AMOD KULKARNI: I think we can get started. So first of all thank you very much for showing interest in the topic and attending this session. My name is Amod Kulkarni. So I'm a developer working with AutoDesk for close to four years now. But I have been working the AutoDesk products, especially AutoCAD and AutoCad Vertical products as a developer for more than that. So before joining AutoDesk also, I have been working on these products.

So today, we'll be covering this API session related to AutoCAD architecture and MEP. And here's my colleague, Vinit.

VINIT SHUKLA: I'm Vinit Shukla. I have been working with AutoDesk for the last six years, mostly with AutoCAD and AutoCAD KCL MEP.

AMOD KULKARNI: OK. So I think the class summary as we can see-- so just to clarify some points, like what this class is about. So here we are going to discuss some fundamentals about the OMF API. I don't know how many of you are aware of this API that is available for AutoCAD Architecture and MEP, but we will be discussing that today. And we'll be discussing some of the differences between the ObjectARX-- I think this is C++ API, which is fairly popular for the customization of AutoCAD. So we'll be seeing what are the differences between ObjectARX and OMF.

And then we will have some live examples of how OMF can be used for customization of ACA-- that is, AutoCAD Architecture and MEP. So what this class is not about. So this-- in this class, we are not going to discuss any programming languages, even though the OMF API is actually a C++ API. But we will not get into the C++ part of it. So we will assume there is some understanding of the language or object oriented concepts.

We will not discuss all the public API which are available, either for ObjectARX or OMF, because it's too vast. It's not possible to cover everything which is available there. So we'll be-- we will be covering the part which is most relevant, or that can give you the overall idea of what OMF can do for AutoCAD Architecture and MEP. So that kind of an introduction to the general API that is available.

We will discuss and resolve-- we will not discuss and resolve the specific issues, because there might be some specific issues that somebody might be facing. We can discuss them offline, but we will not part of this particular session.

VINIT SHUKLA: [INAUDIBLE]

AMOD KULKARNI: Yeah. So as you can see, the prerequisite for this is knowledge of ACA and MEP, because I hope everyone here is somehow related to the products, ACA and MEP, AutoCAD Architecture and MEP. And the basic knowledge of programming or object-oriented concepts as it was put out. So basically this course is for the programmers, but if you have some understanding of the object oriented concept, then you can probably relate to what we are going to discuss in this session. But essentially, it will be a programmer oriented class. But eventually you will get to understand what that can lead to when it comes to the customization of the products.

So general class agenda. So we will distribute this session into two sections. So first is the fundamentals of ObjectARX API. Now why we are looking into ObjectARX API is simply because OMF is an extension of the ObjectSRX API, which are available for AutoCAD customization. So we will understand some of the concepts from ObjectARX API, like what are the different model types. You might have heard these different terminologies like ARX file, DBX file, or CRX file. So what are these different model types?

We will introduce some of the concepts which are very important, like in terms of how the data is stored in drawing. Like, as a user, when you add something to the drawing in the background, how the data is getting actually stored in the drawing database. So those are some of the concepts that we will touch upon. Then for the OMF part, we will be-- as I explained earlier, we will be looking into what are the differences between ObjectARX and OMF.

Now when we say differences, it's not ObjectARX versus OMF. It's not like a fight between the two, but how some of the things are simplified in OMF as compared to ObjectARX. So that is the point of discussion. Introduction to OMF concepts, like what are the different customization possibilities in OMF for ACA and MEP. And then we'll create our own OMF application to give you an idea like, how OMF can be actually used to introduce some of the-- your requirements into the product.

So key learning objectives. We hope, and Vinit and I will try our best to achieve these objectives at the end of the session. So we hope that we will get-- you will get familiar with the overall concept of product customization using API. So that's a very high level statement, that how API can be used for customization of the products. Then you will be able to understand the basics of OMF API, which are very specific to ACA and MEP. And then you will be able to

understand different customization possibilities in ACA and MEP, like what all you can do in product using OMF API.

And then as a next step, probably you will start thinking, how I can make use of these API for-- or simply find some of the workflows in the product that currently may take certain steps, or maybe complicated workflows that you can simplify by writing some customization for the product. So that's the overall objective. Now, let's get into the topic now. The first part, as we said, we will have some introduction to the ObjectSRX, which are the C++ API of AutoCAD.

So to begin with, let me ask this question. So how many of new-- how many of you know what is API? OK. I think most of us know. So, API is Application Programming Interface. So this is a very common term that we hear, and many of the products which are available around us, we have API available for them.

You take Microsoft products like MS Word, Excel, even Outlook, or any product that you name nowadays, we find that they have API to offer. And why they do this, because the functionality is growing day by day, and the requirements of the customers are not-- never ending, basically. So by offering the API, the third party developers can actually write customization on top of the product, and they can introduce the necessary requirements into the-- can work them into the product features. So that's the whole objective of the API.

And its nothing different for ObjectARX also. So AutoCAD, as we know, is having tons of functionality features in it. But to introduce your own customized workflows and features, you can use ObjectARX to customize the AutoCAD product. So that's the whole idea of API.

Now, AutoCAD Runtime Extension. So we hear this term called ARX. So what is this? So basically, it is AutoCAD Runtime eXtension. So that's where the origin of the term ARX comes. It's a DLL plug-in model.

Now what does that mean? So, DLL plug-in model is nothing but, AutoCAD is an executable application. It's EXE. Now, DLL is a Dynamic Link Library, which, you can consider it as a function library which you can load into the AutoCAD EXEs memory space. And executable AutoCAD can actually use some of the functionality from the DLL and that's how the whole entire model works. So there is executable, and there is a DLL. And they work together by exchanging some of the function calls. So that's why it is called as a DLL plug-in model.

It's a set of objects-- C++ object, ARX-- C++ libraries. So as I said, it's a function library.

Basically, you have implemented certain things in a DLL, and those functions you are calling from the AutoCAD executable. So that's how the whole mechanism works. But essentially, it's a framework. So typically, you will see some API which are only kind of a function library-- limited function library.

So they have a bunch of functions available, and it's nothing but a tool kit that is available to you. Very limited functionality, but so to say there is some API available. But ObjectARX is much, much more than that. It's actually a framework. You can relate it to let's say, .NET framework, if you are familiar with it.

So, .NET framework has many things to offer. It has a base class library, so a lot of API available. It has garbage collector component, which clears the memory when it is not in use. There is a memory management that is happening very transparently to the user. So so many things are happening behind the scene.

So ObjectARX is also similar to that. It's not just a bunch of functions that you can use. There are many other offerings which a developer can leverage. So that's why it's a framework and not just a toolkit. So you can have your own custom entities implemented. You can leverage the reactor mechanism, which we eventually will see in some later sections. So those are some of the things which you can do.

So what can I do with ObjectARX? So as a third party developer, what are the possibilities? So you can modify and extend the drawing database. So drawing, essentially is the database where all the engineering data that you're creating as a user is getting saved. There is no external database in case of AutoCAD products like MS Access or Oracle or anything. So drawing itself is a database. So whatever you are storing as engineering data-- whether it's a block reference, or sort lines, circles, anything-- that is getting saved in the drawing.

And as an ObjectARX developer, you can get access to this drawing database programmatically. And then you can manage your data. You can create new type of data through your code. You can delete the data. You can modify-- whatever you want to do, you can do using the API.

You can create new types of objects. There are some existing object types which are offered by AutoCAD. But if you want to have some new type of objects that are specially designed for your purpose, you can implement those. And you can add those to the drawing. So that's the overall concept of managing the drawing database.

So user interface is another important part. So user interface, as you know, the commands are the most important part of how users interact with AutoCAD. So you can manage your commands through ObjectARX. You can write your own commands which can be used by the end users. Then you can modify some of the user interfaces, like Toolbars, Dialogs-- Properties Window is another important part. So all of this can be customized using ObjectARX API.

And then, important part, we just saw in the previous slide there was a mention of reactors. So event notification is nothing but that reactor mechanism. So there are so many things happening when you are working as a user on an AutoCAD product. So you can add a line, for example. It's a simple workflow. But at the same time, behind the scene, there are some notifications which are getting generated. And as a programmer, ObjectARX developer, you can catch those notifications and react to it.

So if certain things are happening, you can get notified and react to it. So that's what the notification, or the event notification, or reactor mechanism, is all about. So how is actually used? That we will see in the subsequent sections.

So development environment, if you are dealing with the latest AutoCAD 2017 series of products, you can use ObjectARX SDK 2017, which is already available. The development environment for this is Visual Studio 2015, which is what we call in programming world, is IDE, Integrated Development Environment. That is Visual Studio 2015, or you can Update 1 also for it. The operating system, Microsoft Windows 7 or anything better than that is good.

Just to get a glimpse of what are the main libraries available in ObjectARX. So AcRx Object and Class Management, this is the first important one. So in C++, if you have some understanding of what is called as RTT, that is a Run Time Type identification. So AcRx is actually taking care of that part. So when you add a new type of object in ObjectARX framework, it has to be registered with the ObjectARX framework. So AcRx is actually taking care of that part.

AcDb is very important library, which actually deals with the drawing interactions. So when you save a drawing, the entire data in the drawing is getting saved in the drawing database. So AcDb module is actually the one which is taking care of all those drawing level interactions. Or when you actually read the drawing, open the drawing, that time also AcDb is coming into the picture.

.

AcGi, the AutoCAD graphic interface, is another important library which is responsible for displaying your entities on the drawing canvas. So a particular circle maybe, for example. How does it look on the canvas? Whether it is a real circle, or it is made up of some small segments created and then you get a feel that it is a circle. All these things depend on the-- how it is implemented in the AcGi. And there is a continuous development happening to make sure the AcGi library is offering better and better quality of graphics.

AcGe is the AutoCAD geometry library. So all the, you can say, kernel level operations like a line intersecting with some other line. Very simple example. So how that algorithm would work. Or there might be very complex algorithm of surfaces, triangulations, things like that. So all the geometry level algorithms are done inside the AcGe library.

AutoCAD Core Library, Accore, is another important-- it has so many core business logic implemented inside this library. And then AcUi and AdUi are the MFC extensions, like user interface. MFC is nothing but a user interface library offered by Microsoft. And inside AutoCAD, there are so many dialogs, user interface dialogs or property palettes, which are implemented using MFC technology. So you have the opportunity to extend onto those existing dialogs or create your own dialogs by using this AcUi or AdUi libraries.

So these are the main libraries which are there inside the ObjectARX SDK. Now, this is an important slide. So in the beginning, we discussed model types. So when you are dealing with ObjectARX, or so as to say AutoCAD, you will come across with these three different types of files. So they ARX files, CRX files, and DBX files. So what are these different models? Or what are these different file types?

So anybody who is familiar-- OK, let me ask this question. Are you familiar with a term called three tier architecture, from software point of view? Anybody?

AUDIENCE: With a what?

AMOD KULKARNI: A three tier architecture.

VINIT SHUKLA: Three level architecture.

AMOD KULKARNI: Three level architecture. OK. So let's try to relate it, and I will explain what this is. So this is what a typically three tier architecture, or three level architecture, would look like. So there is a user interface layer. In the middle, there is a business logic layer. And then there is a data

access layer. And at the bottom, there is a database.

So forget about DWG at this point. But for any general application, this would be true if you are following the three tier architecture. So there is a database at the bottom. There is a data access layer, business logic layer, and at the top there is a user interface layer.

So what does it offer? So this componentization of separating user interface, business logic, and data access, offers you a lot of flexibilities that tomorrow, if some new technology comes and you want to replace your existing user interface completely with that new technology-- for example, MFC is outdated technology now. And you want to replace your user interface with some new technology called WPF, let's say.

So if your components are not separated like this, and if its a monolithic application where your user interface code, and your business logic and data access, everything is mixed together, it will be very difficult to use those newer technologies very easily. And that's where this architecture comes into picture. If you had done that, then you can replace your user interface layer with some newer one. And that's where this comes into picture.

The ARX is actually corresponding to the user interface layer. So typically, the ARX model is supposed to be consisting of the code, or the implementation, which is related to the direct user interaction. It may be command. It may be some kind of user interface like dialog, or any other thing. So if you have to implement that, implement it in a ARX model.

CRX is business logic layer. So its like, it will not have any direct user interaction implemented in it. It has capability to run on its own. It will have some very important business logic that you want to implement inside it. But it will not have any user interface implemented in it. It will work along with the DBX model, which is basically the data access layer.

So data access layer is-- the DBX is the one which corresponds to the data access layer, responsible for directly accessing the drawing contents. So you can read the data from the drawing, you can write the data to the drawing, using the-- by implementing it inside the DBX model. So that's how the overall structure of your ObjectARX application should be.

Now, loading the ObjectARX Apps. Once you have developed the ObjectARX application, how do you load it into AutoCAD? There are simple mechanisms to do that. You can simply use APPLOAD command, or you can drag and drop the module into the AutoCAD environment.

So you write some application, you implement it. Either it's an ARX file, or DBX files, or CRX. What you can do is, you can use the APPLOAD command. It will show you a dialog like this. And you can select the file. And that thing will get loaded into the AutoCAD.

Once it is loaded, the functionality that you have implemented inside that model will be available for you inside AutoCAD. Probably it will be a new command that you have implemented, that should be available for you once that model is loaded. Or you have implemented some new type of object, that should be available for you once it is loaded.

There is another scenario where AutoCAD automatically loads these models. For example, there is a drawing file which has some custom objects in it, which are not normal AutoCAD objects, which are implemented by some third party application. So if you try to open such drawing in AutoCAD , AutoCAD will automatically try to find those applications and try to load those applications. So that's called-- that's what we call as a demand loading. It's not explicit loading by the user. But AutoCAD tries to do in certain scenarios automatically.

There is another time which I purposely introduce this slide here, because there is always this confusion between what is ObjectARX and what is ObjectDBX. So ObjectDBX is nothing but a subset of ObjectARX. So ObjectARX is a whole SDK, which offers you everything, covers ARX part, DBX part, CRX part. Everything what you can do with AutoCAD is available inside ObjectARX. But DBX is a subset of that, which deals only with the data access part that we just saw. So it builds only with this data access part here.

So ObjectDBX is an AutoCAD independent subset of ObjectARX. Now why it called AutoCAD independent? Because if you write an ObjectDBX application, it's not necessary that you have to run it inside the AutoCAD environment. You can actually have-- you can actually implement a separate executable application also, which does only the drawing read and write operation.

So it's not necessary you open it in AutoCAD. It can silently read the drawing, do whatever manipulation you want to do in the drawing-- maybe you can add some objects, delete some objects, whatever, modify some objects-- and save that drawing. So this can happen transparently, silently, without AutoCAD getting into the picture. So that's what ObjectDBX is. But for ObjectARX, you need to have AutoCAD.

Now RealDWG is the license to use ObjectDBS in an own application outside of AutoCAD. So if you want to write an ObjectDBX application that has-- that is independent of AutoCAD, then you have to have RealDWG license for it. But if you want to run inside AutoCAD, then there is

no need, actually.

So what I can do with ObjectDBX? Create object enabler. Have you heard this term, object enabler? Yeah. So object enablers are nothing but DBX models. It's a set of DBX models which are kind of responsible for identifying, let's say custom objects. So inside AutoCAD, if you want to load some drawing which has non-AutoCAD standard-- nonstandard, or non-AutoCAD objects in it, which are separately implemented by some third party, then to recognize those objects in AutoCAD environment you need to have those models which has the implementation done in it. So those models are typically called as object enablers.

So if you want to open, let's say, AutoCAD Architecture drawing, ACA drawing, in vanilla AutoCAD, pure AutoCAD, whether pure AutoCAD will be able to identify the wall or the windows that you have added to the drawing? No, it will not be able to identify it, unless ACA provides AutoCAD with original set of DBX files, which are the ACA object enablers. So that's how any other product which is having custom implementation of objects would have to provide the enablers so that other products, whether it's AutoCAD or any other AutoCAD based product, would be able to identify those objects. So that nothing but object enabler. And as I said, we can also write standalone applications using RealDWG license, which will not need AutoCAD for reading and writing drawing.

So this is how, typically, in AutoCAD database, data is stored. Now we are slowly getting into some deeper technical stuff where, as I mentioned in the beginning, like how the data is stored in drawing when you add something to the drawing. So this is how the internal structure of the drawing database is. So if this is your drawing, and this is how some data has been created in the drawing, actually in the background there is a database which is nothing but corresponding to it. You can say it's a drawing. Then there are some symbol tables. [MUMBLING]

[SIDE CONVERSATION]

AMOD KULKARNI: Highlighting?

VINIT SHUKLA: This one, this one. Pointer.

AMOD KULKARNI: So this is your drawing. So this object corresponds to your drawing. Then there are nine symbol tables, which are-- in the ObjectARX world they are symbol tables. So there is one corresponding to layer table, one corresponding to block table, and there is one for text table-- text style. There is one for view and such-- there are nine such symbol tables available. Each

of those tables have records, like layer table record, block table record, and their individual symbol table records.

Now, this is one data structure where the data is getting saved in the drawing when you add something to it. Another data structure is name of the dictionary and the child dictionaries that it has. So it's basically a container, you can say, for the dictionaries. So symbol tables and dictionaries are, broadly, the two data structures where data is getting saved when you add some data to the drawing. Now, when you have added, let's say, some line to the drawing it actually gets stored inside the block table record. So this is an important symbol table.

The block table is an important symbol table because the entire design data, the entities, basically, that you add to the drawing, are getting stored inside this block table data structure. If you add any layer, that's a new layer to the drawing, it will get stored inside this data structure, the layer table. If you add a new text style, there is another symbol table for it. Now you are familiar with styles in AutoCAD Architecture, right? We use walls styles, windows styles. All those are nothing but dictionary objects.

So whenever you create as a style, it is getting stored in the dictionary. So now we will see, maybe this looks very terrible and you are not able to visualize it, what exactly might be happening here in the scene. But in the subsequent slide we will see what exactly it means. So this is how the general--

AUDIENCE: So that means everything that shows up in Style Manager is getting [INAUDIBLE] object dictionary?

AMOD KULKARNI: Yes. It's coming from the dictionaries, yeah. OK, so I think that's pretty much it. Now, one more thing and then we will move to probably the first demo. So this is the object identity part of it. Now, as a programmer, let's try to relate it to the general database concept. So when you are dealing with, let's say, any RDBMS like Oracle or MS Access for that matter, you add some records. You add some-- there is a table, and you add some records to it.

And then you have some mechanism to identify that record uniquely. There might be a primary key or something, by using which you identify that record. Or it could be a combination of some keys. But in drawing, how do we identify a specific object, because there might be thousands of objects added to the drawing? As a programmer, when you are writing some application and you want to modify that data of a particular object, how do you get access to that particular object in the huge drawing that you are having?

For that, there are two mechanisms. One is the handle. The AcDb Handle is the corresponding ObjectARX class for that, so unique identifier of an object for the life of the drawing. So this thing is actually getting stored along with the object in the drawing itself. So every time you open that drawing, that handle will be present. So you can access that object by using the handle if you know it. So handle is nothing but some kind of number you can see. So you can access the object if you know the handle.

Another thing is object ID. Now this thing is dynamic. It's not static like handle. This thing, the object IDs are generated every time you open the drawing. So they are regenerated, rather. So you will not have the same object ID every time when you access the object.

So these are two different mechanisms by using which, either of the one, you can access the object. And then you can do whatever you want to do. I mean, first you have to get access to the object from the drawing.

You are to open the drawing. You get access to the object. Then you set some property of the object, modify it, delete it, whatever you want to do. And then save the drawing. So that's how the typical work flow would be.

So as I said, this theoretical part may be a bit difficult to visualize, like how exactly data is getting stored in the drawing? What is meant by dictionary? What is meant by symbol table? So we have some snoop tools which are available. Some of them are available publicly. Some of them are not. So we can try to demo one such tool which is available along with the ObjectARX sample, and this tool is called ARX DBG.

So this tool can actually give you a good insight-- good internal viewing of how the data is getting stored in the drawing. So I think-- let's try to have a quick look at this tool.

VINIT SHUKLA: So we have a drawing here, which is having a line.

AUDIENCE: Not seeing the drawing.

VINIT SHUKLA: No? OK.

AMOD KULKARNI: I think you have to [INAUDIBLE].

VINIT SHUKLA: OK, stop the rotation. So we can download ObjectARX SDK from AutoDesk website. And there is a sample application in the ObjectARX SDK called ARX DBG, stands for ARX Debug. So the

command to load ARX is AppLoad. So when we run this command, we get the dialog. We can browse to the location.

For example, I have downloaded ARX SDK here. So inside the ARX SDK there are samples. So in the database samples, we have this ARX DBG sample which I have already built. So this is the ARX that got generated after building the sample. So we can load this. This is like a warning. So here we said-- it says it is successfully loaded. So after this ARX is loaded, on the context menu we get this option, ARX DBG. And there are several options inside this to test the database. So we will first go into the database info thing.

So this is what Amod just inform-- told us, that there are several symbol tables. So block table record, dimension styles, layer styles, and several others. So if you open the block table record, by default there are three block table record present in a drawing. So each stands for the model space stands for the model space lay out and the paper space. And paper space zero stands for the layout one and the layout two in that drawing. So any entity added to the model space gets added to the model space.

So here, this is the information for the model space table record. Inside this, we can see when we open this, entities in block. So we can see there is the object of AcDb line, the line that we see on the canvas. The line-- the class name for this line is SCDB line. And then we see a handle of the object line. This is 484 now. If we save this drawing, and close AutoCAD station, restart AutoCAD station, again open this drawing, this handle will be same.

And then there is object ID, which will change each time we open a drawing.

AUDIENCE: [INAUDIBLE]

VINIT SHUKLA: Yes.

AUDIENCE: Lot of times [INAUDIBLE] trying to use the error code, [INAUDIBLE] name on the handle. So when we open the file, how do we delete that entity which is making our file to crash? Is there a way that we can--

AMOD KULKARNI: One way is, you can do a drawing recovery, actually, that internally tries to delete the hanging objects. But apart from that, maybe you can try to push some of the entities.

VINIT SHUKLA: Apart from that, you can write-- you can write your own small application which will go and read that object and delete it from the tables.

AMOD KULKARNI: Yeah, you can write a custom app. But out of the box, I think there are two ways.

VINIT SHUKLA: OK, so we'll go back to the presentation. I need to move [INAUDIBLE]? Anything more? OK. So [INAUDIBLE]. And if you have objects, instead of going to the block table record and then, you can directly go and debug the object. Select the ARX DBG entity in full, and then select the line. And we can directly see the line information. All the information, like what is the plane, what is the layer, what are the starting points? And there are many other options in the ARX DBG which we can explore later.

AMOD KULKARNI: This, just to give you idea that there is a tool available which can give you some idea on how you can visualize the data that it's getting stored in the drawing. So just a couple of notes left on ObjectARX and then we can start with the OMF part. So I think-- I think this is more like a summary of what we just saw. So there are three different blocks table records, as we explained. So there is model space, paper space, and paper space zero.

And I think you can relate it to the tabs that you see on the drawing. So there is a model space tab, there are paper spaces. So by default, in a drawing, there are three such block table records available internally. And whatever you add to the drawing actually getting stored in the respective block tables records. So only entities added to one of these is visible in the AutoCAD drawing. So anything else, any other thing that is getting stored anywhere else in the drawing, is not visible in the canvas. So dictionaries, for example. They are not visible in the canvas. You can only visualize the things which are added to the block table records.

Yeah I think it's really repetition. Now, slowly getting into-- towards the OMF part. And for that, we need to understand as a developer, when you start writing a new application, typically you would want to write a new class. So class is, I think most of us know, its an object oriented concept, class. So when you write a new class, you need to decide from what you want to do. What is the responsibility of that class?

And based on that, you have to decide, from where I want to inherit that class? Because ObjectARX offers you many base classes, from which you can inherit the custom class that you are trying to implement. And that inheritance gives you lot of default functionality. And depending on what you want that class to do, you have to decide the parent class.

Now, there are many, many parent classes that you can derive from. But some of the important ones which are listed here. First is AcRxObject. Now, AcRx, as we-- I think in the

libraries part, we discussed that AcRx is basically runtime identification.

So you have to register your new class that you want to introduce into the overall ObjectARX framework so that it is identified, because if that class is not known to the ObjectARX framework, ObjectARX would not react to some of the things when you are building on it, when you are using that object. For example, when you add that object, ObjectARX might not, if ObjectARX doesn't know what type of object it is, it will not be able to handle that object properly. So first of all, you have to register that class using-- by driving it from the AcRxObject.

Secondly, it is AcGiDrawable. So if that class that you are implementing has some implement-- it has some responsibility of graphically showing it nside the drawing, then you have to derive it from AcGiDrawable. Otherwise it will not be able to draw itself in the canvas. So you have to do that.

AcDbObject, if you want that class to-- instance of that class. Not class, the instance of that class, that is the object. If you want to store that object to get stored in the drawing-- if that has to be persisted in the drawing, then you have to derive it from AcDbObject, because this particular AcDbObject gives us the ability to store anything inside the drawing database. So anything that goes into the drawing has to be derived from AcDbObject.

Secondly, AcDbEntity, AcDbCurve, these are the special cases for [INAUDIBLE] the object. So this is again persistent in the drawing, but in relation to AcDbObject what other thing it has, is basically graphics that is shown in the canvas. So AcDbEntity is AcDbObject, plus some special graphics. So if you derive from AcDbEntity class, you can have your own graphics that can be shown in the canvas, and you can save that in the drawing also. So it is typically called as a custom entity.

So whatever we saw-- what we see in AutoCAD Architecture, MEP, for example. Like pipe entity, or a wall for that matter, window-- all these are custom entities. So they are derived from AcDbEntity. That's why they have their own special type of graphics that we see in the canvas, which is not available in normal AutoCAD. So those are specially implemented entities. And you can save them in the drawing also. So that's why they are derived from AcDbEntity. And again, this is-- Curve entity is special case of AcDbEntity.

So ObjectDBX, or [INAUDIBLE] ObjectARX also help and host applications like AutoCAD, interact with your object to these interfaces. So if you derive from these-- one of these, or any

of these base classes depending on your requirements, the host application that is AutoCAD would react, or interact with your object through these protocols. So when you, let's say, rotate your entity, there will be certain things happening, certain notifications generated from AutoCAD. And you could react to those notifications in your implementation of the custom entity.

So we will see those things in the coming slides. But it is important to understand that you have to follow certain protocols, and those protocols are nothing but deriving from these base classes. We will specifically see the AcDbEntity protocol, because that's what are going to do in the OMF sample that we are going to look into afterwards. Now, AcDbEntity protocol, these are some of the details of that protocol. So I think-- yeah, it looks very flimsy, but typically what it means is, if you see this part, GetOsnapPoints. So we know what is osnap, right?

So when you select an entity in the canvas, there are some osnaps, object snaps, which are displayed. And other objects can snap to it. So for your custom entity, that you are implementing now, if you want to provide some of object snaps, then you have to implement this function in your code. And you have to tell AutoCAD that, OK, if my entity is selected in the canvas, these are the osnaps I want to show. This is how you interact with AutoCAD.

Similarly, GetGripPoints. So these are all functions inside your custom entity class. And you have to implement it in a way that AutoCAD understands. If this type of entity is selected, I have to show these grip points. So this is how the protocol works. Similarly, I think that I will not go through each of them. I think important one is this, subTransformBy.

Again, there are some transformation operations like rotate, move, all these transformation operations. So how your custom entity should react to these commands? If your engine is selected and moved, rotated, how it should react. All that can be implemented in the transform by a function. So you have to reply to AutoCAD Occurred when AutoCAD asks you, OK, there is a move command happening on your entity. What do you want to do?

You have to reply back to AutoCAD, this is the new transformation matrix after the move command is implemented. Because AutoCAD tells you, OK, this is-- this much is the displacement that is happening on your custom entity. Or this much is the rotation that user wants to apply to your customer entity. What do you want to do? So you have to calculate the transformation matrix, and reply back with the AutoCAD. Then AutoCAD will take care of the things. So this is how it works.

OK, subIntersectWith, that is intersection with some other entity. GetGeomExtents, since there are many, many situations where AutoCAD would want to know what is the bounding box of your custom entity. There might be some intersection with some other entities, or there may be some other scenarios where AutoCAD would want to know what is the bounding box? That is, the area covering your entity. So you have to reply back to AutoCAD by implementing this GetGeomExtents function.

subList function, so I think we know list command, right? You select the entity, and you give list command. In the command window you see so many details of your entity. So if you want to override that part, you can implement the subList function. There is some default implementation available that shows some of the basic data.

But apart from that, if you want to show some additional data, you can implement the subList part. subExplore, I think subExplore-- Explore command, everyone knows. So if you Explore, the custom entity is getting disintegrated into many AutoCAD entities. So how do you want to react to this Explore command? So that is implemented in the subExplore.

And most importantly, the worldDraw, or viewPortDraw. So this function is basically required for telling AutoCAD how your entity looks like, graphical. So your graphic implementation, that my entity-- custom entity looks like a circle with eyes or some kind of smiley inside it. So that graphic detail of how entity looks like, are implemented inside one of these functions. So there is subtle difference between the two, but let's not get into that.

Now we get into the object modeling framework part. OK at this point, are there any questions?

VINIT SUKLA: [INAUDIBLE]

AMOD KULKARNI: So, OMF-- so, Object Modeling Framework, it's a long-- what we call as OMF is Object Modeling Framework. So this is the API for AutoCAD Architecture and MEP, C++ based. It's a layer on top of ObjectARX, just like MFC versus Win32. Just an analogy, if you know MFC and Win32, you can correlate what is ObjectARX. and OMF. So basically, it's an extension. So we saw what is ObjectARX..

OMF is a next step to it So it tries to use the power of ObjectARX. and allows the third party developers for ACA and MEP to customize other products. It's a powerful tool to extend AC, and also create your own ACA like objects and entities. What does it mean? So we saw what is

custom entity and what we need to do for implementing our custom entity. So in OMF, there are a bunch of possibilities, by using which you can have your own ACA like entities. Like, we already have walls and windows and many other-- stairs, roofs, different object types.

So in MEP, we have [INAUDIBLE], et cetera. But if you want to have your special implementation of those entities, because ACA and MEP offer you some default implementation of the object. But if you want to-- let's say, a wall object, if you want to derive your entity from the existing one, but you want to provide some special features, some additional grid points, let's say, or some special style that you want to implement for the wall, that can be done through OMF.

Still can use functionality of ObjectARX., Because you are still in the ObjectARX. world. So it's just an extension. So both OMF code and ObjectARX code can coexist. So it is more object oriented. There are some portions of ObjectARX. which are not fall in the-- 100% object oriented concepts. There are some global functions and stuff like that. But OMF is purely object oriented.

Last-- latest version is 2017, and this API, OMF API is available only for ADN members at this point. Of course there is a .NET available on this OMF, which is available for all ACA MEP users, because it is available as a part of your ACA MEP installation itself. So you can write to your document implementation by using those managed models, which are installed along with the product. But the OMF as such, the C++ part, is available only for the ADN members.

So I think this is-- we discuss more of this. So you can access the building model. What we mean by building model is, in a typical AutoCAD Architecture or MEP drawing, you would have lot of architectural optics like window, wall, et cetera, added to it. By using OMF, you can get access to that data and you can do so many things. So we will try to see what all we can do.

I think the list is self-explanatory. So we'll just-- so just to show the analogy between the AutoCAD and ACA API that we have available. So for AutoCAD, we have, I think many of you might know, there is something called as Lisp, which are kind of a scripting. API, which were available I think for many years they are available now. But we do not have anything corresponding to that in ACA. We don't have anything corresponding to Lisp. For ARX, I think by now we know that there is something called as OMF, which is extension of the ARX in ACA. I'm sorry, there's some skipping, I think-- OK.

AutoCAD ActiveX.. So, ActiveX, it's-- you might have heard COM technology from Microsoft.

So ActiveX is nothing but COM API. So these are mainly used for property pallete customization. So you have property palletes in AutoCAD, or any AutoCAD based products. So using these API, you can customize your property palletes. If you want to add a new tab, let's say, to the property palletes, some new rows for the property pallete, and show some additional properties, that can be done through ActiveX API.

For ACA also, we have corresponding ActiveX APIs. These are mostly C++ COM API. And there is a .NET API, as I mentioned. It's a wrapper on top of the OMF one. So the .NET ones are available, but they do not really-- sorry, I'm talking over [INAUDIBLE]. So AutoCAD .NET API are full fledged available or AutoCAD. But ACA .NET API, as I was just mentioning-- sorry for the confusion there. So ACA .NET API are available, which are wrapper on top of OMF, but they do not cover the entire functionality of OMF. So it's a subset, we can say.

But still, I think people are using these API, using .NET supported languages, like maybe .NET or C#. You can use those languages for writing custom applications on top of ACA and MEP. There's a general component architecture for OMF. Let's not spend too much time on it, but just want to explain a couple of important points here. So by now we know what is DBX, right? These are data access layer. We know what is ARX, so these are mostly user interface related models.

So ActiveX, we just saw. These are X models are typically called as ActiveX models. And there are-- there are some CRX models also available. So these are business logic models which are available inside OMF. So in vanilla AutoCAD, if you load of all these models, basically you can say it turns it into ACA, in a very layman's language if you want to say it. So these models basically have all the functionality which allow you to implement custom entities and do so many things which can be customized inside AutoCAD Architecture and MEP environment.

We will have some quick differences between OMF and ObjectARX.. And as I said initially, it's not like OMF versus ObjectARX.. They are like partners to each other. But ObjectARX has certain areas where OMF has improved upon. So that's the whole idea.

So in OMF, we need to derive from AecAppDbx. So whenever you start writing a new application, you want to write a custom application using OMF API. So you have to decide whether you want to go for DBX application or ARX application. And once you make that decision, you have to derive from one of the available base classes, that is AEC DBX or ARX. Now, it has its own advantages of course.

By deriving from these base classes, your application is plugged into the OMF framework. You get default support for internationalization. So that means automatically there is a resource DLL created once you derive from this class. And that resource DLL can be useful localizing your custom application. So that part becomes very easy. And of course it's ObjectEnabler strategy, you can, if you go for a DBX application, you can very easily create those DBX models and ship it as object enabler.

AUDIENCE: OMF creates its own object enabler?

AMOD KULKARNI: Yes. You can derive you own object enablers. OK, this is very important part, and I think in the demos we will see this. So we saw that in the AcDbEntity protocol, you have to implement your ViewPortDrop or WorldDrop Function to show-- to tell AutoCAD what is the graphic representation of your entity. But in OMF, actually you don't need to implement any of these functions which we saw in the AcDbEntity protocol. Because that part is taken care of in the background by OMF provided you give some implementation of these three basic functions, like Draw, preDraw, and postDraw. These two are also optional, but Draw is the important one.

And most importantly you need to implement the AcDbDispRep. I think those who are familiar with AutoCad Architecture, then maybe you know what is display representation. So this class is actually in a programming equivalent of that display representation. So it's AcDbDispRep. So this class takes care of how your entity is going to look like, depending on how view directions and things like that. So this is how you have to implement it in OMF.

So based on these two parts, that is the deriving from the application base class and the other thing that is how to implement the draw, you will the demo. And we will gradually progress with this application, the demo application, and eventually we will see how the custom entity can be implemented.

VINIT SUKLA: So this is what demo is for, creating of a sample custom entity. So this entity is very simple. It looks like a triangle in the top view, and it has a solid in model view. So this project is created from the ObjectARX wizard. So then we install ObjectARX wizard in Visual Studio. And then we create a new project.

So you can see in the Visual Studio templates AutoDesk, this option, ObjectARX DBX OMF project. So if you select this and create a new project, and then we select-- it will ask, it's a DBX project or an ARX project? Then we can create a lot demo this part. I've created the

projects already. So when we create one project, like the one I created here is OMF DBX.

This is the DBX project which will contain the definition of our custom entity. It will automatically create a dependent project on this is OMF DBX ENU, for English. So all those strengths that are user facing strengths, will be defined in the ENU project, the resource project. And to go translate to globalize this, we just need to translate the ENU. The DBX, OMF DBX model will be independent of the globalization. So after this, so this is a-- this file gets auto-generated from the ObjectARX wizard.

This is the implementation of the class. So we see that this is a derived application, which is derived from DBX because I selected the DBX option. And it implements few methods. So is the implementation of the class, and most of the methods are auto-implemented, so we don't have to implement. As we go on adding functionality to this model, we'll see how to add our views and custom object here.

So this is the project that got created from the ObjectARX wizard. Then we will go and define our custom entity. So in this case, we are creating a custom entity with the name OMF DbEntity. It can be any name that you want for your object, a meaningful name. So we are deriving from AecDbGeom, all the custom entity and OMF are derived from AecDbGeom, which in turn is derived from the AcDbEntity.

And this is AcDbGeom implements the protocol which AcDbEntity asks us to implement, and simplifies the work for OMF custom entity. As part of this, you see we are not implementing any of the AecDbEntity protocol. We have-- only thing we need to implement is a summary info, for to support list command, set from the standard. So when we create new object, it will inherit some of the properties from the standard when scale is done. And then there are some param-- the properties of the class are the length and height, which will be the parameters for the custom entity which will define the size of the custom entity.

And you can see here we are not defining any draw method in this. So this is just the database class defining the database, the parameters of the custom entity. The draw code will be separate. And this is the property of OMF which separates the data from the display. So then we will implement this OMF entity. So the [INAUDIBLE] integral method to support the read and write of the drawing files. That summary info we have to implement to display when the list command displays for this custom entity.

All these things are auto-implemented, like for this scale by, we just enhance the scale the

length. The next thing we'll do is this defines the custom entity, but it cannot be displayed because we do not have-- we have not written the code to display. So we will write a display code, the display class for the plan view. So this is our display-- representation class for the plan. It is derived from AecDbDisplay, which is a virtual class. And we implement these three virtual methods for this. WorksWith we will define what is the AcDbEn-- what is the custom entity this display class will work with, what are the view type name it supports, and the draw method?

So this is the method which will draw the custom entity on the graphics. Then we look into, and we will also define the properties like the proper-- the display properties are also separate, which, are not part of the custom entity. So the display properties we have defined for this entity are-- there are two properties. One is a profile that we'll see in the custom entity. And the name of the m which are two. We can selectively change the color, layout, line type and visibility of these two display components.

Then in the plan implementation, the supported view type is this display representation is for the plan view. So whenever we enter view, this display representation code will be used to draw the custom entity. Then this is the code which does the final draw in the plan view. So whenever we switch to top view, this method will get called to draw this custom object.

AUDIENCE: And all these show up in the display manager with all their options, like the [INAUDIBLE]

VINIT SUKLA: Yes. Yes.

AUDIENCE: [INAUDIBLE]

VINIT SUKLA: Yes. Yes. Yes. By implementing these two methods, this custom entity will be visible in the display manager. We can select the types of the view for--

AUDIENCE: [INAUDIBLE]

VINIT SUKLA: Yes. Yes, yes, yes, yes. So here is a very simple code to draw. This, we just get the vertices from the custom entity, and we just stream it to the graphics. And then we draw the name of the entity. We just say OMF entity, the name of this custom entity, and that is also streamed to the graphics.

So this is the display representation for front and the top view. And for this custom entity, we want to define a-- in 3D or the model, this space view is different. For example, we saw the

door. In top view, it looks like an arc. But in the model, it looks like a real 3D door. So for that, we will define a model display representation.

This class is almost similar to the plan display representation. We implement almost same functions. And the draw is overrated. And this, we are having-- we have a different property class for the model view. In here we are defining three display components.

We are defining as top body, bottom body, and the entity name. These property we'll see in the final demo. So we'll go to the model, the implementation of the model display of that. So this is adding the display property. We will go to this one.

The display there for the model. So we create these other two methods to let the OMF know that this new entity is added, and to display it in the respective display representation and sets. We have it look for the properties of this, so this display representation works with this custom entity and the draw code. So here we are in the body, we are creating a modeller body. This also is in the OMF SDK [INAUDIBLE]. We create a body, which is a box, from this origin and the size of length and height, and we stream this body to the graphics.

And finally, we display the name of the--

AUDIENCE: Where would it implement custom profiles into this [INAUDIBLE]?

VINIT SUKLA: Yes.

AUDIENCE: [INAUDIBLE]

VINIT SUKLA: Yes, yes. We can-- so this I will, for simplicity, I just used a box. But we can do it. We can have a polyline and then we can ask a modeller to do extrude from this, or [INAUDIBLE]. So all the repairs are available in a modeller, to create the design body. And clean ACA, whatever what it is we see are created in a modeller.

So this is the definition of the entity. The definition of the display representation, the plan view display representation, the model view. But still we have not introduced how to create this. So we will you go through some of the command, how to write a new command. And then we go through the ARX project that we have over here, through which we will be able to create this custom entity. And then we will finally demo this custom entity.

So we will go back to the presentation.

AMOD KULKARNI: So I think a couple of more big points of differences between the OMF and ObjectARX. Dictionaries, as we briefly discussed earlier, that whatever styles we see in AutoCAD Architecture or MEP, those are nothing but dictionary objects. So they get stored in the dictionaries of AutoCAD drawing. And Style Manager may be a famliar user interface for ACA users. So all those objects, as you say, are coming from the dictionaries.

So in ObjectARX, there are some more efforts, I would say, for the programmer to handle the style objects, other dictionary objects. But in OMF, it is very easy to handle these dictionary objects if you divide those from AcDbDictRecord. So there is a very base class provided in OMF, and if you want to store anything in the dictionary you can simply derive it from this base class. And most of these issues which are there in ObjectARX will be taken care of for you by OMF.

OK, we will not have this implemented in our custom entity for now, because I think it's too difficult to include everything. But these are some of the theoretical points that we can discuss. So reactors, I think we have briefly touched upon this topic, what is reactor? So it nothing but event notification mechanism. So some action happens, you get notified in your code. For example, reactor would be on an object level. So something happens to the object-- it is deleted, it is moved, or it is modified in some other way-- so you get notified in your code.

The reactor would be at the elite level, that some specific command starts, you can get notified. The reactor could be at drawing level. That drawing is getting saved. It is getting opened. All these specific events, you can capture and you can't react to it in a way you want to. So that is nothing but reactor.

In OMF, reactors have very special significance because in OMF there is something called as object relation graph, which is continuously operating in the background when you are working in an ACA drawing. So object relation graph is kind of a reactor where you move-- for example, you have a wall. And there is a window or door placed in the wall. And you window or door in the wall, or you delete, for example, that wall. So any such operation actually updates the object relation graph, so that ACA always maintains the latest relationship between the objects, depending on the user actions. And this happens through reactor mechanism.

AUDIENCE: [INAUDIBLE]

AMOD KULKARNI: Sorry?

AUDIENCE: Sticky moves, where are those at?

VINIT SHUKLA: Sticking moves is a different animal, but it's something similar.

AUDIENCE: In this case, [INAUDIBLE] relationship, you can have [INAUDIBLE]. Do you know what I mean? You cannot draw [INAUDIBLE].

AMOD KULKARNI: If you place a window or door without a wall, then there is no relationship. It's a hanging wall, hanging door. So there is no relationship in that. So similarly, I think we discussed transformBy. So it is a function which is getting called when there are transformations happening, like moving or rotation of the objects. So you can implement it in your custom entity. And most of the things are handled by OMF when such transformations are happening. So this is very simplified in OMF as compared to ObjectARX.

Now we move to user interface customization in OMF. And commands are the most important user interface, as we discussed, in AutoCAD. Same is true for SEO. So there are so many commands that you can use to add new objects or modify existing objects. So if, in your custom application-- for example, now the custom entity that we are implementing, you want to add a new command to add that entity. Because as Vinit said, what we have done right now is just the database part of it. That we have implemented the custom entity, but we don't have any mechanism to add it to the drawing.

So now what we would like to do is, using this mechanism, we will try to implement a new command using which we can add this entity. So what we can do in OMF is, base class is AAecUiCmd. If you derive your command class from this base class, it offers not only the basic requirement of adding or registering that command in the AutoCAD, because you have to first registere that command, AutoCAD to recognize that command first. Then only you can continue with that command. So it does the basic job of registering that command with AutoCAD. But in addition to that, there are so many other things that you can leverage if you derive it from this base class.

So for example, you can get some special context menu if you implement some functions provided by this base class. So along with if you select the entity, you can get some command available in the context menu also. There are some portable viewer windows that you can show when a particular entity is selected. So implementing-- driving from AecUiCmd proides a lot of benefits.

Command line prompts. Again, there are many scenarios where you want to accept some input from the user, some kind of data that you want to ask the user, provide this data to me. In ObjectARX., there are ways to do that, but in OMF it is very simplified. You have special classes available for accepting special type of data. For example, this is for integer. This is for double. This is for string, that is characters. This is for asking the user for providing a point in the canvas.

So all these things are very simplified. The developer can simply choose what prompt he wants to show to the user, and just use that function. So we will see how it can be used. This is a very simple example of how that can be used. So here we are asking user for providing the radius, which is nothing but a distance. So we use this particular prompt, AecUiPrDist(Def), this one. So just create the object with some default values, and just call a go function on it. And it will show the prompt for the user.

Entity selection, also very simplified. There are many scenarios where you want-- you expect user to select the entity and then perform some operation on it. So that part is also very simplified in OMF. There are pretty fine available base classes, like this AecUiPrEntity, AecUiEntitySet, EntitySetSingle-- each one of these has a special usage, and depending on your use, you can decide what to do. So based on this, we will continue the same demo that we have started. The database part is done now we will add a new command for that.

VINIT SHUKLA: So as a part of this demo, we have defined our custom entity. Now, we again create a new project using the ObjectARX. wizard. This time, we select the project type as ARX. So this will create the ARX project and the corresponding resource project. So all the strings that we will use for prompt and other things will be defined in the resource project, and the command will be defined in the ARX project. So that the default-- the ObjectARX. wizard will give us-- will implement the application class.

One thing we missed in the previous thing was, when we create a new view we have to register the views here. Like, for the plan we have registered in the register views. So for each custom entity, we can have multiple views, multiple display types, depending on the views or depending on the standards for each entity. So add classes. In this case, it is derived from ARX Add Class.

In the previous case, were derived from a DVX class. So we are now defining a new command. So our ARX command entity, which is derived from the AecUi command, and then

there's a virtual function to register the commands. And we will be adding two commands in this class, to add the entity and to edit the entity. So here is the implementation for that.

So this is a common line, so this is the function to register the two commands to the object-- to AutoCAD to know that these are the two commands. One is EntityAdd, another is EntityEdit. So the implementation for EntityAdd command is, allows the user to prompt-- will prompt the user to define the length of the entity with a range. The range is different from 100 to 1,000, and the default value is 500. And then the height, using the same prompt class, distance. And then we'll get the working database, create a new OMF entity, DbEntity, set all the values-- length and height-- and then we will-- this is a global function provided by the OMF through the placement of location and the notation of the entity. And then this is also a global function provided by the OMF framework to add it to the model space and close the entity.

Similarly, the other command is to edit the entity. Here we'll first ask the user to select the entity. So we are using a sort of-- if you use ObjectARX. as plain, or ObjectARX SDK, we have to AecEds as get. A lot of code to convert ADS name to the object AecDb object IDs. So this class wraps all those things and creates a simple one line interface, so this.

And then we select the entity. After selection, we just open the entity for write, modify the length, and get the new length, and new distance, and height from the user, and modify the entity. So these are the two classes. And then I think we'll not be able to, in the presentation, will have description of the grips.

We can create a [INAUDIBLE] object using the grips. That's also very easy. Just to [INAUDIBLE], we can demo this. And the sample code is available with the class resources. So you can download and play with it.

So now we will create a new drawing. I will just load this ARX and show that our custom entity. We are added for the standard display set, so we'll change that. So [INAUDIBLE].

AMOD KULKARNI: 26 [INAUDIBLE].

VINIT SHUKLA: So these are the two-- so this is our OE, which defines the object. So we'll load it first. And then our command classes in the grip classes. So it's got added. So after that, we will see. So this command is of a level that we added to the command class. So the default value is 500. If you give something more than 1,000, you call that the range we have defined. It will say, well, you must be within this range. So let's say we just accept 500 as the default. In the height, we

will define as half of the length.

And then we start seeing the preview of the entity. So this is-- so then we can place it. And then we can define the rotation of it. So we have added our entity to the database, and we can use ARX Debug to see what are the properties of this. And we can list this. So we can see this OMF entity. We tell the length and height, we can see here. This is implementing the summary information method in the DB thing. So this was-- the planned representation is just a triangle and an entity name.

Then we will switch to model space, and then the entity is a different entity. It's a solid. The display of the entity in model space is different. It's a solid. We can see in realistic mode, it's a solid with a name. We will define the different components for this. We can see here top body, bottom body. We can selectively hide the top body or the name of the component. Can go back to our frame. Any questions on this?

AUDIENCE: When we create these objects [INAUDIBLE] space, and there's going to be like rainbow color, [INAUDIBLE]. Can we also see this object go in the system? [INAUDIBLE].

AUDIENCE: OMF category.

VINIT SHUKLA: Yeah. We can-- I think we can do it to the UI with the properties definition. Not a thing from the core library [INAUDIBLE]. Can we add to the property set?

AMOD KULKARNI: Ahh, property set--

AUDIENCE: Yeah, but you have to [INAUDIBLE] wall, the property sets hang on your wall, and a structural member.

AMOD KULKARNI: Oh, you mean for the schedule table purpose?

AUDIENCE: But it's hard, what type of entity [INAUDIBLE] then get classified as.

AMOD KULKARNI: [INAUDIBLE] as an OMF entity.

VINIT SHUKLA: This is a separate entity. It's not any of the existing entity. It's its own entity.

AUDIENCE: Is there a way to specify a name for that classification of this entity?

VINIT SHUKLA: Yes. So here we have it specified as OMF entity. But for your use, you should use some

meaningful name for that.

AUDIENCE: Because if we can assign an entity to this, we gain control of display on the display model here. Or when we create a schedule, we can tell that this is some structural member that we can anchor other structural members. That's what I'm trying to do. That way you get the schedule [INAUDIBLE].

VINIT SHUKLA: OK. I think it will be [INAUDIBLE] consideration.

AUDIENCE: [INAUDIBLE]

AMOD KULKARNI: So since it's a custom entity that we are registered with the OMF framework, actually you can get it listed here if you--

AUDIENCE: Perfect. Thank you.

VINIT SHUKLA: So just because of OMF, everything-- just registering with the OF, all these things come for free.

AMOD KULKARNI: Yeah, that comes for free.

VINIT SHUKLA: We are not done. So whatever code I showed you, that's just the code--

AMOD KULKARNI: In fact, we write here [INAUDIBLE], just because you asked.

VINIT SHUKLA: Yeah, we don't know if--

AUDIENCE: This is the open code, you can create the style mod and then maybe [INAUDIBLE], you can [INAUDIBLE].

VINIT SHUKLA: So we have not implemented the style. But it's really easy to create separate styles for this product. In the display menu, we can see this entity. In the display representation we can go-- this entity is there. So it has two representations that we created, model and plan.

AMOD KULKARNI: So all those things will come.

AUDIENCE: [INAUDIBLE]

AMOD KULKARNI: OK. I think.

VINIT SHUKLA: Yeah. We are all ready running out of time.

AMOD KULKARNI: So now we have someone understanding how a door looks different in plan, and how it looks different in model view. So we can make our entities also behave the same way. So this is a list of helper functions, just to emphasize on the fact that OMF also offers a lot of wrapper functions, helper functions, which save time for the developer to implement lot of ObjectARX. code. So I think in one of the code snippets that Vinit showed, we were using this add to modelspace, add to model space and close. So this is basically, you just-- one function call, and you'll save a bunch of lines of code if you have to implement it in ObjectARX..

So these are some of the helper functions. So these are typical AEC entities and objects that you would see. So whenever you open any AutoCAD Architecture drawing, normally you would encounter these type of objects in it. So display properties, different AEC objects, anchors are another different area altogether. Objects styles, we all know-- display representation. So these are typical object types which we will see in any ACA drawing.

Now, interaction of AEC objects. I think the diagram looks very complicated, but I think the simple concept here is the bottom part. That is the ACA entity and styles. These are physical data. So what I'm trying to show here is, the ACA entity and style, they are not having any graphical information in it. This doesn't give any details of how the entity is going to look like on the canvas. So there is no graphical information.

But display representation and display properties are the ones which are carrying all the information which is related to the graphics. But still they interact with each other. And there is a clear isolation between the physical data and the graphical data that makes us very easy to make any entity look like any other form. Like you can show door as a door, but at the same time, by modifying the display properties and display representation, you can make it look like altogether a different entity.

So that's how-- because of this separation, it is possible in ACA and MEP. Whereas in AutoCAD, or ObjectARX., the entity itself is responsible for drawing itself. So there is only one representation for the entity. So this is what I'm trying to do. So the door would look like this in one display representation. It would look like another in another display representation, because we implemented the way we just did.

Again, display properties. These are the key part which basically define how the object would look like in the canvas. So color, or the line type, all these graphical properties are governed by the display properties. Again, at the center of all this, I think we have been-- we said many

times, display manager. So display manager, the user interface part that we know what it does. But in the background, the class that is responsible for all that functionality is AecDispRep Manager which is responsible for handling all the display related stuff in ACA.

VINIT SHUKLA: This is already done.

AMOD KULKARNI: OK. I think Vinit already did the demo.

VINIT SHUKLA: This was the demo to show you how the custom entity will look in-- how we can create a custom entity and we can review to show how the different display representation changed the display of the same entity.

AMOD KULKARNI: Yeah, in the plan view and model view. So again, there is some concept called streams in OMF. So its a bunch of classes that accept graphical input. And you can-- you can implement it to process the graphical input in whatever particular format that you want. There are already some available classes in OMF there do different kinds of processing of the graphical input. But you can implement your own stream also, by deriving it from the AecStreamAcad base class and decide what to do with the graphical input.

I'll just give some examples here. If you want to draw any entity on the canvas, this is done by the AecStreamAcGi. so you get the graph-- you get the graphical data and you pass it to the stream. The stream will do the job of displaying it. You get the graphical data and pass it to the stream intersect, it will-- it will do the intersection calculation with other entity and return you the result.

If you pass it to the AecStreamExplore, it will give the result of the explore operation. So these streams are basically meant for different purposes. Or depending on your need, you can pass it to the required stream and it will return you the data.

So you can see, if you pass it to the AecStreamExchange, what you get in return is a collect bounding points. And as I said, you can implement your own stream also and decide what to do. For example, I just want to mention one example here that we all know what is 3D printing, right? So Is it basically accept the STL data format. So you get the STL file and you can process it for 3D printing. So Vinit, actually, you want to explain?

VINIT SHUKLA: Yeah. So we can have a separate stream class, AecStream STL. So it will be going to the same-- and we can implement a method on the stream class to take a 3D body and convert

the 3D body using the model APS two triangles. And those triangles, we can write to STL file, which can be input to a 3D printer. So AC objects do not support directly the STL out command, which AutoCAD solid supports, but we can very easily write this command, the two lines of code to convert ACA solids to a 3D printable file in STL out.

AMOD KULKARNI: So we just collected the graphical input from ACA, pass it to the stream that Vinit implemented, and that stream was implemented in such a way that it creates the STL. OK, anchors is another very important concept. So we are probably known from user point of view, that when you have a wall, add a window in it, they are anchored to each other. The wall-- the window is anchored to the wall. So whenever you make the movement of the window, the movement is governed by certain rules. You cannot move window outside the wall.

Why why does it happen? Because there is anchor that is taking care of this movement. And there are so many anchors which are available in OMF, you can derive-- you can implement your own anchor also, and decide the rules that you want to follow for that anchor. So in case of wall and window, there is a predefined anchor that is used. But its up to you if you want to use the same one, or you can implement your own. So whenever there is a get ACS function getting called, the-- whenever there is a transformation or any anchor that is getting filed, the update ACS is getting called.

But the important thing is the anchor is the one which governs the movement of the objects. As I can see in AutoCAD, the entity is actually responsible to respond to the transformation. Whenever there is a movement of the entity, it is the entity's responsibility to respond. Well, in case of OMF, It is the anchor that is taking care of that part. So again, that part is separated. If there is no anchor, the body can move freely. There is no-- there is no restriction on the movement.

AUDIENCE: [INAUDIBLE] two bodies are anchored together?

AMOD KULKARNI: Two?

AUDIENCE: [INAUDIBLE], if they're not anchored, you've got the [INAUDIBLE] individual-- Yes.

AMOD KULKARNI: Yes, yes. Yes. Again, there are available anchor types. There are some are pretty fine, you can--

VINIT SHUKLA: You can skip this.

AMOD KULKARNI: Here we can derive from the existing ones. Again, we were talking about property sets. We are all familiar with it. So all these properties sets and related functionalities is already available in OMF. You can create your own property set definition, add properties to it and so many stuff can be done. You can clear schedule tables using the properties sets. All that can be done using OMF.

OK, so how to get OMF as-- sorry?

AUDIENCE: Can you modify existing code?

AMOD KULKARNI: Yeah. Should be good. Yes.

AUDIENCE: [INAUDIBLE]?

AMOD KULKARNI: Yeah. So OMF is available to download for ADN members. Please visit this-- I'm sorry. OK, in .NET API, I think as we briefly discussed, its a wrap around OMF. But it doesn't cover the entire functionality. So any questions? I think during the session we have been discussing, but if you have any questions we can discuss offline also after the session. I encourage you to give feedback on the session, either online or if you have the AU app, download it, please use it for the feedback. We'll be there at the Answer Bar also. If you have any additional questions, we'll be happy to answer. Yeah.