Title: Session 3 – Standards Based Mobile App Development

Steve Mirman: Hi, my name's Steve Mirman and I'm a IBM Worklight Expert. Today, we're going to discuss standards-based mobile application development using the IBM Worklight.

So, what is IBM Worklight? Worklight is a mobile application platform that's going to give you capabilities to write web-based, hybrid applications, native applications. It's going to give you the full capabilities to test the applications, and even the development lifecycle tools. One important thing to note is that it's a completely open platform, so it's based on HTML5, CSS, JavaScript, but it still gives you the capabilities to write a completely native application, or partially write a native application, if you have an application that majority HTML, CSS and JavaScript, but a few pages that go into the native scope when there aren't any APIs available for that. The second main component to it is the server side aspect, so this is where we're going to bring in some of the security model. We're going to bring in encryption. We're going to bring in local storage on the device. We have a whole set of adapters that allows you to communicate with the back-end. We have analytics, push notifications. There's many parts that provide the middleware aspect that really makes Worklight a differentiator between other application platforms in the mobile space.

So, let's talk about the five components that make up Worklight. First, is the Worklight Studio, and the benefit to the Worklight Studio is that, again, it's completely open, so it uses HTML5. It uses CSS3. It uses JavaScript. It integrates with the SDKs that are specific to devices, so IOS, Android, Microsoft. It allows you to bring in any third party libraries that you like. So, if you're a Sencha user, JQuery, Dojo, if you want to use node.js, whatever you want to use from a framework perspective, it's completing available inside the Worklight Studio. It also has a WYSIWG editor, so it allows you to drag-and-drop, get look and feel, do quick preview in applications, and you could see based on the SDKs, the number of different environments that are fully supported. The second part is the application center. This is where we can distribute applications, so you can get, you know, IOS IPA file, or an Android APK file, make them available to users who manages those distributions, control who can or cannot see them, including even a rating system of the applications that are deployed. The third part is our device runtime. So, this is the physical device itself. Worklight's going to give you capabilities to have a fully encrypted storage that can be synchronized with the back-end. We can have runtime skins, which means you could get a different look and feel based on certain aspects of the application, location-based handling, and even the idea of communicating with the back-end to get the information from your Worklight Server, streamline that data and get the response back. The fourth part is the Worklight Server, itself. Here's where we bring in authentication, and whether we want to authenticate that this is a valid device that's making a call, or it's an application that's signed by the server properly, it provides many layers of security that can really protect any of the back-end calls and any of the back-end services that you want to expose to your mobile devices. It also has, as part of the adapter framework, mashups, and service composition so that you can have multiple database calls yet return one response. You can streamline those responses so that you're only returning a very small JSON response that’s completely applicable to the application. And, what that means is that I don’t want to return a thousand lines on an adapter call when realistically my device can only display ten. Let’s streamline the results on the server side, minimize the packet back and forth so that we can get the quickest, most efficient response. It also has the direct update capabilities, which we’re going to discuss in a little bit, that allow you to update the application after it’s been deployed from the application store, and also hosts the mobile web apps. So, if you want to have an application that’s maybe downloadable for IOS, downloadable for Android, but you just want to host a mobile web version of it for BlackBerry, Windows or other devices, Worklight gives you those capabilities. And, the last aspect is the Worklight Console. So, this gives you insight into your push notification, gives you insight of the analytics and shows you what applications you have deployed, as well as what versions and what current state that they’re in.

So, before we dig too deep into how the actual development happens, let’s talk a little bit about the models that Worklight supports. So, there’s four major models, and we’ll start with the first on the left, the mobile browser. So, browser access. This is an application where the user on their phone is going to open up their browser and they’re going to hit your website, so they’re going to hit your mobile website. This is very easy to develop, easy to create. It gives you the capabilities of kind of dynamic content and form factor so that it fits the device itself and gives you those capabilities, but it’s kind of limited because you're limited as to what native capabilities you can get to on the device. You have everything HTML5 gives you, but you kind of lose some of the offline storage, you don’t the idea of a synchronized database, so it’s a quick and dirty way to get to the mobile space in a short amount of time that’s kind of universal. Anything that uses a webkit browser gives you those capabilities. On the complete opposite end, you have the native applications. So, again, this is something that is completely acceptable within Worklight. You can even write fully native applications in Worklight. So, this is Objective-C for IOS, Java for Android, Java for BlackBerry, Visual Studio for Windows. This is running an application in its native language and still utilizing the APIs that come with Worklight, whether they’re adapters, whether they’re security, whether it’s using some of the encryption, it’s just the exact opposite of a browser-based application. The two in the middle or essentially the sweet spots for Worklight, and the reason is because you're getting the most re-use out of your code. So, the hybrid application web is an application that’s exclusively HTML, CSS and JavaScript, yet it’s running completely on the device, has offline capabilities, has full functionality into the native capabilities of the device, but yet your code base is really shared between Android IOS, BlackBerry, whatever you deploy it to, whichever environments you pick. So, you can write, you know, a call to the camera, a call to the GPS location, it’s all in a common fashion and there’s really no native code written. The hybrid apps mixed is just what it says, it’s kind of a mixed option here. So, let’s say for example IOS exposes a series of

APIs, or you want to take advantage of NFC capabilities on an Android device, something that's not universal among other devices and there may not be a current set of APIs out there available, you can write your application 90% using HTML, CSS, JavaScript, universal among all the different devices, but let's say on, maybe their Android-specific devices, you could add one native component where you take advantage of NFC, or for IOS, you could add another native component that you've written in Objective-C and take advantage of some native functionality that's specific to IOS, that's one of the beauties of Worklight, is that you're not limited to the APIs we give you. If you want to go straight native on a certain set of calls, no problem. If you want to write your entire application native, again, no problem, but we give you the option to take advantage of the functionality as soon as it's released by the vendors.

So, let's get into the Worklight Studio. The Worklight Studio is essentially an Eclipse plug-in, also runs on Rational, and what it does, it allows you to write these applications. So, it allows you to write the native application. It gives you the idea of runtime skins. This is where you're going to integrate your third party libraries. So, if you want to bring in the Dojo, or the JQuery, or the Sencha libraries, and this is just a small subset, it just happens to be the three most popular frameworks, you can bring in any JavaScript framework into the Worklight Studio, and develop for it. It has built-in auto-complete, validation, and it has access to the emulators and debugging tools. So, you can take your application, you can preview it, you can use browser-based development tools, you could push it directly to emulators or direct it to the simulators, and under the covers, it's using the SDKs that are specific to the devices. So, we'll have access to the Android SDKs. On Macs, we'll have access to the IOS SDKs. It's all based on you're writing your HTML, your CSS, your JavaScript, you're native when necessary, and the Worklight Studio is giving you the package that allows you to integrate everything.

As part of the Worklight Studio, there's a mobile browser simulator, and this is really predicated on, I want to be able to test the UI, the look and feel, of all the non-native components of an application, and when I say, non-native, I mean something that's not written in the native code, itself. So, here, we can simulate the phone gap or the Cordova APIs, meaning that, if you want to simulate GPS location, or you want to simulate the compass, or pushing the volume up and down button in an application that's using the Cordova APIs, you can simply test that in the mobile browser, and you can test it against multiple devices. So, here, iPhone, BlackBerry, Android, in this case, you can rotate them, you can compare devices against each other. It's a quick way of validating that this is how the application looks and functions under these certain conditions.

Now, the way the Worklight Studio works and the way that it kind of handles the code is that when you first create an application, what you're going to be given is, in this case we created Worklight demo application and the actual application name is MyApp. What Worklight gives you by default is an application descriptor XML file, which is going to be used to set some of the configuration of the application, but also, you're given a common directory, and this is going to be where the code is stored that's shared among the different applications or environments. So, the MyApp.css, the auth.js, the myApp.js, the MyApp.html, which in this case is going to be your main file to startup with, will be shared whether you do an Android device, whether you do an iPhone, whether you do BlackBerry, it's all based on this common code base. Now, what Worklight allows you to do is override these functions so that you get the specific functionality you want pure device.

This is where the environment optimization comes in. As you can see, we can do environments for iPhone, iPad, Android, multiple flavors of BlackBerry and Windows, as well as desktop and even web widgets. And, the way that it works is, on the right side, you kind of see a screenshot where you see that same MyApp application, the common folder's still there, but now we've added in a BlackBerry folder. We've added in an Android folder, and you can see under Android, we also have the css, the images and the JavaScript directories, similar to what we have in common, but in this case, they're only used for overrides, so the myApp.js is going to be essentially a blank JavaScript file that allows me to either add functionality for Android, or override any common functionality that I want. So, for example, if there's a Steve.function, or stevefunction inside if myApp.js in your common directory, I can override that by putting the exact same function in the Android JavaScript, and the reason it works is because all the JavaScript files per environment, and the CSS files, are just appended to what's in the common folder when an application or a final project is built. So, for example, when I want to generate my APK file, the JavaScript and the CSS from the Android directory, will be appended to whatever's in the common directory. Now, the HTML and the images, those files override, and the reason is because there's no idea of overriding HTML files. They're just sequential. And, the same with images. There's no way of extended an image.

Now, we have a set of integrated device SDKs so that we can actually write code and we can utilize the native functionality of the devices themselves. So, if we want to write something specific to Android, or we want to write something specific to IOS, we have full capabilities and full access to the SDKs so that we can utilize that functionality. And, the Worklight APIs, and the Cordova APIs, again, have simple calls that allow you to interact with those native components as well.

Now, editing with inside the Worklight Studio, there's two ways you can essentially edit the application. You can do it straight with the code, as you can see in this case the HTML on the right side, or you can do it in kind of a look and feel fashion, or WYSIWG editor where you can actually see what's happening and you can, as you're validating the code and as you're writing it, you can actually see the changes occurring as part of the application. So it makes it very easy to see what you're looking for, kind of design, make it look and get the framework looking as expected, get the wireframe setup, and then actually modify the code as necessary to add in the back-end functionality.

Now, the rich page editor, that's part of Worklight Studio, allows you to drag-and-drop. So, here, initially we have Dojo, and we have JQuery, that allow you to, once you've loaded the libraries, drag-and-drop in different widgets from those two frameworks to kind of visually create your application. You always have the option to go in and modify the code directly, but it just kind of gives you a quick jumpstart from an application perspective. You can quickly get in there, make some modifications, get it to look and feel the way you want initially, maybe do some testing and validation of how the back-end code works, and then come back and complete the UI.

So, new in version 6.0 is the idea of these screen patterns. So, what IBM did was they essentially went out and said, okay, what's the most common screens, what's the most common functionality from a UI perspective that utilizing these frameworks, and what they did is they created essentially a widget library that says, okay, you want a simple list. I can drag that in and I can start from there. You know, you want a text input, or you want a form, or you want any one of the things that are out there, maybe some authentication modules, there's a set of 40+ patterns that come with it that allow you to drag in and say, okay, I want to start here, because that's very similar to what I want my application to look like. It kind of gives you a quick start for an application to have a good look and feel, and then allow you to proceed from there and customize it as necessary.

So, Worklight Studio runtime skins. The idea behind runtime skins initially was, let's do kind of responsive design, but from a programmatic perspective, meaning that I can, at runtime, check and see what's, you know, some parameter, and then make modifications to the application itself based on that. So, for example, let's say you have an Android phone and you have an Android Tablet. From the application store, they can both download to exact same APK file, but at runtime, what I could do is I can check and say, hey, what's the resolution here, or what's the version I'm running, you know, and I can key on anything, you know, what's the GPS location, or what users login, and I could change the look and feel based on that. So, now that I have more real estate on a Tablet, for example, instead of having one screen where I swipe left and right to see the individual aspects of maybe a trip, for example, I can show the entire trip in three columns on a single screen for a Tablet, and that's all done at runtime so you push that one, single package. This way you don't have to have an Android Tablet version in the App Store and then just an Android phone version as well. Here, you just have one Android version, you push it out and at runtime it can determine what it needs, what it can expose and what it's going to show to the user. Gives you a lot of capabilities from a good programmatic perspective.

You can also build and deploy directly as native components. So, in this case, you know, we're running from a Mac in these screenshots, I can take the application directly from within the Worklight Studio, just do run as Xcode project, and it pushes the entire project into Xcode. Then, I can use the built-in simulator, I can add additional code if I want to, I can add in some native code, but it allows you to utilize, you know, the device-specific, or the SDK-specific environment that's required for it. So, here, you know, we're talking about Xcode, for Windows it could be Visual Studio, for what pretty much Eclipse covers. With Java, it covers BlackBerry and it covers Android, because those can run directly inside of Eclipse, but what it does is it lets you seamlessly translate directly from I'm in Eclipse and I want to push it to the native SDK's build environment, so in this case, Xcode.

Another new thing in version 6.0 is these functional test tools. So, you know, it's a standardized Eclipse-based tool that allows you to go through and really validate, run through a complete testing suite to do a quick testing of the application itself, and it has, you know, full capabilities of running through a set of scripts that allows you to test the functionality of your application, validate what is or is not working, and quickly record and edit that. So, it's a very quick way of saying, you know what, I need a test on multiple devices. I need to test this functionality. I don't want to have to push it to a real device. I want to run it and validate it, and it gives you that capability in version 6.0.

There's also a complete functional testing suite. So, now you develop your application in here, you create your test workbench, and you can add that project to there. Now, you could start to utilize other projects in the Rational Suite and you can take advantage of testing. So, here, we're clicking on input e-mail. We're doing certain things. We're doing a password. We're basically going through the complete scenario to do a complete functional test of an application. These are all tools that are built into Worklight, starting with version 6.0.

Now, one thing that's been enhanced or makes a big difference from an Worklight perspective is there's always been an issue with how do we, from one server, or one physical machine, how do we create the Windows executable. How do we create the IPA file, the APK file? How do we do it all from one machine and how do we do it in a secure fashion? So, that's where the Worklight build system comes in. Worklight comes with a set of (and? @ 18:31) scripts that allows you to do remote builds against a Mac, remote builds against a Windows machine. That way your developers can simply check their code into the source code repository, whatever you want to use integrated with Eclipse. From there, your build masters, people that actually have the certificates for the server side, can initiate the build. It'll go out to the Mac, it'll build the IPA file, it'll build the APK file, it'll build the Windows file. You can push it directly to the application center. You can check it back into the source code. But, what it does is it means that your individual developers don't need to all have Macs, or don't need to all have Windows machines, or any specific machine that targeted to a specific SDK. You can have whatever environment you run, whenever you're running Eclipse, and still have the capabilities of building, you know, an Apple-specific file, or a Windows-specific file. Makes it very easy to, from a developer, just check it in, let the build happen, and also from a security perspective, you're guaranteeing that that build system assigning it with the production certificate in many cases, and that way that certificate doesn't have to be distributed to your developers, themselves.

So, let’s talk a little bit about the back-end integration. So, inside the Worklight Studio, you can create the adapters. The adapters are going to be your communication between the Worklight Server and your back-end systems, whether you're calling a database, or a web service, or a JMS, or you know, mainframe. What it is is it allows you to create, using simple XML and JavaScript, your communication channel, how you want to parse the data, how you want to get the results, whether you want to merge a larger set of data and return, you know, a smaller set back to the device. You also have full server side debugging capabilities. And, we’re going to talk a little bit more about the adapters when we get to the server component, but it’s all done within the Worklight Studio. Once you create these services, or create the adapters, you can fully test them. From within Worklight Studio, you can call the real back-end systems, you can validate the results even before creating the application they’re going run with so it allows you to really segment your team. You can have a development team working on the adapters that are setting up the communication, making that call, making sure the response that comes back is correct, or you have a separate team that’s actually working on the UI of the application.

Okay, so now let’s talk about the application center. The application center allows you to distribute applications within the enterprise. So, here, what we do is you're allowed to upload, you know, an IPA file, or an APK file. You can allow users to rate that application. You can distribute it to users, and this is kind of in the exact opposite of what an MDM would do. This is more of an App Store pull type solution. So, instead of you pushing out applications to users, users are going to now login, they’re going to see a list of applications that are applicable to the device, but also applicable to the ID they logged in as. So, an administrator may see a hundred applications, whereas a generic user, or a store user, or you know, a guest, even, may only see five or six applications that they’re allowed to download and have access to.

For adding an application, again, it’s as simple as uploading the IPA, uploading the APK filing, giving it a version, giving it specifics. And, you can have multiple versions of the same application. So, I can have version 1.0 or 1.1, 1.2, and users can go in and upgrade, downgrade and manage it themselves. It gives them greater control over what’s running on their device.

So, you have a full set of rule-based administration. So, when you upload an application, you can say, okay, I’d like this group to have permission, or this set of registered users and, you know, you can setup access control and say, okay, you know what, for some applications, hey, I’ll just make them generically available to anybody, for others, I want you to have to be either in this specific group, or I want you to be this specific set of users so that you have complete management over not only what you push out there, but who has access to those applications.

So, applications can always be installed using the downloadable app, but you can also set them as a favorite. So, you can see here in this example, somebody downloaded, they took a look at it, they set it as a favorite application, and it again, you have ratings, you have capabilities of setting it as a favorite, you have a lot of control over what users can download. You can see the users that have downloaded it and what devices are targeted based on those user profiles. You have the capabilities to really serve as a full application store for your mobile applications within the enterprise. Now, when you're looking at external applications, or consumer-based applications, you're still going to need to go through the traditional Apple App Store, or Google Play. This is more for enterprise distribution, enterprise testing, a quick way over the air to distribute applications within your application, or within your enterprise.

Users can also provide feedback for these applications. So, you can see here, users can download it, they can write it, they have leave comments. You can leave comments for multiple versions. It’s really been enhanced in version 6.0 to give you more capabilities, but also allow it to, it gives it a nice look and feel so that if you're using it as an enterprise App Store, users can come in and they can give their honest ratings and you can take advice from that and kind of move forward with the application.

So, let’s talk about the application runtime, now.

So, the runtime, this is the server side component. So, this is going to run on the Worklight Server, typically in your data center, and in most cases this is going to be fronted by, you know, HTTP server or some reverse proxy running in your DMZ. So, the Worklight Server is really simply an application running on a JEE container. It gives you a set of adapters. This is where you're going to control your back-end communication. So, they support SOAP, REST, SQL, JMS, Cast Iron, Node.js, even, and you can even make it directly connect with the Java process. So, example, if you have a Java class running inside your JEE container, you can instantiate that and interact with that Java program. One of the big benefits, again, is the data transformation. So, I can make a call to the back-end and then decide, you know what, okay, here we turned a thousand (inaudible @ 24:49) from the database, I only want to send ten back to the user, or I want to run it through xsl and I want to streamline and take out certain aspects of the results, because they’re not really relevant to the device, themselves and the goal is that, I want the smallest JSON package distributed and sent back to the users so that I get the quickest response. The adapters already are going to turn them into JSON, but you want to limit the response so that you get a quick call in, a quick call back, and you can move on with the application. It also provides a full level of security, both from the device side and from the server side, so whether it’s authenticating against your LDAP, whether it’s validating the application installed is a valid application, or validating that it’s coming from a real device, or a non-rooted device, all these capabilities exist on the Worklight Server side. Another aspect is versioning and controlling the versions of the applications that are out there. So, I can have multiple versions of an application deployed, and I can manage and maintain those individually within the Worklight Server. We also have push notifications, which I'm going to address a little bit later on a slide, and a full set of analytics that kind of tell you, okay, what types of devices are hitting, what versions are being hit, are applications having issues, are we getting errors. It's all supported directly on the Worklight Server side.

So, let's dig a little deeper into the adapters. So, this is a simple adapter model. You have an application that's going to invoke an adapter procedure. That adapter procedure is going to call the Worklight Server, and the Worklight Server, based on how the adapter is configured, is either going to, you know, make a SQL call, call node.js, maybe it's a REST service, or a Cast Iron, it's going to query that back-end system to get the data and it's going to bring back the response. From that point, you can merge the responses. You can take the responses and you can alter them. Maybe you want to take it and mash up a couple of different responses and send a response back to the device, but the key is that you can actually go through and now you can streamline what you're going to send back to the user. So, maybe the user made a quick web services call. Now, you want to send back, you know, a small 10K response that is the exact information they need, let all the processing happen on the server side and view the adapters. But, this also serves to protect your back-end systems, so you're not making a direct connection from the device itself to the back-end databases. You're not exposing your web services directly. If someone were to take your application, let's say they take your APK file and they unzip it, and they look at what's actually happening, what they're going to see is you're making an adapter call and you're passing it some parameters. They have no idea whether it's a database, whether it's a web service. They have no direct connections into it. They don't see port information. All they know is that you called the Worklight Server and you called a specific adapter. So, it gives you a lot of, not only capabilities for performance, but also from a security perspective.

Another feature on the server side is the idea of the direct update. So, direct update means that any of the non-native capabilities, or non-native code of the application can be updated at any point. And, when I say, non-native, I mean, any of the HTML, the CSS, the JavaScript, regardless of if it's in the common directory, or in the environment-specific directories, all that code can be updated and using the direct update capabilities in Worklight, without having to go back through the application stores. So, here's how it works. Initially, when you download the application from the App Store, and this is assuming a consumer scenario, you download it, it comes with a set of pre-packaged resources, so the standard HTML, CSS, JavaScript, whatever makes up your application, the images, everything that came with it is basically expanded on the device. Now, when the application starts up, or on a certain interval, you have complete control over when this event occurs. By default, we typically check with the server when you bring an application to the forefront, or when you start the application, but what happens is it checks, does the package I have on my device actually match what's on the Worklight Server? If it does, great, we'll go ahead and proceed and we'll move forward. If not, then we need to update the application. We don't want to have mismatched resources. So, what we do is we download that package automatically, we expand it and we re-start the application, and you get all the new functionality. So, now you have the capability of actually doing updates without having to go through the whole process of going back through the application center, or going back to the App Store, without pushing out a new complete version of the IPA, or the APK file.

So, how does this function within Worklight, and it’s really the idea of a shell approach. Worklight is really segmented. Although, it’s one single project, one single application, it’s really segmented into the, I guess, the external shell, or the native shell, which has a quota of APIs, the Worklight APIs, any native code that you may have written, and the inner web application, which is all the HTML, the CSS and JavaScript. With Worklight, that code, that CSS, that JavaScript, all your common code is executed exactly as is. There’s no cross-compiling, there’s no converting it to native, so that’s executed exactly as is, and that allows us to really do the direct updates, because now what we can do is, since we’re executing the actual HTML, and we’re actually getting the JavaScript, we can then update that code and still execute it within the Worklight container, or within the Worklight shell. So, this external shell, and this is what’s going to give you the access to the native capabilities, so me calling camera’s now going to give me access to the native camera. You can have branding. You can have security. You can setup individual shells that are maybe a high security shell, maybe one’s that completely open shell, distribute these to your developers and they have a shell that they know has kind of past the security audit for the corporation, and then they can do what they need to with their inner application. It really provides a nice segmented approach to application development.

Another thing is this mobile data support. So, HTML5 always had the idea of the encrypted storage. The problem is it had somewhat of a limitation on size. So, HTML5, you know, you're talking 5, 6 megabytes of data that you can store, kind of in a JSON name value pair. Well, this has been expanded on Worklight to give you essentially an on-device mobile database with full encryption so you can synchronize. You could have a server-to-client synchronization, which means if a modification happens on the server side, it can update the database on the client side, and the exact opposite. Let’s say, you're working offline, you make some modifications to this database, that can then be synchronized back with the actual Worklight Server and then pushed down to the database, itself. So, it allows you to keep information synchronized between devices in a fully encrypted storage capability with virtually limitless size, because now you're running with an online database, or an on-device database, I should say, which is really limited just based on how much memory your device has, so it gives you a lot of storage capabilities. It’s a fully searchable database. It allows you to set the keys for the database, and it really allows you to think about a scenario where you maybe have a catalog, or you have a complete offline set of information that you need to keep in synch with all your different users. This gives you an easy way using a set of Worklight APIs to do so.

All right, finally, let’s get into the Worklight Console. So, the Worklight Console is kind of going to really give you your view into what’s happening inside of Worklight. So, looking at the screenshot, and it’s kind of small here, you can see we have two versions of an iPhone application, this, flight ticket application. We have an Android version. We have a mobile web version. We can manage and we can monitor push notifications. This is where we’re going to get our log and we’re going to get our administrative dashboard, and we can also get our analytics through here. So, we can see, you know, what’s happening with the application. We can see exactly how it’s set, what type of results that we’re getting, but more importantly, this is where I can go in and I can set some of my security tests, where I can set my app authentication. I can actually enable, disable applications, or I can send out notifications. This is all sent directly through the Worklight Console, and it gives you this capability so you kind of have an insight to your application, and you have greater control over the management capabilities of Worklight.

One of those capabilities that you can see through the console is the push notifications. So, using push notifications is a very complex thing, because it’s not just that I need to use APNS, or I need to use the GCM servers, or Microsoft Push, when it comes out, or SMS, or the RIM servers. The key is that there could be users that, you know, maybe in my case, you know, they have a BlackBerry, they have an iPhone, they have an Android phone, how do I distinguish, you know, which users to send it to, which actual back-end systems to utilize. So, Worklight takes care of that for you. You utilize a simple unified push API that can either be invoked by pulling a back-end system, maybe it’s pulling a queue waiting for something to happen, or you’re actually making a call in and just invoking the API. They can say, okay you know what, you want to send a notification to Steve. Okay, let me lookup and see what devices he has, I mean, look at the device IDs. Okay, he has an Apple, he has a Google, he has a BlackBerry. Let me use the appropriate services and I’m going to send out the notification. Let me get the response back and I’ll record that. Or, if I want to send a notification maybe to every user that’s using application XYZ version 2.6, it gives you fine grain capabilities that are out of the box, easy to use, polished and really ready to go. You just setup your communication with APNS, with GCM, with any of these services, and Worklight takes care of the hard part, which is determining when to send the notifications, who to send them to, what to do about the notifications, whether they’re actually registered. A lot of times with consumer applications, the user will decide, hey, I don’t even want to have push notifications so I’m not going to accept them, so it knows whether to even use the service or not, all built into Worklight, very simple APIs, makes it very easy to use right out of the box.

Another thing is application disablement. So, what you can do within the Worklight Console is you can say, you know what, maybe I’m having an issue, or maybe there’s a new version of an application out, I can go in and I can actually disable an application. And, as you can see on the screenshot on the right, I can say, you know, hey, this version’s no longer supported, or a new version’s out, send that notification to the user, so next time they bring up the application, or they bring it to the forefront, next time it makes a communication with the server, they're going to get your disable notification, and they're going to be forwarded to maybe the App Store to get the new version. You can also set it just to be a notification, meaning that maybe you've released version 2.0, but you don't want to force your users at the point to upgrade, so instead of saying, disabled, you can set it to active notifying, which means when they bring it up, they're going to be get a notification. This is different from a push notification, because this won't pop up when they're not in the application, but it's also guaranteed to be delivered, meaning that with a push notification, you can always opt out. In this case, you won't be able to opt out of a notification coming from Worklight, so you'll get that notification and it'll say whatever it needs to say and let's you still continue with the application. Again, one of the real benefits to the middleware is the capabilities around disabling, around direct updating, around controlling kind of the user experience.

So, unified client and analytics. The analytics in Worklight version 6.0 have been greatly enhanced. So, now you have a lot more insight into what's happening, whether it's hits, whether it's visits, whether it's type of applications coming through, whether it's problems that are happening, you have really good insight into what's happening with your application and you can kind of take advantage of these built in analytics to determine what to do, whether it's you're getting excessive load, or maybe one application isn't getting enough load, it's all stored in built-in Worklight databases so you can send them, whether you're sending it to another data repository, whether you want to view it in a completely separate analytics tool, that's completing acceptable as well, or if you want, you know, parse it and look at it, just using BIRT reporting, completely acceptable, and it allows you, again, you want to know how many people are using the application, how many different types of applications are hitting it, what your hit rates are, maybe where people are getting stuck in your application, these unified client and analytics give you those capabilities.

And, the last thing is kind of the integration with Tealeaf, so integration that allows you to analyze the users of the application, analyze what path people are taking as they're going through it, see how the application and server are communicating. So, you can see complete transactions, see what's happening, and maybe for some reason you're not closing deals, or not closing sales, because there's an error somewhere along the path, Tealeaf can help you identify that path and is included in the new version of Worklight. So, it kind of give you, you know, again, an easy way to look at what's happening, get analytics, store them in a way that it's easy to parse and analyze the data, itself.

Well, that's it from a presentation perspective. I hope you learned a little bit about IBM Worklight, and also about mobile development in general.

Now, I'm going to show a quick demonstration of the Worklight Studio. If you don't have Worklight Studio already installed, all you need to do is download Eclipse, go to the

Eclipse marketplace and do a quick search on Worklight. There, you'll find the development plug-in. You can download that and install it automatically into Eclipse. It's a very simple process and gives you all the aspects needed to do the demonstration I'm performing today. There's also a full set of sample applications on the ibm.com/worklight site that will allow you to test, authentication and test, integration with back-end systems. There's about 50 or 60 samples that kind of go through almost virtually every mobile scenario you may want to try and give you complete code samples from the Worklight perspective. But, for this demonstration, we're going to go with a very simple application and just show you how to create one from scratch. So, first, we're going to create a new Worklight project and in this case we're going to pick a hybrid application. The hybrid application is essentially a combination of the inner and shell components and we talked about this a little earlier during the presentation. You can also create a straight native application by selecting the native API option. For here, we'll call it the IBMDemoProject. Once we've given it a project name, we're asking for an application name, which we'll just call IBMApp, and here's your opportunity to bring in some third party libraries. If you want to add in JQuery or Sencha, you need to download the necessary libraries ahead of time and here you can specify the exact location so they can be included in your project. If you want to use Dojo , Dojo was already included with the download. You just have to select it so that it's included in your individual project. So, I'm going to add in Dojo and then I'm going to click finish. After the application initializes, you'll be presented with the application descriptor file, or the application itself. We're going to look at it in source mode so I can identify a couple of key things you'll want to be aware of. First is, it's simply going to identify the display name and the description of your application. One key thing is the main file. This is going to be the individual file that's identified. When your application starts on a specific device, this is the first file that's going to be loaded. The other thing to know is the Worklight Server URL. This is the Worklight Server that's identified for direct updates for actual back-end communications. This is where the application's going to look when it needs to make communications with the Worklight Server. Look at the application itself, the first thing you'll see is that under the apps directory, is our IBM app, and we have our common folder. In the common folder, we have our IBMApp.html file. This is the starting point for our individual application. We can look at it in design mode to get the actual widgets that go with the application. So, we're going to get a standard set of HTML tags, HTML form tags, and then all the Dojo-specific information. So, in here, what we're going to do is we're going to start from scratch, I'm actually going to take out all the built-in information, and we'll start with a blank application. The first thing we're going to do is add in two views so we have an application that basically has two pages. We're going to simply just drag-and-drop the information, we'll call this view0, set it as the default view, we'll give it a heading of IBM Test App and click finish, and you can see we get our drag-and-drop, our look and feel of the application, our visual perspective, and we also going to have the code down here. The next thing I'm going to do is I'm going to add in, now that I've added in one view, I'm going to go ahead and add in a second view, which is going to be our second page. This, I'm going to call, Information, we'll include the heading, and we'll do the heading information, for back button, we'll set it to Home, and that's going to forward back to our view0, which was our initial page. Now, to look at the multiple views inside of an application, you could see on the left, we actually have our views here. Right now we're viewing view0, but we can look at an individual mobile view for each page. So, here on our Information page, we simply right-click on the view and show the view. Now, if we want to add a button just so that we can go back and forth between the pages, we'll add a simple RoundRecList and we'll just bring it inside this page. And, now we're going to modify the title, we'll call it Information, we don't need an icon, we'll go ahead and send it to the Information page and we'll just leave the transition as is. Now, we have a very basic application and we'll go ahead and save it. Going back to the Java EE perspective where we can look at the individual files, one key file is this initOptions.js. Inside that file, it's actually going to set the connection startup status for your application. By default, it's set to false so that the application on startup will not check with the Worklight Server. For demonstration purposes, we're going to go ahead and set that to true and we're going to save that page. Now, we're going to go ahead and build and deploy our application. So, to do that, we're simply going to right-click on IBMApp, run as, build all and deploy. What this is going to do is start to Jetty Server, deploy the application and go ahead and set it and deploy it into the individual console.

So, now if we load the Worklight Console, what you can see is we have our IBMApp, and we can preview its common resources, and here's our simple page, IBM Test App, it's information, goes to the Information page, comes home, a very simple application. There's not much to it. Obviously, a simple design based on Dojo. But, now let's say we want to start to target individual applications or individual environments. Now, we could start to add in those environments to our project simply by going to Worklight and adding in Worklight environments. You can see the number of environments that we have available, iPhone, iPad, BlackBerry, Windows, mobile web. For this demo, we'll just pick iPhone, Android and mobile web, the three most common environments. Now, that those have been added to the project, there's a couple of things to notice. If I open up the Android folder, you can notice that we have our JavaScript and we have our CSS, and these are going to give us the opportunity to override functionality that's in the common folder. For the most part, they're really just placeholders at this point, because I haven't added in any overrides and I haven't added any new functionality that's specific to Android, but you'll also notice that there's a native folder. So, if I want to look at the Java, Java-specific for Android, I can go into the directories and see the individual Java files, and the same thing is true with iPhone. If I want to see the individual Objective-C files, I can go under native and you can see that I have the specific file structure that required to generate an IOS IPA file. So, back to our project, now I'm going to re-build and deploy this application, because I've added in additional environments. Now, the build and deploy is complete, I'm going to go back to my Worklight Console so we can see what's been added. Now, you can see that we had an iPhone, an Android and a mobile web version of the application out there, and now I have my security protocols, I have my versioning information, I have the ability to set it to active notify, to disable the application, and I could also look at these applications through the mobile browser simulator. So, bringing it up I can see here's my application running on an Apple iPhone, 3GS, I can add an additional device. So, for iPhone in this case let's say I want to add in the iPhone 5, it's a little bit larger form factor, it's a little bit cutoff based on the screen size or the resolution I'm using now, but you have a lot of capabilities here. I can rotate the phone, I can compare it against multiple different versions and really just see, you know, how does the application function in certain form factors using a certain code base. I can also simulate Cordova events. So, let's say for the compass, for example, or for the volume up and down button, I can simulate events, and if my application's designed to listen and react to those events, I can test out that functionality in the mobile browser simulator.

I also have the capability to push it directly to Xcode. So, for the iPhone version, what I'm going to do is I'm going to go back into Eclipse and I'm going to push it directly into Xcode, and that simply requires clicking on iPhone, run as, Xcode project. This is going to send the code into Xcode, both the hybrid and the native aspects of it, and allow me to use the IOS simulator, and now you can see my application running on the IOS simulator. Again, it's a very basic sample, but with two minutes coding, I have a functional application that I could push to a real device. Now, that I have an application that's actually running, I can start to explore some of the maintenance and some of the middleware functionality that comes with Worklight.

So, the first thing we're going to show is the capabilities of notifying from an application. So, I have an iPhone version out there and I could set it to active notifying and maybe I want to say, new version out. I could make that modification in the Worklight Console, save the change, bring the simulator back up, and I have it set to trigger on two events, either starting up the application, or bringing it to the forefront. So, right now I've essentially sent the IBM, or sent the IBM app to the background. When I bring it back up, I'm going to get my notification that simply says, new version out, and at this point, I'll say whatever I happen to type in on the console. The good part about this is that users are only going to see this notification if they bring the application up while you have it set inside the console, so it's different from a push notification. This is going to pop up if the application's not running and the other benefit is that let's say you send out a notification, or you set a status that the application's going to be down from 10PM-12AM, maybe you have some maintenance going on. If the user doesn't happen to hit the application in between those times, they'll never see the notification and never realize there was an outage at all. So, I'll go ahead and close that so we can move on with the application, and we'll set it back to active. The other option we have is we can disable to application, which means that users can no longer access it.

Now, let's talk a little bit about the direct update capabilities of the application. If I go back into my Worklight Studio, and let's say I want to change this, and instead of saying, IBM Test App, maybe I want it to say just, IBM App, so I'll make that simple change, I'm going to save the code base, and now I'm going to build and deploy the application again. What I've done is I've added in, I've simply made an HTML change to the code base itself and I'm going to show or demonstrate the direct update capabilities. Now, I've changed the server side code so when the simulator hits the server, it's going to realize that the code base running locally no longer matches what's on the server and it updated necessary. So, again, when I bring it to the forefront, now I'm going to get an update message, because again, I've changed the code and I cannot hit an older version of the application. So, I'm going to click update, it's going to unpackage the application and you could see my title now. It just says, IBM App. I didn't have to go back through the idea of re-deploying through Xcode. I didn't have to generate a new IPA file. All I did was make a modification direct update took care of the rest and I didn't have to wait the standard one or two weeks process of going back through the re-certification of my application. But, this doesn't just rely on simple HTML changes.

Let's say I wanted to go into the application and let's say for iPhone specifically, I wanted to add in an alert. So, we'll go into the IBM App JavaScript file and inside my WLENVINT I'm just going to add in a simple alert. It says, Hello IBM. Now, there's two things to know here. So, I've saved my change. I may just change only in the iPhone code so this is only going to affect IOS versions of it. It's not going to affect the Android versions. I'm going to go ahead and do my block and deploy again. Now, this time when I bring up my simulator, again, I'm going to get an update. I'll go ahead and click the update and you'll see the first thing that pops up. It says, Hello IBM. I made a code change this time. Now, it's a functional change, because I modified the JavaScript, but again, the direct update capabilities still function as normal, the update occurred, I get the pop-up message as expected. The key is that this is just an IOS change.

So, if I go to the mobile browser simulator and I click iPhone, you're going to see I get my pop-up message. Again, we're just doing functional testing in the browser. If I do the same thing and I try it with Android, now Android doesn't have that code change, that's an environment-specific change for IOS, so again, we're not going to get any pop-up message here, because it's code-specific to IOS in this case.

The last thing I'm going to show, since this is a rather short demo, is the idea of versioning. So, let's say I want to make another modification and I'll say, instead of saying Hello IBM, let's say, Hello World, I'm going to save this change, but instead of just pushing out and doing a direct update to all the current versions, what I'm going to do this time is I'm going to actually create a new version of the application. So, inside my application descriptor, for iPhone, I'm going to change this to version 1.1. Now, I'm going to build and deploy my application. Now, that it's deployed, there's two things you're going to see. First of all, when I bring up my simulator and I take my application, I send it to the back and I bring it back up, you'll notice that no update occurs, and the reason is is because my simulator's still running version 1.0 of the code. The other thing you're going to notice is that inside the console now, I now have two versions for iPhone. I have version 1.1, which when I look at it it's the mobile browser simulator, it's going to say, Hello World, and I have my version 1.0, which is the version running on the simulator, which is going to say, Hello IBM. I have the capability within Worklight to run multiple versions at the same time on multiple devices.

And, the real key here is that, let's say for example I made six or seven directory updates and I finally get to a point where I say, you know what, I want to make a final version change so that when people go to the App Store, the official App Store, they're going to download version 1.1, now, so I need to support all those users, but I still need to support the users on version 1.0, because there's no guarantee that they're going to automatically update. They're going to get a notification from the App Store to the new versions out there, but there's no way to force them to update. So, this is kind of a common problem that we handle with versioning, but we'll support multiple versions of an application and we'll slowly, based on analytics, see which users and how many users are still on older versions before we begin to sunset them. At some point, we may want to disable the application altogether and say, new version out, please upgrade, and here, I could simply give it the URL to the App Store where the new version is. Now, that I've saved that change, when I bring up my simulator this time, now the application is disabled. I have determined that maybe 1% of the users are still on version 1.0, I want to force them to upgrade, so now I've made that possible. They've got two options. They can go to the new version, go to App Store and get it, or they can close the application, because from the Worklight side, I've completely disabled it. I know this was a short demo, but we only had about ten minutes time before, and we wanted to get into some Q/A. So, let's go ahead and open up the lines for Q/A and see what kind of questions we have.

Darin Rich: All right, Steve, thanks for your presentation and demonstration. That was very good. I appreciate you going through all the detail on the Worklight and both how it can be used from a development perspective, but also from the runtime and management. So, now, we're going to go ahead and start the question and answer part of the call today, and we'll also take some polling questions so, first, so let's just kind of mix those up. First, we have a question, Steve, about the requirement for having a separate server or if a customer already has WAS, is there something they can do to just add it to handle the Worklight runtime?

Steve Mirman: So, the Worklight runtime, you can either run it directly on Tomcat if you want to put it on its own server. If you already have WebSphere, it's essentially just the server side component, or a war file that run on top of a JEE container. So, if you already have WebSphere, you want actually put it right on top of it, setup the Worklight database and run with that.

Darin Rich: And, if you don't have WAS, is WebSphere a requirement for Worklight for the runtime?

Steve Mirman: No, so the two primary supported platforms are Tomcat, so you could run it on the Tomcat JEE container, or you can run it on WebSphere, or WebSphere's

Liberty Profile. It really just requires a very lightweight JEE container. So, if you don't have anything currently, is you could always get Tomcat, or WebSphere's Liberty Profile if you're looking for something very lightweight.

Darin Rich: Very good, okay. Before we take any more questions, let's do a quick polling question. Alex, why don't you go ahead and put the first one out there. So, everyone that's on the line should see the polling question. This is just to kind of get a handle on the current status with regard to those participating with regard to mobile development. So, are you currently building or planning to build mobile applications today? I'm assuming that will be pretty high since you're on this call today. And, tell you what, as we're waiting for the results of that, let's take another question. Steve, you know, given your background in development, could you just, I guess, opine a bit on, you know, for a web developer, web developer skill, how easy is it to learn Worklight?

Steve Mirman: For a web developer, it's actually very simple, because Worklight's completely open, so you're really, the skills you need are HTML5, CSS, JavaScript, and most specifically, whatever JavaScript library you decide to use for your framework, so if you're a JQuery developer, or a Sencha, Dojo, whatever you want to use, you pretty much already have those skills, so integrating it with Worklight is just accessing the APIs, so you should be able to pick it up really quickly. The native aspects, that's when you have to have a little more experience with Objective-C and Java if you're trying to write native code specifically for certain applications.

Darin Rich: Very good. Thank you, Steve. Okay, and there's the results from the first question, and the first polling question, it looks like it's from a 2:1 margin, and again, not real surprising. Let's go ahead and put out the second polling question. So, the first one is, you know, are you planning or are you doing mobile development? The second question is along the lines of how you're doing it. So, go ahead, and Alex, put that one up. Are you doing it in a native fashion today, or hybrid, or whatever combination, or I don't know. So, if you can respond to that and we'll ask another question here. We'll go through another question that was asked about how, Steve, would you say Worklight compares with some of the other mobile development tools, or meet offerings that are out there in the marketplace? How is Worklight different or unique?

Steve Mirman: One of the biggest things for Worklight is that it actually runs or executes the hybrid code that you write. So, you write HTML, CSS, JavaScript, that's what's actually executed, even on the client side, it's just running within the Worklight container. A lot of the competitors, what they do is they try to cross-compile or take your HTML and CSS and convert it into native code to run on the device, and the real limitation to that kind of approach is that when you want to write new code or a new functionality comes out, you're really required to wait for the vendor to give you up to date APIs, because they're converting your current code. With Worklight, we run exactly what you write and we give you access to fully invoke and call and write native code so there’s no limitation. When a new SDK comes out, you could immediately get access to that functionality.

Darin Rich: Very good. Thank you. And, I think that we are close to getting the results here for the second polling question. There they are. Looks like, wow, even up on native, hybrid and web and then the combination is by far the biggest winner with second in place, I’m not sure. So, one more question here. Steve, I know that you talked about this before, but if you could repeat, now there was a question about the platforms and client devices that are supported with Worklight, both on a client side and then on the back-end, you know, runtime server side.

Steve Mirman: Right, so on the server side, it’s really wherever you can run your EE container, so Linux, AIX, Windows, it really doesn’t matter. I run a full Worklight Server on a Mac, because again, you know, I’m running it with Liberty Profile, which is very lightweight. As far as the devices that we support, you know, we support Windows, BlackBerry, multiple flavors of BlackBerry and Windows, as well as IOS, even Java ME, Android devices. Basically, as the market changes, we just take advantage of those SDKs and write it so that the code can utilize them. So, that’s one of the benefits to Worklight is that we can adapt as the market changes. If a completely new thing comes out, we can adapt and we can make use of that, because in the end, you're writing either small portions of native code, or you're just writing common hybrid code and just utilizing the APIs that we make available.

Darin Rich: And, then, and again, you mentioned this before, but had a question about the supporting of new releases, you know, a new IOS release, or you know, cut a lag time there.

Steve Mirman: Right, and that’s actually pretty quick on our side, because when a new SDK comes out, you can bring it directly into Worklight Studio and begin using it. So, think about for Android, a new SDK comes out, we can program natively directly against it. Cordova APIs, we’ll, you know, pick up that change, we can bring in a new Cordova set of APIs. So, we have a lot of opportunity when new things come out, we get the pre-releases, we try to configure it and we update as quickly as possible to give you APIs, but you always have the capability of writing native when a new functionality comes out, and the SDKs are released as soon as the device comes out, essentially.

Darin Rich: All right, thanks, Steve. So, that’s all the questions I have queued up here. Just a couple of housekeeping things and again, if you have some additional questions, just go ahead and put that in the Q/A section there on the webcast screen. A reminder. On the bottom left corner, you can access this presentation and other related assets by clicking on the content button and then on the files section. So, you can go ahead and download the actual charts if you’d like. The other reminder I would say is that this whole webcast will have a replay option for your colleagues that weren’t able to attend, so you can get to it by the same, you know, basically the same websites you registered to begin with, so, you can pass that information on to any colleagues or friends and they can hear this replay, or obviously the other two replays from the one last week and the week before on the web standards and WebSphere application and server version 8.5.5. So, all right. I do not see any additional questions queued up so at this point I will thank you all for attending this Developer Days webcast and thank you for your time.

[END PRESENTATION]