BRIAN EKINS: Hi, I'm Brian Ekins. I work for Autodesk, and I don't really ever know what to put for my title. Whatever my official title is doesn't really ever describe what I do.

So I work for the ADMT, Modern Desk Developer Network, but I've been on loan to the Fusion team for the last couple of years to work on the API. And before that I worked on the inventor API. And before that I did the Solid Edge API. And before that I was an application engineer for a long time, actually building parts and doing designs. So I feel like that's actually why I got drafted into doing Solid Edges. I was a user of the system and when they started Solid Edge development they said, we want somebody that kind of understands the end user perspective to work on the API. And that's how I got into this big mess I'm in now.

So pretty simple, what I want to get out of this class is to jump start you on playing with Fusion API. And so how many here use Fusion? OK. How many have tried writing a program?

AUDIENCE: In general?

BRIAN EKINS: Well with Fusion. How many have played with the API at all? OK. So let's take a look. So this is more details. We'll look at how the API is exposed in Fusion and how do you create a program. And how to use the documentation because, of course, we can't cover everything in this hour and a half so where do you go from here. And there's a piece that's maybe a little more complicated about creating commands and so I thought I'd go over those. And especially kind of a little trick to make it easier to implement.

So first a little bit on how the API works. So you can think of Fusion-- I mean when you start up Fusion you see this nice user interface, but really underneath that there's this engine. And what that engine does is it knows how to create the Fusion data and read and write that data and create it. And inside that engine there's something we refer to as requests. And that's really where the real logic is that knows how to create that data.

So through the user interface we have this nice thing to guide us through doing stuff. So if I run the extrude command it pops up this dialog and it's collecting, guiding me through everything that's needed to create an extruded. And so as this command runs it gets the input from the user, and once the user is provided the needed input, then it calls that request. So it takes all that information and passes it to the request and then the request does the work and creates what you see as an extrude in the model.

So the API is really the same thing. It's very similar, except for a user interface, and I have a programming interface. And instead of having this command dialog, I write some code to get those different inputs and then through the API I make a function call which also calls that same extrude request, passes that same information to it, and the same results created. So it's just two different interfaces to the same engine. And so one is pretty easy. The other one takes a little bit to get used to but I think the key thing here is to look at the similarities. So if you understand the UI the API will be kind of familiar it's just getting over that how do I code hurdle.

That looks nasty. Looks good up here. So a big part of being able to write programs initially, is a lot of the programs you write you're just going to be duplicating the same things that you could do interactively, but writing a program to do it. For example, you want to create a gear. And so you could do that interactively. You could go look up the formula for an involute. And so you calculate some points on the involute and then you fit a curve through those points and you mirror it the other side to get both sides of the gear. And then you pattern that around and now you've created a gear.

But now you need another size. So you start all over and do that again. So if you could write a program-- so the program's going to do the same thing it's just going to do it really quick, so now I can do it over and over. So to create that gear I'm using just standard inventor functionality, but now I need to access it through the API.

And so what this chart is trying to show, which just kind of washes out there, but here's all the commands inside the modeling environment. And the ones in this darker green, if you're familiar with Fusion you kind of know what's there, but the ones in the darker green have API coverage. So I can do those same kind of things through the API. The ones in the lighter green, which is supposed to be yellow, are ones we have some partial coverage. And the red ones right now you just can't do that through the API. And we're catching up on all of this.

And here's in the patch environment, so there's pretty complete access through the API in those.

Some other things that you can do through the API is all the preferences. You You have access to that so you can manipulate those. There's a data panel and you can import and export files just the same as you can through the UI. You can access the timeline. But today there's not support for these other work spaces. There's an initial API for CAM coming up, but

the other ones there's not currently support, but it is all planned for sometime in the future.

So there are some other things that are very useful in an API that you don't necessarily need in the user interface. And so the API has some additional functionality. Because you kind of think of it in the user interface, if I draw a box and then need to fill it a corner, it's really easy. I draw the box and now I'm ready to fill it. So I just say, OK that edge. It's easy to pick that edge. But now I'm writing a program how do I tell at that edge. So it's kind of like you're driving blind with the API. And so this is a piece-- this B-Rep and geometry query is the ability from the API is to look at that model and find out, oh, it has these 20 faces, this is a planar face, and those faces are connected by this edge. So to be able to just find out really what geometry makes up a model.

For some applications they want to get a simpler version of that model just as a triangular mesh and so you can do that.

I can access the assembly structure through the API just the same thing like you see in the browser.

Through the API you can create custom commands. So I can create commands that interact with the user to gather input and then do what I want with it.

And I can insert my commands into the UI so I can manipulate and if I want I can even customize the UI. I could write a program that would go and delete out a whole bunch of the Fusion commands and simplify it of if I wanted to. And I can access the camera so I could do little animations or whatever like that if I want.

So this is the key to how to access all that functionality. And so this is kind of the UI for the API, is this object model. I printed out a few extras of these so if you want one after feel free to come up. So it's a pretty big, at first maybe a little overwhelming, eye chart here. And so every little box on there represents an object in the API. And most of the objects correlate to something inside Fusion. So there's a box on here for an extrude feature, there's a box for a sketch line, there's a box for a button that you would see in the user interface, that kind of thing. So we have all these little objects in there.

And the structure of this chart indicates ownership and we'll look in more detail on the next slide what I mean with that. So if you're used to programming in some other languages and you're doing it, well C++, a lot of times you'll see class diagrams. This isn't the class diagram.

So it's showing ownership rather than how classes are derived from each other.

And we'll go into this in a little bit. So the structure is you traverse that structure to get to an object that you need. And the same thing-- so Fusion, you can use it, we'll talk more in a minute too, you can use it with three different programming languages, but the API is the same regardless of which language you use.

So a little more detail on this object model. So every box on there is an object. And every object supports methods, properties, and a few of them support events. And there are some special objects on here you can kind of see. There's some of them that have rounded corners. And those are referred to as collection objects. And so those don't have an equivalent in the user interface, they're an API only thing. And they're used to really provide some structure to this. And so what they do is provide access to a similar collection, similar set of objects, and they also provide the ability to create new objects of that type.

So for an example, right up here we have this documents collection and so if I go to that collection-- so every collection without exception supports at least two properties, item and count. So if I go to a documents collection and get the count property that tells me how many documents are currently open in Fusion. So that's how many things are in that collection. And if I get the item property I can pass in a number so I can say give me item zero, item one, item two, and so that lets me get the things out of that collection. And some collections support other ways to access data inside it too. So for example, a parameters collection has item, but it also has item by name. So I can say give me the parameter named length. So these collections are important.

Well, let me go back. So let's just talk about the structure of this. So this top level object, the application object, so that represents all of Fusion. And so that's really our door into this whole world. So somehow, and we'll see in a minute, we get a hold of that application object and now once we're there we can get to anything else in the whole model. We just have to know how to diverse through the other objects to get there. So from the application I can get the documents or what we'll see is really common, is I can take a shortcut.

The application has a property called active document or even active design, so what's the user currently working on. And so I might jump down straight to here or jump down to the document. So from an object I call properties on it that return other objects. So that's how you traverse. So from the application I call some property so I could call this that gives me the

documents object and I call something on the documents object to get a document and I call something on the document to get the design. I call something on the dot design to get the component. So I'm having to make calls to traverse down to what I care about and eventually I could get to a sketch where I could get to sketch lines collection where I could then call a method to create a new sketch line. So it sounds a little confusing and we'll see some codes and hopefully it won't look as confusing as maybe it sounds.

There's another kind of object, it's actually not shown here, but I'll show an example of it called an input object and that's kind of the API equivalent of the user dialogs. So when I create an extrude object or an extrusion for example, I first create an extrude input. And so nothing's happened in the UI, nothing's changed inside Fusion, I've just created this programming thing and it's really the equivalent of that dialog popping up. So creating the input is the same as just clicking the extrude command. So nothing's happened in Fusion yet, it's just going to collect the information it needs.

And so on that extrude input object it has properties so now I start setting them. I specify what's a profile, what's the extent, I want to go three inches in this direction and I define whatever I need for that and then I pass that extrude to another function and say OK now create the extrusion. So it's like clicking the OK. OK, now do the work. So internally now it's taking that information, calling the request, and now we have a real extrusion. Just the same as if I did it through the UI.

So I said before that you can use three different languages to program Fusion. The first is JavaScript and in Fusion-- so you think of JavaScript, and its true, that Java scripts this language to program HTML. It's the logic that sits behind an HTML web page. But it's not used that way in Fusion. It's used just as a general purpose language. And in fact you can't use HTML for a GUI.

And built into JavaScript are some limitations because you don't want to open a web page and have it reading all the files on your disk and shipping them off somewhere, so there's just some built in limitations with JavaScript. But when you're writing a program for a CAD system, a lot of times you need to build to read a file. You want to read this CSV file off the disk to draw a curve through the points to find or something. And so the API we've added some functionality that's in the Fusion API to let you read and write files and do some things that are missing in JavaScript. So we've kind of worked around some of the limitations there.

You can use any text editor to write JavaScript, it is just a text file. But with Fusion there is an editor called Brackets that provides some better help. And then debugging is done in Chrome or Safari depending on if your in Windows or Mac. So you write in one thing and then you debug in another environment.

And JavaScript runs out of process in a separate invisible browser that you can't see. And this has some effect on performance which I'll talk about in just a second, in a bad way. So it's going to be slower because of this. And you can use external JavaScript libraries. There's a lot of stuff out there and so you're not limited in that way. You can use those. And then this uses asserts for errors.

And another limitation with JavaScript is as the API-- there's not a whole lot of advance right now exposed through the API, but as we add more and more events they for the time being anyway won't be supported in JavaScript. So that's a limitation if you would choose JavaScript.

AUDIENCE: [INAUDIBLE]

BRIAN EKINS: Pardon me.

AUDIENCE: [INAUDIBLE] events or?

BRIAN EKINS: So an event is when something happens inside Fusion and it notifies your program, hey, this just happened. So an example, so a mouse event could be-- so there are some events already inside Fusion and right now almost all of them are supported in JavaScript. But there's with some of the events there's some potential problems just because of the way our JavaScript implementation works today. And so to avoid those problems we're just not going to expose it the events through JavaScript.

AUDIENCE: Well call a [INAUDIBLE] problem in JavaScript. So because it is running our process [INAUDIBLE]. I think the event trigger happens in Fusion [INAUDIBLE] JavaScript and that JavaScript in response calls back [INAUDIBLE] limited our exposure of events in JavaScript. [INAUDIBLE]

BRIAN EKINS: So Python runs in process. There's a Python interpreter built into Fusion and mainly because of that in process, out of process stuff Python is going to be faster. And just to give a little idea, so I wrote a program that just made a call and it made as simple a call as I could find. Something that fusion didn't really have to think about. It just would have the value already and just pass it back. So really all this was testing is what's the overhead to make a call.

So Python was 275 times faster than JavaScript. But then when I start actually doing something in Fusion, then that time's going to go down. So on this next one I created a program that drew 100 lines, 100 sketch lines. And so then Python is 23 times faster because now the call takes longer in JavaScript but the work that's happening in Fusion is the same regardless of if it came from Python or JavaScript. So the harder Fusion has to work, the less of a difference. But I think Python's always going to be faster than JavaScript just because the in process, out of process.

And then Python is built to be just a general purpose programming language, so it has a pretty rich set of functionality and it has built into it full access to file system. You can read and write files, access the registry, do whatever you want. And again its Python, what you're creating is just an ASCII text file so you could use any editor you want, but we do deliver one with Fusion called Spider. And I'll show that in a little bit.

And then Spider is also used for debugging so you have this one environment that you use for writing and debugging. And you can use other Python libraries besides what's built into Python. And then it also uses a search for error. So if you're familiar you can just try except with the catch errors.

So C++. So it runs in-process like the others. And here you have Dll on Windows or dylib on Mac. And so it's going to be the fastest of all of them. So it's running in-process like Fusion, but then it's also compiled. Python isn't known by itself as being a fast language because it's interpreted.

AUDIENCE: [INAUDIBLE] I wasn't sure when you stop.

BRIAN EKINS: I can post these on-line too if you want. So it's going to be the fastest and just build in a C++ until you have full access to the file system. So that's nothing to do with the Fusion API itself just build into the language. And then the default, when you edit one of these is on Windows, it will use Visual Studio, and on Mac use XCode, but you could change it to another editor if you want. And then C++ and C++ is typed. When you declare a variable you tell it it's this type so you get better code hints as you're writing, and so it can be a little bit smarter.

And then in C++ we chose just because it's more standard in C++ programming to use error codes instead of asserts, so it's a little different that way. But they all are using calling into the same API regardless of which language. So with Python and JavaScript I create this program

and then it's interpreted when I actually run it. So that same program can just run on Windows and run on Mac. I have to do anything special for different platforms. But with C++ I have to compile it on Windows to create a Dll then I have to have a Mac that I compile it on to create a dylib to run a Mac.

So here's a program and so this is going to demonstrate what we were talking about before with the structure and how we navigate through that. So this little program ends up creating a new-- so I have a design already up and then it creates a new sketch, draws a circle in that, and then adds a dimension constraint to it to control the size of the circle. And this is a JavaScript example. So this first line it's calling this GET method on the application object. So this is a static function that it's calling. And so what that does is that gives us the application object. So now we're hooked in to this big chart.

And then on the application object it calls this property, called active product. And the way Fusion is, is you have a document and then inside that document there's chunks of data. And so as you're designing when you have all your models and your assembling everything, that's part of the design chunk of data. And then if you do some CAM then that's going to be in a CAM chunk. And if you do some analysis, that's going to be in another chunk. And so when the Fusion opens up that document it just has to open up whichever chunk is needed at the moment. And API refers to those different chunks of data as products.

And so this active product is what's the user currently working on inside Fusion right now. And this is assuming that he has a design open. If he was working in CAM this is going to fail because it's getting-- well it doesn't know yet to fail. It will later. But it's good to get whatever the active product is and then we're going to assume it's a design and use it as a design and then something later is going to fail. So we get the active product. So now we've got-- we're to this object so we shortcutted those other two and jumped down to here.

And then on the design it calls this property called root component. And the root component is the top node in the browser. And so now we're here. And then we want to create a new sketch. So on the root component it has this sketches. So it has a sketches property on it that returns the sketches object. So now we're down to here. And the root component also has a property that returns the XY construction plane, just the base plane that you see listed in the browser. So we get that and it gets it and assigns it to this variable.

And then on the sketches collection which we got up here, we call the add method on it and

pass in the XY plane and that creates a new sketch and returns it. So now I have a new sketch that's just been created. And so now we're here, so we have a sketch. And then on the sketch it calls it the sketch curves property which gives the sketch curves object. And then on the sketch curves object it calls the sketch circles property which gives a sketch circles object. So now we're down here.

So there are actually two calls here. It's calling the sketch curves property on the sketch and that returns a sketch curves object, and then it calls the sketch circles property on the sketch curves which returns a sketch circles object so that's that. So now we have a sketch circles object and then it calls add by center radius. So there can be different kinds of adds depending on how you want to create something.

And so here we are creating a circle by center in a radius it defines the coordinate 000 of the center point and the radius and then passes back the circle that just got created. So now we have a circle. And then from the sketch, it's getting sketch dimensions and it's calling it's add diameter dimension method, passes in the circle that just got created, and then defines another coordinate for where I want the dimension text to go. And so it creates a dimension on it. So does that kind of make sense or is a little overwhelming or?

So the key is that chart and your just traversing this chart. And a lot of times you kind of do that-- I mean you do that in the UI but it just becomes natural and you don't think about it. So if you wanted to access as a specific line in a sketch, what do you do? You first have to open the document that you know that sketches in, so here I'm kind of connecting up here, and then I have to go find that sketch in the browser and so I'm still traversing down manually to that and I open up that sketch and then I look through it and find the line that I care about.

So here's the same program in Python. And I'll go back. So there's really no difference except for just syntax differences between languages. How I define a comment I don't have semi-colons on the other. JavaScript requires a var when I declare a new variable. Python doesn't.

Here's C++. So C++ still the same thing. It has the same objects, calling the same methods, just syntax again is a little different. How I declare variables I use are different than-- so it's all the same API it's just accessed in different ways through different languages.

Here's an example just to carry on what we had a little bit that creates an extrusion to illustrate that input object how that works. So we already did this up to here. And as you're drawing in Fusion, in a sketch, you're drawing shapes. And as you kind of close the shape you see it fill,

that transparent fill. And so what Fusion's doing as you're sketching is it's looking for these closed areas and those are called profiles. So because then when you go to create an extrusion and it's asking you to pick a profile you can go pick any of those closed areas one or more of them. And so once we create that circle, Fusion has already figured out, oh, there's a closed area so I'll create a profile. And so this is what this line is doing, is from the sketch it's going to this collection of all the profiles that exist in that sketch and it's just getting the first one. And so you could have a whole bunch of profiles in a sketch, but I know I have one here because I created the sketch in the program and I created one circle, so there can only be one profile. I'm just getting it.

And then here, from the root component, I'm getting the features collection and then getting the extrude features collection, and now on that extrude features collection I'm calling this create input. So that's creating that input object. So nothing's happening inside Fusion. Think of the dialogue, the extrude dialogue just popped up.

And so here I provide the things that every extrusion needs. So it needs the profile or profiles that could be more than one too. And then for every extrusion I have to define am I creating a new component, am I subtracting, am I adding, joining. So you define that. And then I can optionally define some other stuff.

Sometimes optionally, some of it's required but different options-- so on an extrusion I have to define is it going to go through all, is it just going to go a certain depth and so that's what this is doing. So I want to create an extrusion that's five centimeters deep.

And so this is filling out the stuff in the dialogue, the equivalent of that but doing it through the API. And then once we've done that then we go back to the extrudes collection, call the add method and pass in that input object. This is the equivalent of hitting OK on the dialog. So now an extrusion is built.

So where do we go for-- might reorder a little bit. Let's go ahead and look inside Fusion here. So if I want to create a new program-- so back to languages a little bit. So there are the three languages but if you are kind of new to this I would recommend Python because it has less limitations. I think it's probably easier to use. Definitely easier to use than C++. And it's just more complete, faster than JavaScript. I would recommend Python. So that's what I'm going to focus on here. If you're going to deliver something more professionally than C++ it's probably a better option. So you can deliver your binary and be faster.

AUDIENCE: Is C++ and JavaScript [INAUDIBLE] updates indefinitely or-

BRIAN EKINS: So the question is, so with C++-- so with Python and JavaScript you're basically delivering your source code to somebody. And it's interpreted at runtime. And with C++ you're delivering a binary so the code is less flexible I guess. And as we do new releases, is that same binary going to continue to work.

So we've done actually a lot of work. That's why C++ was later in coming as we had to figure out how do we provide this binary compatibility across releases. So you should be able to create a C++ Dll today and it should work a couple of years from now. And part of that depends on how much does Fusion changed between now and then.

That's the really big question. So we may have some API on something that just can't exist two years from now because Fusion has changed so much. But as long as that functionality is still there the program should still work.

So I want to create a new script or an add-in which we might talk about a little bit.

So the heart of the API in the UI is the scripts and add-ins command. And inside here there are two tabs and add-ins and there are two folders in each of those tabs. So on your machine right now that's probably empty, unless you've installed something from the app store potentially. So these are the scripts you've added or have installed. And then this folder, you'll have the same ones as I have. So these are what get installed just when you install Fusion. So these are sample scripts. So you can go look at run use and look at for examples.

And then add-ins the same thing. So you got some samples and then the ones you built. And so I can create a new script or a new add-in of any language right here. So I want to create the new Python script C and I can optionally give it a description author and create. And there it is. And let's look at it. So this is Python. So I brought up this Spider editor. And there's the initial script. So I could run it right now. But let's just go ahead.

So it brought up hello script. So let's look at that a little bit. So it's importing some libraries. This run function, so that's kind of the main function. So Fusion automatically calls that so it loads up your script and it calls that run function. So that's where you'll put your code. This is using this try accept thing so if any errors happen in my program it's going to jump down to that accept, that message box at the bottom and report the errors to me.

So at the top it's getting the application object. Just like we saw before on the application object there's a property called the user interface that returns a user interface object and it supports a bunch of stuff, including this message box. So it's just calling that and this is the string that it shows us. So pretty simple.

AUDIENCE: Do you get IntelliSense with that?

BRIAN EKINS: Yeah good question. So we'll get into that. So in fact, let's go ahead and look at some things here.

So IntelliSense and in Python-- how many of you have programmed in Python? Alright, a few of you. So I hadn't before I started working on this, so I can still consider myself kind of a newbie to Python a little bit. Some of it's good some of it I don't particularly like. So these variables, they were just declared on the fly, I just assigned something to this new name and it made this new variable app. And it's smart enough to know what created-- what I called on this side to create app. So the editor is kind of keeping track of that and trying to figure out, oh well what is that variable. And so it knows it's a Fusion application object. So if I do app dot, those are the properties and methods that it supports.

But let me show you where we start to fall apart. And so it still knows what app is and then I'm calling user interface on that, so it knows UI is a user interface object. So it's still working. But if I do this, design equals app dot active product and design dot and I want to get the root component from design, it's not here. And I'll show you in just a second why it's not. But I know it's there, so I'll just go ahead and type it in. And then the root and it doesn't even know what it is at all.

So here's a big tip that makes, I think, using Python a whole lot easier is you can help it. So the reason that it didn't know what design was, is I remembered these products are these chunks of data inside the file and so a product by itself is kind of generic. Inside the API there's a thing call the product. And so that object has everything that every type of chunk of data supports. So it's only the common stuff the everything would have. And then the design product is derived from that. So it supports everything product has plus it has all the stuff that's specific to a design.

And so what's defined in the API is active product returns a product object. And so when I did dot I'm just seeing the stuff that a product has. But I know I'm really getting back a design object, not just a product. But the editor doesn't know that. And so I can help it to do that by

right here.

So the API is divided up into two libraries. There's a core library. And how things are divided up probably don't necessarily make logical sense always to you, sometimes not even to me. So you just have to look it up probably in the help and we'll look at the help and little bit. So there's a main ADSK namespace that everything's in and then there's this core and Fusion library. And then there'll be some more so they'll be a CAM library soon. So we've got Fusion dot product or design and then all of these support this function call cast.

So this doesn't change at all how my program works. All I'm doing really when I do this is I'm giving a hint to the editor, hey the thing that's coming back from this is a design object. So now down here when I do design dot, now it has a root component. And now root has all the stuff it has.

AUDIENCE: So this Spider [INAUDIBLE] Fusion [INAUDIBLE].

BRIAN EKINS: So if you haven't programmed at all when you first edit a Python program, Spider will load. So it's not actually installed with Fusion but it'll automatically load the first time you need it. And so I wrote a paper for this class. You can get it online if you go to the AU Website and look. And I have a link. There's a professor, I'm not sure where he's from, but he published on YouTube several videos on kind of getting started with Python and he uses Spider with that. So it's Spider and Python. So it's maybe a good but kind of dry introduction to everything. And it's free. So it's just up on YouTube.

So that cast is really nice because especially if you've been using Visual Studio. You just get so used to these code hints. And it does make things go a lot better because it helps avoid typos and because all of these languages are case sensitive, and other kinds of typos. So code hints are really good.

In fact maybe while we have this up. So I've got a program. So here to access an existing one, so from this script and add-ins so we saw I can create I can go here and edit an existing one. I can just run a program. So this is where you run scripts is through this. This is your UI to run a script, pick it and run. And I can also start it in debug mode from here. But I just want to edit. And so here's a little script that draws a line, actually it will draw-- it asks you to pick a face and then it finds that the normal on that face and draws a line sticking out of the center of that face sticking out on longest normal.

But I wrote this-- right now it has a lot of problems. And a lot of these problems were stuff that I really struggled with as I was getting started and just figuring out Python, mainly Python things. And so I wrote this with all these different problems in it so we could kind of step through and fix it and see. I'm sure what you're going to run into too if you started playing with some of this stuff.

So the first thing to do that Spider helps you with is over here on the left hand column, are these little warnings and sometimes errors depending on what it is. And with Python where you don't have to declare anything you just create new variables on the fly and stuff, it can be easy to kind of mess up there. So we can fix some of that.

So for example, here it's saying design is assigned so basically the designs being created, I'm assigning something to this new variable, but it's not used anywhere else and so I probably have a mistake somewhere. And if we go down here, undefined names. So this hasn't been set anywhere else, I'm using this name that you haven't created. So I've got a typo, I just left out the i. So that one goes away. And then down here I've got undefined name offset point. So what's going on there. So it doesn't understand offset point but here it is. Anybody see what the problem is there?

AUDIENCE: [INAUDIBLE] it's not the right case.

BRIAN EKINS: Right. So it's case sensitive. If you're coming from VB-- VB isn't case sensitive and you could get really lazy so then switching from Visual Basic over here you got to be careful. So this is a capital S where this is a lowercase s.

So there we go. So now we have no more warnings. So let's go ahead and run it. And let me get some kind of geometry we can try it on. I showed you in that scripts and add-ins dialog I can run and debug from there but I can also run and debug from the Spider ID. So I'm just going to run it. And so it's running so it's gone down. We're on that face equals select entity line.

So this user interface object supports this method, select entity. So it's asking the user to pick something. And here's my prompt and this is a string saying what kind of things can he pick. So I'm saying he can pick any kind of faces. I could say solitical faces, or a sketch line, or different things, so I'm filtering what he can pick. So it's asking me to pick a face. There we go. So that's kind of overwhelming that mess of stuff up there.

So let's look. So you want to start up here at the beginning. So something failed and a nice thing is this line, line 18 and face evaluator. So what's line 18. It's a face dot evaluator. So that's failing. So how do I even know what the face supports. So if we do face dot. So here we're not getting any IntelliSense either so I'm not getting help here so we can use that cast up here.

And now down here I should be able to face dot evaluator, so there it is. So let's try it again. I just had some goofy mistake. Pick a face. Still no evaluator. So what's going on. So let's debug. Get a little more help. So debugging I can double click over here and create a break point and then we can debug. So when I debug it automatically breaks at the top but I'll just let it run. Now it's running. We're down to this line.

And so I'm debugging-- I don't know if you can see this, I was able to make this bigger but not the console window. But a couple things as I'm debugging, is this window over here is nice, the variable explorer, and this is showing me all of the variables that the Python has created and their current values. So I can look over there. But I can also use the console window to kind of do the same thing. So if I just type app, here it shows me so app is of this type, its application type. So I know it's been assigned and it's an application so it's good.

And so UI, it's getting the active product. It's getting the root. And I can do other things too here. So root-- so it tells me that's a component, so I can say root dot name and it comes back unsaved right now because I haven't saved this document yet. So you can kind of run the UI-- or run the API over here too as you debug. But let's see. Let's step again. So now it's asking us to pick.

I got sidetracked there somewhere. Got out of sync. Let's put the break point after we do the selection. I just ran it and didn't debug. Debug. So we've broken here and when we call face dot evaluator that's what's failing. So why is that failing. So let's look and see what face is. It's nothing. So we didn't get back what we expected from this selection.

Let's see, let me take out this cast first. And let's debug that again just to show you something. That was from the previous run. And I'm not sure what's happening here. We have some little problem but it's hard to reproduce consistently and if you get in this you just have to shut Fusion down and start it up again. But it's something that we need to figure out and fix on our side.

Got our break point. Big face. Now let's see what face is. And I still get caught with this in my

program so that's why I wanted to highlight this. And you wouldn't know this, this isn't necessarily a Python thing this is something with the API. So from this function, the select Entity, returns a selection object. So it doesn't return the thing that got picked, it returns a selection object and then from it I have to get the thing that got picked. And so I'm missing a step here and I can do it all in

this one line. I'll do it in a separate line. So face equals face dot entity. So a selection has two things, it has a property to let me get what was actually selected and it has another property to give me the point in space where it was selected. And we may add more stuff to it in the future. So I get back this wrapper that has all the selection information including the thing that got selected. So now face is good.

So let's run our program again. Pick a face. So now we're lower, down on line 22. So we've got some problem in there. So let's look at line 22. So a useful thing on all these-- let's go look at the help. And so in Fusion, just from Fusion itself, I can go to learning and help and then here is a programming interface. And in the programming interface there is this welcome topic that provides some links to some other sources. There's this what's new that lists what's new in the latest update.

And then there's this user manual that has some overview topics for all the stuff. So how to create the script and add-in and then a lot of what we're going through here. So it's some Python specific issues. So this talks about doing that casting we talked about and other things. And then using the other languages.

And then the heart of it really is this reference manual. So that big object model chart-- so there's over 500 objects there and I can't remember how many thousands of functions, methods, and properties each object has and those are all here.

So if we go to say this, whatever it is. Well, let's look at something we'd be familiar with. Here's an extrude feature. And so here are all of the methods and properties that the extrude feature supports. And so if I click on one of those then it takes me to information specific for that. And at the top of each of those is the syntax for each language of how to call that.

And so several the problems I think I'm going to skip going through some of them more of this is for sake of time. But a good share of the other problems we were going to find here could be fixed by going and looking at how the call is here. So one of the things in the program where Python is different than the other two is the way Python is, is you can't return a value

through an argument. So in the API it's defined in quite a few places where I call some method and then there are several things that I want to get back. And the API defines those as arguments to get returned from that function call and Python doesn't support that. So everything is returned as the return argument, the single return argument.

So the way multiple things are returned is as a Python list or a tuple because in this case, which is just an unchangeable list. So then you have to extract the stuff out of the list to get the results that you want. So that's something to watch for too. So that was in that sample program but I'll skip that for now.

So this syntax is good. We'll get to this some more in a second. So one other thing on the help is a few months ago they changed the format of the help, so how this looks. And let me show you the old style which is still available. And the old style still works better for accessing the API. So the content is the same. So it's not old content, it's just the format of how the help is displayed.

And so I would recommend using the old format for the API information and it just works better. There is some a scroll bar here on the side is independent of this. And here you can see it's all one piece and this has a lot of white space around it so I don't get to see everything.

So let's go back. So the old help format, if you Google for this, it'll take you to this blog post that I made that gives you a link of how to get to the old format. And so this is what we already talked about, it's in the help. So there's a lot of information there. So there's that many topics over 6,000 topics in the help for just the API.

So you can create scripts or add-ins. And there's really not much difference technically. They can both do most of the same things. Really the difference is with scripts you access them through the scripts and add-ins dialog and you run it from here, it runs and it's done. And you can use commands from scripts but it's probably not all that common, but you can there's no limitation to not do that.

But with add-ins, Fusion looks for add-ins when it starts up and if an add-ins is there it runs it at startup and then it's what an add-in can do at that point is it can create a custom command and then it can go insert it into the user interface. So with an add-in, if I have an add-in on my machine and it just looks like native Fusion functionality. I start up Fusion and I just have all these new commands available. So a user really doesn't know is it native Fusion functionality

or is it an add-in command.

And so the add-in runs at startup, it inserts its commands and then it's still running in the background just waiting for one of its commands to be run so that it can react to that and do something. And this is what I said. So commands in general are typically executed by the user but you can also run a command from the API, just an event or Command I could or Fusion command. I could execute extrude from a script. But what's going to happen is it's the equivalent of clicking the button but then I can't provide any input through the API. So it would just start it and then it's just going to stop there waiting for the user to interact. No interaction with the command is supported through the API. And all commands are represented by this object, this CommandDefinition.

So whether that's a custom command you created through the API or a standard command extruder line are all represented through the is one of these objects. So I can access the Fusion commands. And all commands I can get to through this user interface object that we saw a program a minute ago use, and then this command definitions property. So that returns a big collection that has every command available both add-in commands and Fusion commands. And then there's some add methods on this command definitions to let you create new commands.

So what a command definition does-- so a little bit on commands, I wanted to spend some time on this because I think it's a little bit confusing. But if you end up doing very much at all with the Fusion API you'll probably want to create a command because that's really how you're going to get input from a user. So the command definition defines what a command looks like. So it's not the command itself really but what it looks like in the UI.

So the command definition defines the name, so what you see in the user interface, it defines the icon, the tool tip, and commands optionally can also have a tool clip so it's just an image that shows up if the user hovers long enough. And it defines the behavior in the UI, so whether it's visible or not. So you can have a button there but it's just not visible. Maybe you turn it on and off based on some state in the system but more likely you'd enable it or disable it. So you can control that. So all that's controlled from the command definition.

And then the command definition provides this event, command created. So what happens is in your programs you create a command definition, so you're defining what the name is, its icons and all that stuff. So you create that and then you use the API to go find in the UI where

you want to put that. So Fusion is divided up into workspaces so that's all of these. So I can go access all of the workspaces that exist. So I'm going to find the workspace I want to put my button in workspace or workspaces I want to put my button in.

And then there are Toolbars. So that one's called the QAT, the QATRight, and the NavToolbar. There's a couple others but they're special case ones and they're accessible too, but these are the ones you'll probably use. And so you can access those through the API. And so you could add buttons into any of those.

And then there's a Toolbar Panels. So associated with each workspace is a set of Toolbar Panels. So these are all the panels where there's a single panel. And then a panel has a list of commands. And in this case this is a special type of control that has another list of commands and you can access all of this stuff through the API. So I can again traverse through this to find a specific place that I want to insert my command and then I can insert it in there. And then as users of Fusion you're familiar command I can click that little arrow and it'll pop it to the top or I can pull it down. So you can do that same thing through the API.

So here's how I insert a command in to the API. So from the UI I'm getting all the Toolbar Panels, so that's a list of them. And I want the sketch panel. I'll show you in a second how did I know that name. So I want the panel called sketch. I want to insert a command into here. So I get this the sketch panel just with that. And then I want this thing here. I want to insert my new command into the pop up that shows up. Yeah, into the pop up next to that.

So the sketch panel has a list of controls. So this is the collection of controls and I'm asking for the one named this, Project/Include drop-down. So now I have that guy. And then I am inserting my command. I've created this somewhere else. So this is my command that I've created and this is a variable referencing it. And then this I'm referencing this existing command to tell it I want it to go next to it. And this is saying if I want it to go before or after basically. And so there now I insert my command into there.

So how do I know these names. That's really the key to this. And there's a sample. So in the API help there's this topic in there to sample programs. And in here there's this program write user interface to file. And what it does is if you run this program then it creates a file, a text file on disk, and it has all these names. And it's organized how you see it in the UI.

So I can go out there and find the design workspace. Probably in this case, I could just search for sketch and then I could go down and find the sketch panel, then I look underneath that and

I see all the controls and that. It builds up this text file that duplicates what I see in the UI with all the names and then I can use those.

And it's been asked why don't we just publish that. But the problem is it changes real frequently so it's a maintenance thing. But it's also different depending on what you have on your machine, depending on what other add-ins you've installed, you're going to see something different on your machine.

So that's the first part of creating a command as I create that definition. So that's just building my icon and putting it somewhere in the UI. But what happens when the user clicks that. And so Fusion has this really well-defined concept of what a command is. And so usually you have something in the UI to run it and then when it's run, a dialog pops up to prompt the user for whatever is required. And then typically a preview is shown of the result as the user is changing things in the dialog, just like extrude.

And then you can cancel and nothing happens or if you click OK then it completes the command and you get the result. And everything can be undone in one undo. So that's typical behavior. And so that's kind of built into Fusion as part of its command architecture. And so the API has just exposed that same architecture. We just wrapped over that and to let you use it to.

So you create your command definition. And you connect to an event. And I want to demo this real quick before we run out of time. So you connect to an event on that. And so what happens when the user clicks your button Fusion creates a command internally and then calls this command created event. So it tells you, hey your button was just clicked and here's the command that I'm creating for that. So your add-in waits. User runs it. Fusion creates this command, fires the event and then on that command object you just got back, you can create what's called command inputs. So you're defining your dialog. You're building up your dialog that's going to be displayed for the user.

So I need the user to select a face and I need him to enter a distance and to type in this string or whatever. So you define all the inputs that you need for that command. And you connect to some other events. The most important one here is you connect to one called the command executed. So that'll be called when the user clicks OK on the button. And then in reaction to the command executed you do the actual work.

So Fusion displays the dialog. The user interacts with it to do whatever input. And then

through those other events you can validate the input, you can filter what's happening, you can show a preview, and then when the executed you create the final result. So this is kind of elegant in the design the way Fusion did it I think, but it's maybe a little hard to wrap your head around the first time you try and do some of this. And especially because it involves events.

So you're having to create quite a few little functions to do this command because with an event you connect to the event and then you've got some other code somewhere in your program that's the handler for that. The Fusion calls when that event happens. So let me just show you kind of a trick to do this and show that it can be not that hard.

So let's create a new script, AU add-in. So here's all the command definitions and I want to create-- there's three different kinds of command definitions I can create. I want to create a button. And I give it some ID. So that needs to be a unique ID amongst every command inside Fusion. So a lot of times you might prefix it with your company name or your name or whatever just to be sure.

And then the name tool tip and resource folder it said. So name-- maybe I got those backwards. No, ID name. So this is what will show up as the name that the user sees and the tool tip can be whatever you want. And the resource folder I will leave off for now. So that's a folder where your icon files are going to exist and you can read in the help of how those are actually defined. And let's see. I'm going to skip some stuff here just to get to another point.

So in here I would be inserting my command into the UI. So that code we looked at earlier. But the thing I need to do is I want to hook up to this command created event, so I can react when the user ran this.

So how do I do that to make that easy. So this is an event and the help is where I want to go. So I'm going to go down to command definition. And so here's a list of methods, properties, and events that this object supports. So the command created event is what I want to listen to and here is what I want to use. So if this is for all the languages but here's where copy and paste can make it just a whole lot easier.

So this part I need you to copy and paste up here. So this is a global variable. And so I'm going to create an event. So any variables inside Python-- it's like this command def, once run finishes that command Def variable goes away. And whatever it was referencing it's just not referencing anymore so that variable goes away. And these events when I hook up to an

event I have to keep it alive if I didn't do what I'm going to do in a second it would at the end of run that variable would go away and the event wouldn't function. So I have to keep a referenced and keep it alive. So this global is just a list of variables essentially to keep them alive so they don't go out of scope them and get released.

And then this handler-- so I'll just grab that. So there's the code that's going to get run that Fusions going to call when my command gets created. So then if you're used to Python you know what some of this is and most of it you can just ignore. But this is the main part, is this notify. So I would put my code in here for what's going to happen in reaction to when my command gets run.

And then the last chunk of code is right here. Not there, down here. And I just need to change that. And that's it. So now I have a handler for that event. And so you can do that. So that exists in the help for every event. So you can just copy and paste to build mainly that and it helps with this too. So you don't have to do that from scratch every time you want to implement an event.

And so let me show you a couple of commands that do some of this. So this is a custom command. So I inserted it into here so that's the name and the icon I built and whatever. And it could be promoted and have a big icon at the top and so it just behaves like a Fusion command.

And so when it was executed it got that command created and then it added these three inputs in reaction to that. And so I have a selection input, I have this drop down input, and a value input. And so it looks like every other Fusion command basically because they all share the same architecture, they all do the same dialogue kind of stuff.

And what this command does-- here's just a little demonstration, but I mentioned before how the API can access the geometry, what does a model look like. And that's what this is doing, as you pick a face it's extracting data out of that and then drawing this mesh over the top of it. And so it's previewing, in this case, so just as I picked it is jumping around. I could change some values.

Or there's another option here to display normals. And then if I do cancel nothing happens. If I click OK then it creates it as a result. It created a sketch actually and so there's a whole bunch of sketch lines. It's what it drew. And because it was all created in that command it's a single undo and it's gone. So I got that for free, the undo.

So let's look at another one. So here's a little program that again uses some of the geometry stuff, inserts itself. So see you can insert yourself any where you want, whatever makes sense. This one just has a single selection, pick a body. It's analyzing the solid, figuring out what size, creates parts, represents [INAUDIBLE], inserts them, adds a joint between each one, and just fills all the holes.

So a couple of things just as I finish up. I want to just point you to a couple of things. So in this welcome there's a GitHub site. And there's a bunch of videos and some samples here that are real good. There's those topics in the help that you want to look at that I showed you already, those overview articles. This is a good one too.

There's quite a few videos that go into some more detail on commands. But they don't talk about that copy and paste. I need to publish some of that because that's fairly recent that I put that in the help. I'm using this stuff and it was a pain every time I wanted to do an event it's just hard. It's easy to make simple mistakes and then stuff doesn't work. So that makes it a lot easier.

So I have some of these charts if anybody wants one and good luck with the program Fusion and thanks for attending.

AUDIENCE: Is it possible to make them parametric? The sketches [INAUDIBLE] if you went back and modified the shape, your sketch [INAUDIBLE] you'd have the sketch, you'd have the normal filter to use.

BRIAN EKINS: I'm repeating because it's getting recorded here. So it was asked, is it possible to make the result parametric. And it depends on how you create this stuff. Some things you can create parametrically, like if I draw a work plane parallel to a face then it's going to stay parallel just because that intelligence is built in to the command when you create the work plane.

But what I did there with the sketch lines and normals, that's not associative. It's just a sketch that right then happens to lie at the normals of those. And today there's not a way to do that associatively. So that will involve us exposing some more events. So you could be watching for changes that happen in the model.

And so you'd be listening to some new events, we would tell you, hey this face is changed, and then you would probably delete or modify or whatever but redraw your normals. So that's not

there today. Any other questions?

AUDIENCE: [INAUDIBLE] connected the Dynamo [INAUDIBLE]?

BRIAN EKINS: And connected the Dynamo. And yeah, we haven't looked at Dynamo at all yet, but there probably is some-- maybe some place for some good interaction there.

AUDIENCE: You've got access to A360 doing API, so can you open the projects [INAUDIBLE]?

BRIAN EKINS: Yeah, so there is some API access. It's not complete yet in this area but enough for most things that's the equivalent of the data panel. So through the API I can query and find out what projects do I have. And then in a project find out what files and folders are inside that. And if there's a folder, go into that folder. And then once I find a file, I can open it up in Fusion. And when I create a new file and save it I have to tell it which project, which folder do I want that file to go into and so that that's all possible today.

AUDIENCE: Can for example, [INAUDIBLE]

BRIAN EKINS: Yep. Yep. Any other questions? I'd like to hear-- we're out of time, but if you want to stay after or even contact me later if you're wanting to do something I'd like to hear what you're trying to do. And especially if you can't do it because of things missing in the API or potentially because Fusion doesn't do it either.

What is there. I mean we know there's a lot of stuff we still have to do. And so it's prioritizing what do we do first. And if we know somebody is going to use this-- we have a lot of experience so our guesses are probably pretty good about what's more important than others but not always.

And sometimes you're doing something kind of unique but this is pretty cool and so we could prioritize that little higher. Any others? We might be done. But I'll stay a little while after and as I say I'm happy to if get in touch with me later we can chat too. All right, thanks.

[APPLAUSE]