forum a lot, is somebody who ask this question that really wasn't iLogic specific. And I would write a little sample in VBA, because that was the fastest. Post it back, and say, oh I can't use that's. It's not iLogic. so talk about that.

So what is iLogic? And so this is the marketing definition. If you go to the help, this is how they say it. iLogic enables rules-driven design, providing a simple way to capture and reuse your work. Use iLogic to standardize and automate design processes and configure your virtual products. Obviously a marketing guy wrote it, because you read it 10 times, you don't have an idea-- clue what it said.

So that's the marketing definition. Here's the technical definition of what iLogic is. So we've got Inventor. And then sitting on top of inventor, really kind of part of inventor, but it's layered on top, is the API. And I'll go into more detail about really what the API is. But then on top of the API we have what are called Add-Ins. So these are programs written to add functionality to Inventor. So you've Tube to Pipe, Cable and Harness, Inventor Studio, pretty much all the apps that you would go to the Autodesk app store and download our add ins. And iLogic is an add in.

So it's not really part of Inventor core, it's this add-in that's been-- it's being delivered with Inventor. So just like Tube and Pipe and those other things, it kind of feels like it's part of Inventor, but it's just this add in. It's running outside. And it talks to Inventor. It does things in Inventor by calling through the API, just like the other add-ins.

And iLogic also has a big dependency on Visual Basic NET, so this and the NET framework. So my definition of what iLogic is, real simple, we'll go into more detail of things. But it's this development environment that was really designed and built for doing simple configurations of parts and assemblies right. with very little code-- and sometimes no code. Right? I can go into iLogic and make this little dialogue and not write a single line of code. So it's pretty cool for these kind of things.

And it also provides some simple functions specific for a configuration, and I'll go in more detail in what I mean with that. So iLogic provides this development environment-- you'll hear this term, IDE, if you start looking around. This-- you hear it more and more lately because there's tons of IDEs for web development, for doing JavaScript and stuff. There's all these different IDEs available now. But so it's just an Integrated Development Environment.

And so this is the iLogic IDE, right? We've got this area where we actually write the code, and then this shows me things that are in the document that I can use inside my code. And then these snippets that let me bake stuff into my code. So they're all utilities to help me build up this code. So that's the iLogic IDE.

And here I can-- here's some iLogic-- what I would say is real iLogic code. And for some reason, a lot of iLogic code looks like this. This is a little bit of a pet peeve of mine, because-- and a lot of the documentation encourages this, I guess. But it's just this nasty formatting. So much easier to read then to maintain. Some indentation and a little bit of white space isn't going to hurt anybody, but anyway.

So same thing, just with-- cleaned up a little bit. And so I'm sure everybody here is using iLogic a little bit, so this is all really familiar. So we're just changing a parameter. And what iLogic does is-- so iLogic has this IDE, and really what it is, is it lets you write Visual Basic NET programs. It kind of hides that from you. That's really what's happening-- is that code you write is a Visual Basic NET program. And to make that a little bit simpler, is iLogic has created a library, a programming library, that you use.

And so these things that I've underlined those are actually classes or objects inside that iLogic programming library that they're using inside Visual Basic NET. And so these things are iLogic specific, because it's coming out of the iLogic library. And the other stuff, the ifs and-- where I'm concatenating and doing assignment, that's just all basic VB NET stuff, right? We're taking advantage of their--

So a little bit too. So what is iLogic's big conspiracy, right? But a little bit tongue in cheek on that. But it's this kind of a trick to get people that don't program to program, and-- which I think is a good thing. So I'm not saying that's bad, I just-- that's part of what iLogic is doing. So people that-- I mean, a lot of people are just scared of programming. It's just this new big thing that's just seems a little overwhelming. And iLogic was able to-- because it is easy to use and through kind of the marketing convince people that it's really easy. So you should use it. And so a lot of people that wouldn't have programmed otherwise now are adding functionality-- customizing Inventor and making it more of what they need. So that's a good thing. Yeah, the gateway drug to VB NET. That's good.

Yeah. So here's that marketing statement again. So if you read that, does that say anything about writing a program? Nothing about programming, right? And so they play these little games. So rules instead of programs. And snippets instead of functions or objects. But-- same thing. Just the marketing world, right? You just change the name of something and then it's some big new thing, right? So we've got-- it's the cloud instead of the web. What else? Oh yeah, it's meet-ups instead of user groups. Right? Same thing, just a different name.

So have you ever seen a rule-- I mean, not this rule, but rules that look something like this? So this was from the news group, I can't remember if this was an answer I gave or just one I found that somebody else had done. So somebody was asking, again, an iLogic logic question, and this was the result from that. And here, this is pure API. So it's not calling any of those iLogic specific functions. And so I want to go in to some of that.

So iLogic in the API. So here's my analogy, is this is iLogic. So I want to take a picture. So here's a simple point and shoot, hardly any settings. It's just turn it on, point, click, and I get my picture. All right, so really simple. And for a lot of people that's all they ever need. It's all they ever want, because they don't want anything more complicated than that. But for others, they do want eventually want some more, right? I want to change my focus. I need to zoom in on some sports. I need a little quicker. And I want to do some low light stuff. And so I want to get something like this.

So this-- so on the right, I've got more the Inventor API. So it's full featured, a little harder to use because there's more, right? But it's going to do a lot more than what I can on the left. And so neither one is bad, right? It's just, what you need?

And so what is the API? So we've got Inventor. And so when you run Inventor, behind Inventor there is this engine running inside it. And that engine knows how to read and write the Inventor files. And inside that engine there are a whole bunch of things we call requests.

And so there's an Extrude request. And the Extrude request is what knows how to really make and extrude. So it knows how to modify the faces, add new faces, trim the existing faces. And so I end up with what looks like an extrude to the user. Add the information to the browser, whatever is needed for an extrude that extrude request is the thing that has all that intelligence.

So now I want to create an extrude as a user, then I go to the nice UI, I've drawn a sketch, and so I run the extrude command. Line And the extrude command guides me as a user through what input is required to create the extrude. So it asks me, pick a profile, do you want to create a new body? Or do you want to add to an existing body? Or you want to cut? And then, how do you want to-- how deep do you want to go? Do you want to go through all? Or a certain distance? All that kind of stuff. So it's guiding me through everything that an extrude needs. And when I've defined that, and I click OK, and then that command calls the extrude request inside the engine and passes that information that it's collected, and then the extrude request creates the extrude.

So API is the same kind of idea. But instead of having a nice graphic interface to guide me through, I as a programmer, I'm kind of responsible to go and collect that information. So I-- there's functions, so I could ask the user, pick a profile or pick something. But now I can also use API to go and query and find things in the model that I need and figure things out for myself. But regardless of that, I still have to collect the same kind of information that the command collected. And then once I've collected all that, then I call a function in the API that it ends up calling the extrude request, passes that information, and it creates the extrude just the same as the command. So I end up with the exact same thing, just I went through the programming interface instead of the user interface.

And so the important thing to get out of this is that these really are pretty similar in what they're doing. And so when you're writing a program, it's good to go and look at the UI and go through the same steps that you want to automate. And just go real slow and look and see all the different options that you're really specifying when you go through the command in the UI and-- because you're basically going to have to do the same thing in the programming interface too.

So this is kind of the heart of the API. And iLogic hides all this from you. So simplify things you don't really know this exists if you're just using iLogic. So that iLogic library makes a lot of assumptions, and so you don't have to mess with this. But-- so this, all these little boxes are objects in the Inventor API. And most of them have a one to one match to things inside Inventor that you're already familiar with. So there's a box there for an extrude feature, there's a box for a sketch line, there's a box for a work plane. Those kind of things. So there just a one to one match.

And you can find out more about this. Inside Inventor there's a programming help and it takes you to this. And we'll look at this a little bit more. But so here's the topic for the parameter object. It shows you all the functions associated with it. So there's a lot here.

So let's look at an example of doing something with iLogic versus doing it with the API. And remember, iLogic is build on top of the API, right? So when I do something in iLogic, it ends up calling the API to do whatever it does in Inventor. So it's going through this anyway, but it's simplified things.

And so here I want to change a parameter called base length, right? So one line, really simple, really nice. So now I want to do the same thing in the API. And now I have to know about this object model. So we saw that great big giant object model, but the thing is nobody uses all of that. So it depends on what you're doing. So if I'm doing some stuff with assemblies, then I use this little piece of it. If I'm doing stuff with parameters, I use this little piece. When doing stuff with drawings, I use this little piece. Right? And then I can ignore all the rest. So you don't have to understand all of it, just that little piece that you're using right now.

And so we're using this tiny little piece to access parameters, so I don't have to care about all those other objects on them. And so with the API, there's this application object and that represents all of Inventor. So that's the important thing to get, because if I can get to that, then I can get to any other object. And so what I need to do to use the API is I need to get to the object that has the functionality that I need to use.

So in this case, I want to change the value of parameter. So I need to get to the parameter object that represents the parameter named base length. And so this is how we're going to do that, is we need to use this object model.

So first, I get to the application object, and here's the code to do that. So I've declared a variable to be of this type, part document type, so that's one of the other objects there. And this is a function inside VBA. It's also a function in iLogic too, that returns the application object. And so just by calling this, now I have the application object.

And then the application object has a property called active document. And so I'm calling that, and that returns whatever document is currently open inside Inventor. And so now I have this object. And with iLogic, you don't have to do this because it always just assumes you want to work with the active document. So you don't ever have to specify. But that-- and most of the time you do. So that's kind of a good assumption.

But sometimes you might not want to. I could write a program that opens a document invisibly, so it never becomes active. And I could access it, change parameters, have it update, and do a Save Copy As in some other file and insert it into my assembly, right? So I can do that, because here I'm specifying which document to go get the parameter, change it, do things like that. So I have more flexibility here, but it's a little harder.

So then from part document, I won't spend really any time on why this exists, but there's this other object called part component definition. And this actually does exist in the UI, but you don't really do much with it. But the top node in the browser for a part, that's the part component definition. And that guy really owns all the stuff that we think of as being part specific. So he owns features, parameters, sketches, work planes. All that stuff is owned by this part component definition object.

And so I have to get it. So, in front of the part document, I call this property that returns a part component definition. And then-- so one-- stepping back here, one basic thing with this object model-- so we've got these objects. You notice some of them have the rounded boxes? And so there's some special objects in the API called collection objects. And those don't represent anything that you're used to in the UI. They're only for the API.

And what collection objects do is they provide a set of similar objects. So the documents collection, for example. If I get that from that collection, I can get all of the documents that are currently open. And so every collection has two-- at least two properties. It has the property called count, so that tells me how many things are in the collection. And it has another property called item, and that lets me get things out of the collection. So if I have a collection and the count is 3, then I can get item 1, item 2, item 3.

And so here on the park component definition, I call a property, called parameters, that returns a parameter collection object. So right now, I have a parameters collection and it supports this property called item. And the item, like I said, I could ask for item 1, I could ask item 2-- but sometimes item also will let you pass in a string if the things inside that collection have names. And so I want the item named this. So I want-- I'm asking for the parameter name and base length. And it passes it back, and so now I have this parameter object.

And then I can call this expression property on it, pass in 6 so that changes it. I don't see anything change in the part yet, because I have to click the lightning bolt, right? So that's this part document dot update. And now the model updates.

So-- Yeah. One line for all of this. What do you like better? So yeah. So this is really easy and really nice. But it kind of goes back again to the point and shoot or the big SLR. So I have more capabilities here, but maybe a lot of times I don't want them. I like this simple interface.

And the reason that API is so complicated is that parameters are more complicated than just setting a simple value. So if we look at the parameters dialog, there is a lot of stuff here. So there's different kinds of parameters, they have different kinds of units in the expression, and they have nominal value and model value because they can't have tolerances associated with them. I can set what-- in the model if it's using the max or the min or the nominal. I can export them as iproperties, or when it's derived, then I can define formats. And so here's the dialog set up so I can control tolerances. I Can format when I export as an iproperty. So parameters are fairly complicated really when you start digging into the details.

And so this little object model-- actually here, I simplified it. This is the real object model to parameters. And-- but it's exposing all of this stuff. So I've got different kinds of parameters. So-- parameter tables. So if I derive in a part, parameters get created. And so those are available here. Just model parameters-- so when I'm creating stuff, model parameters get automatically created. So they're in this collection. If I import an Excel file, parameters get created. So they're available through here. Reference parameters are available here, and user parameters. So there's different kinds of parameters. And so this exposes all that functionality.

And so one more thing I want to touch on right here is that when you look at the object model it can be useful. Notice up here now, this my parameter has this I and it's in bold and brackets, and then these have an I just in parentheses and italics. And what this is saying is, this guy is referred to as a base class. Any of these that have a bold number-- letter by him is a base class. And these are derived classes.

And what that means-- if we go back to high school biology and taxonomy. So if we have mammals, right? So that's our base class. And we've got dogs, kangaroos, and whales. And so they're all mammals, right? And so a dog and a whale, they share common things that all mammals have. So a mammal has a set of properties that every mammal has. But then a dog has all of those, plus it has a few things that are specific to dogs. And a whale has some things that are specific to whales.

And so that's what these are. Is this parameter object, which are code that we just looked at--

got, so I have this parameters collection and I can use the item on it to get a parameter of a certain name. And that groups all of these together into one collection. So I don't care what type it is. So that makes it easy, right?

And so this parameter object is like the mammal. So I'm just-- I'm going to use stuff even though it's a model parameter or a user parameter, I don't care. I'm just treating it as a mammal. And-- but then, if I do care, I can go down and get the specific type. I can get a reference parameter and then do things that are only specifically available for a reference parameter for a user parameter.

And then these are all the functions that are supported by a parameter object, to support all of that capability. So iLogic is designed to solve a very niche thing really. The configuration of parts and assemblies, where the API is meant to do anything you want, everything you can do in Inventor. So if I want to write a program, it's going to create parameters and add tolerances to it and all that stuff, then I want that full capability. But if I'm just doing-- configuring parts, then iLogic's a great solution. So just back to this. What do I need.

So couple more slides, then we'll do some demos. But-- so there are some differences-- just some basic differences with how a couple of things work in iLogic versus the API. And one of those is how units work. So in iLogic-- well first, just in Inventor, the user defines what units he wants to work in. You open up a document, you go to the documents settings, and you can pick what units you want to work in and you can change that at any time. And because the user can change it, it's not consistent.

So if I have this iLogic rule and a guy that has the document is set as inch, what's this going to do? Set it to six inches, right? And what if I give that rule to another guy who likes to work in centimeters? Then what's it going to set it to? So six centimeters, right? So this same rule isn't going to work consistently for different people.

So generally, I think iLogic is used within a company and they probably standardize, and so this isn't an issue. But if I'm writing a program to put on the app store or just share with a lot of people, I don't know how they're going-- what their environment's like. I want it to work the same all the time, and not hear reports that this doesn't work for this guy, it's working for this guy, and it's just because he had one unit type set versus this other guy.

So Inventor has internal units that are consistent all the time. An inside Inventor, distances are always, always centimeters and angles are always radians. And so the API uses these internal units. So if I'm using the API and I go and find a sketch line and ask the line how long it is, it's always going to tell me in centimeters. Doesn't matter what the document units are. So I'm always working in these internal units when I'm using the API. And-- but I'm going to show you how this ends up being actually easy to use. And there's this-- inside the API, there's this object, units of measure, and I'll give you an example.

So when I worked with the users and I ask-- I need to get some information from him. So a length.

So when the user types in that length, it's a string. And any time he's entering information into a dialog, that's a string. And I could end up converting it, so like this. 7.32, so that's a 7, a period, a 3, and a 2. A character 7, you know, a period, a character 3, a character 2. And then we can convert that into a floating point of 7.32.

But the user types it in as a string. But he could type this in. That's a valid value too. Here I'm even specifying what units and mixing units. But that's a valid input. You could do something like this, if the parameter length exists. Or you can do this, if height and length exists. So those are all potentially valid.

And I'd like to not worry about what he types in, because figuring out some of those could be kind of a mess, if I had to deal with it. And then if I need to display some information to the user, that's always displayed as a string. So I want to display that correctly to him.

And that's how I can use this UnitsOfMeasure to make it easy. So when I'm getting information from the user-- the way this works in the API is, it really makes it so I don't have to care about units. So I think the first reaction to some people is, when I say API, is always centimeters. Or you're, like, oh, no, I always work in inches. It's, I'm not used to thinking in centimeters.

But what it really means is, I don't care about units. And I'll show you why. So I want to get some input from a user. So I'm going to ask him to enter a length that I'm going to use for something.

And so here I've gotten the part document, the active document, and I get the UnitsOfMeasure object from it. So every document has its own UnitsOfMeasure object, because this document could be inches, this document could be centimeters, this one millimeters. It's a per document setting that I set the units for.

And so here I am asking the user, enter a length, just using this VB function called the InputBox. Box. So it brings up a little box. He can type in whatever he wants. And so let's say he types in any one of these. And I don't care, he can type in whatever he wants.

And then all I do is, on that UnitsOfMeasure of measure object, call this function, GetValueFromExpression. So I take whatever he typed in. And also I pass in whatever the current document units are that the unit of measure knows about.

And then it passes back the floating point, the double of what this got turned into, in centimeters. So I get whatever value he typed in using the current document units, and I get back centimeters.

And so now I go and do all my Calculate, whatever I'm going to do inside Inventor using centimeters, because that's how Inventor wants to talk. And then now I get some result that I want to display to the user.

So now, again, I get the UnitsOfMeasure object. And here I've gotten something, so I just-- for example here, I've gotten the length of a flat pattern. But, whatever, I've got some length that I either calculated or got from Inventor. And I want to display that to the user. So the units of measure, now I use this function, GetStringFromValue. I pass in that length, and I tell it that I want it in the current document length units.

And it gives me back a string. And it's using whatever the user set up for how he wants units to display. And so then I get back-- so if the user is in inches, he'll get back this, 2.496 inches, because he wanted three decimal places accuracy. And it shows inches because that's what he's working in.

If the user happens to have millimeters, this is what he'll get back. I don't care. I didn't even look to see what units he's using. I just called this function, passed it a value, got back a string, and displayed it to him. Or the other way, I just got a string, passed this function, and got back a value, and then I go. So it just takes units out of the equation.

So this is, to me, the big Achilles heel of iLogic, is that IDE that we looked at. So it's just really horrible for writing code. And especially if you're writing API code, or even just standard VB.NET code, it doesn't help you at all. If you haven't seen some other editors, I can demo in a second. And you'll see what you're missing if you're trying to write a lot of code.

Some people have got to have just the patience of Job with the programs that I've seen on the forum. They'll post this big, long iLogic program that, really, is just all API programming. But they've been coding it inside this editor that gives them just no help at all in writing a big program like that. And I just think, oh, man, that's tough. So there's some other options for this. We'll talk about VBA and Visual Studio.

And so now for a little fun to break it up. Everybody's still awake. So something to go home and try. So have you ever taken a spoon, kind of a big spoon. Put it in your mouth, and turn it around 180 degrees. OK, so kind of hard, but doable.

But now try this. So without your hands, you want to turn the spoon around just using your tongue. And so hands in your pockets. Behind, out the side. Ta-da!

[APPLAUSE]

It's a little easier, instead of the spoon, if you use a fork.

[LAUGHTER]

So the object lesson from this is there's different tools. So use the right tool for the job.

So we've got the VBA. The VBA is built into Inventor. So it's not an add-in. it's really embedded into the core of Inventor. So VBA is from Microsoft. It was designed initially for Word and Excel, develop a program Word and Excel.

And then they started licensing it to other people to use in applications too. And it was designed for APIs like Inventor has. So Inventor's API is exposed in the same way as Excel and Word APIs are exposed. Because it's designed for APIs like that, it works really well for programming Inventor.

So when somebody asks me a question, how do I do this in the API? Then I almost always pop up VBA and start coding there. Because it's the fastest to work with and to just get a quick result. Even if I'm writing an add-in, and I need VB.NET, I'll oftentimes just see if I can do it at all, first in VBA, and then port the code over.

And because it was built for an API like Inventor, it's really the best for debugging the API. Dot net's not bad, but VBA's a little better. I'll show you some things in a second.

It has something called an Object Browser. iLogic doesn't have an Object Browser, because it hides that whole object model thing. But this has an Object Browser, lets you look at all those objects. 1 net has it too, but it's nicer in VBA. And I'll demonstrate this. What I mean by this is we can kind of look at a live model of that whole object model inside VBA.

This is the big problem with VBA, is it's based on an older version of the Visual Basic language. There was VB 6, and it's really based on the VB 6 engine. And VB.NET is the next generation of Visual Basic. VB.NET has some really nice features in the language that we just don't get with VBA. And they're, I don't know what percentage, probably 98%, the same, but just that little bit of difference. And so they're not totally compatible with each other.

And most of the samples that are in the Help, and a lot of stuff you'll just find on the web if you're searching for API information, are going to be written in VBA. So it's good to have some understanding of VBA. And I just threw this in. But so if I use VBA kind of like I do iLogic, if I'm just writing little utilities, I can write-- in VBA it's called a macro, rather than a rule-- but I write this little macro, and it is fairly easy to add buttons into the ribbon so I can easily execute them. So there's some functionality built in.

AUDIENCE: So I think it was the 2015 AutoCAD VBA stopped shipping, and you have to install separately with AutoCAD. So any concerns moving forward with-- we've got a lot of existing VBA customizations. And, of course, I'm just asking to [INAUDIBLE]. I'm wondering-- I [INAUDIBLE] guess, but I'm wondering time line? Is this something that's eventually going to go away because of those limitations?

BRIAN EKINS: So good question. So the question is, so with AutoCAD, AutoCAD had VBA, and they quit delivering it in 2015, so a little while ago, maybe even before then, I think. But anyway, they quit delivering it with AutoCAD. You can still install it separately and use it with AutoCAD, but it's not delivered with it anymore.

And so is something like that going to happen with Inventor? And there was some interesting action going on with Microsoft a while ago. And so when VB.NET came out, Microsoft came out with something called VSTA. So it was like VB.NET to use inside a product like you use VBA now. And it was a failure.

And they were trying to kill VBA, but VBA didn't want to die. And VBA at that time was only available as a 32-bit application. And so Inventor did some crazy workarounds to still allow VBA to work with 64-bit. And I think Microsoft can't kill VBA because there's tons of Word and

Excel macros out there. And so now they have a 64-bit version of VBA.

And Autodesk used to pay a lot to license VBA. And now it's free. So VBA is going to be around for a while. It's not going to improve. It's just in maintenance mode. But it's so heavily used, it's not going to disappear.

AUDIENCE: That's why our applications [INAUDIBLE]

BRIAN EKINS: So let's look at VBA here for a second.

AUDIENCE: So we don't have a lot of [INAUDIBLE]

BRIAN EKINS: So he asked, if I don't have a lot of existing VBA programs, do I care about VBA? I guess, right? And I would say you probably not necessarily want to write a lot of programs that you're going to use daily. But I'm going to show you something. I think you still want to be familiar with it for some reason that it has here.

So here's Inventor. And I can access VBA through Tools, VBA Editor, or alt f11. Pop it up. So here's VBA. And so it's an IDE. It's a development environment. I've got this area with my code. I'll get rid of that for a second.

Over here shows me I can have different groups of code. I won't go into details really describing this, but just some of the things in the Editor itself. So I'm going to write it really simple. And this is in the notes too, something like this.

And so first off, it's one thing there is VBA Help Me code. I did Subtasks, and it just added the unsub. And I don't get any of that with iLogic. So it's helping me to code. And this is a VB thing to debug.print. So this application is the application object. When I do dot, now it shows me all of the properties that the application object has.

So again, it's helping me to code, and they call this Intellisense. So I can pick something here. I'm just going to pick, there's a property called Caption. And then this is a VBA thing called Stop. But now if I run this, it stops execution right there. So this is in the middle of running right now. We've halted it, but it's still running. It's kind of paused.

And now if I click on this application, if I right click on it, I can say Add Watch. And I put this in the notes. But this is, I think, one of the most important things you might get out of this. And unfortunately, I can't make this bigger, I could make my code bigger, but not this. So I know you won't be able to see this very well.

But down here, it's showing this application. And then over here, it's showing the type of this object, and it's saying its application. But what's cool is I can expand this, and now that shows me all of the properties of the application object and their current values. So this is like a live view of that big object model.

AUDIENCE: I have a question. What about hidden properties? Are they hidden for a reason-- kind of being replaced? [INAUDIBLE]

BRIAN EKINS: Yeah, so in the API, there are some functions that are hidden. But there is a way that you can-- like in the Object Browser, it shows you all the objects and functions that they have. And you can just right click and say Show Hidden. So they're not very hidden.

But the hidden ones are hidden because a couple of cases. So one is-- well, maybe three. One, we added something, and it's not quite finished potentially. And so we don't want to officially support it. That's really the big answer, is hidden ones aren't supported. So it's maybe-- we're still working on it.

Two is it's an old thing that got replaced by something new. So you really shouldn't use it. You should use the new thing instead. But it's still there to support backward compatibility for people that did use it. So we don't want to break those programs.

And a third one is sometimes we wrote stuff that we didn't really think had general usage. But one of our add-ins, like tube and pipe or somebody, needed this kind of advanced functionality that was pretty complex. So we added it to the API so they could use it, but we didn't really want to support it in general, because it just probably wasn't very useful to most people. And it would have been pretty hard to support. So we hid it.

So here I have this application. And I can see here, one of the properties is Active Document. And it shows me over here the active document is a part document. So I expanded that. And here now, I see all the properties that part documents have. And I won't spend a lot of time here, just because I know you can't read it.

But this is really powerful. Because I can see things, how to get stuff. So this is one way I use it a lot. So let me stop this.

So somebody will ask a question of how do I do this on a drawing dimension or something.

And I don't remember. And so the easy way-- and, in this case, I'll say it's a work plane. So they want something on a work plane. So I'll just select whatever object I'm interested in. Select it, come over and run this code, expand this application.

I go to Active Document. Active Document has a property called Select Set. So that's whatever is currently selected inside Inventor. And right now, there's one item selected. It tells me it's a work plane. Now I expand that. And now I see all the properties that a work plane has.

So it's just a quick view, again, of that object model. But I see a live version of it. And if one of these properties returns an object, there's a little plus by it. And I just expand that, now I see what that object supports. And so I am just navigating down through that object model structure, but a live view of it.

So I use this all the time when somebody asks a question. That, and then just it's a quick way to type in code, because I get all that help. So I get the active document. I want to get the part component definition.

You can see how it's helping me. So it's as I go along. And it helps format things too. So VB.NET even more so.

So another big thing here. So let's go back to Inventor. And Help, programming help. So here's that programming help. This is broken up into a few topics.

So there's this user's manual as, kind of, overview topics of how things work. And then there's this reference manual, and that's where the guts of-- here's all of the objects that are in that object model, and then detailed description of each one of those.

And then there's a bunch of samples. So from these topics, they link to samples if they're demonstrated. But you can also just go to Samples themselves. And these are almost all VBA. There's a handful of a couple of C# ones, I think. But mostly they're all VBA. So here's a sample. And I'll just grab this code. Copy. Paste.

So now I just want to understand how the API works. Maybe you found a big block of code in the forum or one of these samples. I want to better understand really what's going on. So here's a great thing to do. So I just pasted it in.

Right now I want to run it. But I don't want to just run it. I want to see what's happening as it runs. And so I right clicked before, and I turned on this debug window. But I could also access the same commands here or use function keys. But I'm going to step into this function.

So this is the debugging. So I'm running one line at a time and seeing what's happening. So on this line, it's using the documents collection. And it's going to add a new document. And it's saying I want to create a part document. So let's-- there we go. So it just did it.

And now it's getting the component definition guy and another object. So now it's adding a new sketch. And I can see it appear in the browser. It's adding a circle to that sketch. It created a profile. Now it's creating a 3-D sketch. It's adding a bunch of work points to that, and then some lines using those work points.

Then it's setting up some things to build a loft. So it's basically, like I said before, it's gathering the input that's needed to create a loft. Just the same as if I ran the Loft command, but it's through the API. Now it's calling this Add method on the loft collection. So that's going to call the Loft request inside the engine to create a loft. And there we go.

So that little program just created all that geometry and built the loft. But to be able to step through and dissect what a program's doing, that makes it easier to say, oh, well, this is what I need, so I'll copy that chunk out. Then from this other program, this is what I need. So I can copy that out and use it. So that's VBA. Any questions with that?

So the other thing is Visual Studio. So Visual Studio is using VB.NET, so the new version of Visual Basic. And there are three versions of Visual Studio available right now. So there's Visual Studio Professional, but it's not cheap, so $500 to pay for it.

There's Visual Studio Community. This is new. This only came out about a year ago. And it's free, but with everything free, there's always some catch, right? So here it's the licensing is limiting. So it's free.

Basically who it's not free for is if you're writing code for your company, and your company makes more than $1 million mi year. Then it's not free for you. But everybody else-- well, there's even some exceptions to that, is if you're using it for training, then you can still use it.

Or if you're writing open-source code, then you can still use it. But if you're just an individual using it, or say in a training environment, then it's free to use, or a startup, small startup, that's not making more than $1 million.

And then there's also this version, which is free. It doesn't have those limitations as far as the licensing, but it has some functionality that's been taken out. But for the most part, you won't care. If you're writing add-ins for Inventor, then it's a little bit more painful with Express . But otherwise it doesn't matter.

AUDIENCE: [INAUDIBLE]

BRIAN EKINS: Yeah, so their comment was that Community replaced Express. And that's probably what's going to happen. But they haven't officially said that yet.

So if you go, so just what I was going to say, if you go to the Microsoft website, they just really talk about Community. And you have to dig a little bit to find Express, but it's still there. And so I would recommend going and getting a license of, grabbing Express while it's still available, before it probably will go away in their push community. But right now, it is still there. And they have the latest version available as Express.

So VB.NET, so that is the same language that iLogic uses. So he said iLogic is really running VB.NET underneath. But with Visual Studio, I can write add-ins. So I could write iLogic, or I could write tube and pipe, or something simpler.

But I can write these more advanced applications that have commands in the ribbon and all that stuff. I can also use something called Apprentice. So I can't do this with VBA or iLogic But Apprentice, just real quickly, is this separate little tool that you can use in your program that lets you do some pretty minimal read and a little bit of write to Inventor files without having Inventor.

Properties is one thing. That's something you can read and write. So if you have thousands of files, and you need to update some iProperties in it, you could write a little Apprentice program that would do that really fast, because it's not opening them inside Inventor. But it can directly read and write the files. And I can access model information and some other things.

So from here, I can use Apprentice. With Visual Studio, I can write EXEs, E-X-Es. So if I want to write some kind of batch processing program, I want to probably start up an EXE, and then have it start Inventor to go do whatever processing I need. And it might even restart it periodically, or whatever, to clean up memory. But I want to write an executable that talks to Inventor from outside. And I can do that with Visual Studio.

And with Visual Studio-- so if I create some fancy add-in, I can create an installer that I just give to somebody. They run that installer. And now all of a sudden that functionality is magically available to them. So it's a nice way to share code.

So now I want to just talk real quick about how to use Visual Studio. So I found some sample that's really close to what I need, but it was written in VBA. And I want to use it in an iLogic rule. And so now I need to clean it up. And if I just do that in iLogic Editor, I run it, I get this list of errors that just says something happened on this line. It's just hard to do.

But I can use Visual Studio to make that a lot easier. So let's take the same program we just looked at. This is the Express version. And I'm going to create a new console. And I go through these steps in the notes.

So here's the program. And I'm just going to, right below it, paste in that VBA macro. And in dot net, here it shows me, without me running it, it's showing me all the errors I currently have. So I've got 34 errors and 11 warnings in this program we just brought in.

And Visual Studio doesn't know anything about Inventor or iLogic, so I have to tell it about the Inventor library so it understands. So here I am referencing this object called part document. Visual Studio doesn't know what that is, but I can tell it.

So I just go to Project, Add a Reference. And I talk about this in the notes too. So the first time, I would have to browse. And I go into where Inventor's installed and point to the library. Since I already did it, it remembers that. So now I can just click it here.

So now Visual Studio is aware of the Inventor library. So nothing changed here. So I could say Inventor, so I tell it which library it's in. And now it knows. That error went away for that line. But to make it a little easier, instead of having to type Inventor dot everywhere, up at the top, I can say it imports Inventor. And now I don't have to do that. It'll just look in that library for any of those names.

So now we're down to 22 errors and 11 warnings. And in VBA, the language itself had some quirks, and they cleaned up a lot of those with VB.NET. So the language in VB.NET itself is a lot nicer than VBA.

But one of the things in VBA, whenever you assign something to an object, you have to use a Set keyword. And in dot net, you don't have to use it. In fact, you can't use it. It's not supported in dot net. So I just have to delete all those. Should probably do a replace, right? Well, maybe I should have. There's quite a few here. One more. And you notice, as I cleaned that up, it was in [INAUDIBLE] form. It keeps my coat nice and clean.

So what have we got? So we have eight errors now. And the first one is it doesn't know about this application. So this application is something specific in VBA and iLogic. So Visual Studio doesn't know what this application represents.

So what I'm going to do is up here, I'm going to declare a variable as inventor dot application. And there's a VB function called GetObject. And I'll do inventor application right there. And that will get the application object for the Inventor that's running. So that gets the application object, assigns it to that.

And then I'm going to call this function and pass in the inventor application object. And I'll set up an argument. So that fixed the application. So this application now is just this argument, which happens to be an application object. So there's another error is right here.

So in VBA, there's something called an enumerator that's defined in the object library. And it's just a general programming thing. But so here, when I create a new document, I need to tell it what type of document I'm going to create. And I do that by specifying a value out of this enumerator.

So an enumerator is just a list of valid values for something. So it's a list of these names, and the list itself has a name. So when I use it in VBA, I only have to specify the name of the thing in the list. But in VB.NET, I have to specify also the name of the list. But Visual Studio makes it easy for me to fix this.

So I want to use k part document. So I'll just delete that. And now this Intellisense knows that it's supposed to be something out of that list, and it shows me all the options. And so I can just pick it from there.

So the Intellisense really helps me to know what the options are. So that one's the same. And down here, we've got one, k surface operation. There it is. So that was out of this list, this part feature operation. One more. And so here-- and actually, I got ahead of myself here a little bit. Let me go back here and run through some slides.

So here are some differences. So we already talked about Set statement isn't used. So I just need to delete those. Parentheses, so this code didn't have it. But another thing you run into is, in VBA this is legal. So I can call a function and pass in arguments, and I don't have to have parentheses around it.

And if I want parentheses, which I kind of like it, I'm just used to seeing parentheses around argument calls. In VBA, I have to put Call in front of it if I want to use parentheses. This is one of those goofy quirks of VBA. And if I'm assigning the result of this to something, then remember I have to use Set. And then I have to use parentheses.

In VB.NET, you have to use parentheses all the time. So this just isn't legal. And you can use Call, but you don't have to. So this is what you'll typically see in VB.NET. But you could run into just lines like this, and you just have to put parentheses around them.

And then enumerators, what I just talked about. So here we're just specifying the name of the item in that list. In VB.NET, we need to specify the name of the list and then the name of the item in the list.

And arrays. So that's the next thing I was going to fix there. So there's a big change in arrays. In VBA, you would see this kind of thing quite a bit. And I'm declaring an array, and I would say what, that it goes from 1 to 10, or whatever.

And, in fact, in VBA, you could do really screwy stuff. You could say I want to create an array that goes from minus 5 to 7, so any range of values. And in dot net, you can't. They always start at 0, and basically that's it. They always start at 0. And there's one screwy thing with VB that they still kept I wish they would have changed, is this isn't an array that can hold one thing. This is an array where the top index is 1. So it really holds two things, 0 and 1.

When I had VBA code, and then poured it to dot net, this is the one that's the hardest, typically, to deal with. Because if they've done this, and you'll see it in this code that I'll show you in a second. That one problem, we've still got to fix. So this is complaining about you can't do that anymore.

So I want an array that's-- so this is creating 1 to 6. And so I say 5, right? So 0-5 is 6. But now I've got to go through and correct all of these indices, because they're assuming 1 is the start. So I just need to reduce all of these by one. So this is the place where you can mess up.

AUDIENCE: Instead of going and correcting all of those, can you just add a zero reference at the beginning of it [INAUDIBLE]

BRIAN EKINS: I don't know, we have to look at it.

AUDIENCE: [INAUDIBLE]

BRIAN EKINS: I don't know if there is. I'll have to think about it a little bit. But I don't know if there's an easy workaround for this.

AUDIENCE: Here, you've got six of them and it's not that big of a deal. But if you an array with 100 it would be more of an issue.

BRIAN EKINS: Yeah. Well, and usually if you add a big array, it's in some kind of loop so that you're just changing a few things in that loop. A lot of times here, you might make some changes, then you run it. And it's not running right. And it's because you goofed up. You missed an index somewhere. So you tend to find these problems at runtime.

But the thing I wanted to show here that we've seen, is the Visual Studio just has found all the errors, and pointed them out to me, and helps me correct them. So it's pretty easy to go from VBA code to dot net. So let's finish these real quick. I think that was all of them. So let's run this. So I can actually just run it from here.

So this is creating an executable, an EXE, and it's connecting to Inventor. And start. And there we go. And so now let's take this code and copy that. And let's go in here. Add a rule. And I can get rid of this.

In iLogic, if I just have some simple calls, what iLogic is really doing behind the scenes is it's creating a main function like this. And it's putting those function calls inside that. But now I need to add that sub main, because I have a function outside. And inside that sub main, I'm going to call this. So let's run this rule. And there we go again.

So there's tons of that VBA code in there that you could use. I think fairly easily It can be converted. And this one, the one I did, is probably, I would say, a little worse than average on converting them. A lot of them, it'll just be deleting the Set statement, and then it'll just work.

And so this is just showing that, how you need that sub main thing if you're going to call the other ones. And so this is one example. This is what kind of got me thinking about this class, of doing this class, was this question that came into the forum. In fact, here it is.

So he was asking a question. And this is where I just have to, I don't know, I guess applaud him for having the patience to write all of this stuff inside the iLogic editor. And so if I copy that.

Say you say I've written this program, and it's not working. And iLogic just isn't giving him a lot of help. And so I need to paste it in here. And yes, 73 errors, right? So here, right off, I just pasted it in. Now all of a sudden, I know what the problems are. And now it's--

AUDIENCE: [INAUDIBLE]

BRIAN EKINS: Yeah, so in this case, yeah. So I'm using VB.NET to help me edit iLogic code. So this is kind of like, I guess, from my standpoint, telling the children don't play in the road, but then teaching them how to be safe when they play in the road.

So I think in a lot of cases, this is probably better, not as an iLogic rule, and it'll do it in VBA. Or just maybe make an add-in if he uses it a lot. But a lot of people still are going to use iLogic, so here is a nice tool. If you're using the API, it's a lot better to write that API code in Visual Studio than to try and do it in iLogic Editor.

AUDIENCE: [INAUDIBLE]

BRIAN EKINS: Right. Well, first let's--

AUDIENCE: [INAUDIBLE]

BRIAN EKINS: Yeah, so I think I'm-- I usually go way over on these things, and I don't get to cover everything. So I was shooting to have material that would last an hour, or an hour and 15 minutes. And I think I'm about done. So we can spend a little bit of time. So let me just check.

So yeah. So here's just the end summary. So iLogic is really the simplified IDE to do, really, iLogic rules is what it's designed for, well, to call those iLogic functions, to stay in that iLogic world. And it's intended, really, for doing configurations, but it can be kind of painful if you want to expand out of that. And there are other options. So if you end up using the API, then there are other options.

And so let me just hop back into Inventor. And so maybe the first question is an add-in, what is an add-in? So an add-in-- so in Visual Studio, I can write an EXE, like we just wrote, that calls and talks to Inventor. But I can also write a DLL. And so that actually gets loaded inside Inventor and runs inside.

And for the most part, you probably don't care. But the thing that's important there is that if I have things running in two different processes, it's going to be slower. Because to pass data back and forth between two processes takes quite a bit of work.

But if it's running in the same process, then it's just a simple function call, and it doesn't-- there's not-- they call it marshaling, the ability to encapsulate it to pass across. And then it gets unbundled, and then used. And then you don't have to go through all of that. And so an add-in is a DLL.

And so I have this DLL that I've written, and it supports certain things that Inventor expects an add-in to support. And so when Inventor comes up, it goes and looks in certain directories for any of these add-ins. And when it finds an add-in-- you have the DLL, and then you have another file called a manifest with it that describes information about that add-in.

And a lot of that information you see in the Add-in Manager, it's name, description, that kind of stuff. But it also says if it should automatically load at startup, and typically they do. So when Inventor's starting up, it goes and finds these add-ins, and it says, oh, OK, here's an add-in, and it loads it. And this is before Inventor's up that you can use it. It's while it's still coming up.

So it loads the add-in. The add-in is coming up, and it calls into a function inside that add-in, and passes it the application object. And the add-in, in response to that, goes, and it creates its commands that it has and adds them to the ribbon. And then Inventor finishes coming up.

And so that add-in is running in the background, but all it's doing is waiting to get notified from when one of its commands gets run. And it could put its commands anywhere. So if they have some sketch commands, it puts them in the sketch ribbon, wherever those commands make sense.

And now when the user clicks one of those commands, Inventor fires an event to the add-in to say, hey, this command was just clicked. And then the add-in uses the API to do whatever it's supposed to do.

So here's a couple of examples of stuff I wrote a while ago. Now I can use the bottom of this. Well, let me create a new one. So just draw a four-sided polygon here, some shape and extrude that.

And so you notice anything different around here? I know it's a little hard to see, that bulge feature. So that's not an Inventor feature. So I have an add-in that added that. So when Inventor loaded, it added that new button to the ribbon. But it just feels like an Inventor command to the user. It's just there like all the other commands, nothing different. And so I run that.

Now it's asking me to select the face I want to bulge. So I can do the previews. So we have a bulge feature over here. So most of this is really just the API. The add-in provides the ability to have that command and have it just-- to the user, it just feels like part of Inventor.

Here is something. Maybe some of you guys will be interested in this, or maybe not. So let me create a sketch.

So Carl Bass, a few years ago, had seen something, an article online that this university in Europe had posted. And they had this thing, they called it zip shapes. And it was for creating banded shapes in wood. And let me just show you what.

So we played around with it a little bit. And so I wrote this add-in to make it a little bit easier. So it's asking me to pick a sketch. Well, that's good enough. But I can, well, I'll just show you. So I changed some of these, the two thickness, so I can get in this preview.

And what it is, is it generates these shapes down below. So if I could cut this out of wood, these two pieces, and then I zip them together, the way that they're-- you can see how the teeth are different as we go along. And the way that they would form together, they would end up bending into that shape.

So this was an easy way to how to get that job. We played a little bit, first, we'd just see if it would work, 3D printed those. I need to post this on the blog, because it was kind of fun. But then I actually went down to San Francisco, and we have a water jet.

So I cut some of these pieces out of wood, had some oak with the water jet cutter in them. But I'm not sure if it's really practical, but it's kind of fun to play with. And I think in a few cases it would be kind of interesting, especially if you had contrasting wood and had those kind of shapes on.

So any other questions on API, or iLogic, woodworking?

AUDIENCE: [INAUDIBLE] but I haven't gotten to the part where I can figure out how [INAUDIBLE]

BRIAN EKINS: No, I think that would be a good application for iLogic. That's really falling into that configurator kind of category. So it's a little more involved. You're writing some more code in iLogic, but it's still using those basic iLogic functions. You're just going to have more logic than you're using

VB.NET for, some more ifs and--

AUDIENCE: [INAUDIBLE]

BRIAN EKINS: OK, another part. So the API supports that, yeah. From iLogic, well, we saw that there. I can access the full API. It lets me get to the application object. So I can access the full API from inside iLogic. But, say, in general, that's probably not a good idea. If you're using a lot of the API and very little of the iLogic, then it's probably better to do it somewhere else.

But I think in your case, it's probably the opposite. You're doing mostly iLogic stuff and need a little bit of the API. So maybe that's a good place to stay, is in iLogic.

AUDIENCE: [INAUDIBLE] Would you do anything like that in iLogic?

BRIAN EKINS: So you ask the buttons for iLogic rules. Not with plain iLogic. Somebody told me, I haven't seen this. But just yesterday, somebody told me there was an app, I don't know if it's on the app store, I think so, that will create buttons for iLogic rules.

AUDIENCE: [INAUDIBLE]

BRIAN EKINS: A global form?

AUDIENCE: [INAUDIBLE]

BRIAN EKINS: Yeah, but can you have them in the ribbon, buttons and ribbon?

AUDIENCE: [INAUDIBLE]

BRIAN EKINS: OK, so you're saying that maybe a global form would be kind of a workaround, I guess.

AUDIENCE: [INAUDIBLE] created a form, and then button 1 had a form that it would--

BRIAN EKINS: It would run stuff, yeah. But you have to bring up the form first, right? All right, anything else on this?

AUDIENCE: [INAUDIBLE]

BRIAN EKINS: Yeah, I don't have any experience with that. I don't really know.

AUDIENCE: [INAUDIBLE]

BRIAN EKINS: It hasn't. No guarantees, but it's coming up a lot more about doing something with it. So one of the things that I've been lobbying for just recently, especially since I started working on this class-- as I said, iLogic has its own library with those iLogic objects-- is I think that should be rewritten so it could be used in Visual Studio.

Because that's a problem, is if that program that I was debugging in Visual Studio was trying to call any of those iLogic things, it won't work. Because Visual Studio doesn't know about iLogic, and there's not a library I can reference in to make Visual Studio aware of iLogic either. So just kind of hosed. There's two worlds that don't mix.

What I'd like to see is that we add some more simplified stuff into the API too. So then it's for everybody, not just for iLogic, but everybody could use some simplified things. So we'll see. And I'll be talking to the guys. It's probably pretty safe to say you'll see some enhancements. What those fully end up being, nobody knows right now.

So thanks for coming, and I appreciate it.

Pardon me?

AUDIENCE: Is this OK?

BRIAN EKINS: Yeah.