KEVIN VANDECAR: All right, so I think that's our cue to get started when the door closes. So welcome everyone. Thank you so much for getting up at 8:00 in Las Vegas. How many people went to the beer bus last night at the expo hall? Well that's why you're here so early, you didn't go to that.

So I did walk through. Some pretty cool things out there. If you haven't had a chance to go through, definitely check it out. I especially like the electric cycle that the *Orange County Chopper* guys put together. They use Fusion 360 for that.

So this is the 3ds Max custom tool development. And specifically we're going to talk about .NET API and Python. These are the newest API environments in 3ds Max, and we're going to kind of compare those two things.

My name is Kevin Vandecar. I'm with Autodesk. My title recently changed to Forge developer advocate. So we go out and we do these kinds of events to spread the knowledge around the APIs. So Forge is our new cloud-based platform, so that's why we're kind of focusing on that as the newest technology.

So this is a programming course. So in the past, I'm not sure if I've advertised it incorrectly or whatever, but I've had some negative feedback saying, well that was a programming course and I expected something else. So feel free, if it's not your thing, don't feel like you have to stay. I'm fine with that, because this is kind of a little bit of dry material because it has to do with programming.

I'm from Manchester, New Hampshire. My email address is here. I have a Twitter handle. And we also have a blog. The blog is not as active as I would like it to be, but we're trying to get better at that. So certainly check it out from time to time if you're doing regular customization in 3ds Max.

So this is the class summary. And again, we're going to specifically talk about the .NET API, and then we'll talk about Python. At the beginning we'll spend a little time comparing the two. And it's meant to kind of give you an idea of maybe which direction to go if you haven't started customizing in either of the environments yet.

So a quick show of hands, is anyone using .NET API in 3ds Max today-- someone. OK, how about Python? OK, one. OK, great. So this is probably the right course if you're interested in

programming topics. So these are the class objectives. I'm sure you read those. So I'm going to get started.

So customization is such an important aspect of our products. 3ds Max has been around a long time. You can kind of put it in the same bucket as AutoCAD. In fact, it's probably been around nearly as long as AutoCAD if you go back to the DOS version 3D Studio. Even then it had a programming environment. It was called IPAS and it was basically four different plugin types. So customization has been just really, really important to the success of 3ds Max.

So before we dive into the two new environments, I just wanted to cover a quick overview of some of the other customization aspects. Because generally when you decide to make a tool, you're going to not just program in one environment, but you're going to probably use some of the other environments as well.

So first of all, the Customize UI, how many people are familiar with that Customize User Interface dialog box? OK, a few of you, that's fantastic. That Customize UI dialog box is really your entry point to customization.

So it's found on the Customize menu. And this is 2017 I'm running. Things kind of moved around in recent versions, so you may find it in a different place. But when you when you bring up this dialog box, you'll see on the left, in most of the tabs, the actions that can be customized. What that means is basically you can take those actions and move them or assign them to various UI elements. So this first tab is the Keyboard tab. So any of these actions, which in a sense are commands, can be assigned to the element of the tab-- so in this case, a keyboard shortcut.

Now the sample code that I'm going to be showing in this, I've not programmed it to show up in the UI. I've simply used this dialog box to create a menu and put my actions on it, because it's easy. If you're just creating tools, scripts, that sort of thing, it's a lot of extra code to actually build, for example, an action that programmatically goes into the menu. Now if you're creating a commercial tool, you may want to do that. But if it's an in-house tool, drag and drop it and you're done.

So the nice thing about the Customize UI facility is that you can also take this, and you see down here at the bottom, you can say save. You can save these CUI files. And if you're kind of an IT manager or something like that in your office, you can take those CUIs and you can share them as well.

So maybe you have a roomful of artists or a roomful of designers, and all they need is maybe the modeling tools or the rendering tools. You can make them a nice little menu, and then you can share it and everyone will have kind of the same set of tools as well.

All right, so just as an example, you can see, this is my custom menu that I've created. And down here is where it was actually customized. So to customize something-- and this is something that's not always obvious, how to use this dialog box. But it's a drag and drop kind of thing. So I can pull it and drop it over here.

And if you want to create a new menu, you'd come up here and say New. It's going to ask you for a name. And at first glance you'd say, well what happened to it? Well it actually goes down here. And let's see, what did I call it-- Fred? And it's here now. So this isn't actually on the menu bar yet. It's just a menu that's in the menuing management system.

So if I want that to be an actual menu, I can drag it over here. And you'll see now I've got the start of a dropdown menu. And then I can take one of these commands-- I need to open it first-- and I can pop it there. And now you'll see it there.

So it's pretty simple. But I have to say the layout of this dialog box is less than ideal. So you start up here. You go over here. You go down here. You go over here. And then you go back up here to bring it over here. So it's a little bit funky, but it's a fantastic system once you spend a few minutes to figure out how it works.

So something that came to 3ds Max in the last few releases is the MAX Creation Graph. Anyone using that today-- so a couple of you. So MAX Creation Graph is a visual programming environment. It allows you to drag and drop these different behaviors and combine them with different object types. And it's basically a flow graph. So you have an output and an input, and the flow through the graph defines what the behavior is. So you pass in a cube. You do some modifications to it. And maybe you have a bendy cube with animation at the end.

So, I would encourage you, if you're not a strong programmer but you want to do some customization, definitely take a look at MAX Creation Graph, because it can be a lot of fun just playing with it, dropping things. And you get immediate kind of feedback on that.

MAXScript, of course, has been with 3ds Max for a very, very long time. How many people use

MAXScript? So in order to use Max well, you're probably driving some MAXScripts whether you wrote them or not. So MAXScript is pretty powerful environment.

It is a proprietary environment. So learning MAXScript isn't going to get you a job using Maya, for example. So you have to kind of keep that in mind. And I think that's where the power of Python coming into 3ds Max helps that, because you learn Python. And that's a skill that, even though the APIs are different, you can take that programming skill with you.

And then the last one is the C++ SDK. So this is the heavy lifting area. If you want to do anything with performance in mind, you want to do anything where you're protecting your code base-- so maybe you're a commercial developer and you're writing something really important with some IP, some algorithms that you don't want to share, C++ is really the environment to consider.

The drawback to C++ is that C++ itself is very difficult to use. You have to do things like manage memory. So you have to create something, and then you have to remember to delete it. Otherwise you can actually cause Max to get into a bad state, consuming too much memory. You can cause it to crash very easily. And so those things kind of raise the barrier to entry.

So we're going to talk about .NET and Python, in detail. And so the first thing I'm going to do is compare the two and kind of give you an idea of which one might be better for your environment.

So first of all, the environments are quite different. So on the left here we have the .NET environment. It requires Visual Studio to be productive. You can get open source compilers that will build different .NET languages. But Visual Studio has a community edition, and it's free, so there's no reason not to use Visual Studio.

It does come with a really nice debugger. So that's built in. There's a package manager called nuget. This was one of the limitations early on with .NET in general is that it was hard to kind of consume these larger packages of functionality. So they've added this now, and what that does is it allows you to kind of pull these packages from other sources and manage them for you. This is kind of similar to the Python pip system.

So in order to use it, you have to use a framework. So this is the thing that, when 3ds Max installs itself, if your computer doesn't have the framework, it will install it, because 3ds Max

uses it as well. So somewhere along the line-- and Windows now with Windows 10 is also using it very heavily. So the framework will probably already be there.

In order to build a .NET API plugin for Max, you need to use the same framework version as 3ds Max. So there's a version associated with these frameworks, and you have to use at least that minimum version to be compatible. You can use newer versions, but it's kind of limited to the functionality that's in our assemblies as well. So if the new features in a later framework don't require anything to be implemented by the APIs themselves, it'll work fine.

On the Python side, it's changed quite a bit. So first of all, in 2017, we added the ability to write Python using the MAXScript editor. So this a brand new feature. There is a type on the dropdown menu in the MAXScript system, in the editor, that says I want to do Python. And it does syntax checking, has the nice lines that show you where your tabs and indents are, and so forth.

But you can also use PyCharm. So prior to 2017, you had to use some other tool. So you could use Notepad, whatever you wanted to because it's a scripting environment. But PyCharm I would highly recommend because it's designed for Python, and it also has a debugger. So that can help you. It will work with 2017 as well, but as we'll talk about a little bit later, there's a problem with the debugger in 2017.

So as far as adding extra functionality, it's the standard Python pip mechanism. So with 2017, we are now providing a custom Python distribution. So it's no longer a standard Python.exe, standard C Python distribution. So that makes it a little bit trickier to install, but I'll give you the instructions on how to do that later. With 2016 and previous versions, you just use the standard Python tools to install any extra libraries.

The main thing to keep in mind is versioning. If you've installed a library and it doesn't seem to be loading or working for you, you may have installed a version that's not compatible with the version that Max has. So currently in 2017, we're using 2.7.6. And this is documented. Prior to 2017 it was 2.7.3. So you just have to keep that in mind. It can get a little bit confusing, especially when the difference in the version is on the third number.

So another piece that's interesting, that you may not ever use, but it's good to know, this one time inspection concept. So .NET has a facility called reflection. And .NET in itself is a pretty strongly typed language. Even though you can do things like var equals and have those morph into other types, that is managed underneath by the framework, where every API has to return

a specific type. And you can cast, similar to in C++, to types that are in that hierarchy.

With Python it's called introspection. And this, because of the more loosely typed nature of Python, is not as interesting. So when you query for an API there, you may not get back the exact information that you're looking for.

Now in both cases what this allows you to do is, at runtime, you can ask for information about an API that you don't know about. So it's good for writing test harnesses, for example, because you can run this against, let's say in .NET, against an assembly, and you can query for everything without having any documentation or knowing anything about that assembly.

So we use this a lot internally to write our test harnesses for our APIs. But you might find this interesting if you maybe are using a third party API, and not everything is well documented. This concept could help you out.

Now the tools that you will use, especially in .NET, actually use this concept. So if you use the Object Browser in Visual Studio, it uses reflection to be able to list all those APIs. That same idea doesn't exist in C++, for example. So that's a huge benefit with .NET is you can basically open an assembly and see everything that's available in that API.

So the evolution of these tools, just to give you a little bit of history-- and this is why I say these are relatively new environment. So we're going back to 2012 for the first complete .NET API. But if you look at 2012 compared to 2014, they came relatively at the same time in the legacy history of 3ds Max.

So the .NET API kind of came in chunks. So they started writing .NET utilities to help them with different parts of their UI, especially. So the ribbon, for example, is a managed .NET tool. So they kind of started writing them for their own purposes. And then they realized, well this is kind of useful. So then they exposed it to MAXScript so that you could actually load DLLs through MAXScript and access other people's .NET utilities.

And then they said, well, we need a .NET API. So they went out and-- there's a company that was producing the State Sets that we acquired. Their name is Ephere, E-P-H-E-R-E. And they had their own .NET API that they consumed for their feature. And that piece came with it.

So the interesting thing about this is this assembly, Autodesk.Max, is what we call autogenerated. So there's a tool that looks at the C++ SDK and builds the .NET API wrappers

from that. So it's kind of an efficient way to build an API, but it's not the best, from a user standpoint. So it's not very user friendly. And we can run into problems.

So one of the biggest ones is this IntPtr problem. So an IntPtr is a type in .NET. IntPtrs are a wrapper around a native C++ pointer. So because we're wrapping a C++ API, if we can't map that C++ type to a managed type, it defaults to IntPtr. And there's not a lot you can do with an IntPtr by itself. So for example, if you see an API that is-- maybe it's a function that one of the parameter types is IntPtr, inside that function, you may not know what the IntPtr stands for.

So the first thing to do is look for what we call marshaller code. And that is basically a class that will take the pointer and convert it back to a managed type. So node is a pretty good example of this situation. In some cases, node doesn't get mapped properly, and you can use the marshaller code to convert that IntPtr to a proper managed node.

The other drawback here is that the docs-- we have much fewer docs. So it's actually part of the C++ SDK. And let me just click here to show you real quick how this works.

So the latest Max help has now got this really nice table of contents. The SDK documentation is in the 3ds Max developer help section. And in that developer help, there is a .NET SDK section. So this gives you information kind of at the high level of how to work with .NET and use it in 3ds Max, but it doesn't have an API reference.

So the reason for that is because we are wrapping the C++ SDK. They decided not to put a lot of effort into specialized documentation. So what you really have to do if you want to learn about a particular API is go to the C++ reference and put your C++ hat on in terms of what it looks like. The naming conventions are the same. But you'll look at it and you can read the documentation and how it's supposed to behave that way.

Now, we go back to the Python side. So this is more recent. So this first showed up in the 2014 extension release. And at that time it was definitely not complete. So they started with this MaxPlus API. A good example of where it wasn't complete was they exposed the standard TriMesh, but they didn't expose MNMesh, which is the quad mesh or polygon mesh. So you could do things with tri objects, but you couldn't do things with poly objects. So it was a huge limitation, and people complained immediately. So that's the case of where, over here, having a full wrapped API can be more useful because it's there, as opposed to being more nicely formed but not complete.

So in 2017, we made a significant change. So one of the things we did was we wanted to make the Python API more complete. And without creating new APIs for every single function, we decided to expose the MAXScript API through Python. So now you basically have a complete API, but you have to think in two places. So the first MaxPlus API is a C++ oriented wrapper. And the rest of it is a MAXScript runtime wrapper.

And this did get some special documentation. So the nice thing here, so when you go into the Python help, it's actually a top level section in the documentation. So it's got its own section. It's got its own API reference, much better formed documentation.

So let's talk a little more specifically about .NET. So why would you use this one? Why would you choose this one over Python, for example, or over C++? First of all, it's a very mature environment. It's in other products. Does anyone use any of our design products as well, Revit, AutoCAD? So .NET becomes kind of a natural choice because those products also have .NET APIs. So you're already probably familiar with the .NET environment. So use the same environment, just learn the API.

The other great thing is that .NET has been on the market for a long, long time. Microsoft is continuing to support it very well. They use it as well. So it's a very good, highly mature environment, lots of community people, lots of network or internet resources. So if you get stuck, you don't have to rely just on our help, but you can go and find community help.

So what can you do? You can automate tasks. So you can interact with MAXScript. So this is an interesting concept is if you want to expose, for example, some functionality out of .NET to MAXScript, you can do so. So you could write some of the maybe more performance-oriented algorithms in .NET where you're getting kind of that C++ level of performance, export those functions, and maybe continue to build your UI or have some other basic scripting things going on to call your .NET functions when they're needed. So that's definitely possible.

If you're a C++ programmer and you have a plug-in already, you can also use C++ CLI to provide .NET APIs. So if you, for example, might be a commercial developer and you wanted to provide your customers with a .NET API, this is the facility to do that. So it's kind of a heavy area to learn, but if you're already in C++, it's not not too bad.

And from a UI perspective, so we definitely support the older Windows Forms. But the newer UI mechanism is Windows Presentation Foundation, WPF. This uses a combination of .NET code and what they call XAML code. And you basically build these partial classes that bind

together at runtime. And you can build some really nice UIs. The Visual Studio tools provides WYSIWYG, so you can basically drag and drop controls and get some really nice behavior there.

And you can also use other .NET API components. So features of the language, for example, Linq is a really powerful facility. It's a simple idea, but it basically allows you to do database-style queries from runtime data structures. So let's say, for example, you've iterated the scene and you have a list of the nodes in the scene, and you want to select all the lights or all the, I don't know, the cameras out of that list. Linq will give you the ability to do that in one line of code. So it's basically a query. You say something like, for all the lights in this list, give me a collection back. And it's as simple as that. You don't have to iterate the list. It's really powerful.

So to set up for .NET, you're going to create a new class library. So this is called a Windows class library. It basically produces a DLL assembly.

Typically people use C#. But the nice thing about .NET is that it is language agnostic. So if you prefer VB.NET, for example, you can use it.

Know that C# is probably the most dominant language used in this environment. All of our code samples are written in C#. So if you haven't started and you don't have a preference, I would highly encourage you to consider C# as your language. Very useful, nice syntax, it's a good language.

So once you have that class library, you then simply reference our assemblies. So we'll talk about the assemblies next.

So the most important assembly is this Autodesk.Max DLL. So you'll reference that probably in all cases. This is what provides you with that full functionality of the Max API.

These other ones are useful. So don't exclude those. But know which ones are useful and which ones are not. When you look at our samples, you'll see a mixture of this one, CSharpUtilities. And the ManagedServices is also one that you'll see.

And then this one is really important for creating an action item. The name itself is kind of a weird name. And it was for another project. That's why they created it. But it has a utility in there that creates these user actions that register themselves back with that CUI system so you can drag and drop them on the menus and toolbars.

So how is your code loaded? .NET is a little bit restrictive in this capacity. You have to place them in this folder. That is your main option. So the problem with this is that it's part of the install location, and so there's the Windows security that you have to deal with. So every time you put something there, it's going to ask you, really, do you want to do that-- yes. Are you an admin-- yes. Are you sure-- yes. And then it will go.

For that reason, because I'm primarily developing all the time, I usually put my install location outside of Program Files. So that's just a trick. If you're not just a daily user and all you're doing is kind of writing code, that might be an option, because you get away from that security then. It's not a Program Files issue anymore.

And this location basically is there already. So you don't have to create it. We have some components in there. And you just simply copy your DLL into that location and it will load automatically when Max starts up.

We also have the app store folder that you can use. And that's another alternative. But it requires a special XML package to define what that assembly is doing and where it's at and which version of Max it can run with. So it's definitely doable, but there's a little bit of more work to do to get it to--

AUDIENCE: Is that similar to [INAUDIBLE]?

KEVIN VANDECAR: Yeah, it's exactly the same. Max has a few tags that are a little bit fussier than AutoCAD, but it's the same idea. So you just create that XML, put your assembly in that folder structure, and it will autoload then. The sample code that I'll show later, the Explode Geometry, has a bundle, so there's a great example of a .NET assembly going in that location.

So once the assembly gets loaded, how do you get your code to actually execute? So there's two fundamental ways. The first is the CuiActionCommandAdapter-- terrible name. I don't why they called it that. But you can see it's at least CUI and action. So those are your main clues that this is an important class.

So this allows you to easily hook into the UI using that action system. And it simply asks you to implement a derived class with a few strings that show up in the menuing system, or the UI system. And that functionality is provided in this assembly. So you need to reference this assembly just for this functionality, at a minimum.

The alternative is the Assembly Loader. So this allows you to do something similar to the global utility plug-in. It basically will allow you to execute code when 3ds Max starts, and execute code when 3ds Max shuts down. That's its primary function. It also has a loader facility. So if you want the majority of your assemblies to be located somewhere else, anywhere else, you can actually use this. The main class in this assembly will allow you to load from other locations as well. But this one has to be in that bin assemblies folder in order for it to load at start. And then whatever code you have in here can manage loading the other assemblies.

The limitation to that is that the loader here will not initialize this. So you don't get the CUI action behavior if you load your assembly through the Assembly Loader facility. So it makes it kind of useless.

I have seen people use it though. They tend to use the Assembly Loader mechanism and execute code on startup to set up their scene. And so then they have a whole bunch of these assemblies that do different things, and they load the one that's appropriate.

Again, interop-- so by default, any classes and functions that are defined as public can be exported to the MAXScript environment. The drawback here is that only primitive types are shared, so things like ints, floats, doubles, types that are built into the language. A node, for example, you cannot return a node to the MAXScript environment.

But there are ways around it. So node is the most popular question. I need to get my node passed back and forth. So instead of returning the node, return the node's anim handle, which is an integer, and then you just convert it back to a node on one side or the other. So that allows you to pass the node identifier back and forth, and then work with the node on either side. And then again the, C++ CLI. So there's good help topics in both of those cases.

Now to debug, there's two ways. So to use the direct debugger, the first thing you have to do is you have to point the debugger at 3dsmax.exe. So you can't just debug a DLL without having a process to drive it.

So you point the debugger at 3dsmax.exe. And then you press debug. And then any of your breakpoints will be hit once Max loads your assembly and starts executing your code.

In order to use it in Visual Studio 2015, you have to set this specialized flag in the debugger within Visual Studio. This is because Visual Studio came out with a better debugger. And if

you're just working in the managed world, it is better. But when you have-- [COUGH]. Excuse me-- a mixed mode application like Max-- Max is written in native code and it has this managed layer-- you have to check this in order to get the legacy debugger. So if you miss checking this, the behavior you'll see is that as soon as you start the debugger, you'll get the [INAUDIBLE] dialog. So you'll know oh, I've got to remember what Kevin told me.

You can also attach to process. And this does not require you to change that flag. This is a little bit more effort to get your debugger started. So you will start 3ds Max. You'll be in a fully running instance. And then you'll go back to your debugger and say attached to the 3dsmax.exe process. And you'll want to use the managed native code option, because again, Max is a mixed mode application. And that sets the debugger to the right mode to debug.

So samples are a little bit limited. We don't have any samples that come with the .NET SDK. So we've provided a few from my group, from the ADN group. So first of all, this Explode Geometry, it is available as a full app. So you can go to the app store and you can just download it and try it and it's there. And then the full code is provided in a GitHub repo here. And this is using the CUI facility.

So let me just show this as well. So I've already started 3ds Max under the debugger. So you can see Max is actually running here. Let me see here, let me start this.

OK, so this is the dialog that accompanies the plug-in. This was written with WPF in mind. I can basically have a node selected in the scene. When it comes up, it's going to check to see if a node is selected. And if not, it'll warn you, say we need a node. And then the dialog displays. And here's where you can select different options.

So I'm just going to go ahead and run this. And you'll see that I have the debugger active, and it just immediately steps into my code. So this is a very nice environment for doing detailed applications because you have a nice debugging tool that's easy to set up. Python-- I'll show you that in the next section-- is a little more difficult to get this to work in and be useful.

So the first thing you're going to see is the entry point into the API. So this code right here, you'll see is using our Autodesk.Max. So that's the namespace from the Autodesk.Max assembly. And what I need to do is get the global interface for the API, and get the running instance of that. So this gives me that global instance of the highest level entry point into the API.

Once I get that I can drill down. And Max has this idea called the Core interface. That's why my blog is named *GetCOREInterface* because that is really the entry point into a lot of the things you might want to do with the editor, geometry, et cetera. So usually you'll have these two lines of code to start any type of programming in the Autodesk.Max interface.

And you'll see here, this is a function that is exported to MAXScript. So it's receiving an integer. It's an unsigned integer. And it's the node handle. So the first thing I do is I convert that node handle back to a node. So you can kind of see how you can pass identifiers back and forth, but you have that limitation where you can only use the primitive types.

And then as we drill down through here, you can see there's a lot of code that does specific things to the geometry. And in the end what it's going to do-- let me just go ahead and run this-- it takes whatever you've selected, and it iterates the entire mesh, looks at every face in that mesh, and builds a new face, and then applies-- oops, let me select one again-- and then applies these various options to that individual face. So it shows a lot of basic ideas that are useful in the API-- iterating a mesh, converting that to something else, building new geometry, the faces, adding a modifier, collapsing the modifier stack, so things that you might want to automate. So hopefully that's a pretty good sample.

It's pretty detailed and we've we've had quite a few downloads of it. So as a utility, I think it's useful in itself. So feel free to go get it.

The other samples here are just showing various UI aspects. So I'm going to just demo this particular one. So first of all, I want to show the CuiActionCommandAdapter. So let me go back up here. So you can see what I've done is I've created an abstract class off of that command adapter. And I've implemented some of the things that you would repeat each time you create a new command. So it's just kind of a base class to provide a common category for that CUI dialog.

And then from there, I implement my specific command. In this case, this is the command text. And this is what-- let me just go ahead and run this. This is what you see up here, WPF viewport sample.

All right, so this particular sample is an interesting one. It's not well-formed. If you have a lot of things in the scene, you're going to get this big long list. But it shows the power of WPF. In this particular case, it's got a transparent dialog holding the controls. So you don't see the outline of the dialog itself. And then the controls have a transparent characteristic applied to it so you

can see through the controls to the viewport.

I've also implemented listeners, so using the event system, to know when the active viewport has changed. And then I move that dialog into the viewport that was selected. So this shows some UI behavioral things that you might be interested in.

It also uses what I mentioned earlier, the Linq facility to subset out of a big list. What I can do is, you can see here, this is the scene list of all the nodes, and there's a lot of faces now. If I wanted to list only the lights, there's one line of code that actually does that work to subset the lights out of that big list of nodes based on its type.

So the sample code shows how to work with that as well. And it also shows doing scene selection through a dialog. So by name, I can say I want to select those nodes. And using the Core interface API, I can actually make those nodes selected.

So before we move on to Python, any questions with .NET so far? Yeah?

AUDIENCE: Is both 3ds Max from the [INAUDIBLE], or do we need to add [INAUDIBLE]?

KEVIN VANDECAR: I didn't catch the first part of the question.

AUDIENCE: Can you basically do a headless [INAUDIBLE], like headless, you don't want the UI from Max at all? You want to just [INAUDIBLE] the Max [INAUDIBLE]?

KEVIN VANDECAR: So Max itself has the ability to run headless. So using the assembly loader where you can execute code on startup, you could definitely do that.

AUDIENCE: [INAUDIBLE].

KEVIN VANDECAR: Yeah. There's no extra .NET functionality though that tells Max at runtime to go headless or switch modes or anything. But certainly at startup, using Assembly Loader, you could execute some code and drive it that way. Was there another question over here? OK, all right.

So just a few tips and tricks before we move on. One of the big things in .NET is that there is this idea of a Global Assembly Cache. I call it the GAC, which is a nice word to say, GAC. Oh, you can't see it very well.

So over here you see, for example, there's the CSharpUtilities assembly. And in its properties,

there's this flag called copy-local. Now whenever you reference an assembly from disk, like you have to do for 3ds Max or AutoCAD or whatever-- it's the same ideas as in AutoCAD and Revit-- this flag, by default, is set to true. The reason they do that is, they make the assumption that when you build an app, whatever assembly you're referencing needs to be copied to the project folder of that app so that there's never a version change. You're always running with that assembly that you started with.

But that doesn't work well for mixed mode applications like 3ds Max or AutoCAD. So you need to set that to false anytime. Just get in the habit of remembering, reference assembly copy-local false. Because otherwise what will happen is, you'll be copying an assembly. And then when you move to the next version of Max and you update your project, it won't get the next version of the assembly. And you'll have the wrong assembly. And now it will crash Max when you're running. So you want to reference this directly from the Max EXE folder, and you want that reference to stay for Max EXE folder. So when you move forward, you'll get the new DLL automatically. And that helps you to manage your projects going forward. So just remember that. That's an important one.

If you actually build your code to that folder, it can cause even bigger problems, because it copies also all the dependent modules along with it. And it can pollute that bin assemblies folder. So this is really an important thing to keep in mind.

Max has some environment variables too, if you're not aware. So I encourage you to get used to using those. They all start with ADSK. And for example, for 2017, there's the environment variable. So you can use this environment variable in your projects to point to your assemblies. And then when you upgrade your project, all you have to do is change the environment variable to reference the new assemblies. You don't have to go through the UI and unreference and rereference. You just edit the Visual Studio C# project file and change your environment variable there.

So think about adding a post build event to copy your assembly rather than just build to that location. So build it locally and then add a post build to copy it to the bin assemblies folder. And that way you eliminate some of that copy local issue. And then, like I showed, create an abstract version of the CuiActionCommandAdapter that has the base stuff filled in, and then you only have to implement the specific command name in the following command.

Use reflector tools. So Object Browser is a type of reflector tool. But there are some third party

ones as well that are really useful. ReSharper is one that's from the same people who make PyCharm.

Make sure you error check. So you'll see in the sample code, it's not perfect, but every single API call is at least wrapped in a try catch block. OK so that's important to trap any of the exceptions that might occur so that they don't get passed to the Max EXE and cause it to crash. So if you capture them in the ManagedProcess within your code, you can handle it, and then Max doesn't have to freak out and die.

And make sure you use all the assemblies. Don't just think that Autodesk.Max is the complete deal. You'll definitely mix assemblies.

And the last thing is something we talked about before the class started is, you want to be careful to not derive from this Max plug-ins type. In theory you could create a full plug-in in the C# world, but there's a lot of problems with it. And we're not supporting it. So we've even documented, it's not advised to do this.

And last one is, the Python MaxPlus API actually has a .NET counterpart. So you'll find it maybe if you're poking around and say, wow, there's some useful APIs here too. So I like to use them, but be aware that we are not officially supporting that particular assembly for .NET, because we might remove that .NET aspect of it at some point in the future. So it's useful for the version you find it in. Just know that ahead of time if you're going to use it. Don't expect it to be maintained going forward.

All right, so let's switch over to Python. All right, so Python, same kind of thing. It's very mature. The thing with Python that I like to think of is that it's kind of the M&E choice. There's Maya, MotionBuilder, FBX SDK all have Python interfaces. So if you're interfacing with those tools already, then Python's a logical choice.

It has a very large community. If you're a person who writes in Python, usually you love Python. You dream about Python. You talk to your wife about Python. You get divorced over Python. So it's just a very, very powerful community out there. And it is open source as a technology as well. And that's why a lot of companies have chosen Python, because for free you can go and get a language environment and put it in your product. So that's just one aspect that makes it really, really popular. And there are tons of third party libraries.

Now the thing you have to be careful with here is, is that third party library-- does it have a

compatible module that works with the 3ds Max version of Python? So you have to be careful. Before you get really excited, try to install the library and make sure you can import its namespace. That will tell you if you can use it or not.

And as opposed to MAXScript, Python is open, right? If you learn Python as a language, you can use that skill other places. And that's the main advantage over MAXScript. It's very closed. And even though the syntax is very simple, it's proprietary. So it's not a skill that you can transfer easily with MAXScript.

So what can you do-- automate tasks. So this is a scripting environment. So you're basically going to script tasks that you would be repeating over and over again, like you might create a MAXScript for. Python can do the same thing.

You can certainly interop with MAXScript. So if you have a ton of MAXScripts already and you want to switch to Python, it's absolutely no problem. You can start writing new, call some of your MAXScript, and your MAXScripts can call your Python.

Other Python pipelines-- so the whole Python file system library works. All of that stuff is compatible. So it's very easy to connect to any other Python tools you have.

Python uses the Qt environment for its UI elements. And like WPF, it's a very powerful windowing system or user interface system. The challenge with Qt is there's no real easy the use WYSIWYG editor. So there's a plug-in for Visual Studio that doesn't work with 2015 and there's some standalone tools and that sort of thing. But there's nothing really integrated in the environment. So you kind of have to find your own way in terms of building your Qt interface.

You can use other Python libraries. So the PySide, which is a version of Qt, is included. So we give you Qt interface tools as part of our standard Python distribution. And of course you can install your own libraries. One of the samples is using NumPy and PyQtGraph to show some sophisticated things with Python.

So with 2017, we now have this built in scripting editor. So with the MAXScript editor, you can now switch to Python very easily. And the MAXScript listener now has a Python command line. So let me just show you those two features really quick here.

So you can see right away at the top-- and this is 2017 only. So it's new to 2017. 2016 and previous versions do not have this. But you can just simply switch to Python. And now you've got a console. So this is a traditional Python console. You can type code in there and it will

execute.

So the editor-- so to create a new Python script, you would just go to Language. And so you'd say New, Language, and tell it that it's Python. And once you switch the Python tool on, you get syntax checking as well. So you can see how, like in MAXScript, you'll see your indentation lines. You get syntax coloring. It's really adapted to Python well. So you can very easily write your Python scripts here, and you can-- excuse me, actually you can evaluate the Python script directly from here as well. So if you're used to using this environment for MAXScript, it's now pretty much the same for Python.

So the functionality for Python is in two distinct modules. So the MaxPlus module, which as I said before, is exposing some parts of the C++ functionality-- in a better way, I would, say then with the .NET API. So they actually take some care to transmit or translate that C++ API into something more pythonic. And then new to 2017 is the pymsx module. This module is exposing the MAXScript runtime.

OK, how is your code loaded? So Max looks in the following directories. So there's a User Scripts directory and a User Startup Scripts directory. So this is probably one that you would use the most often as an individual. You put it in your user profile area and you don't have to worry about mucking around with the Program Files location. But you can also put them in the Program Files directories.

And then as a last resort, when you try to execute a Python file, if it can't find it in these places, it will actually search your path environment as well. So if you try to load a file and it takes a while, takes a while, takes a while, takes a while, and then it says it can't find it, well it's probably because it also searched however big your path is. And that could be huge. And then finally it will also look in the app store folder. Again, you have to have a package. And you actually use MAXScript to get the Python to execute from the app store folder. But it's just like a one liner, Python file execute and go.

So the first thing to remember is that Python is very integrated with MAXScript. So you execute and you work with Python through the MAXScript interface. And that's a big difference between .NET where it's kind of independent. You can work with it without touching MAXScript at all. So with Python you kind of have to understand a little bit about MAXScript. Even though all your scripts might be written in Python, you'll have to know just enough to get them to execute and how they load.

So first of all, in MAXScript, there's a Python object. And it has different functions. But for example, executeFile is the most popular. And you just point it at your Python script.

Now earlier we talked about the search folder structure. If you provide a full path here, it will find it as well. So you don't even need to put your scripts in those search places. You can tell it exactly where they are. So if you have a library that you want to keep independent of the Max install, whatever, you can do that as well.

You can also execute from the command line. So the question about .NET being able to run code on start up, this is how you can do it from Python. So you can pass a script in. You just use the -U, tell it that it's going to be the PythonHost running it. This could alternatively be MAXScript of course. But then you tell it where your Python script is. It will start up and it will run that script.

If you want Max to kind of run in an automated fashion, in this mechanism you actually have to tell Max to exit as well at the end of the script load, or run. So you might have a separate script that you call with that code to shut Max back down.

If you want to hook it into the CUI, a macro script is a really handy thing. So this is kind of a not a MAXScript, not a Python thing, it's a special other thing that allows you to create an action item. So you just simply create this macro script. It's saved as an MCR file. This idea is well documented in the Customize User Interface section of the docs.

But here, this is the category, like we saw in the .NET example. And then the text that will show up in the menu is actually the buttonText here. You can also specify a tooltip. And here you just tell it, through MAXScript, what to do. So fileExecute and my Python script. You have to place this MCR in a specific folder. But when Max starts up, this category and command will be part of the CUI, and you can put it on a menu or toolbar just like in the .NET example.

So like I said earlier, there is a Python console. That's kind of a nice way to try things. You can just type your Python in there as you're writing your script and see what executes well and what doesn't.

So, next let's just consider why there are two different modules. So first of all, it was about completeness. So they started down this path of MaxPlus, which is a wrapper over C++ SDK. They use a different tool called SWIG to produce this. So a similar idea of autogeneration, but it has much better configuration options to build better formed APIs.

So it really does expose the C++ low level side. So for example, here you're also going to use things like the Core interface, class IDs to identify things, just like in .NET. And this is also the module that integrates the PySide UI into 3ds Max. So even if you choose to use the other module for most of your programming, you'll need this module just to tie whatever UI you have into the Max window.

So the pymsx, again, remember this is only in 2017 and going forward. This basically exploits the entire MAXScript runtime. So anything that you can do in MAXScript, you can now do in Python, using this new library. And it's a two way street. So the interop is really good going back and forth.

So again, you can execute these things from each other. So going back to anyone with legacy MAXScript that wants to move to Python, it's very easy to mix these things together. MAXScript can also import and use Python libraries. So if you prefer MAXScript but you want some Python functionality, you can keep using MAXScript. And now you can bring those Python modules in-- for example, NumPy, some of those other utility type libraries, bring them in and use them in MAXScript, no problem.

And then data can be easily shared between MAXScript and Python. So that issue of primitive types in .NET being the only thing you can pass back and forth, because these are so tightly integrated, all of the types are just, by default, available in both environments. So you don't have to worry about that transfer of information between the two.

So debugging-- so this can be done with PyCharm. There are other IDEs out there that might be useful. But PyCharm seemed to be the one that worked the best when we explored this a few years ago. The community edition of PyCharm is free. But unfortunately in order to get the remote debugging capability, you have to buy the full version.

So if you're interested in this, I would encourage you to go get their trial version of the full version. I think it's a 30 day trial. So play with it. See if you really want it before you spend the money to get it. It is useful. In 2016, you can debug your Python scripts very easily. You'll step in just like I showed you with the .NET side.

There's a little bit of setup. You have to basically create a couple of helper Python scripts to help PyCharm to know where to find the Max process. But we provide those scripts. Actually it was a third party developer who figured that out, and he let us redistribute those.

So like I said, it's pretty easy in 2016 to get this set up. But unfortunately, at the moment, we can't get it to work in 2017. So we're having some trouble there. If you need debugging, you have to be careful about the 2017 environment. As soon as we figure it out, if we figure it out, we will post a setup on the blog, on the *GetCOREInterface* blog. So you can check there. Feel free to email me as well if you don't see any news and we can try and escalate it to engineering.

So samples-- so first of all, the nice thing here is that there's a bunch of samples that come with 3ds Max. They're in the scripts folder under the Python subfolder. We've also reimplemented the Explode Geometry as a Python script. So the cool thing there is you can kind of look at the .NET version and look at the Python version, and you can get a pretty good idea of some of the differences between the two environments just by comparing the code. And then last we've got this Energy Monitor tool, which is kind of interesting.

So let me just show you these real quick. So I've got the Explode Geometry code here. And you'll see that this example is importing both pymsx, so the new library for 2017, and MaxPlus. So this sample is only going to run in 2017 going forward.

From a UI perspective, because you don't have that nice WYSIWYG facility out of the box, you tend to just code it yourself. I think that's what a lot of people do. And you'll see here, this is the actual setup of the UI. So it's kind of a lot of pretty dry code. But this shows you what you kind of need to do in code to get a reasonable UI set up using Qt.

So this is where we actually launch the form. And then this do_explode is essentially the mirror of the code from the .NET. That iterates the scene-- iterates the mesh, converts those mesh faces to individual faces, adds the modifier, and does basically the same thing.

So let me go ahead and execute this. I'll say here, Evaluate All. So it's executing my Python script now. And I'm going to select that sphere. Yeah, that should be fine. And my colleague Dennis wrote this, and he chose to do the opposite that I did. So I have to go through on his example and select all of the suboptions, where mine were defaulted. But it's the same idea. Select those options, and then you click explode. And it will iterate through that mesh and create the faces. So you see there, it was done. So basically the same utility implemented in Python.

So the other one is kind of an interesting one. And this just kind of shows some of the cool UI

stuff that Qt provides. So this is a scene that he created that is basically kind of mimicking one of these new devices that people are experimenting with where you float a grid in the water, or you set a grid in the water, and these bobber go up and down based on the current, and they generate electricity. Not much electricity, but if you have enough of them, if you covered the ocean, we might be OK. So it uses that. The scene is kind of set up to animate that idea.

So let me run this. And you'll see I've also got it on my menu. So this is implemented through using one of those MCR macro scripts to get it up here. So I'll launch that. There we go.

So right off the bat you'll see it's got a graph at the top. Now that graph comes from that PyQtGraph module. It's not something we provide. You can go and install that library and then use it. So this is showing a good example of a third party library being used.

And then I basically select on one of those beacons. So when the Python loads, it actually iterates the scene and finds all of those beacons that are identified by name. So it's kind of the same idea as the other sample iterating the scene and finding all the lights, so a similar idea.

And in this case you select on any of the beacons and you'll get a graph for the energy that is being produced at a particular point in time. So if I run this through, you can see the little circle running along the graph to show the energy at that particular point in time. So kind of a cool way to generate a UI and use a third party library.

Now the nice thing here too is that with Qt, it docks very easily. You don't have to do a lot of work to get that panel to actually dock in the Max UI. It uses kind of the standard docking and floating mechanism that the Max UI does.

All right, so tips and tricks on the Python side. One big one is, just be careful when you get started to not pollute the global namespace. So it's very easy to say import a library, and then you basically just import that entire library and all the namespaces within it. That can consume actually quite a bit of memory. So keep that in mind. It's better to say from that library, import a specific namespace. So kind of look at the documentation for the library, see where the functionality is that you want to use, and import just that namespace. If you need another one, go grab the other one. And that will keep the memory consumption lower.

Also, because Python scripts can also be libraries-- and there's really no way to kind of tell the difference just from the filename-- get in the habit of using this __main__ idiom. This is kind of a standard Python thing. So just be careful to use that. Because if you don't and somebody

else loads your script, it may execute right there on start, or it may be considered a library depending on how you've written your code.

And also, don't get stuck in Python. So if you are writing Python code and for some reason you run into a roadblock, remember that MAXScript is there. So if you like to use the MaxPlus, well, switch over and use the pymsx module and pull in that MAXScript functionality to complete your job.

And .NET and C++, of course, are also available. So if you just run into something completely where you're stuck-- for example, maybe you want to use a very specialized native library that can only be consumed in C++. Well, there you go, you've got to switch to C++.

Installing packages-- so again, the versioning is very important here. So Python doesn't really care which version you're installing to. It'll just install it. So you have to make sure you get the version that matches the version you're running from. And in 2017, we now have this specialized Python executable. So prior to 2017, again, it was the standard CPython distribution. So Python was being run by Python.exe. Now it's being run by the special version called 3dmaxpy.exe. And that's what controls the 2017 Python environment.

So in order to install a module here, you have to use this executable, the -m. And you do that from the max install folder, and then specify the standard pip instructions to get the module. So you'll see an example here, 3dsmaxpy -m pip install and these other options. And right there you see the version is really important. So it's the version.

And notice that this doesn't match 2.7.6. So you have to look at this utility to see if it's compatible with 2.7.6. They'll say, oh yeah, the 1.10.4 version is compatible with 2.7.6. So gets a little bit weird to think about, but you just have to be careful. If something doesn't run or import right away, it's probably because the version is mixed up.

Also, run these commands from an admin command line. That's a way to avoid these security issues, and it will help to have a successful install.

So any questions on Python before I kind of wrap up? Yeah?

AUDIENCE: If you want to distribute your script and you need to [INAUDIBLE], how do you go about that? Do you this command lines?

KEVIN Yeah, so Python does have a package manager kind of facility. So you can actually write a

VANDECAR: script to do the installs. If it's going to be a commercial app, you definitely would want to do that. If you're just going to share it within your office, you might go and help them to get it installed, because it can be tricky with the admin stuff, et cetera. So just depending on how much time you want to spend setting it up and getting it to work. So as a commercial tool or widely distributed tool, yes, you could use their package manager and do that as kind of an install of your script routines. Any other? Yeah?

AUDIENCE: [INAUDIBLE] performance differences to see between MAXScript, Python, and [INAUDIBLE]?

KEVIN VANDECAR: So MAXScript and Python are about the same. I don't know of any differences there. They're basically using the same thing. Python, actually executing some of the MaxPlus APIs is probably faster than doing the equivalent with MAXScript APIs. But that would be the only minor difference.

So there, I would think about using the MaxPlus APIs if you have an algorithm that is taking a fair amount of time, working with geometry or whatever. Consider using MaxPlus instead of the MAXScript API.

As far as .NET goes, it's a performance win over both environments because it's not a scripting environment. It's actually executed. We usually refer to it as a thin wrapper around the SDK. So it works pretty much as fast as the C++ SDK does.

A good example is we had one of our commercial developers had written a lot of their functionality in MAXScript, and they had a very complex process that they needed to run. And we suggested-- they didn't want to go to full C++, so we suggested, well, maybe just implement that process in .NET. And they got like a threefold performance increase. And it was enough to kind of get them over that hurdle of being stuck. Any other questions, Python, .NET?

OK, so I just want to mention the App Store real quick, because as a consumer, there's a lot of great functionality there already. So don't forget that. If you're looking for a tool, go check out the App Store. A lot of things are there for free. And anything that does have a fee is usually pretty low cost.

And to give you an example, the Explode Geometry plug-in that I just showed you is available from the App Store. So if you decided, oh, this is all too much, I don't want to do any programming, but that Explode Geometry looked pretty cool, you can just go get it, install it,

and it's done. But if you do want to work with the App Store, it's also a great sample to figure out how that works.

And now as a publisher, so you get started down the Python or .NET route and you think, man, these utilities that I've written are awesome. And maybe I'd like to start sharing them. So the great thing about the app store is, first of all, we don't take any piece of the pie. We just provide the App Store as a facility to share utilities, or plug-ins, whatever. You can post it. You can publish it there for-- gosh darn it, I'm sorry-- a free trial or for fee. So if you have a set of utilities that you think are useful and you want to sell them, there's no reason not to consider doing this.

The monetary aspect of it is handled through either PayPal or Blue Streak. And you just simply need an account in one of those environments. And then the user will pay you directly. We just facilitate that paying. So again, we don't take any part of that.

There is an entitlement API. So if you're worried about proprietary algorithms or whatever, you can use that. But it's really only useful from C++ because most of the other environments, even if you encrypt them, can be decrypted if someone wants it bad enough. So it's not worth wasting time in those environments.

And resources and support-- so I've included all these things in the handout. The handout actually has more step by step instructions as well. So if you want to get started in .NET, there are some steps in there that walk you through specifically how to create an assembly, for example. So make sure you get the handout and use that as a place to get started. All of these resources are documented there.

The last thing I wanted to just mention is Autodesk Forge. So how many people who have heard of Forge? There's been quite a bit of talking about it. So to start, Forge is a cloud API stack. And 3ds Max is moving into that environment as well. So if you use 3ds Max today and you've ever thought to yourself, well how can I easily share my assets or demo some of my assets to my customers, that sort of thing-- the nice thing about the Forge environment is that it has a WebGL-based viewer. So you simply convert your geometry to this SVF format using the Forge translation service, and then you can point people and view the geometry online. And it's a pretty lightweight, nice way to share things without them having to install something special, that sort of thing. So you can provide pretty cool experiences that way.

If you're interested in Forge, we have a developer web site. We have a developer conference.

So the next one will take place in June of next year in Fort Mason in San Francisco. We also have a booth on the show floor this year, first time. So if you're interested in learning more, make sure you drop by.

And that's pretty much it. Any other questions, comments? Yeah?

AUDIENCE: When you say that the [INAUDIBLE] Max, with that translation in the Forge, do you mean that they would produce Max files?

KEVIN VANDECAR: Yeah, so the long term goal with our translation service is to be able to receive the file format, so in this case Max, and translate it to something else, and to take something else and translate back to Max. That's the long term vision. The first revision of this will be Max coming in, and then translate it to SVF so you can do the lightweight WebGL viewing. So that's our first phase.

AUDIENCE: So are you planning on that have an update condition, through Forge, you can create and [INAUDIBLE]?

KEVIN VANDECAR: So like the AutoCAD I/O and the Design Automation? So we are talking about that. And if you have a specific idea of how that could be used, I'd love to hear you, hear what you think. Because the engineering team, they're a little resistant to doing it because they're not sure what people will do with it. And my opinion is, put it out there and they'll figure out what to do with it. So it's the cart and the horse problem, always.

So yeah, feel free to contact me afterwards and let me know what you'd like to see there. We've definitely been trying to influence them into getting a Max I/O type thing going. Any other questions?

AUDIENCE: Forge-- this is the first word I've ever heard about Forge. Can you explain it just from layman's terms. Is that a P word that you can [INAUDIBLE] MAXScript working?

KEVIN VANDECAR: Well first of all, to go back to the beginning, so Forge is basically our brand name for cloud-based APIs. So what we're trying to do is provide APIs to the other services that we have for users. So for example, have you ever used the A360 viewer? So the Forge viewer is the same idea as that, except now you can wrap it into your own application and kind of take it outside that A360 experience into something more unique that you want to provide to your customers.

But underneath it's the same technology. So the API layer basically gives you access. We

have the Data Management API which has two and three-legged authentication. So as a developer, with two-legged, you can access your own data that's in the A360 system or from Box or some of the other cloud services. With the three-legged, it gives you the ability to ask the user to authenticate you to access their data. That whole process is kept private from you.

So as a developer, all you get back is a yes or a no. But we will post the dialog to the user asking them for their Autodesk ID information. And then it will say very clearly, your app is asking for read access to your customer's data. And they can say yes or no. And once you get authentication back, you can then go and access their data and do things with their data online.

So you might decide that your application needs a Max file, for example. And you're going to take their Inventor file, run it through a translation to get the Max file. You run some process. If we have a Max I/O, you could maybe postprocess that Max file and then transfer it back and give it to them as a Max file. So that's one example.

So Forge is really just a set of cloud APIs that kind of mirror our core technology that's in the cloud already. Is that-- and the viewer part is just-- so the viewer, it's based on WebGL, at the lowest level. We actually use a library called three.js, which is an open source WebGL library. And then we create our viewer on top of that. And we provide you with this kind of rich viewing experience online.

Any other questions? So was this type of class-- I know a few people left throughout. But was this type of class useful for you guys? Do you like to see this kind of background history, one versus the other kind of thing? In the future, is there anything you would like to see more in depth? Would you like to have maybe a .NET in-depth or a Python in-depth, just so I have an idea of what I can submit for next year?

AUDIENCE: I really want to see even more complex workflows and kind of common tasks, kind of [INAUDIBLE].

KEVIN VANDECAR: OK. So maybe with the Explode Geometry, as an example, I can walk through that code and just show the code in a little more detail and the techniques and-- OK.