**How to Develop Dynamic Mapping Applications with IBM ILOG JViews Maps**
**Event ID: 236882**

Unidentified Speaker: Welcome to today's webcast, Dynamic Mapping Applications with IBM ILOG JViews Maps. This presentation is being recorded and will soon be posted to IBM Developer Works for on-demand viewing.

During the presentation, you can submit your questions at any time, but we will answer all the questions at the end. Our presenter today is, Dan Jeffrey, Technical Manager for ILOG Visualization in the IBM Software Group. Dan manages technical presales of the visualization products at IBM. With 20 years experience in software development specializing in user interface design and implementation, Dan has a passion for designing usable applications for rich user interfaces.

Dan has broad experience with user interface designs, working on all popular platforms on a regular basis. I will now pass the controls over to our presenter so we can get started. Dan?

Dan Jeffrey: Thanks, Wendy. Hello everyone. This is a presentation for technical people, actually more specifically for programmers, to learn how to develop mapping applications that display dynamic data using IBM ILOG JViews Maps. So, our objectives are to give you a hands-on introduction -- this is what I'm calling a "nuts and bolts" review of the process developers follow to develop these kinds of applications.

And fair warning here, it's light on theory. I know all of us are sort of either in one camp where we like the hands-on nuts and bolts stuff, and others prefer the declarative sort of architectural view. This is not the latter and information is available though to teach you more about the architectural view and if this sort of doesn't make -- if you're a little uncomfortable with that my apologies in advance. Please ask questions if I get too far down into the weeds without giving you enough architectural background to understand it.

But I think it'll be helpful and it gets you over a lot of little hurdles that you might encounter when developing these applications. When you're done you'll understand the base classes involved for doing 2D graphics and mapping applications. And there are tools that we provide with this suite of products that you'll understand when and where, how to use those when we're done. So, that's good. We're going to whip through this very fast but we're actually going to cover how to develop a desktop application and a web application. Applets, of course, fall very close to the architecture for desktop, so those people will be ready to go as well.

We assume you know Java and that you have some kind of understanding of computer graphics. Although, I bet you could get by without that. We also assume, and this is a big one, that you know J2EE web tier architecture. But I will cover some of it. Again, more from the nuts and bolts view. However there are excellent documentation and tutorials on JSF and J2EE web tier applications available on the Internet, and specifically actually on

developerWorks. Some of the best documentation I have found on J2EE in particular on JSF, I have found on the developerWorks from IBM. I'll show you some links later, too.

So, I forgot this was an animated slide. Let me just get it on the screen because I think animated slides are aggravating for the audience. So, there is a sweet spot for doing mapping applications and this is important because we have competitors or you know -- not really competitors, actually, because we don't compete with companies like ESRI who do an excellent job of providing a framework for doing mapping applications.

And unless you fall in the sweet spot for a JViews Maps application, it's usually better to opt for the alternatives, which almost exclusively are covered by ESRI. So, the first thing to make clear is when and where JViews Maps applies. We are used in applications where there are some sort of very rich graphics required to depict customer-specific data. And these graphics need to be animated. In other words, if there's no animation on top of the map, if it's a static map view, move on. This is not the right product. ESRI has better tools for doing that and Google Maps has some as well.

So, animated graphics depicting custom data. And typically these animated graphics will be fairly rich. You'll show more than just a pin in a map or something moving around. Usually it's showing state as well. You'll see some examples and this will become more clear, in fact, in the next slide. If you have those requirements though, I believe we have an excellent way to attack them as a programmer and I'm going to show you how to do that today.

So, here is a slide showing an example of what I just described. Let's imagine you have GPS data streaming in from somewhere. We usually by the way, are used by talking to databases or web services. That's the most common use case. But we can talk to data anywhere. We use the Java APIs to load things up so when I show you examples of data coming from a GPS or a database, don't think of that as the only use case. We can take data from anywhere, literally.

Okay, so if you imagine GPS data streaming in and you've got a map on the screen with a number of objects, in this case just one, but typically you'll have more. Also in this case we're showing an icon. Very sad that this screenshot isn't animated. It normally will show more than just an image as we're showing here. We'll have text and decorations on that image.

But that would be a very typical JViews application, this Jeep moving around the screen. And I'll show you examples of that as we go through. So, what does -- assuming that the requirements are a fit -- your requirements are a fit for JViews Maps, what can we do to help you? We offer tools for doing map preparations to do complex layered map overlays that you know you're going to need at run time. So, you can prepare a map -- a background map -- using these tools and it's fully optimized. We'll talk about that a bit more but just to make a big subject small and simple, if you use a map builder you know at deployment time what you need is a map background. Map Builder is capable of producing that in the most performant way possible.

Memory consumption, load times, all that is very nicely handled by Map Builder. Okay, so then -- but regardless of whether you know it at run time or not we also have tools for doing

loading of maps at run time, in a dynamic way, at user request. And both of those scenarios, we do a great job or well a perfect job in my opinion literally, of minimizing the memory footprint and maximizing performance for the user. This is a key issue in mapping applications and we do a great job of it.

And then we layer, on top of that map, symbology. In this case in the lower right of the slide you'll see some little icons. This is a network diagram and one of them is showing red so, it's indicating a state change of some kind -- an alarm state. We provide facilities for doing that. In addition, and this is specific by the way for the defense product. There are two products; JViews Maps is part of the JViews Enterprise Suite. But there's another suite called JViews Maps for Defense. It layers on top of everything outside some terrain analysis capabilities including 3D views.

We won't spend much time on those today, but you can explore them using the Map Builder as I'll describe shortly. And also in addition to desktop applications, we make it very easy to produce a web application with specific user interactions and that sort of thing, and we will cover that today. So, in the end this is a class library. Make sure you understand that. We'll be talking about some applications like Map Builder that are tools for developers but you can throw all that out the window and we've got -- without all those things, that would make it easier -- but without those you still have a very rich API for producing mapping applications as a programmer.

And on top of that the symbology management stuff is a generic 2D graphics management framework. So, this is really making us unique. You can get a lot of mapping tools and frameworks and development tools that are good putting maps on the screen, but in varying degrees, not as good as we are at animating objects of very rich and intricate data objects on top of that map. So, let's get started with the details. Enough overview.

When you're developing with Maps for Defense, there are four key steps. If you remember this slide and then you get the evaluation you'll be many steps ahead. This is the -- consider it the author's message for this presentation. Okay, so the first step is to create or import maps into the Map Builder. I've never seen a mapping application where at least one of the views wasn't known at deployment time, where the background map that is for one of the views, was not known at deployment time.

And it's that kind of thing that we're talking about here. You're going to import maps to use either for your proof of concept or for your users to start with as a map background. Again, APIs enable you to change that background any way you want to or any way your user wants to at run time. But this tool is still important for a starting view even in the simplest case. Right, then you're going to define symbols to represent your data objects. What I do is I take a sample of the data that I'm going to display. I try to have it to be very representative and I either print it out or put it on the screen to the left of my main screen, and I look at a list of the data and then I start thinking about now, what kind of graphics do I need to depict this.

And I write simple little PoCs that don't have to be real pretty but assuming you're writing a commercial product then I recommend you do another step in addition to what I'm talking

about and that is to mock-up representative graphics very simply and hand that to an artist and have an artist create the graphics that we'll bring into the symbol editor for creating dynamic objects at run time. This is a great use case and probably the biggest value we can provide to you. If you take advantage of it, it will be a good thing. I have some questions I'll take a moment to answer here. Let's see.

Can ILOG Maps integrate directly with a Google Maps server so that Google provides the maps and layers and then ILOG provides the displays and the data overlays. The answer is yes. Google Map data will -- actually I'm not -- I'm a little rusty on Google Maps server, but we read the Google Map file format. So, one way or another, the answer to that question is yes. I've only used static Google Map files myself. I think the extension is kml. Anyway, but the point is that's exactly what that capability was designed to do because the Google Map server is a handy way to get map data.

Can you work with an RDF Owl especially in N3 Turtle format? So sorry I don't know what N3 Turtle format is, but yes, RDF Owl was the basis for the symbology layer that we created so that data is very easy to work with in our -- in fact I we even have a data model. We have this layer for symbology and I'll talk about it in a minute, and one of the instances of -- one of the implementations of our data model is an RDF data model. So, I think you'll find it fairly easy to do that. But I'm going to give you advice later that might not totally agree with that, so, I'll come back to that when we get to the data screen.

Where can we get the software to play with, is it free? And the answer is yes but the path to it depends on -- well the easiest way is an evaluation through developerWorks and there will be a link for this later in the slides.

I believe I’m caught up on the questions. If not, could you please -- yes, I am caught up. So -- but if not, please ask again.

Right, so here's an architectural diagram. I said no architecture but even a nuts and bolts guy like me still needs a little bit of context, so let's have it here. This is a really good one. This isn't an idle block diagram that I just threw up to have something to talk about, it really kind of concisely combines everything you need to know about the pieces of the puzzle for producing this app. We'll go through these in detail but I'll go through it now quickly.

Let's assume you have some raw map data as shown in the upper right. You will pull some of that into Map Builder. The purpose of that is, well twofold, but the primary purpose of that is to prepare maps that you know at deployment time your users will need when they're using your application. Assuming you have existing graphics and in the scenario I described where you're hiring an artist to create them, you'll import those into Symbol Editor. Even I though, as a developer, I prefer to use Corel Draw or Adobe Illustrator is another option, or there are free ones like, I'm blanking out. There's a pretty good free vector drawing editor out there in Open Source. They're a little easier to use. Symbol Editor can be used to do all the drawing you want, but it's not as -- I find not as easy. So I recommend a third party graphics editor that does vector graphics.

You don't want to use Photo Paint or a paint program because those produce images. We're not talking about images, we're talking about vector graphics. All of them -- all the tools that do that can export to svg and that's what you want. Then from Symbol Editor the graphics go into Designer. It's a declarative tool I'll talk about in more detail, but it's used to define a view in a declarative way.

You don't need to know all your data. Usually you're using a sample of data in the Designer to define the view and instead you follow the bottom line, around the bottom of the slide, from the application data model up into the APIs where the output of the Designer is used to render graphics. Don't worry too much about details. We'll spend a lot of time describing them. So, you're going to write code in the end to pull things from all these sources and have a running application. And then when it's running, these are the layers -- this is a bit of a gratuitous block diagram but it does show you the layers of the API that are in play and how they sort of stack up.

Okay, so first step, Map Builder. This is an application that would -- that justifies a presentation of its own and in fact, there is one recorded and a link to it is shown on this page at the bottom. But, it serves actually three separate purposes. The primary one I've talked about already is to prepare maps that are static, that are known to be needed at run time and it does this in a very performant way.

I wish I had time to drill down on that. It is to me amazing. There's no compromise. If you do need a map that the Map Builder can produce, there's no compromise on performance to create it here. It handles loading data on demand in three different ways. I'm not drilling down on that today, so sorry, but there are three different dimensions on which you can load maps on demand and it covers them all. It's really kind of amazing.

And it's not hard to use. So, that's a great way to prepare maps. Also, it's a great example of the capabilities of the map product. So, if you're wondering, can JViews Maps do this? Can it do that? A really kind of good place to start is the Map Builder and just sort of get to know its capabilities and that will give you a very good view of what the API can do. There are exceptions but they are rare. And the third thing is that this is a code sample and it's the best code sample we can offer because it does almost everything.

You'll see sort of a big stack of tabbed controls to the left of the main view there on the screen. Those tab controls are each great code samples to just copy and paste into your app. I do it all the time. I don't -- with other products we sell, I have to create my own little samples and show people this and that. Almost everything you want to do with JViews Map Builder is done -- I mean with JViews Maps is done in JViews Map Builder.

So we're going to talk a little bit more about how it's used but we're not going to really give you a comprehensive picture but let's talk a bit more. Okay first of all, so, we can view - we call it fusion here a fancy word, we're bringing data in from all kinds of different sources. And this is an advantage if the data is known before you deploy your app. So, you can bring in vector data like shape files, tiger format, map info format, VMAP data which is also called VPF sometimes from the US government. DAFIF is used in some defense and what do you

call, like a TSA kind of application -- I'm using the wrong term, but all kinds of aircraft management applications use DAFIF. Then also you can use GeoTIFF is a great format for bringing in really colorful background maps. I think this is one of the things I see people skip all the time and it's so sad, is if you get a real colorful background map, users will respond to your app very differently.

Now, I know it's not substantive but this is a reality that users, especially when they're new to an application that good graphics can make a huge difference in their perception of its value. You still have obviously, you have to have some substantive value but if you want to get a head start use a really rich GeoTIFF background and they're available all over the web for free so you don't have to pay for these things.

Okay, so anyway, GeoTIFF is a great format. I'm drilling down too much. I mentioned KML and other Google Earth formats are available. DTED data is a very important format for doing elevation data and other things. DTED is used not just for elevation but to depict things like soil contents and other things that are--can be depicted with a bit in the map. So, it's a good format for a lot of things but actually for elevation data there is a better one -- I'm trying to remember it. It's a great little tip if I can remember it. I'll have to come back to it. Oh, I think it's GTPO30. Yes, that's it. Then the last one mentioned here is if you can get the same data in that format instead of DTED, it's more performant.

You can -- because a lot of DTED data can take a lot of memory. So, that becomes an issue. Remember there's a faster format. Sometimes you might have to pay for that data. DTED is good because it's frequently freely available in all countries. Most governments produce DTED data. But GTPO30 is a good alternative performance-wise. So, let's say now we've got our background maps. We set those aside. We're not going to use those for a few minutes. We're going to now go create symbols to overlay on top of the map.

You know I kind of skipped ahead here. I wanted to show a screenshot of -- you know I didn't include it. Let me jump around here for just a second to give you some context for our ongoing discussion. I want to find the screenshot of what we're producing here and this is it. I'm jumping ahead. I'll come back, sorry to be erratic like this. This is a sample application that I used as the context for all of our discussions, all the steps to this process and I broke my own rule and used a really plain map as a background. My apologies.

But it's depicting a bunch of icons on top of a map connected by links -- that's not always the case and sometimes things are moving around. But for sample applications, to keep the code simpler, I usually avoid animation. You're all going probably be doing animation. It's easy to handle. It's not really specific to maps. But in this case I avoided animation so I added links to show some sort of a distribution network. Anyway that's the application I'm thinking of when I describe these different slides.

I jumped to the wrong spot. Almost there. Oh, I see, I'm sorry I didn't show you the right slide. I'm going back now to do it again. My mistake. So, let's push to audience, here we go. So, here is the screenshot I was just trying to talk about. Give you a minute to sort of soak it in. It's a web-based application showing a network on top of a map of the US. Okay, and

back to my symbol editor. So, step two is to create symbols to populate that. We added some green and red and different kinds of icons on top of them and they're very typical animated sort of data. And so we have this thing called a symbol editor to create them. Now think of it as producing -- a symbol to me is like an icon that's alive. It's a vector graphic intended for animation, let's say it that way.

So, one symbol is usually associated with one chunk of data in your application. So, it could be a jeep, it could be a computer -- desktop computer in a network diagram. It could be an airplane flying over in an air traffic application. But it represents a concrete object in your application. So, you'll draw things to depict it. Or you can pull things from our palette and use them.

Or you can pull things from our palette and change them and save them to your own. The output of this Symbol Editor is a .jar file. We call it a palette, because it's not just graphics. It's a vector graphic plus a list of parameters -- think of them as data syncs if that term helps. I like that term myself. They're data syncs that your code can later manipulate. Not directly, I'll show you, but indirectly to produce animation.

So, you'll add something using this tool and again, I wish we had time, but there are other presentations -- there's a link shown here -- that give you all the details. I can, though, to give you the picture here, I can say, okay I want a needle on the middle of this graphic. I can draw a needle or I can pull one from somewhere else and paste it in. And then I'll say, okay I want this needle -- here's a new data sync, and it will say speed let's call it speed. I'll make up any name I want. I'll call it the float value and its got a minimum of zero and a maximum of 200 and I'll say, straight up and down the default value is going to be 100 miles per hour.

And when the value changes now I want it to rotate the needle about -- its axis and axis happens to be at the bottom and now I've got a speedometer. This is the kind of thing -- this is a more complex kind, but the kind of thing you can do here. Anything in 2D graphics can be declaratively set up as a reaction to a data change for the symbol. When we're done here we have a .jar file. The next screen I'm going to show you, what we do with that .jar file and with the map file we created previously.

By the way I skipped it; the map file is in a proprietary format. It's good to know that. It's an .ivl file. That's why the Map Builder is used for maps that are known at deployment time. I'll show you how to do the run time case as well. Okay, so, I missed something. Oh, I see I didn't push that last slide to the screen. So sorry. I’m new to this software. So, this is the Symbol Editor I was just talking about and the link at the bottom is the one that shows you in great detail how to use it. So, now the next step in the process is the Designer.

The Designer is to Java code what editors are to HTML pages. That's kind of convoluted. Let's try again. The output of the Designer, the important output of it is a CSS file -- a cascading style sheet. We stole the idea from HTML where cascading style sheets are used to match tags in an .html file, like an h1 tag for a heading and declare how they should be depicted graphically -- make it red, make it this font or that font and so on. We use exactly the same idea. Exactly the same concept only we're not matching HTML tags, we're

matching entities in our data model. We have a class called ildsdm model and we go through that file and say, okay give me the next object, and following rules in the CSS file, we render it on the screen and animation then results.

So, the tool -- you could write the rules yourself and sometimes that's a good idea, not often, but sometimes it is. But more often you'll use this tool called the Designer. It's an application we deliver with the library to set up these rules. And you can say when there's an object in the data model and it's of type ABC, and it has a value of X and the value of X is greater than five, then make the background of this graphic red. That's one example and you can do this very slowly and iteratively to bring your application alive without a lot of -- the code equivalent of this is enormous because you have to consider all the different cases. You end up with huge if blocks and this is a better way to do it.

It's a declarative tool for setting up the graphics. It's a great tool. Okay, the output of this is a CSS file as I said before, also a project file. I'll talk in a minute about that. You're not going to use the project file very often. And by the way, when I use this tool, the documentation will tell you, oh yes, use the tool to connect to your live data. I don't find that often works and I'll talk about how the code works better if you do it a different way. But to make use of this tool you'll need some kind of data for -- to play with, to write your rules and that's again where I come back to the sample of data that I talked about earlier for the Symbol Editor.

I used that same sample in here as my data source. I kind of like using XML files but you can use flat files. You can query a database -- all that sort of thing. And those declarations can be used at run time. I don't want to mislead you. But more often the data is very dynamic and the ability to connect to databases in this tool is somewhat limited. For example union statements are not supported. So, you can't connect to two tables to produce one big table of data to project, predict -- to display as nodes. And a few other things make it more complex. I like using the APIs and I'm going to show you that very shortly.

So, the output here is a CSS file. That's what really matters to us. Actually I forgot one key point. I'm going back one slide. One other thing you do in the Designer is you declare what the background should be of the diagram. In this case what we're looking at now, it has no background. It's just a white color background. But what we want to do for our application is say put a map background on here. And in fact also in Designer we'll tell it, use these fields from our data model as long values for positioning objects, and then you're done positioning objects. Believe it or not, that's all you have to do. It's a great way to do all this stuff that used to take a ton of code. It's very efficient.

Alright so let's say now we've got a diagram with a background -- oh and this is a quick description of the rules and in the interest of time I'm going to refer you to a link where you can learn more about the Designer on developerWorks and here it is at the bottom of this page. This is reiterating some of what I've said. In this case it's using an XML file for the data and it's saying -- sorry I didn't explain, the iDPR file is a project description file and this slide shows the contents of one of these. Just so you understand the mechanics of it. So, what I do is I use Eclipse for my development environment. I'd create a data folder and I save my project into that data folder.

Now my code is only going to use the CSS file but this gives me a great way to debug and iteratively work on the graphics by opening that iDPR file directly from Eclipse using what they call the system editor in Eclipse and then I bring up the Designer for JViews diagrammer and it's a great way to develop with these tools.

So, now it's time to write some code. In the end we're going to end up with this web application that I've shown you previously. But first we're going to show you how to do it with the desktop because it's part of the process. Our web components use Swing to render the web pages. Now on the client side, there's no Swing, there's no Java, it's just Java script, HTML, pure web browser application.

But on the server side we're using Swing to help to support that and I'll show you. But first let's talk about desktop apps. Now here's the same application, not the prettiest application, but a good simple example. This is a desktop application. So, let's talk about this code. The first think you're going to encounter in the documentation is the class called IlvDiagrammer application. If you can use that it's good for POCs. If you can use it for your deployed application, great, but in most cases it's too limiting. So, I'm going to show you code that's more -- that's closer to a real world application.

The slide shows a couple of lines to create a Swing panel and then we create a diagram. This is a core class of everything we're going to talk about for maps. IlvDiagrammer is the bulk of what you see in the screenshot there. Everything below the tool bar is IlvDiagrammer. It's a Swing component. Once we have it on the screen it's empty, it's going to produce a white screen so we have to add data. So, we call this "get engine method" and it's kind of -- sorry, sorry, I digressed. Before we get to the data model, let's talk about getting the map to render correctly.

We have to add this one line of code to set the coordinate system for the diagrammer and it's a cut and paste it, you're done. This assumes you're doing a geographic coordinate system. The details of geographic coordinate systems might be interesting to some of you and they're covered extremely well in the documentation. So then, I also want users to be able to move things around in my application. So, I said set editing allowed, true. I added a toolbar. It's predefined. Diagrammer has a great set of toolbars and so forth. IlvDiagrammerViewBar is what you see at the top of that screenshot at the upper right.

And then I set a data file. This is the easy sort of canonical way to use the product by using the whole project file that was produced by Designer. It's an iDPR file. We're going to deviate from that though, because the real world applications do things a little differently. I'm going to show you some background information to support that. So, underneath everything is something called an IlvSDMModel. This is an interface for storing any kind of Java data. I'm going to argue that you want to do it in a particular way, but rest assured we have a whole family of implementations of IlvSDMModel.

Once you've loaded data into that model, the diagram component will use what we call the SDMEngine to render those data objects as graphics in what we call the view. Well, IlvSDM

view, I guess we don't just -- that's not a -- that's a generic term. But our term is IlvSDM view. So, by programmers loading and manipulating the data model and creating a style sheet feeding in here from the bottom in the diagram, they produce an animated application. It's a very elegant, in my opinion, way to do this and we've really nailed it I think with this architecture.

So, let me show you how to use it. By the way, the model -- this is confusing to a lot of people -- the model is owned by something called the engine, which is owned by something called the diagrammer. So, IlvDiagrammer is a method called getEngine and with that you can get or set the data model. That little indirection confuses people so I always make sure to point it out. Right, so now when you have your reference to a data model -- oh in this case we're doing it the other way around. This is an initialization code for an application where we're creating the diagrammer and then the model on line two.

The third line is starting to create a node. And I'm using a class here -- generic. Oh no, this is an interface class. IlvSDMNode is an interface. But we can tell our model to create a node for us. And most of the time you can write your own model to create any kind of node you want. So, you can actually just use the interfaces and completely wrap objects you've got in Java. We have all kinds of classes like Ilv, I think it's called Ilv JavaBean data source. There are lots of classes to support all different scenarios. I have to tell you I don't use them. I use something called IlvDefaultSDMModel all the time.

When you call create node on that model you get something called an IlvDefaultSDMNode or when you call create link you get IlvDefaultSDMLink. And I just use those classes. They're very light weight and typically what I'll do is wrap them or extend them to embed, to create my IlvDefaultSDMNode and I'll embed one object in that, that is of the type from my real world application. And then I just write a few methods inside of there -- I'm digressing quite a bit but it's useful. I write a few methods inside of there to get properties and set properties on those objects to support the SDM system.

So, in this way with a very thin layer, a facade of top of my own personal -- the classes I've written previously, I can create a diagram view and it's completely a two way -- data feeds back to my live objects in other parts on my application and it reads in directly, they’re all connected. So, actually the short version of all that tirade I just gave you is remember that IlvDefaultSDMModel is almost always the best way to deal with these data models.

Okay, so we're moving fast. Let's see. The next step -- let's -- what I've just shown you before will put a diagram view on a screen in a regular Java application. Now let's talk about getting it on the screen in a web application. This is an architectural view of JSF and I don't remember any of this. When I need to remember it I come back and look at it. Most of the time I'm working with a few APIs in the JViews diagrammer and the JViews maps libraries and I don't have to bear this in mind. I'm not doing a lot of stuff on the server side. And part of the beauty of that is that JSF makes it easy.

With JSF you have a plain Java object that you define on the server side and your web page goes and talks to that through this cycle we're looking at here. It goes and talks to that and

either gets information back or re-renders the view or both. The view itself is a tag in a web page. The model is a JavaBean on the server and it's usually contained within one of these POJOs, or plain old Java objects, on the sever that JSF supports. So, it's a very simple application for most cases. Or at least the first rendering. No graphical applications are simple in the end but from a programming perspective, it's really a pretty simple path.

So, your tasks when your creating a Java, a JSF application using JViews, are to add JSF to the project and I'm going to show you a diagram of this. There are a bunch of .jar files that you're going to dump into this folder, WEB-INF/lib. You'll add some content to the web.xml file -- I cut and paste it. I'm really -- I've never become an expert on all the declarations in a web.xml file and I don't have to because I can cut and paste.

Then I'm going to write a JavaBean -- you're going to write a JavaBean, a plain old Java object with methods that will react to user actions on the web page. Will also provide data for the web page -- we'll see examples of this, it will be concrete in the next slide. Then you're going to write the web page, a JSP that adds some JSF tags to support -- or to invoke rather -- the libraries from JViews and also it pulls down -- when you do this -- it pulls down a whole bunch of Java script to support user interaction.

So, your part of this task, at least for the first round until you start customizing things, is really simple and I'm going to show you these simple examples. And then you're going to configure JSF and the required step is you write a file called faces-config.xml. Again I usually copy and paste but these are so simple you don't have to that in this case. For the getting started example without a lot of fancy rendering and stuff going on, and in fact for some many actually deployed applications, all you need is what's called a backing bean or the more accurate term is a managed bean. I'll show you that as well.

And then when you're done you build all this into a war file whose structure you'll see in just a moment. Right, here you'll see it in fact. So, this is a picture from my Eclipse development environment of the application I've shown you a screenshot of previously. On the right hand side here I'm going to go through these folders and talk about what's in each. And the key pieces here though first are our class files for each backing bean, in this case if you can squint you'll DiagrammerBean.class in the upper right there under the WEB-INF class's folder. That's actually in Eclipse. It gets built for you. Most development environments will move that class file there for you from the source folder when it compiles.

And then we need a license -- oh, sorry, this is an old slide. You don't need a license file with JViews Enterprise or JViews Map for Defense. We don't have a technical licensing mechanism any more and so you don't need that. So, skip that. By the way there is a declaration you do need. It's described in the documentation under the introductory section where you add a static declaration in your application to acknowledge -- its says that you realize there are deployment costs sometimes for JViews and by putting that in your code you're acknowledging it. So, when your application doesn't run the first error you're probably going to see is "no license." That means you need to add a static declaration to your application.

Then, so let's see, then we're going to create some web pages. They are JSP usually. It can be XHTML or whatever but I always use JSP. That's the most -- I think almost everybody does in JSF programming. And so you'll see those here at the bottom of our web content folder. This folder that you see at the right is the complete deployed web content folder. And really you can think of it as the contents of a war file that's being deployed to the server. So, at the bottom you'll see index.jsp, map1, map2, and so forth.

Then we're going to add some data. I have a data folder here. That's what I like to call it. Call it what you want but it will contain the CSS file. That's the most important piece. It will also contain the background image map, that's an .ivl file and sometimes other files described in the Map Builder presentation referenced earlier. In addition it will contain other graphical support things like .svg files or that sort of thing. And then I also have you'll notice here an image file where I keep HTML-specific images because I don't like mixing them up with my data images. It can be confusing, at least for me.

Then there are a bunch of JAR files here to support the JViews graphics and they are shown at the right here under WEB-INF/lib. But in practice I don't usually worry about all this stuff so much. I'll copy an application I have that's working and change it. On JSF programming, in fact with all web programming in my experience, working from success is the only way to go. It's overwhelming otherwise. Let's say you do all that that I've described. Here's sort of a mechanical view of what's going on at run time when the user brings up map5.jsp, the JSP page at the bottom of this slide.

Now let's work backwards. The user opens that to look at your application. There's a tag in there called jvmf:mapView and jvmf is what we usually use for the tag name. It doesn't matter. It's a jsf convention established by JView but the type of the tag mapView is the important part. Java script is provided that's being downloaded automatically by the structure I just showed you to support that mapView tag. I'm giving it an ID so I can reference it in Java script and in other tags. Giving it some dimensions in the style but here's the key, the key to all of this web stuff is that I'm setting an attribute called diagrammer equal to this thing called diagrammerBean.diagrammer.

If you hear the background noise, that's my phone, sorry for the background noise. So, that's actually -- diagrammerBean was declared in my faces-config.xml file. I said here's a managed bean. I'm going to refer to it as diagrammerBean but I'm instructing the JSF architecture to always pull up this class called package.diagrammerBean. And so I'm saying here's a Java class. Every time I reference something called diagrammerBean in my JSP files, you go and get this class called diagrammerBean. That's shown up here at the top. And I have it in diagrammerBean.java.

So, that's the connection that hooks up the background, the POJO, or the managed bean on the server side, also called a backing bean sometimes, on the server side JSF with my web page. That's the plumbing. And it makes web programming very easy because I can save a state in my backing bean and use it on the server without a lot of plumbing, a lot of effort. What this let's me do -- now this is a super simple example. It's not really doing anything interesting. We're saying get the diagrammer instance from the backing bean and we’re not

doing anything with it in this code sample here. The most interesting case is when you need to get data from a database or more commonly, poll a database or poll a web service to look for updates in the state of some system and when you get updates you have -- you're getting those updates on the server side. You're running code on the server side to do that in a timer or whatever is convenient. And you can think of better ways than timers but anyway, that's the simplest scenario when -- a good one to think of.

We've got a timer, it's going out and looking at some external data source and is getting new stuff each time it looks. What's it going to do with it to update the view? It's got a reference in that backing bean and it can say, okay diagrammer give me your data model, as diagrammer get engine get models, you recall from a previous slide. And then we manipulate that model using IDs. We can either update objects or if they don't exist yet add them. And this produces a very nice simple programming model for animated diagram applications. And remember diagram applications are map applications because we -- just using a CSS file, we set the background of the diagram with a map.

The mapView tag, I didn't reference previously, is actually creating a diagram. I think that's obvious but I should say it explicitly. So, the mapView tag extends the diagram view and it adds things like layered -- a layer of visibility controls shown -- it was in the previous screenshot. Yes, it's the middle one, layer of visibility with a red title bar just above the main map you see in this one. We also have an overview control shown in the lower right. That always shows the whole view that you've created and the user can then zoom and pan in the main view or they can do the same thing, they can pan using the overview control. Its got a square showing where the main view -- it's connected the main view. It's got a square showing what the main view has visible and you can drag that little black rectangle around or you can make it red or any color you want. But you can drag it around and the user -- it's an excellent way to keep a global view of the whole map.

Okay, let's see. So, there's also a zoom tool. It's an interactive way for -- somewhat interactive way in HTML to zoom the main view. We also have a toolbar to support things like Pan and Zoom and Zoom to Fit and all kinds of basic user controls. But also we support popup menus. We provide classes to make that easier. Usually you'll want to start from a sample because that gets to be a bit complex. It's not just a matter of dumping a tag into your web page. There's some backing code to support it and so you look for examples in the product and copy those for starting points for yours.

But all kinds of extra stuff is given for the web-based applications and this is a pretty comprehensive list of that. So, here's a little more detail on what the JSP page looks like. The tag declarations that need to be at the top are shown at the bottom ironically of this slide. And then the main tag though is mapView tag shown at the top of this slide. We've been through that in that rather complex diagram I showed earlier of how this all works. It's good to know, getting started you don't need to know this, but it's good to know that JView spaces uses servlets to generate the views.

So, the web.xml will contain in addition to other things the tags you see in the bottom of this slide. So, behind the scenes when all the JSF stuff is working for you it's going out and

invoking a servlet to generate the image. And there are cases where you need to extend that servlet and there are examples of that in the product. But especially for getting started, and for many finished applications, that's not necessary. It's just good to know it's there and when you see references to it, if you're just getting started don't worry about it. Worry about it later when you get into more complex requirements.

So, in summary, building a JViews maps application is easy for animated data or the intention is for animated data display over a map. We call them asset maps. In other words something is on top of the map. It's not just a pin in -- we're trying to talk about, show information about, an object that floats on top of the map, not just sticking a pin in it. And it's also not for static reporting things where if you don't have animation, you know, this is not the product. It's overkill for a non-animated application.

When you get started then, you use IlvDiagrammer to display your map and you'll populate it by first preparing background maps with the Map Builder but then defining symbols for your data types then by animating those symbols, in other words by connecting them to your data using that thing called the Designer. Those three steps are described in other presentations on developerWorks as well as anybody can do it and so I recommend those instead of wasting time here. I didn't choose to spend a lot of time here on it because those demonstrations are excellent and they'll show you how to use those tools.

Once you have these outputs from these tools, you'll define a view in code either for a web page or in Swing for your desktop application. And sorry, I misspoke. This is -- the point here is saying you'll define the view using the Designer and specify a background map and you'll assign symbols to data using the Designer. Then you'll write the code to invoke either the CSS file or the full iDPR file to produce your finished application.

So thank you for your time. I haven't received any new questions -- oh yes I have, so sorry. Haven't been keeping --

Unidentified Speaker: Yes, there's two new questions.

Dan Jeffrey: Thanks, Wendy. Okay so we have one question. When the user uses the software, does the user have to download it and install it? No, the user library. So, the output are some data files, the output of the tools I've described are data files. You'll bundle those into your application and your application will include the JAR files that JViews provides and then for web applications, the final output is something called either a war file, w-a-r, which is web application re -- I don't know what it stands for. And then -- or it can be an ear file, enterprise application resource, whatever it is. Wars and ears are J2EE standards. So, you'll have one of those files when you're done.

That gets deployed on a server. That's all that happens. If they're doing desktop applications, you'll build -- the typical --there's a lot of ways to do this. But typically you'll end up with a JAR file, Java R file -- I don't know what R stands for, resource I think, and -- that contains everything. And you'll just deploy that JAR file. Sometimes you'll deploy it with a jvm but your customers or your users have no notion that IBM ILOG components are embedded in

those files. It's your application when it's done. And that's why we see a lot of applications developed by third parties and we have OEM software venders who embed these products. In fact, before we were part of IBM, IBM was our biggest customer because the Tivoli products use these libraries just all over the place and I think that's a good testimonial for the kinds of things we can do because those are great products.

Another question. Do you have a demo application based on JViews allowing us to play around and test the performance? Yes, we do and those are available on developerWorks. So, if you go to developerWorks and search for JViews, there are -- there's a link under JViews. You can also go to the ILOG JViews main page on the main IBM site, it'll get you there as well. There are things called demos and movies. And there are actual live sample applications you can play with. Some of them are web-based. Some are applets or Java web start applications that are invoked when you click on them from the web page.

One thing about the web applications, the servers that support those, that we use to deliver those at IBM, are doing a million things. So, please be understanding. Those are very weak servers serving up our web applications and the performance you'll get from our demos on the sample site on developerWorks really are a kind of worse case scenario. There's really very little hardware supporting those applications for whatever reason. But anyway that's the case. So, don't be too judgmental about their performance. It's really good to see it for yourself by building your own little sample application.

Great, I think I've got all the questions so far. Are there any more?

Unidentified Speaker: I think there's one more. There are -- if you hit your refresh --

Dan Jeffrey:  I keep -- I'm forgetting my refresh button. Okay, here we go. So, here's one, very good. In case of desktop applications how do I dynamically load maps based on the relative position of the majority of the objects, provided we are using Google maps instead of building our own? Okay, how do I dynamically load maps based on the relative position of the majority of objects? Okay, that's good. This is a common requirement. So, let's say we have -- let's start with a world view. You deploy the application, user starts it up and there's a whole world and you've got a cluster of vehicles -- I'm making up requirements -- a cluster of things. I'll just say they're vehicles and they all happen to be in the state of Nevada.

And so we have the whole world view, we don't want to show the whole world. The user is not interested in the whole world, he's interested now in the state of Nevada because that's where those objects are. You can request, with one API call, the location of all the objects currently visible, not counting the map, I mean just the -- what we're calling assets or data objects -- what are currently visible. You get a rectangle back in XY coordinates. You convert another API, recall we're up to two now, you convert that rectangle to latitude and longitude coordinates.

And this is how I would do it. I would say okay, I would create some sort of custom class to quickly figure out what maps I have available based on that rectangle. Another way to do it is extrema points -- upper left lat/long, lower right at lat/long. But basically we're talking about

maybe three or four API calls of code you've written to figure out what map you want to go load. And then we have these APIs go get your, in this case the gentleman's interested in Google maps, go get the particular map file you've decided you will use at that zoom level and so forth. Now, that's how you do it with APIs. You can actually do all of the stuff I've just said using the Map Builder.

You start with a world map and then you have individual -- you include individual maps for areas you think might be interesting at run time. The result of this is a pretty huge file. And so that's one caveat. If there are too many of these possible variations of what the user might want to zoom in on, then you might think about it twice. But for a lot of applications this is a good way to go. So, you'll load, in this example, you would load maybe a map for each US state and Canadian province, I'm imagining we're only interested in North America because I'm in North America I guess is why, but that's a good scenario, wherever you are.

Usually there's an area of interest you know ahead of time and that's exactly what I wanted to get to there. Once you've loaded each of these maps you'll define a zoom level that's about appropriate for -- in this case the state of Nevada, and I'll have that on my screen then I'll declare this is a new area of interest and I'm calling it Nevada. And you can -- that gets stored in the map file as output. So, you have these areas of interest. Now those areas of interest make it easy programmatically to zoom on that area but also they're an optimization. The image of that area of interest is actually stored in the map file so when you request it at run time, it doesn't have to be rendered. It pops it right on the screen in the background. Your assets are still being rendered dynamically but the map file knows, oh, he wants that area of interest. I don't have to go run out and use the APIs to generate it. I can just pop this on the screen and blurt it right up on the screen.

And that's one of the performance improvements I described earlier. So, the answer to your question in simpler terms is you have two choices -- use the APIs and the algorithm for figuring it out is easy to implement because we provide all the conversions from XY to lat/long. Or use the Map Builder.

Okay. Documentation on RDF Owl support, where can I find it? That's a good question. It's in the API docs. I would just do a search for RDF and you'll find the class. Now, I promised to say something further on that subject and I will do so now. I want to go back to another slide. RDF Owl support was a dream that we had when we developed the diagrammer component. That was in the back of our minds -- not our minds, I wish I could claim this -- the minds of the developers when they created IlvDiagrammer. I'm trying to buy time while I search for what I need here.

Let's try this one. So, we had this earlier. No, that's not it. Next slide. Maybe one more. Yes, this will work. So, I recommended earlier that you use IlvDefaultSDMModel even for an RDF app. It also depends, my opinion is, the amount of work it takes to actually use the classes other than IlvDefaultSDMModel is not worth the effort because this one is so simple to use in so many cases. Just by taking an RDF Java object -- and I'm sorry I don't work with RDF on a regular basis, so let's say you have an RDF -- some Java class describing an RDF object. That class can be embedded in inside of what we call IlvDefaultSDMNode. So, I

extend IlvDefaultSDMNode, say embed this other guy here and then I write a few methods inside of there to route requests of the SDMNode to this RDF object. Now you might find otherwise, but I remember investigating this once for a customer who wanted to do RDF implementations and we decided that was a better way to do it was to use IlvDefaultSDMModel.

But do please look for RDF in the referenced documentation and see for yourself but I'm just giving you this as a kind of a warning ahead. I have two more questions coming in but I have one here about -- oh, someone is setting me right. JAR equals Java archive -- web, war equals web archive, the r is part of archive. So, thank you. And let's see, I think I'm caught up.

Unidentified Speaker: I think that's all the questions.

Dan Jeffrey: Thank you for those. That really adds to a presentation when we have a lot of good interactive questions. I appreciate it. I'll jump to the end. Wendy, I've got it.

Unidentified Speaker: Okay.

Dan Jeffrey: Okay, so over to you, Wendy.

Unidentified Speaker: Great. Oh, wait a second. No, that was it. Okay, so thank you Dan and thanks to all the attendees and we know you're very busy and you have busy schedules so we appreciate you spending this time with us today. This presentation has been recorded and will be available on IBM developerWorks soon. But I believe you'll also receive a link to the recorded version as an attendee today in a day or two. So, this concludes our presentation. Thank you, and good-bye.