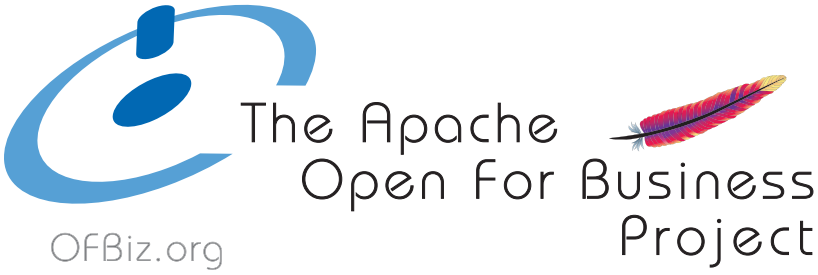
Apache OFBiz Advanced Framework Training Video Transcription

Written and Recorded by David E. Jones

Transcription and Editing by Nathan A. Jones

Brought to you by Undersun Consulting

Based on:



Advanced Framework Overview

Introducing the Series

Starting screen site/training/AdvancedFramework.html

Hello my name is David Jones, and this is the Advanced Framework Training Video from Undersun Consulting. I'll be recording and narrating this. I'm one of the administrators of the Open for Business Project, and offer various services including things like these training videos through a company that is run by myself and Andy Zeneski called Undersun Consulting.

This video is an overview of the Advanced Framework training. Here on the screen you can see the outline for this. This outline is available through the OFBiz.org website, also the html file for this is in the website directory in the OFBiz source code repository. The target audience for this video is developers and architects, especially those who are senior level and who will be helping to supervise and support those who will be doing OFBiz based development.

00:01:24 screen changes to site/training/BasicFrameworkToolsOverview.html

Some prerequisites for this, the OFBiz introduction videos, which follow this basic framework and tools overview outline, are very important.

00:01:30 screen changes to site/training/AdvancedFramework

These videos cover the OFBiz framework at a high level, and so should be reviewed before going over these videos so you have an idea of what the big pieces are, what the major things are in the framework, and how they fit together. In these videos we will be going into more detail on each of the videos, as well as providing more background information on Java and J2EE and some of the other infrastructure that's in the open for business project.

While extensive knowledge of Java and J2EE is not required, some familiarity of basic Java and some J2EE specs like the service specifications JDBC for Database Connectivity, the JNDI, the Naming Directory Interface, and JTA, the Transaction API, are important. We will be going over some of those in the first section, this Just Enough Java and J2EE, talking about Java Classpath, the various things about these J2EE APIs we just mentioned.

We'll also discuss in this video the OFBiz base, especially the Containers and Components and how they are used for getting OFBiz started when OFBiz is running as the main program with application components such as the servlet Container and TransactionManager and such, running inside OFBiz. As well as when OFBiz is running inside an application server, this infrastructure still applies. The Containers and Components, they're just used in a slightly different way. The easiest way to run OFBiz is out of the box, using the OFBiz start executable jar file, which uses the container and component infrastructure to load everything up, get all the tools started when the application server shuts down, and to stop all of the tools and such as well. And also to configure each of the framework and applications components that make up the open for business project.

So we'll talk about those, how they're configured, the different things to do plus some conventions for the component directories and such.

In the next section we'll talk about the web application framework, including the widget library, the various widgets; the Screen Widget, the Form Widget, the Menu Widget and Tree Widget, as well as MiniLang, which is the main one here in the simple-method, is what it's called. MiniLang is sort of a category that it goes in, and we'll talk about that, how to implement logic, especially for validation, data mapping, those sorts of things, in that language.

We'll also talk about other user interface level things here such as BeanShell which we use for data preparation and scripting, FreeMarker which we use for templates, for various things including user interfaces including generating XML files and such. And one specific application of that is Generating PDFs. We'll use the screen widget, and other such tools FreeMarker, BeanShell. So with the FreeMarker files they'll actually generate the XSL:FO, formatted XML file, rather than an html file, and then we use the FOP project from Apache to turn that XSL:FO file into a PDF.

We'll also briefly discuss JPublish, which is what we use to separate logic from the layout before we have the screen widget, and also to use decoration pattern to compose more complicated pages from different files.

We'll also talk briefly about the Region tool in OFBiz, which is used basically depends on these servlet API, and is used for JSPs, since they are not a generic templating tool and must be executed in a servlet context. In order to do composite views we have this Region tool which is based on a servlet context running, with a request and response objects being passed throughout these things. So we'll overview briefly the Region Framework and the OFBiz Taglib that's used in JSP files. So Jpublish and the Region stuff you'll find some of this code still in certain parts of Ofbiz from older things that have been written that way, although if you want to write new things you can use these tools, although very much the recommended tools to use for user interface things are the screen widget and the related widgets there, but we'll talk more specifically about when these various different tools are best used.

In the MiniLang we have the simple-map-processor and the simple-method, as two of the tools there. Simple-map-processor snippets can be imbedded in simple-method files, and that's often how you'll see them, but simple-map-processors can also be used on their own. And we'll talk about the various operations, the semantics, and how these things fit together. Also when these are best used versus implementing services or the control servlet request events, when those two things are typically best implemented using a simple-method versus implementing them in Java or some other scripting language.

In the next section we'll talk about the Entity and Service Engines. The Entity Engine is what we use for persistence and we'll go into detail about the design goals and how this is configured and used, and also some of the more advanced topics, view-entities and details about the delegator API and related objects there, and also about a little bit about the GenericHelper interface and what the delegator is capable of handling, and the Entity ECA rules, which are used to trigger services based on different data level operations that are happening.

And I should mention just as a preview here, we'll certainly go into this in more detail later, these are mainly intended for data maintenance and not business process management. For business process management that's best left at the logic level, with Service ECA rules, which is covered in the next section about the Service Engine.

We'll talk about the design goals and features of the Service Engine, the different things you can do with it, and the various things you typically want to do in an enterprise application in the logic tier that this facilitates. We talk about configuration of the service engine, very detailed information about the service definitions, remote service invocation, various ways of implementing service, different options for calling services, and we'll go look at sample code and various things for these.

Some of the more advanced things like Service Groups, Service ECA rules, and service engine remote dispatcher, which can be used from another Java application to call service as if they were running locally on the remote machine on the OFBiz server. So that can be handy if you need to interact with an OFBiz server and another Java application that will be running in a separate process, and do not want to have all of the infrastructure of OFBiz in that separate application.

In the next section, part five here, we'll be talking about the workflow engine, the data file tool, and some of the other miscellaneous things in OFBiz. I should note here that while we will be covering some information about business process management in general, and some workflow specific concepts, we'll also talk a little bit about the workflow engine that's implemented in OFBiz, and the new workflow engine we're moving towards which is a separate open source project called the Shark workflow engine, which implements this same specification, the WfMC and OMG specification, that then will be the target of the OFBiz Workflow Engine.

I should note before we get into this, before we wrap this up here, the Workflow Engine is based on WfMC is very much of this workflow style, where we have states and transitions between them, as opposed to the message-flow style. This is very good for managing a large volume of process that need to be both automated as well as manual.

For the most part in OFBiz what you'll see though is this other tool, the Service ECA rules, which are the Event-Condition-Action rules, which is kind of a rule language which is event based, or in the case of the OFBiz Service Engine, business event based, and this can be used to tie processes together, and we'll talk in more detail about that. That's what you'll see used for the most part in OFBiz, because it's a very nice model for managing high level processes and is very easy to configure and change when customizing the service and application that are in OFBiz.

The DataFile tool is the last part here, and this is a tool that helps with something that is often run into when you're integrating or communicating with other systems, and that is dealing with Flat Files, especially in cases where you need to do a data level integration with an older system. It's very common to run into a need for different varieties of flat files, whether they be fixed width files, or character separated, or these different sorts of things, the data file tool can help you interact with those types of files, reading and writing them in a very declarative way.

Normally when we do this as a live training session, we'll do some review, and a question and answer section. I will do some review at the end of this, but for question and answer, that is where one of the packages that Undersun Consulting offers is these training videos combined with a number of hours of remote training, that you can use to get your questions answered, as you're going through the training or once you're finished with the training, so that for things that are left out, or that you're not sure how they would apply to your specific situation, you can ask about those.

And that is what we will be covering in this advanced framework training series, so welcome, and enjoy!

Part 1: Just Enough Java and J2EE

Introduction

Starting screen site/training/ADvancedFramework

This is the first part of the advanced framework training video series, and we will be covering the “Just Enough Java and J2EE” section.

This will include a short overview of the Java classpath, what the packages are, the different entries on the classpath and how they are organized. Servlet API and JSP, how they fit into the big picture, these are basically http for the most part, request handling libraries. JNDI, the Naming and Directory Interface, for handling registration of objects that various parts of an applications or even multiple applications can share. JDBC for connection to and communication with the database.

We'll talk about some of the basic concepts and some of the main objects in the API, and we'll also talk about Transaction Isolation Levels because these are important for understanding how the OFBiz or any application interacts with the database when you have possibly in some cases even hundreds of concurrent transactions open in the database, so seeing how those transactions will be isolated is important.

We'll also talk about the JTA, the Java transaction API, and specifically the UserTransaction and TransactionManager objects. Some specifics about Transaction control, including the differences between rollback and setRollbackOnly, and we'll talk about Pausing and Resuming transactions, which is an alternative to a nested sort of Transaction, which is not really supported in the world of JTA, but which you can get close to it with pausing and resuming transactions, and in general doing it that way is significantly less confusing and or error prone.

Part 1, Section 1

The Java Classpath

00:02:17 screen changes to
http://java.sun.com/j2se/1.4.2/docs/api/

Okay so let's start at the beginning with the Java Classpath. Most of this we'll be looking at in terms of the online documentation that is on java.sun.com. This is the overview for the JavaDocs for J2SE 1.4.2. Right now OFBiz, because of some library dependencies that only run on a 1.4 series JVM, OFBiz will only run on a 1.4 series JVM, not a 1.5 series yet, in other words.

00:02:50 screen changes to
http://java.sun.com/j2ee/sdk_1.3/techdocs/api/index.html

Also certain parts of this will be from the Java 2 enterprise edition, API Specification. Certain parts of even the lower level J2EE stuff is now in the 1.4 series, in the standard Java virtual machine, the libraries that come with the standard package. But many of them are still part of the enterprise edition extensions that are application server specific, especially the servlet implementation and JSPs, as well as JTA.

00:03:37 screen changes to
http://java.sun.com/j2se/1.4.2/docs/api/

JDBC is actually part of the main J2SC now, so there are some extensions to it that are part of the enterprise edition, but the main stuff is right here in the standard Java API.

Okay so the first topic I wanted wanted to talk about quickly is the Java Classpath. And this is very important to understand how Java packages and Classpath entries work. When a Java application starts, it starts out with a certain Classpath.

We'll go over what OFBiz does in particular in the next section, in part 2, when we talk about the OFBiz based, but for now it's important to understand that Classpath basically consists of multiple entries, each entry will either be a directory that contains Classpath resources, they can be properties files, XML files, any type of files, or they can be compiled Java classes.

00:04:43 screen changes to containers.xml

If we look at the OFBiz tree, we have certain directories like this, the config directories, that are put right onto the Classpath, so that if we look up cache.properties, or component-load.xml, or any of these other files on the Classpath it will be found, because this config directory is on the Classpath. If there are multiple directories or jar files, which are basically just like directories but they're put into a zip file with the extension .jar, or Java Archive, then there are multiple entries on the Classpath that have the same resource in them, like multiple cache.properties files in the Classpath, the first one encountered would be the one that was used.

So other things that are on the Classpath, it's very common that you'll find source directories in OFBiz. This source directory has two source directories underneath it. In many cases the source directory itself is the root of it. Inside the source directories we'll typically have other directories, like org, ofbiz, base, start, for example, going into here. And then in the start directory we actually have various Java files, as well as various properties files.

So if we wanted to find this properties file, we'd have to refer to the full location, which would be org/ofbiz/base/start/install.properties. One thing to note, though, is because this contains Java files, this directory is not put directly onto the Classpath. Instead it is built using the Java compiler, and the convention in OFBiz is that those are then put into a build directory.

The equivalent of what is under the source directory will go into the classes directory, so here we have the base and start directories, and under here we have org, of-

biz, base, start, and now we have instead of classpath.java.start.java and so on, we have classpath.class, which is a compiled Java class file start.class, and so on.

These ones that have a $ sign in them are inner classes that are inside the start.java file, kind of like sub classes. So then once this is put together in these classe directories we take those and put them into a jar file for more easily managing it, we have an ofbiz-base.jar file that corresponds with the base directory here, and the ofbiz.jar file corresponds with the start directory. And this is actually the executable jar file that we use to get OFBiz started when we're running out of the box using the OFBiz container loader as the primary starting point of the application server.

If you're running a different application server then the ofbiz.jar file isn't used. The way the Java Classpath is set up for OFBiz is a little bit more specific. Everything that we put initially in the Classpath is right inside this jar file, so we can just execute it without specifying additional Classpath entries, and then the configuration files for OFBiz inside here will tell it how to find, based on the OFBiz root directory, other configuration files which will tell about the other Classpath entries and such to load, or to put on the Classpath.

We'll look at those in more detail when we get down to the OFBiz component.xml files, and the component loading in general. So for now just know that jar files as well as directories like the config directory can be put on the Java classpath. And when we're searching for a Classpath resource, we just give it the name of the resource and it looks through all of these entries on the Classpath, jar files or directories, to try and find the resource that we've asked for.

So this is a convenient way for kind of keeping track of a large number of resources. As you can see with OFBiz, especially where we have almost well over a hundred Classpath entries, both directories and jar files, including dozens of libraries from other projects, and hundreds of class files from OFBiz itself, especially in the framework, its very important that things be split into packages.

So we use this org package, and an OFBiz package underneath it. And then underneath that we have various packages like base, or entity, or basically that correspond with each of the components of OFBiz, so that within their jar files or directories that are put on the Classpath, we can avoid collisions with the names. Now when referring to something on a Classpath, let's look at a Java file real quick. Let's just look at a simple one like this start.java.

00:10:19 screen changes to start.java

Each Java file will declare the package that it is in. These are basically the directories from the Classpath where it can be found, so org.ofbiz.base.start. It can also import other classes, if they're in the same package they're imported by default, if they're in other packages then we need to refer to them explicitly.

So java.io.bufferedreader, where we include the full Classpath or the full location from the root of the Classpath where the class can be found, or where any resource can be found but specifically the classes. This is called the fully qualified class name. Now when we're loading other resources from the Classpath, how we refer to them is a little bit different.

So if it's not a class file, so with class files we're using we have these directories inside of each other so there's a Java directory, an io directory, and then inside that a file called bufferedreader.class. And that would typically be in a jar file somewhere, in this case this is one of the main Java libraries, so it'll be kind of inside the JVM.

00:11:49 screen changes to entityengine.xml

But if we're referring to a resource, actually let's look at the entityengine.xml file. I think I have that open here. When we're referring to a resource that is on the Classpath, we do things a little bit different.

00:12:05 screen changes to controller.xml

Actually a good place to look is the controller.xml file. These were introduced in the framework overview videos. You'll see where all of these Java locations, these Java things, they have the fully qualified class name so from the root of the Classpath; org.ofbiz.common.commonevents, and then the other part of this, this is specific to an event, declaration, the other part of it is the name of the method in that class to invoke for the event. And now let me see in this file, I should have looked this up in advance, the other type here when referring to a resource on the Classpath instead of a Java file will use a slash. And it looks like this is all fairly modern.

00:13:06 screen changes to controller.xml

Let's look at the eCommerce one, I think it has a little more variety in it. And there's got to be one somewhere. Okay so here's an example. When we're using a simple method, those are not compiled Java class files, but the files are typically on the Classpath, and so we're trying to find this XML file on the Classpath, so we need to refer to the full path inside the jar file or directory, that is one of the classpath directories.

So we have org/ofbiz/ecommerce/customer/customerevents.xml, and that's how we can refer to a resource on the Classpath. The same is true typically for properties files, XML files or any other resource that is that way. Notice that this is different from the component locations, and we'll talk about these in more detail that start

with the component://, and these are relative to whatever component is named in the first directory spot right here.

00:14:33 screen changes to entityengine.xml

So those are some of the ways that we refer to entries on the Classpath, and in general another thing to note about application servers is that sometimes they will have an hierarchy of Classpaths, so rather than having a single Classpath for the entire application, a lot of application servers will have a Classpath for the application server, and then a protected Classpath for different sets of components that are deployed on the application server.

Like an EAR file, an Enterprise Archive File, when it's deployed will typically have its own class space. Also web applications. They always have their own application space. So each web app has an independent class path space that is separate from the other Classpaths, that is why most of the OFBiz resources, rather than going on the web application specific Classpaths, go on the global or shared Classpath.

This is important especially for things like the caching functionality that's in OFBiz, that is in the base util package. These are all basically shared utilities that are used in different parts of the OFBiz libraries. But it's especially important that the caches be shared between applications.

So this utilcache and all of the related files are in this jar file that is deployed on the global Classpath. To make it easier to share and refer to other things and so that the service engine can always find the class that it's looking for, for example, we'll typically put even application specific stuff on the global Classpath, rather than on a per web application Classpath.

00:16:42 screen changes to site/training/AdvancedFramework.html

So that's something to keep in mind when you are creating things in OFBiz that are located through the Classpath, and also when you are deploying OFBiz, and may have Classpath issues to learn about to deal with. Different applications handle the Classpath differently, so when doing an OFBiz deployment, the first thing to research about an application server is how you can manage the Classpath with it.

In some cases you have to copy the Classpath entries into the application server's directories. That's the case with Tomcat or Geronimo, which makes an external deployment in those, rather than using Tomcat or Geronimo, in an embedded mode like we do by default in OFBiz. But deploying OFBiz inside of those tools would require copying a significant number of these Classpath entries, like I mentioned there are over a hundred in OFBiz, into these directories that these application servers can recognize and use.

So that's always something that's fairly important to consider. When you end up with a class not found type of exception, then it means that you referred to something that you expected to be on a Classpath, but it couldn't find it, and so these are some things to consider when trying to figure out why it's not on your Classpath.

There are some more advanced things to know about Classpaths and managing Classpaths, and you can actually do a lot of this management programmatically, and that is actually done in OFBiz, we do a fair amount of Classpath management programmatically. But that's outside of the scope of what I want to cover here.

Part 1, Section 2

Servlet API & JSP

The next section is the Servlet API and JSPs. We'll talk about some of the main concept. The servlet API basically handles the requests coming in, processes those requests, and sends back a response. While servlets can be used for other protocols, the primary protocol that is used for the servlet API is the http protocol. And of course the derivative of that, the related protocol of https.

So when we're looking at the request and response, we'll be looking at an http servlet request, and an http servlet response. We'll also talk about the context. This is an important part of the organization of the servlet API. You have an object that represents each application, an object that represents each user's session, an object that represents each request that's coming in, and there's a corresponding response object for each request.

And then depending on what you're using for rendering a view, you will typically have a page or screen context. In a JSP you'll have a page context, using the OFBiz screen widget we call that the screen context. It's a very similar thing it just happens within a single part of the view. With the screen widget it's a protected context, so you don't have to worry about declaring a variable and overriding it in a screen that is at a higher level than the current screen or things along those lines, and we'll go over that in more detail when we get down to the screen widget.

00:19:59 screen changes to http://java.sun.com/j2se/1.4.2/docs/api/java/sql/package-summary.html

So let's look at some of the servlet API. We're going back to the Java docs on java.sun.com. The specific package we're looking at is javax.servlet. There's a lot of functionality, a lot of stuff in here, if you're interested

in more complete coverage of the servlet API, and haven't already looked through something like this, you might consider getting a book that covers it more thoroughly.

But the main objects that you'll typically need to know about when working on a lower level with OFBiz, and concepts that are helpful to understand even working at a high level with OFBiz, are things like request and response objects that i mentioned. The servlet context corresponds to a web application, and the session which we'll talk about in a second, corresponds to the session.

Now this is the generic javax.servlet package. This API is meant to be used for multiple types of protocols. More specifically there's a sub package of that, javax.servlet.http, that is used for http requests and responses, and http session management. Since http is a stateless protocol, in other words there is no session management inherent in the http protocol definition, that is not actually true with https, that actually does have some session management in place, but http itself does not.

So application servers that handle web applications will generally have to have some kind of session management tool where they use a cookie or a url parameter that is passed around consistently, so that as page requests come in, the application server knows who the user is, and which session they are associated with.

So on the server side, we can keep variables in the session, and keep track of things on behalf of the users, such as who was logged in, or things they've selected through a process that's more complicated than will fit on a single page, or those sorts of things.

So the HttpServletRequest will come in as the object that represents the request coming in from the browser client. HttpServletResponse is the object that represents everything that we need to do to send the response back to the client. HttpSession has things like attributes and such that we can manage on behalf of the user, but do so on the server side, and then.

00:23:03 screen changes to site/training/AdvancedFramework

So when we're looking at the different scopes of these ones over here, we're looking at the page and screen, which are kind of outside of the servlet API but those are the most local scope, towards the most global. So we have request attributes, session attributes, and application attributes.

00:23:22 screen changes to http://java.sun.com/j2ee/sdk_1.3/techdocs.api.javax/servlet/http/package-summary.html

And those are stored in these different objects, mainly the HttpServletRequest object, the HttpSession object.

00:23:31 screen changes to http://java.sun.com/j2ee/sdk_1.3/techdocs/api/javax/servlet/package-summary.html

And going back to this more generic library we have the ServletContext object, that can have attributes as well.

00:23:37 screen changes to http://java.sun.com/j2ee/sdk_1.3/techdocs/api/javax/servlet/ServletContext.html

All of these four attributes have sort of a getAttribute method, and for the other side of that a setAttribute method. And then other things related to attributes like removeAttribute, getAttributeNames, and so on.

Now with the application or servlet context object we can also get resources that are inside the web app, similar to how we can get resources that are on the Classpath. So it has this sort of thing for getting a resource, getting a url that represents the location of the resource, relative to the web application, getting a resource as a stream to actually read it in and resource paths. So various things like that, there are also init parameters. These come form the web.xml file, and we do use those for certain things.

00:24:47 screen changes to site/training/AdvancedFramework

Okay so that's basically it for what I wanted to cover for the servlet API. For JSPs these are built on top of a servlet API, and are not really a generic templating wing, which. I just wanted to discuss them briefly and how they fit into the world of the j2ee specifications. So what I mean by they are not a generic templating tool is that the only output that a JSP engine, according to these specifications, has to provide, is output into a servlet response object.

So that mans you cannot easily get a string, or a stream of characters, out of a rendering of a JSP that can be used for other purposes. That's one of the reasons why we don't use, we have moved away from JSPs for the most part in OFBiz.

There are still some older tools, especially in the webtools webapp, that are written with JSPs that have not been modernized of moved to the different tools that we prefer. But most of the application stuff, and all of ecommerce, has been moved away form JSP to use FreeMarker, which is our preferred templating language, and is a more generic templating engine.

The nice thing about, and there are of course others like FreeMarker like Velocity and such. These templating tools can provide output to a writer or a stream of character, so you can also get a plain string out of them with a rendered template.

The nice thing about that is building tools for composite views, putting together more complex screens, or

things that have decorators or this sort of things or include other screens, frameworks for that are significantly easier to write and are more flexible.

So the Regions tool in OFBiz supports JSP and other servlet resources, and uses these servlet API on certain parts of it, to do what it does in the composite view. But it is not nearly as flexible or powerful as Jpublisher or the screen widget, which use these string writer or text stream based templating engines that are more generic.

The nice thing about using templating engines that are more generic like that is we can use the same templating tool for transforming data, for importing and exporting, we can use it for email templates that are used to generate the text for outgoing emails, and various other things like that.

Part 1, Section 3

JNDI

Starting screen site/training/AdvancedFramework

Okay the next topic we want to cover in Java and J2EE section is JNDI, this is the Java naming directory interface. The main point of JNDI is to have a registry or directory where you can put things, put objects or other resources so that other parts of the application or other applications even running on entirely separate machines can have access to those objects.

It's very common to put objects that implement remote interfaces into JNDI so that other machines can get them from the JNDI server and then call methods on them and communicate with the machine they came from.

00:00:51 screen changes to http://java.sun.com/j2se/1.4.2/docs/api/javax/naming/package-summary.html

So let's look a little bit at the basics of the API, this is the javax.naming package. This is part of the j2se Java docs. Some of the main objects here, the Context is probably the main one you end up dealing with for more simple uses of JNDI, or the InitialContext is an instance of that that's commonly used.

Down here at the bottom they have some helpful stuff. Good description of JNDI and how it's used. This little code snippet right here is a very simple example of doing a JNDI lookup. You get a context object that implements the context interface and call the lookup method, pass in the location or the name in the directory, and it will return the object, and you can cast it to whatever the class is that the object is an instance of, and then the call methods on that class. So as I mentioned this is often used for objects that represent different resources in an application server, on a lower level for j2ee stuff.

00:02:17 screen changes to site/training/AdvancedFramework

It's very common that some of the things we'll talk about next, let's hop back to this outline, like JDBC objects, especially the data source objects, and JTA, you'll typically have UserTransaction and TransactionManager objects that are located somewhere in JNDI, when the application server starts up it will typically put them somewhere so in the application you can get those objects in a fairly standard way, by just pulling them through JNDi, and then using them that way.

When you configure the entity engine you can tell it to get objects from a data source in JNDI, JTA interfaces from JNDI or we have some embedded one that skip the JNDI interface for when you're running OFBiz out of the box and you use some of the embedded components. So those are some different uses for JNDI. Another common one is for rmi, when calling methods remotely using the java rmi tools, you'll typically do a JNDI lookup to get the rmi remote interface object or the object that implements the remote interface.

And then when you call methods on that object from rmi it's really underneath kind of in the background talking to the other server. So that's basically JNDI. It's a fairly simple thing in its application but provides a lot of flexibility and power to application servers and other application infrastructures.

Part 1, Section 4

JDBC

Okay the next one we want to talk about is JDBC, this is the Java DataBase Connectivity API. Some of the basic concepts here: basically we have a set of interfaces, a general API, that can be implemented for different databases. So a JDBC implementation is typically called a JDBC driver, and most databases these days have one.

And it allows your Java application to interact with the database through a standard API. And it does a lot of things for you, it allows you to get a connection to the database, send sql and get results back and get metadata about the database.

One thing it does not do is give you an abstraction that would give you an intermediary for sql. There's no query language that's part of JDBC that you could use as a generic thing that could then be translated into sql to get sent to the database. The approach that the entity engine takes relative to JDBC, and it takes advantage of the power of JDBC for communicating in a generic way with the database.

But the entity engine basically operates with higher level operations, and based on the metadata for the different data structures involved in those operations, it generates sql on the fly. And in the entity engine there's configuration to change the behavior in certain ways for the sql generation to support the variety of databases that we do.

Sometimes as new databases come out there are quirks in the JDBC drivers, or new data types or new syntax changes and stuff like that, little things that need to go in. those can go into the entity engine and then the applications that are using the entity engine don't need to be aware of those changes.

00:06:08 screen changes to
http://java.sun.com/j2se/1.4.2/docs/api/java/sql/package-summary.html

So let's look at the API now. Let's go back to our Java docs. So most of the JDBC API is in the java.sql package, and it's part of the j2se stuff.

00:06:19 screen changes to
http://java.sun.com/j2ee/sdk_1.3/techdocs/api/javax/sql/package-summary.html

But there are some extensions to it that are very important for enterprise applications in the javax.sql package, and these are in the j2ee Java docs. So these are part of the j2ee specifications.

00:06:34 screen changes to
http://java.sun.com/j2se/1.4.2/docs/api/java/sql/package-summary.html

Let's look at the basic ones real quick. Some of the important things here are Connection, the connection object which represents connection to the database, and the ResultSet.

So typically you'll have a Statement, or a PreparedStatement, that you get from the connection object, and using that PreparedStatement or Statement you send some sort of command or operation to the database, the results of that come back in a ResultSet.

00:07:04 screen changes to
http://java.sun.com/j2se/1.4.2/docs/api/java/sql/Connection.html

So let's look at the Connection briefly. We'll look at this some more in just a minute because these transaction isolation levels are defined here in the connection object, and those are important. And basically like I mentioned we have things like the CreateStatement, PrepareStatment if you want a prepared statement, and those are some of the common ones to use.

There are a number of their methods in here, also some transaction related things. Because we use jta and we'll use xa aware connections, or transaction aware connections, rather than these plain connections, connections will be enlisted in the transaction already, so we don't typically work with these Rollback and Commit methods that are directly on the Connection object.

So typically we'll just use this to get metadata from the database, this DatabaseMetaData object has a whole bunch of operations for getting information about table structures or whatever else in the database. So we'll come back to this in just a minute to look at the different transaction isolation levels.

00:08:19 screen changes to
http://java.sun.com/j2se/1.4.2/docs/api/java/sql/ResultSet.html

Let's talk for a minute about the ResultSets. So usually once an operation is done, we'll get a ResultSet object back, and with the ResultSet we can scroll through the results and use the getter methods to get values for each columns in the results, not that the columns could be in a single table or across various tables for a join or whatever.

A number of methods here for moving back and forth between the results, moving so you can move forward and backward, got to the first one, after last, before first, different things like this, various getter methods for different types of data, some of them are streamed over, some of them are just small values, there's a lot of variety in here.

This is actually a fairly large class because of all the different data types, and it has getter and setter methods for all of them. Some of the important things when you are executing a query, you'll typically specify how you want certain things to be handled, like the type of ResultSet object. For the scrolling, do you want it to be forward only, so you can start at the first result and scroll to the second one and to the end, and never move backward, that often means also never jumping forward to a certain spot, a number of databases restrict that for the forward only.

So if you do want to jump to a certain spot or move backwards, scroll_insensitive or scroll_sensitive are necessary, scroll_insensitive means that you can scroll forward and backward but it will not be sensitive to changes made by others.

In other words it's not constantly talking to make sure that the rows it's showing you are up to date. If you do scroll_sensitive there's a bit more overhead, because it is more sensitive to changes made by others, depending on the transaction isolation level of course, which we'll talk about in a bit. So it is talking to the database as you move back and forth through the ResultSet, just to make sure that the values you are getting through the ResultSet object are up to date.

Okay some other ones that are important here, the Concurrency, you'll typically have a read only or updatable. Usually for the most part we will just use the read only Concurrency in OFBiz, with the updatable one as your iterating or moving a result set.

You can not only get values but you can set values and have those get sent back to the database. We don't typically do that in OFBiz; with the entity engine we're basically just reading information or writing information, and we keep those fairly separate.

So for the concurrency we typically just use read only. Okay so there's a quick summary of the ResultSet. One thing to note about the ResultSet interface, the entity list generator in OFBiz allows us to handle very large sets of data basically as a wrapper around the ResultSet.

For the most part the entity engine, when it gets a set of results back like this, will go through them. And for each result it will make a generic value object that represents that result, be it an entity or a view entity or a dynamic view entity or whatever, basically what the entity engine uses to make it easier to interact with the data.

For very large queries, where you might get potentially even greater numbers of results back then you can fit into memory on this server, it's important to have something like the entity list generator that just wraps around the ResultSet; instead of just pulling all of the results back it iterates through the results. And for each one, so you can get a generic value object for one result as a time, or a number of results at a time, or you can have it get a subset of the results like the first ten or results 100-110 or whatever as you're paging through things in the user interface or if you have an algorithm that can look at things one at a time, just kind of iterating over them. That sort of thing.

With the entity list generator it keeps a connection to the database open, by keeping the ResultSet open. So just as the ResultSet needs to be closed when it's done being used so the network resources can be freed up, and the object the connection can be freed up, the entity list generator needs to be closed so the ResultSet that it wraps can be closed up.

00:13:03 screen changes to
http://java.sun.com/j2se/1.4.2/docs/api/java/sql/package-summary.html

Okay so that's it for the entity list generator. Typically with JDBC, in order to get a Connection rather than creating it directly from the JDBC driver, you'll have some sort of DataSource that you can use, and it says here it's basically just a factory for Connection objects.

The XADataSource is a factory for XAConnection objects. This XA prefix is referring to being transaction aware. So the XADataSource, typically you'll have a DataSource in a full enterprise application that is not only transaction aware but is also Pooled. The reason to use Pooling is that connections are very expensive to produce, so you can establish a connection with the database that stays idle for long periods of time without any problems, and these connections can be reused over and over and over.

So they're basically used, pulled out of a Pool and used, and then when they're done being used they're put back in the Pool so that other processes going on can use the same Connections. For efficiency purposes we use a PoolToDataSource that has Pool Connections.

For transaction management we want a XADataSource that supports XAConnections that can be enlisted in the transaction so that you can have multiple connections along with other transaction aware resources that are all enlisted and therefore considered in the same transaction. So these are some of the more enterprise oriented things.

00:14:50 screen changes to
http://java.sun.com/j2se/1.4.2/docs/api/java/sql/package-summary.html

But when it comes right down to it basically through all the fancy infrastructure underneath, all you're getting back is Connections and ResultSets and to do things like PreparedStatements and Statements.

Now with the entity engine you can get a connection through the entity engine that takes advantage of all that infrastructure and then do database operations directly on that, writing your persistence code in JDBC.

The Connection will be enlisted in the transaction, so everything you do is basically transaction aware, included in the transaction and such. So you don't have any real problems there, it's just you can use those Connections as if you were writing your own JDBC code in your own application somewhere.

For the most part the main things to remember about these, because it is generally best to just run your database operations through the entity engine, the most important thing to understand is how the flow of these things are.

00:15:53 screen changes to site/training/AdvancedFramework

And one important aspect of that, the next topic I want to cover, is the Transaction Isolation Levels. Transaction Isolation Level refers to how isolated transactions are from each other. In a server you may have hundreds of transactions that are active at any given time, accessing different parts of the database and sometimes the same parts of the database. So the transaction isolation level configured for a database connection

will determine how insulated each transaction is from the other ones.

00:16:28 screen changes to http://java.sun.com/j2se/1.4.2/docs/api/java/sql/Connection.html

So let's go ahead and look at these. There's some documentation on these right here in the Java docs for the Connection class. Here's the url if you want to look at this further (highlights url http://java.sun.com/j2se/1.4.2/docs/api/java/sq;/Connection.html). Up here at the top it has a quick summary of these, and then there are some more, we'll come back to that in a sec.

00:16:47 screen changes http://java.sun.com/j2se/1.4.2/docs/api/java/sql/Connection/jtml#TRANSACTION_NON

Down here there's some more verbose text about what some of these different things mean, especially talking about dirty reads. They have it split up if you're looking through here so we have a little bit more details about a dirty read there, a non-repeatable read there, and a “phantom” row, or a phantom read down here.

00:17:13 screen changes to http://java.sun.com/j2se/1.4.2/docs/api/lava/sql/Connection.html

Let's hop back up to the top and I'll briefly go over what these things mean. So perhaps the most lenient in the transaction isolation levels is the TRANSACTION_NONE, where there's no transaction supported at all, there's no protection between the different threads.

The next level is TRANSACTION_READ_UNCOMMITTED. With this one dirty reads, non-repeatable reads, and phantom reads can occur. Now as we go through these different levels we get protection from these different problems.

With TRANSACTION_READ_COMMITTED we get protection from dirty reads, they're prevented, but we still have issues with non-repeatable reads and phantom reads.

Stepping up to the next level, TRANSACTION_REPEATABLE_READ, dirty reads and non-repeatable reads are prevented, but phantom reads can still occur.

In the TRANSACTION_SERIALIZABLE level dirty reads, non-repeatable reads, and phantom reads are all prevented, so this is the most strict isolation of the transactions.

The thing to remember about these is that as you go through the levels, uncommitted to committed to repeatable-read to serializable, it becomes more and more costly for the database to manage all the information necessary to provide that isolation between the transactions.

When you get to the point of serializable, there's so much overhead that for high volume environments it actually gets to the point where it doesn't work. Because you have so much locking going on, including table level locking and a lot of nasty things like that for performance as well as for functional behavior because you'll run into a lot more dead locking situations and that sort of thing.

Stepping back on how much more isolated transactions a little bit can help that. The recommended level that we typically go for is read committed, so that dirty reads are prevented, but non-repeatable reads and phantom reads can occur. Read uncommitted is also good. Repeatable read is okay, but this also has some performance issues and you tend to run into deadlocks more often, so it's not really recommended, nor is serializable at all.

One thing to keep in mind with these is that most databases do not support all of the different options, so you have to pick the best one among the supported. The first choice would typically be read committed, and then read uncommitted, unless you have a very low transaction volume, very low concurrency in your application, then you might consider repeatable read to protect things more. And for highly sensitive data you might do a separate database that has serializable transaction isolation.

Let's talk about what some of the settings mean a little more. A dirty read basically means that with two transactions running , the first transaction reads something, let's say the first transaction is going through and doing operations, and writes something to the database. While the transaction is still active, in other words it hasn't committed or rolled back yet, and in fact it doesn't know if it's going to commit or roll back yet, the second transaction comes back and reads the record that the first transaction just created or updated, and sees the data that the first transaction just dropped into the database, even though the first transaction hasn't committed or rolled back yet.

So that would be called a dirty read, and the reason it's called a dirty read, it's kind of referring to the fact that the data that's coming back that the second transaction is getting has not really be confirmed by the first transaction. So the second transaction could do something based on that, and then the first transaction could roll it back, making what the second transaction is doing invalid. That's basically the idea behind a dirty read.

Non-repeatable reads and phantom reads are very similar. They're basically referring to say a transaction, transaction one, reads a record from the database, transaction two comes along and updates that record,

and transaction one comes along and reads that record again.

If between that first read and the second read in transaction one, it sees the data has changed, that's a non-repeatable read. So if you get up to this transaction isolation level of repeatable read, you can read those same records over and over and be guaranteed every time that you'll get the same data back.

Now the overhead for enforcing that is fairly high, and the need for it in applications is usually not too serious. That's why we usually don't make such a big deal about repeatable-reads. Phantom reads are a very similar scenario, except instead of referring to a single record it's referring to a set of records, so it's basically a repeatable-read over a set of records.

So consider our scenario again where we have a first transaction that does a query, and gets back a set of results, a second transaction drops in a new record, the first transaction does that same query and gets another set of results, so now it has two sets of results for the same query, and comparing them all of a sudden this new record has appeared.

Because another transaction has dropped it into the database and committed it. And so that sort of phantom read in the repeated read isolation level it does not protect against phantom read, where new records will suddenly pop up in result sets, but for a single record the read will be repeatable, just not for a set of records coming back from a query.

If you need that level of isolation then the serializable level is what you're looking for, but again just to warn with that, it's very rare that any application will be deployed using a serializable transaction isolation because the overhead in the database is so heavy, so on any active server where you have an number of transactions going on at the same time, it becomes a problem with deadlocking, and all sorts of things become problems. Full table locks are sometimes necessary, all sorts of ugly things like that.

Okay so those are the basic transaction levels. If you want to look at those in more detail or review this, of course you can always listen to what I said again. If you want to read more there is actually pretty good information in the Java docs for the java.sql.Connection interface. Of course there are various books that cover this sort of material as well.

00:24:08 screen changes to site/training/AdvancedFramework

Okay stepping back to our outline, that's pretty much it for our JDBC section. Let's move on to JTA.

Part 1, Section 5

JTA

This is the Java Transaction API, and the two main interfaces a transaction manager should implement are the UserTransaction and TransactionManager interfaces. It's very common that a JTA implementation will use a single object for UserTransaction and TransactionManager because they actually have quite a lot of overlap between them, and we'll see that in a minute. The TransactionManager is basically a superset of the UserTransaction in fact.

So we'll look at those, and specifically we'll talk about Transaction Control. You know different methods being responsible for transactions and how they should respond to error situations with a rollback verses methods that are not responsible for managing the transaction that should just use a setRollbackOnly. We'll talk about why and some of the patterns there, some of the flow. We'll also talk about suspending and resuming transactions.

00:25:28 screen changes to http://java.sun.com/j2ee/sdk_1.3/techdocs/api/javax/transaction/package-summary.html

So let's look at the JTA API. The package in J2EE for this is javax.transaction, and as I mentioned the main interfaces we're looking at here are the UserTransaction interface and the TransactionManager.

00:25:45 screen changes to http://java.sun.com/j2ee/sdk_1.3/techdocs/api/javax/transaction/UserTransaction.html

Let's look at UserTransaction real quick so you can compare these two, and we'll look at more detail of the flow and such from the TransactionManager. Both of these, by the way, UserTransaction and TransactionManager, are wrapped in OFBiz by a class called utilTransaction that provides some higher level access to these sorts of things and supports some of the flow patterns that I'll be talking about.

That's typically what is used in the different tools, as well as application methods for tranaction management in OFBiz. So this has the basic things; begin a transaction, commit a transaction, roll it back, or SetRollbackOnly on it.

GetStatus for the current thread, is there a transaction that's active, is the transaction being rolled back, or set for rollback only, or being committed, so there are various states that a transaction can be in, or the current thread can be in relative to the transaction, may not be an active transaction. Then setTransactionTimeout. TransactionTimeout is basically how long from the beginning of a transaction it has to complete its operations before it will automatically be rolled back.

So that's a value in seconds. Typical value there is sixty seconds, if you're doing a large import or very large data moving operation or something, then a much longer TransactionTimeout is typically necessary.

One thing to keep in mind if you are using these over level interfaces is that when you set the TransactionTimeout it does not change the timeout for the current transaction, just the next transaction that is begun or starting with the next transaction that is begun.

00:27:39 screen changes to http://java.sun.com/j2ee/sdk_1.3/techdocs/api/javax/transaction/TransactionManager.html

Okay let's look at the TransactionManager interface. This basically as I mentioned before is a superset, so it has; begin, commit, getStatusRollback, setRollbackOnly, and setTransactionTimeout. It adds a few things like getTransaction to get the transaction object.

This is necessary for most of these things, the additions in this TransactionManager class, is necessary for more advanced behavior related to transactions, the transaction object is helpful for a couple of things. We'll look at it in the context of suspend and resume in a minute.

00:28:16 screen chagnes to http://java.sun.com/j2ee/sdk_1.3/techdocs/api/javax/transaction/Transaction.html

But this getTransaction method, usually when you get the transaction object here it would be for doing something like enlisting or delisting a resource, like XAConnection for example.

00:28:26 screen changes to http://java.sun.com/j2ee/sdk_1.3/techdocs/api/javax/transaction/TransactionManager.html

So a typical flow for a transaction, a method will begin it. The method that does begin it is responsible for ending it, either committing the transaction or rolling it back.

So the method that does that should use the tri-catch and finally clauses to make sure that if it begins a transaction, that the transaction is committed or rolled back before that method loses flow control in the execution. Of course commit is used in the case where everything is completed successfully, and rollback is used when there is some sort of an error, so that everything in the transaction can either succeed together or fail together.

This way when you're making changes to the database you can go along making changes, and if you find an error somewhere down the line, you can just rollback the transaction rather than trying to keep track of the original state of everything along the way, which would be enormously tedious.

So transactions are fantastic to manage that sort of situation. If you are in a method that is participating in the transaction that did not begin the transaction, and you run into an error, you should never run back the transaction, only the method that began the transaction should roll it back.

What you should do instead is call setrollbackonly. This will set a flag on the transaction that tells it it should only rollback regardless of what happens in the future, it should only rollback.

If the method that is controlling the transaction that began the transaction, somehow the exception doesn't make it back to it, the error doesn't make it back to it, so it doesn't know that there's an error, when it tries to commit the transaction, the TransactionManager will see the setRollbackOnly flag and rollback the transaction instead of committing it, as well as of course throw an exception to let the method know that the transaction was rolled back instead of committed.

Of course if the error message or exception does get all the way back up to the method that began the transaction, then it will typically call the rollback, so the setRollbackOnly flag which protects it from being committed won't do anything in that scenario because it's being rolled back by the controlling method anyway.

So that's basically the point of having rollback verses setRollbackOnly. Onto the next topic of suspending and reusing transactions. Sometimes you'll have a larger process that has separate data to maintain or separate things to execute, that should not be in the big block of operations that will succeed or fail together. In those cases what you can do if those little operations need to happen before the big operation is finished, and in the same thread, what you can do is suspend the current transaction.

When you do that basically it will pass you back an implementation of the transaction interface, so once it's suspended there's no transaction associated with the current thread, you can begin the transaction, do your stuff, and either recommit or rollback that second transaction. Then the first transaction, once you're ready for it again when the second transaction is finished, either rolled back or committed, then you can resume the first transaction by passing the transaction object into the resume method, and then go on with that and eventually get back to the point where it's finished by either being committed or rolled back.

That's the basic idea with suspend. For the most part managing these things directly is not necessary, because tools like the screen widget and the even more importantly the service engine handle TransactionManagement for you. The service engine actually has pretty flexible options for how you want it to behave relative to the transactions. For example if you want a service to run in its own transaction, regardless of what

the service calling it or the method calling it might be doing with transactions, then you can tell it to always run in a new transaction. Then if there is an existing transaction the service engine will suspend that and begin a new transaction for that service, when that service is done if it's successful it will commit the transaction. If not it will roll it back, which will finish the transaction for that service and then it resumes the original transaction and lets the flow continue for the original service or method that called it. It can handle some things like that.

So that can be very valuable in certain cases, especially if you were going through a big process and have information related to the process but not theat you want to keep in the same transaction that you need to save over time, things like that. One of the places where this is used in the engine, actually, is for managing sequences every time a new bank of sequence values is needed. Every time we need to get a new bank of sequence values from the database it will suspend the current transaction, get those sequence values, committing to the database so the updated sequence is there. Part of the point of this is to isolate that as much as possible, because those sequence records can be a major point of contention between different threads. So it gets the bank of sequences in the new transaction, and then it can use those in this other outstanding transaction without making the operation of getting a bank of sequences take so long, or be involved in this larger thing.

So that's one example of where a suspend and resume is of value.

00:34:54 screen changes to site/training/AdvancedFramework

Okay that's basically it for JTA, for TransactionManagement and such in OFBiz. So these are all very low level tools that are part of the Java and j2EE specifications, and we use various implementations of these specs. Of course for J2SE you can run on different JDKS as you wish, and the same thing for the J2EE level specifications, which are implemented in an application server typically.

So we have embedded libraries for those things to run OFBiz from out of the box, or you can run OFBiz inside a more traditional application server, and just configure it to get things through JNDI and such in order to have access to the resources of the application server.

And that's basically it, so if some of this was somewhat confusing, or was going over your head or going too fast, you might want to consider listening to this section again. Or don't worry about it too much; we'll see a lot of these concepts used in context of the higher level tools that are part of the OFBiz Framework, so you'll get somewhat of a review of them there.

Also a lot of these things are not really necessary for those that are doing higher level programming or higher level application creation based on the Open for Business framework.

Of course on any team it's helpful to have at least one person on the team who has a good understanding of these lower level tool as well as the OFBiz tools, so that if problems arise, or if a little creativity is needed for deciding how to solve a business problem or meet a business requirement, is maybe a better way of putting it, then that sort of knowledge is available.

But for day to day operations, day to day things for creating or even doing data modeling and defining and implementing service and creating user interface for the most part you don't need to worry about these things, understanding the basic concepts is enough, and the details can be left to exceptional circumstances where the tools are not really not sufficient for the type of functionality that you need to create.

And of course with that type of person, that kind of expertise, a good way to get that onto a team, especially a smaller team, is to have a development support contract with someone who can offer help in those areas.

Okay that's it for the Just Enough Java and J2EE section of the Advanced Framework training. The next section we'll go onto is to talk about the OFBiz base, which involves various utilities in the OFBiz loader, and the Containers and Components which are used for configuring OFBiz and make up the structure and layout of all of the different parts of OFBiz. Which is important because it is a fairly large project, and almost makes it easier for you to add your own things to OFBiz without disturbing too much of what already exists; you can build on top of it fairly easily.

So we'll look at some of the details of these things that apply whether you're running OFBiz out of the box with embedded application server components, or if you're running OFBiz inside another application server. It's a slightly different flow but the Containers and Components are still used.

And that's it. So we'll see you in just a bit, getting onto the OFBiz base.

Part 2: The OFBiz Base: Containers and Components

Introduction

Starting screen: Advanced Framework Training Outline

This is part two of the OFBiz Advanced Framework training course. We'll be talking about the OFBiz base,

and particularly in the OFBiz base about Containers and Components, what those mean and how they're used to configure Open for Business to add in your own things. These are of course used for the out of the box tools and applications that come with OFBiz.

Part 2, Section 1

OFBiz Base Utilities

So let's first talk about the OFBiz base utilities and the loader.

00:00:39 screen changes to containers.xml

If you look at the base directory which is right underneath the OFBiz directory, here in the base directory, in the source directory, we have a base package and a start package, underneath each of these we have org, ofbiz, and base. This is base and it has various other things in it. And this has base with just the start part of it.

So there are various utilities that are in the org.ofbiz.base package. The various component configuration classes, and other configuration classes for classpath management, JNDI things, all sorts of different things: resource loaders, different types of them from URLs, files, and various other things are in here.

There are also the main classes for managing containers. Basically a container is just a class that implements the container interface. These are various things, some of these are actual containers, the BeanShellContainer, the ClassLoaderContainer, we have a cached ClassLoader that is used in OFBiz. Some application servers include that, so you don't need to use that container if that's the case. If your application server doesn't use that, it's very nice. Also out of the box having a cached ClassLoader in place helps performance significantly.

I guess that's the container. We have some cryptography things. In the location we have some location resolution stuff. Some of the main things you'll see in here are the component location where we have the component://, and there's some other things like the standard URLs from the OFBiz home relative to the OFBiz home, different things like that.

Splash. This is for splash screen stuff, I believe that may only be used in the point of sale component right now. These unitTests and baseUnitTests are just some general unit tests related to the base package, and then the Util directory has a number of things in it.

This is where the main OFBiz cache manager is, we have some collections and string utilities. Some template rendering things, these are mostly just for FreeMarker right now, although we do have some XSL stuff in here. But FreeMarker work are in the FTL transform things in here. Using FTL is an alternative to XSL basically.

And then we have some other general utilities, here's the cachedClassloader I mentioned before. There's the container that loesd it is up in the Container directory. Http client, file utilities, BeanShell utilities, objectType for conversion and checking to see if an object is of a certain type or inherits a certain class. So this is some extensions to the reflection API mainly. Just various miscellaneous things in here, you'll see them used in various parts of OFBiz. Basically just lots of different utility classes that are used to kind of extract common functionality.

Some of these things are actually fairly similar to what you'd find in the Apache Commons, and some of those things we've replaced, a lot of things though that still remain, though are different from what is in Apache Commons, but anyway the concept in general of these utilities is somewhat similar in many cases to what you'd find in the Apache Jakarta Commons.

Okay so let's talk about the start, how OFBiz starts up. We'll talk about this for just a minute, and then after we go through this briefly we'll look at a startup log to see briefly what is done on startup.

So when you start OFBiz it's typically done using a command like java-jar ofbiz.jar. This will basically just use ofbiz.jar as an executable JAR file, and starting up this way with that command will run the executable JAR file.

00:05:34 screen changes to containers.xml

The executable JAR file itself actually comes from this directory right here, the start directory. So when OFBiz starts up this is the only stuff that will be on the Classpath, which is why we have certain configuration files sitting in this start directory here, namely these properties files. The default one is the start.properties file, and there are various other ones.

If you do “-test” then it will use this test.properties file, “-pos” will use this one, “-jetty” will use that one as an alternative to Tomcat which is the default, and the install properties file for running “-install” which is the data loading routine and such.

Let's look at the start.properies file for just a second so you can see what is in here. First of all we have an option; is the tools.jar required? It'll check that and if it is required then if that's set to true and if it doesn't find the tools.jar...in other words if you're running with just a JBM that has the Java runtime environment but not the JDK, the full Java Developer's Kit, then it would throw an error if this was set to true.

These are some optional things, an admin host and admin port. There are other ways of doing this administration now so those aren't necessary. These are

some of the general things, directories relative to the OFBiz root, just to tell the Startup Container where to find different things. Such as where to find the libraries, where to find the Config directories. This is put on the Classpath, all the JAR files in here are put on the Classpath. The base JAR can tell it explicitly where it is located so it can put it on the Classpath.

The logs directory. Again this is relative to the OFBiz root. That's the login directory. And the Container Configuration XML file. And this is the file that specifies all of the Containers that should be loaded when it starts up. And we'll look at what that means in just a second.

One of the differences between this start.properties files, this test.properties file, and install.properties pos is that they have different container.xml files, so different Containers that are loaded when they start up. Some of this other stuff is basically a shut down hook, autoShutDownAfterLook. These sorts of things do vary somewhat, but for the most part these settings will stay fairly constant, and one of the main things that changes is just pointing to different container.xml files.

So this is the first thing that happens when it starts up. The executable JAR will actually hit the main routine in the start.java file, and then we get going from there.

This Classpath.java file basically does the initial Classpath setup. So once the startup file is there, basically it just starts loading, or once the executable JAR file is run, it basically just starts loading Containers, according to the containers.xml file specified here.

Part 2, Section 2

Containers

Here's the default on when we start OFBiz, here's the Containers file. The first container that we load is the Component Container, and what the component container does is it reads in the configuration for all of the different components that are configured here, and it reads them in according to the component-load.xml file.

Let's grab the root one. And that's the framework one. We'll look at the rest of these in a minute. Here's the main Component Loader XML file, so we're loading all the components in all of these places in OFBiz home/framework. We'll load all of the components there, or if it has its own component-load.xml file, we'll just load the components specified there.

00:10:03 screen changes to component-load.xml 1

So we look at the component-load.xml file in the framework directory, and here it is. First we load Geronimo in the entity engine, Catalina which is Tomcat, Security Component, Datafile Component, and so on all the way to the Example Component.

00:10:16 screen changes to component-load.xml 3

Next thing it does is load the Applications Component. Now when it's loading the component I should remind you that basically all it's doing is loading the configuration for these components and keeping that in memory, so as other parts of OFBiz are started that configuration will be in place for it to use.

00:10:38 screen changes to component-load.xml 2

Then it loads all of the Application Components, which are here, from the Party Application all the way through the Point of Sale Application, just loading the component configuration files.

00:10:45 screen changes to component-load.xml 1

Then we load the Specialized Components, HotDeploy Components, and this one, with a single load component, we'll just be treating the website directory here as a component.

You'll notice all of these basically use this singular load component to load these individual components and specify the order they should be loaded in. The order is important for certain things like the Classpath order, and also so certain dependencies in compile time will typically use that same order here.

00:11:33 screen changes to ofbiz-containers.xml

Okay going back to the OFBiz-containers.xml file. Once it loads all of the component configurations the next thing we do is set up the ClassLoader. The ClassLoader container will put in the cached ClassLoader that I mentioned. And now that we've looked through all of the components, we know what all of the classpath entries are that are needed for each component, and this ClassLoader container will put them all on the classpath. And then that for a new ClassLoader, and then that ClassLoader will be used as the main ClassLoader for this application as it runs.

Now one thing to note about these containers is that in principle they are very similar to UJMX, so when you look at a JMX file it will typically have something a lot like this. And underneath you'll have something like this sort of property, a name value pair, which is just a way to pass parameters into the container.

This is starting the namingService, and we want to tell it which port to attach to. This one is starting the RMI Dispatcher that's part of the service engine.

Another thing to note about each of these containers is that they have both a startup and a shutdown method in them. And you can implement both of those methods (they're part of the container interface) to do things that are needed for setup and tear down of the application.

So the RMI Dispatcher has various parameters to pass to it. You can see here, this is part of the service en-

gine that we'll cover later on when we're talking about the service engine itself.

We load the javamail-container, this is a JavaMail listener basically, that will basically pull the mail account periodically, according to the delay time you set here. I believe that's in milliseconds, so that would be about five minutes. And you can configure where to do this.

When mail is received through this JavaMail container, this is also part of the service engine by the way, so we have the service package here, the service.mail package. There are various rules you can set up to determine the service to call as the email is received, and there is a service right now to take in incoming email and create a communication event from it, and associate it with the parties. Like it'll look up the parties by email address and associate an event with it, things like that. Of course you can add custom email handlers to do anything you want with this.

So this is another thing that's a container, basically something that's loaded at startup. This is something important that you want to load at startup, which is the web application container, and this is just an embedded version of Tomcat, or Catalina, from the Apache. And there are various parameters to specify. Most of these parameters, there are some OFBiz specific ones like the delegator name, but most of these things are specific to Tomcat.

With all of these different access log things, and then setting up the AJP connector, setting up the HTTP connector, setting up the HTTPS connector, each of these basically just has all of the parameters, just the standard parameters that are part of the embedded API for running Tomcat.

Okay here is the last container we have here, and this is one that is kind of important to know about. While it's useful in development and pretty much everything that comes directly out of SVN, or the source code repository for OFBiz, is meant to be in a development mode.

So we have timeouts for caches and configuration files that are very helpful during development, but are a very bad idea, can cause a lot of problems, for production. There are various changes described in the basic production setup guide, that should be done.

We'll talk about some of those changes that should be done when we're talking about the configuration deployment and maintenance, and that's another one of the advanced courses related to OFBiz. And that is kind of appear to this Advanced Framework course.

The Beanshell container is one that opens up a telnet-port, and then adding one, or port-1 rather, will be the HTTP port, which has a Beanshell client which runs in the browser in a little applet.

So this is nice to have for kind of testing things on the fly or whatever during development. But in production, or on a production server, this should typically be commented out, or the ports at least should be protected by a firewall.

Okay those are the containers, these are basically all of the things that are loaded when OFBiz starts up. So as different things are loading, as an example as the Catalina container is loaded, it will load not only the embedded Tomcat server, but it will also mount all of the related web applications that are configured in the various components. So when we're looking at the startup log in just a second, that's basically what you'll see, is it'll load these containers in order.

As soon as it loads, gets to the point where it's loading the Catalina container, you'll see it mounting all of the webapps, and as the webapps load you'll see it doing other things as well. At some point along here, it's probably when the RMI, as soon as its needed the entity engine will be initialized.

There's nothing that explicitly initializes the entity engine or the service engine here. But as soon as the RMI Dispatcher, the RMI Service Container, is loaded it will need the entity engine, so it will end up initializing an instance of the entity engine and also a service engine instance. So we'll see that as well as we are going through the startup script.

00:18:05 screen changes to terminal-bash-bash-162x60-1

So let's go look at that now, let me just scroll this back to the top. While that's going through the install routine let's hop down to the second running of this, that just has the basic stuff.

So basically I'm just in this case starting OFBiz with the start ofbiz.sh script. The first thing it does, as it mentions here, is loading the containers. You'll see in the UtilXml class here that it well tell you every time it reads an XML file. We do this so you can see what's going on in startup and during execution as well as so you can check on performance things, because most of these files should be cached in production. If you see that an XML file is being loaded over and over than something is wrong when you're in production.

The first one we're looking at is the ofbiz/base/config/ofbiz-containers.xml file. And that's the one that we just looked at. Then it's reading this component-load.xml file, and the first thing it does there, based on that XML file, is auto-load the component directory for the framework. So we end up reading the component load file in the framework directory here.

Part 2, Section 3

Components

Then we go on and load the ofbiz-component XML file. This is the component definition file that every single OFBiz component will have. So ofbiz-component will look for that file in each directory, and if it finds that file then that directory is considered an OFBiz component directory.

We see with Geronimo, entity, Catalina, Tomcat, security, datafile and so on, these are basically just the components as listed in order in this component-load.xml file. The first thing we'll see and hear as soon as it's done with the framework directory, the example being the last component, we have the, will be auto-loading in the component directory of applications.

So it finds the component-load.xml file there, and based on there starts loading the components that are listed in that file; party security, extensions, content, workeffort, and so on. You'll notice some little warnings here of some things that need to be cleaned up in OFBiz right now, like the billed wide directory does not exist in the human resources component.

This is something that it's noticed that we've said that there would be a billed wide directory to put on the classpath but it does not exist, so it's putting a warning message about that. That's because right now in the human resource component there aren't any classes to compile and deploy, so that can be cleaned up right now, just by removing that from the list of directories to look in for JAR files to put on the classpath.

Okay so the next thing we run into is the specialized directory. See the component load there, hot-deploy directory. I do have a couple of components sitting in the hot-deploy directory, these are the open source strategies OFBiz extensions, CRMSFA component and financials component. We'll be talking a whole lot about those, but a little bit in the OFBiz application overview. These are extensions to OFBiz and separate open source projects, but very handy to have, especially the financials module if you're doing any sort of accounting for automating posting, and also reports and stuff.

And the sales force automation module can be helpful as well, especially if you have a sales force. Okay so once all the components are loaded, you can see the next thing we're doing. It created the cached ClassLoader, so all the classpath configuration stuff is done. And we're hitting something that needs an instance of the entity engine, the GenericDelegator, so we're initializing a new one.

So you'll see its reading all of the different stuff, first of all the entityengine.xml file, which is the configuration file for the entity engine, and we'll talk about that more in detail in a bit. And then it's reading the entity model files, and all of these entity model files are specified for each component in the ofbiz-component.xml file.

So each component is really meant to be a stand alone self sufficient directory that can be plugged into OFBiz, and that's accomplished by being able to configure everything about that component in the ofbiz/component.xml file. So all of the classpath directories or JAR files that you want to put onto the classpath can be specified there, and all of the entity engine resources, entity models, entity ECAs, entity groups, and those sort of things. Same thing with service resources.

One other entity resource is DataFile to load the XML import files to load as part of the install process. Service engine resources like service definitions, service groups, service ECA rules, the location for the files of all these sorts of things can be specified for each component in the ofbiz-component.xml file. And web applications can be specified there.

The last thing that's been added fairly recently is we have some XML files that describe tests that can be run for each component. So the test runner in OFBiz looks at the test execution XML files that are in each component as well.

A component can have all of these things and kind of be self sufficient and plugged in. And then when those global processes are run in OFBiz it will pick up what each component needs done and take care of it. That's certainly the case with the entity engine; we have all these entitymodel.xml files from different components that are being added into the entity engine for entity definitions, so it knows about all of them.

So these are the entity model files, the entity group files, and we'll talk about what all these entity model and group files mean. We did talk about them a little bit in the framework overview, the Framework Introduction videos, so you should know more or less what those are, and we will be covering those in more detail in these videos and in the entity engine section.

So as soon as it loads all of the entity definition files we do an entity definition check, which does kind of an internal consistency check on the entity definitions to make sure that relationships to other entities are all valid. The types for each field are valid according to the fieldtypes.xml file that's being used for this database that's being initialized, and various other things. So there's quite a bit that it checks to do kind of a sanity check on your entity definitions.

And it also does a checkReservedWords. We're seeing a little message about the array of reserved words being initialized. So there are 1023 words reserved words that we have in there right now, and that will probably grow over time.

Okay so the reading to the entity egnine.xml file. Going through we're creating a default delegator looking up the helper that we need for the group org.ofbiz, and

we'll be loading the database associated with that helper. And we do a database check, we're showing here some information about the database, some of the different options and such. And then it gets all table information, all column information from the database, and it checks that against the entity model that we have.

Now these are things that the entity engine does to be sure that the metadata in the database matches the metadata that it has. And this can be turned off, and there are various options related to them that you can specify in the entityengine.xml file, just like everything else here. And when we go over the entity engine configuration we'll be looking at more of that.

With the default delegator we actually have two helpers with two groups, so we have the org.ofgiz.odbc group as well. This group has a few entities in it in its data model, not very many, just a few that are used for integration pirmarily. The main target for these right now is just the Worldship or I think that's the name of the package.

But anyway there's a specific set of utilities that uses this separate database, and the entities are kept separate from the main database because they'll typically be in some sort of an integration database that both OFBiz and the other application look at.

So here's an example of how a separate database can be configured using the entity engine, but accessed through the single delegator, and then depending on the entity that these different operations are being done on it will choose one database or the other. But anyway for both databases it initializes and checks them here as the entity engine is starting up.

Now the entity engine is done initializing, and we're ready for creating a new ServiceDispatcher. Here you can see it loading all of the service definition files. This is another set of files that is specified in the ofbizcomponent.zml file, so that's how the service engine knows where to look for all of these different configuration files. That's true for service ECA files, that's true for service definitoin files themselves, and also for service group files.

All of these things, I guess the initial ones are ECA and group files, down here we have all of the services.xml files, the typical name you'll see for those, sometimes with an extension like this; service_test in that case. And there are a number of other of these.

So here we're loading the service definitions files, you can see there are quite a few of them. Let's get down to the bottom of that. Okay so basically we've loaded all the service definition files at this point. We have another dispatcher, a LocalDispatcher, that it's creating.

The RMI containers that I mentioned would be the first things that need the entity engine and service engine, which is why they're being initialized here, and here's the rest of the initialization for the RMI dispatcher.

00:29:35 screen changes to containers.xml

Okay so the next thing that we had, let's hop back and look at it real quick. In the containers file we saw the component files being loaded, the class loader being set up, and the naming container. There wasn't really anything special going on there but we saw that being set up. And we saw a whole bunch of stuff, the entity engine and service engine initialization going on, and then the RMI dispatcher being set up.

00:30:03 screen changes to terminal-bash-bash-162x60

Once that's done the next thing we should see is the Java mail container being set up, so let's look at that. Here are some messages from the Java mail container being set up. The service MCA, Mail-Condition-Action is what that stands for, kind of a variation on the ECA rules but mail specific rules that you can use for this.

So here's some stuff for the mail setup, and there are of course some error messages here, because of course we just have a bogus set of server locations and passwords. We have this user and host and such, so it's not going to be able to communicate, unable to connect to mailStore and so on, because we just have bogus data in there. And when you configure it properly then it will connect, and you'll see different things in this file. Okay so the next thing that's happening here is we're starting to initialize, first of all the context filter. We have some web.xml stuff that it's mentioning here.

00:31:01 screen changes to containers.xml

So we're getting into the section where we're done with the Java Mail Container and we're getting into the Catalina container. And as the Catalina container initializes, and as I mentioned before it will initialize Tomcat, also known as Catalina. And then as it's doing that there are a whole bunch of web applications that are configured in each ofbiz-component.xml file, and it will load all of those web applications.

00:31:27 screen changes to terminal-bash-bash-162x60

As it loads each web application we'll see a whole bunch of messages for it. So you'll see this one, the example/webapp/example, or the webapp example that's in the example component that's being loaded there. We have the party manager in CRMSFA, it has a surveys webapp.

These are various web applications that are being loaded, so basically it's going through and loading all of those, there are various context parameters you can

see from controller.xml files. You can see where it's loading all of the configuration things, and config handlermaps for the different handler types. All of the request maps are in here, all of the view maps, you can see this application for example, which would be loading control servlet mounted here.

This is the content manager, and it has 309 requests and 129 views in it. No, sorry, we're looking at the manufacturing one here, so it's for the next one. The content one has, we'd look up here to see it, the content one has 87 requests and 42 views. So it's going through and loading all of the different webapps, so eventually we'll get down to where that's done, where it's finally created its last view-map for its last webapplication, then Tomcat actually starts up the different connectors.

In this case we have an AJP connector, an HTTP connector, and a TLS or HTTPS connector all configured. So it's loading all those connectors, and then it reports that it has successfully started Apache Tomcat. In this case it's 5.5.9, this log that we're looking at is actually from a couple days ago, and 5.5.17 is actually the version of Tomcat you'll see in there right now.

Okay these messages are from the BeanShell container, where it's starting its things on ports 9989 and 9990.

00:33:49 screen changes to containers.xml

And that is the last thing on the containers file down here, the BeanShell container.

00:33:57 screen changes to terminal-bash-bash-162x60

So that basically means we are done with the startup of OFBiz, and of course there is a nice little warning message in here that “these ports are not secure, please protect the ports.” As I mentioned, for production usage these are typically commented out or protected by a firewall or some sort of thing.

Okay the rest of these messages are some things that are happening. It's common that when you start OFBiz in certain cases, especially where it hasn't been started in a while, you'll have a whole bunch of automatic jobs being run. So back order notifications being run, Update Shipment From Staging is being run, Purge Old Store Auto-Entered Promos is being run, Clear EntitySyncRemove Info is being run.

These are just automatic services that are configured. There are settings for them sitting in the database, the job control settings for the service engine are in the database. So it's reading those and automatically running these things. In this case what I did when I started up is I just let them run, as soon as they were done I shut down.

So this is what you'll see when OFBiz is being shut down, that it's shutting down the containers. So in reverse order it just shuts everything down, shutting down the dispatcher for the service engine, stopping the Java mail puller, stopping the RMI dispatcher, and then it's done. That's basically all that's involved in a full cycle of running OFBiz.

This that we're seeing now is just another execution of OFBiz that I did, and basically the same things are happening. In this case there weren't any services to run by the service engine job manager after it was done, and so after BeanShell was open there you see that it's just going through the shutdown process.

So that 's the basic process for the loading, the startup, and the shutdown for OFBiz. Here's a more recent one that I did, so we can see the actual current version of Tomcat that's in subversion 5.5.17

00:36:23 screen changes to containers.xml

Okay so that's where the containers and the startup executable JAR files fit into the picture.

00:36:30 screen changes to Advanced Framework Training Outline

Before we go on to start talking about components and the details of components, I'd like to mention one more thing about containers. We just looked at the startup process for OFBiz being run out of the box, where you're using the OFBiz startup JAR, that executable JAR to run OFBiz. So application server components are embedded in OFBiz, rather than OFBiz being run inside and application server. It's also possible to run OFBiz inside an application server.

00:37:10 screen changes to containers.xml

And when that is done there is a separate containers.xml file that is used. All of these containers.xml files are in the config directory here. And if I remember right this limited-containers.xml file is the one that is used.

00:37:25 screen changes to limited-containers.xml

So the only things it's doing, you'll notice there is no Catalina being started up here, nothing like that, because it assumes that being deployed in an application server, that you will already have a servlet container, a transaction manager, and all of these sorts of things should already be set up as part of the application server.

These containers will be loaded automatically if they haven't been already loaded, so as soon as the first web application is mounted it will check to see if the containers have been loaded, and before it does anything else if they haven't been it will load those containers.

When the first web application is mounted in the servlet container of the application server, it will go through and load all of the component configurations, basically just run through and load this container file, set up the Classloader, set up the RMI Dispatcher, and set up the BeanShell container.

When you're running in another application server you can see there's less that gets set up here. Notice this doesn't have, this is kind of an example thing, typically when you're doing this you'll have a fairly custom configuration. If you wanted to use the Java mail listener then basically you would just copy that from the main OFBiz container.xml file. You would copy that into this limited-containers.xml file.

And that's what happens when OFBiz is run inside an application server. One of the consequences of that is that we're telling it not to update the Classpath here, when it's running the components, because typically in an application server it will not allow you full access to modify the Classpath.

If you do your web applications or other web applications, or anything that's shared between, the web applications would not be able to see those classes. There are certain things that are done automatically, pulled directly from the component.xml files when you're loading OFBiz in the way it's run out of the box with an embedded application sever.

When you're running in an external application server that OFBiz is running inside of then you have to configure those things manually, and those are Classpath entries. All Classpath entries must be loaded through whatever the application server supports, they will not be done automatically just because of the component configuration here in the OFBiz component.xml files. That is also though for webapplications.

For service engine and entity engine resources, and the testRunner resources, those are all handled automatically because they are things that are used by different OFBiz framework tools. So you don't have to worry about doing anything special with those. But the Classpath entries, the web applications, those must be configured through whatever means the application server supports.

00:40:37 screen changes to terminal-bash-bash 162.60

To help with that there is a component in OFBiz. Let's just look at this real quick in the framework directory. There is an appserver component. And in this directory there is a Readme file, which basically tells you how to run these appserver setups. And basically what this gives you is a set of templates to generate configuration files, basically to generate configuration or deployment script or these sorts of things for different application servers, based on templates and using the information that is in the ofbizcomponent.xml file for each component. So this process is not so very manual.

Let's look at the templates real quick. Here's some templates for the Orion application server. We also have some for WebLogic. This is certainly an area of OFBiz that could use some more use. It's very common that when people do an application server deployment they do a very custom one, so it's not very common that these templates get contributed back to the project. But these ones can be very helpful for these application servers, and they can be good examples for other things.

Let's look at this one. This HTTP site file is one of the standard Orion files, and here you'll see this is basically just interpreted with FreeMarker. So you're seeing some FreeMarker markup right here. It's going through the entire webapps list, creating a variable through each pass through the loop called webapp.

This is just a template, and we're creating one of these webapp lines, one of these webapp XML elements or tags for every single web application. This is where all the webapps are configured. And all we have to do is tell it the name of the webapp, and the context root of the webapp.

So those are some of the things that are available to these templates for automatically generating the configuration files. There's also here an application.xml file, this has all of the Classpath entries in it so we have all the Classpath JARs and all the Classpath directories.

This is one way of configuring them in Orion. It's very nice you can specify external JAR files and external directories to put on the Classpath in Orion. Some other application servers don't do that, so you have to get a little bit more fancy in the types of deployment scripts or configuration files that are used for those application servers.

Anyway these templates are helpful even if you're not using the out of the box way of running OFBiz. If you want to run it inside an application server of your choice, be it WebLogic or WebSphere or Orion or the Oracle application server, or whatever application server that you want, even JBoss or Geronimo, you can use these templates to help make your deployment easier in that application server, and more automated.

00:44:09 screen changes to Advanced Framework Training Outline

Okay so that is pretty much it for the OFBiz base utilities and loader, or the OFBiz executable JAR files to start it up, and Containers. The next section we'll talk about more detail of the components.

We talked about the component-load.xml files as we were talking about the load order, the different things

that happen as OFBiz starts up, and we looked at the one that's in the base Config directory which points to the different component-load.xml files for each of these other directories: the applications, framework, specialized, we also saw the hot-deploy one. I guess those are the only other main ones.

Now let's look at component definitions and then we'll look at the component directory structure conventions.

Here's a little thing I'll just mention real quick. For various URIs in OFBiz you can refer to things that are in Components by using Component:// component name and relative path. This is the relative path within the component, and we'll look at some more of that when we cover the other topics.

So let's look at the component definition. These always exist in the root of the component directory in a file that's always called ofbiz-component.xml.

These are different things we'll see in there. I've already mentioned all of these, but let's review them real quick: Resource Loaders, Classpath Entries, Entity Resources, Service Resource, Test Resources, and webapps. So those are the different things that are declared in a component.xml file.

00:46:11 screen changes to ofbiz-component.xml

Let's look at one. So this is the ofbiz-component.xml file for the entity component in the framework that has the all of the stuff for the entity engine. The first thing we're doing is specifying a resource loader. The loader, every loader will have a name. There are different types of loaders, and we'll see these here: Classpath Loader, Component Loader, File Loader, and a URL Loader.

With File Loader and URL loader and I guess with the other ones as well, but mainly with File and URL loaders, you can put a prefix on the loader so they're always prepend something. It's basically a matter of convenience.

Basically this is a component loader, which means all of the locations inside the loader will be relative to the root of the component. And we're going to call this loader the mainLoader. That's not used for the Classpath entries, but for entity resource, test suites, service resources, and so on. We're basically going to specify two attributes that work together. So the name of the loader, this loader name of Main corresponds with this Resource Loader name of Main.

Loader and location work together, so this location is the location of the entity resource that we're looking at, entitydef/entitymodel.xml, and that will be relative to the root of the component because this loader is a component-type resource loader. This is the most common one you'll see used. It is of course possible to use other types of loaders for these different resources that are part of an OFBiz component. But typically the component is meant to be self contained, so having things relative to the root of the component is a good way to go.

This has various things: an entity model file, an entity group file as we talked about in the Framework Introduction, and we'll talk about those more when we get into the details of the entity engine. We also have some test entities in a grouping file for those test entities that are sitting here. And then we have this test suite that has refers to this entitytests.xmll file which tells the test runner a number of tests suites to run for the entity engine.

Right now they're just unit tests of the entity engine, and we'll get into this in more detail later as well. But basically in this XML file it's just specifying a number of tests to run. There are a number of different types of tests that can be run as well, but it's just specifying a number of tests to run when we are running the unit test.

Each component can have its set of tests, set of entity resources, set of Classpath resources, and other things that are run as OFBiz is taking care of these other parts; taking care of the Classpath resource, taking care of the entity resources, and taking care of the test suites to run.

Let's look at details. We kind of glossed over these Classpath resources. There are two different types; directories that we put on the Classpath, and JAR files that we put on the Classpath. When you're specifying the location of a JAR file you can specify the name of an individual JAR file. These are always relative to the root of the component.

There's a little shortcut for type = jar classpath entries, where you can say, for instance, lib/*, and it will look for all of the JAR files in that directory and put them on the classpath. That's a very handy thing. This one actually has three directories of JAR files; one for libraries, one for JDBC specific libraries, and it's very common that you'll see this build/lib/*.

So we have a build structure that does similar things like this for ANT. It has ANT build files that will do the global build will actually build each component, as well as the base directory and such.

And each component will typically have a build/lib directory where the JAR file that is a result of that build that has all the compile Java class files in it, where that will be located. That's how it knows to put that on the classpath.

00:50:50 screen changes to ofbiz-component.xml

Let's look at another example of the OFBiz component.xlml file. This is for the product component that is part of the applications. Just the same it has a

resource loader, and it has some classpath entries for the configuration directory and the script directory. Those are directories that will be put on the classpath, and all of the JAR files in this build/lib directory will also be put on the classpath. These are basically the compiled Java source files, the Java class combined class files that are in this JAR file we put on the classpath.

This has quite a few more entity resources; entity model files, entity group files, and then entity ECA files. You can see the different type of entity resources with this type attribute. The reader name is something that is specific to the entity engine, and then we have this loader location pattern that we we were talking about before that relates back to the resource loader. We'll talk about what the different readers are and such when we're going over the entity engine configuration and look at that in more detail.

These are all the entity resources and ECA rules, some data files, that are part of this component. But when we are loading all the data files seed data or demo data, or extension data rather, are the three main reader names that we have set up in the entity engine and again when we go over the entity engine configuration we'll see more of that.

We also have all the service resources, service model, service definition files basically, service ECA rules, and service group definitions. Down at the bottom here we have something we didn't see in the other one. Notice this doesn't have any test resource to run, but it does have two web applicactions that are part of it; it has the catalog manager and the facility manager in it.

Basically we're giving the webapp a name, we're telling it the location relative to the root of the component where the webapp directory is located. This is basically just an exploded (war?) file in that directory, and we're also telling it the mount point.

This is one of those two things I mentioned if you're using OFBiz out of the box; when it starts up it will automatically put all of these things on the classpath, and with whatever embedded servlet container (by default it is Tomcat or also know as Catalina), it will load all of the web applications that are defined in various ofbiz-component.xml files in that servlet container. It will use the embedded API specific to that servlet container to mount all of these webapps.

So it uses information in here to determine where to put the mount point, the directory to mount. These are used to create the information that is needed to be passed to the servlet container in order to actually mount these webapps.

We have a couple of other OFBiz specific things here like the base permission that you can assign to each webapp. In this case it's saying you have to have the OFBiz tools permission and the catalog permission in order to, the user must have those permissions in order to see and use this webapp. That's part of the OFBiz security model, and that's very specific to OFBiz; it has nothing to do with the servlet container.

The server here corresponds with the server name so you can actually have multiple servlet containers or multiple servers set up in your OFBiz containers.xml file in case we just have one, so it's pointing to the same one for each of these webapps.

The nice thing about that is you can have different servers set up to run on different ports, or on different virtual hosts, and then have different web applications mounted on one or more of those. When you're configuring OFBiz and installing it, you can actually set up an internal server and an external server.

The internal server can only be opened on the internal network, and the external server on the external network. So you have different applications deployed on different servers.

00:55:28 screen changes to ofbiz-containers.xml

This default server corresponds to the string configuring. Here is the Catalina container or Tomcat configuration, and this is the default server name that that configuration is matching up to. The value is engine here, which is the type of sub-property that this is looking at.

To specify some general things the default host, this can be an IP address or virtual host name, JBM routes or access log things. These are various Tomcat or Catalina specific things.

00:56:15 screen changes to ofbiz-component.xml

Okay so back to the component.xml file, back to the product one. That's basically it; those are the different things that we can specify in an ofbiz-component.xml file.

And again the purpose of this ofbiz-component.xml file is to reconfigure everything on a per component basis that is in that component so that extensions to OFBiz like the financials module I mentioned earlier.

00:56:57 screen changes to ofbiz-component.xml

We can look at that OFBiz-component.xml file real quick, all we have to do is drop this into the hot deploy directory and then when we run the install process all of the seed data from this can be loaded. All of the web applications, entity definitions, service definitions, classpath entries, all of these sorts of things are configured in this file. So all you have to do is drop that directory in the hot deploy directory, and then everything that OFBiz needs to know about related to this component is specified right here in this file. That is the beauty of the component-based architecture in OFBiz.

00:57:18 screen changes to Advanced Framework Training Outline

So now we've looked at the component definition files. Let's talk a little bit about the component structure conventions. As we've looked each component has a root directory, and under that directory we have various other directories. Of course certain components may or may not need these directories.

Let's talk about the purposes of each one. The config directory is a directory that will go on the Classpath, typically. It is how the various configuration files in that are made available to the Java application. Typical files to put in this directory would include configuration XML files, in some cases, and configuration.xml and properties files. We also put all of the localization properties files and all of the message properties files in this directory, and that's how they are put onto the classpath, made available through the classpath as classpath resources.

The data directory has the type data entity resources that we've seen in the ofbiz-component.xml file. These are basically the seed data, demo date, and extension data XML files that are imported as part of the install process.

Okay the DTD directory. These are for XML data type definition files, although we also use these for XML schema files. Most of OFBiz has moved over to XML schema. In the various framework components you'll see the XML schema files, the XSD files, inside the DTD directory. This convention is just a carryover from the days when we were only using DTDs.

The entityDef directory has all the entity configuration files, as well as all of the data model files, entity model files, entity group files, and all of the entity ECA rule files. Those will all be in the entityDef directory.

The lib directory is typically where you'll put JAR files, either right inside this lib directory or you can create various sub directories under it, like the JDBC directory that we saw for example when we were looking at the entity component XML file.

The script directory, is also a directory that is put on the classpath as a directory that's intended to hold classpath resources. They will all typically be script files, like BeanShell files and simple-method.xml files. All of these sorts of things will go in the script directory.

In a service definition for simple-method files, we will actually be specifying the location on the Classpath for the XML file the simple-method is defined in. It will find that by looking at the Classpath because the script directory is on the Classpath.

The source directory is not put directly on the Classpath. This is where all of the Java source files will go, in sub directories under this directory. Same as the script file, they will also have sub directories under the directory.

And typically the sub directories under the script directory and under the source directory will follow the same pattern, which is typically something like org.ofbiz.product of the product component. Underneath the product you'd have things like product.category or product.product for the main product related files, product.facility.product. All of these sorts of different things.

That same directory structure is in both script and source directories. As I mentioned the source directory is not put directly on the classpath because these are Java files that need to be compiled before they are put onto the classpath. So it's the compiled class files that are put on the classpath.

What happens with this in the ANT build.xml file will be building everything under the source directory and putting it into a JAR file. And then that JAR file, which is typically in the build/lib directory, as we saw over here.

01:01:52 screen changes to ofbiz-component.xml

Let's look at this again real quick. So here for example in the product component we have a build/lib directory, we're taking all of the JAR files from there and putting them onto the classpath. That's how all the Java source files make it from the source directory onto the classpath itself, through a JAR file in the build/lib directory.

This build/lib directory is basically the default location where the ANT build files will be put. This is kind of another OFBiz convention by the way. But it's also a convention that's used more widely, that there will be some sort of build directory where built artifacts go. And then under that there will typically be a classes directory and then a lib directory. So the build/lib directory is where the JAR files will go that came from the source directory.

01:02:39 screen changes to Advanced Framework Training Outline

Okay the next directory is the serviceDef directory. This has all of the service definitions, service.xml files, SECAs, Service ECA rules, secas.xml files, and all of the service groups will also typically go in this directory. The typical naming convention for them is groups.xml.

Okay the next directory is the templates directory. Some applications will have various templates that are not part of the web application or anything. These will typically be for email messages or surveys. We also have special templates for surveys in OFBiz, which is part of the content component.

There are various entities you can configure surveys, and then those can be rendered by those survey tem-

plates. If you look in the e-Commerce component there are some examples in there of email and survey templates, so those will go in the templates directory. These are just subdirectories, typically an email subdirectory and survey subdirectory and other subdirectories for other types of templates.

Okay testDef is like the serviceDef and entityDef directories. This is where the test definitions will go. The OFBiz testTool testRunner will look at these test.xml files and run all the tests in them. This is how on a per component basis we can configure the unit and other tests.

Okay the webapp directory is where you'll put all the web applications, so there will typically be a sub directory under this webapp directory for each web application.

01:04:21 screen changes to ofbiz-component.xml

As we saw for the webapps in the product component, they're located in the webapp/catalog and webapp/facility. So that's where that convention comes into play.

01:04:34 screen changes to Advanced Framework Training Outline

The widget directory will contain all of the widgets that are associated with OFBiz, or with an OFBiz component. These could be Screen Widget definitions, Form Widget definitions, Menu Widget or Tree Widget definitions, and in some cases their related resources such as BeanShell scripts that is used for data preparation.

One thing to note is that the widget directory is kept separate from the webapp directory, for the reason that these widgets are meant to be more generic than a specific type of application. So they are reusable in theory. In design they are reusable in other types of applications other than webapps, although we don't have any widget interpreters for other types of applications. But the design is intended for that purpose, so these are kept separate from the webapp directory.

Okay so that's the basic stuff for the conventions for the typical directories that you will find in an OFBiz component.

One last thing we'll review real quick; the OFBiz Component URI. This is another thing that's important to know about the components. In various places in OFBiz you can use this component://. It's kind of like a URI or URL that you see with other things, such as http or ftp or file or various others of these.

Component:// only works in certain parts of OFBiz where they have been extended to recognize this type of URI. After the component:// the first thing you'll run into is the name of the component, then another / and the relative-path within that component. No matter how the component is loaded, OFBiz will be able to find a resource within that component using the component:// URI.

01:06:54 screen changes to controller.xml

Let's look at a couple of examples of this real quick. We'll just open up the first controller.xml file which is in the accounting application.

Here's a place where we are referring to the location of a Screen Widget screen definition file, so we have our component://, the component name, and then the location of the XML file relative to the root of the component. In many cases these will be referring to, in the web applications, screen definitions in the same webapp, but they can certainly refer to screen definitions in other components as well.

One place that does that quite a bit is the e-Commerce application, which shares a lot of stuff with the order component. The order manager is kind of a higher level. Actually all of these screen definitions are for the most part specific to the e-Commerce application, so they're all referring to the component://ecommerce for the different screen definitions. If we look at these screen definition files, catalogscreens.xml for example, in e-Commerce.

01:03:13 screen changes to CatalogScreens.xml

Okay we'll see certain parts here like this where it is reusing a resource that is in another component, in this case the order component. The productSummary.ftl file is being used, shared by the order manager and e-Commerce application.

This is a pattern that you'll see for reuse; it's typically the higher level component that reuses things in the lower level component. So between e-Commerce and the order component, the order component is a lower level one, so it's okay for the e-Commerce to depend on the order component, but it's not okay for the order component to depend on the e-Commerce component. We want the order component to be able to run on its own without the e-Commerce component in place, but the e-Commerce component can share things with the order component.

Now what if you wanted to customize this file? What you'd want to do is copy it over into the e-Commerce component, or into whatever component that you're creating based on these other components, and then modify that file wherever you've copied it to, and then change the location reference right here to point to the location of the new file.

So here you see we've got FTL files, script files, BeanShell script files, and various other things that are shared between these applications. This is one of the nice things about this component://, you can refer to

any resource in any component from any other component.

Just be careful, as I mentioned before, to watch for dependencies between components. Just to reiterate in this case, we do not want the order component to be dependent on the e-Commerce component. So we only have references going in the other direction. So any resources that are shared between the two are in the order component.

01:10:17 screen changes to Advanced Framework Training Outline

Okay so that's basically it for the ideas behind the base of OFBiz, the utilities and the loader that are there, the containers, the container configurations, components, how to define components and the various parts of components and also the directory structure conventions for components. And this new type of URI that is OFBiz specific to refer to artifacts in components.

Okay one more thing I'd like to mention, and I've left this to the end because it's a little bit more of an advanced thing, is how to implement a container.

01:11:19 screen changes to Container.java

If you are not involved in low level implementation stuff, then don't worry too much about this part. This is the container interface. There are basically three methods: an init method, a start method, and a stop method, that you need to implement in order to implement a container that can then be loaded when OFBiz starts up using the containers.xml file.

01:11:40 screen changes to ofbiz-containers.xml

So for example here the component container. Let's look at that class real quick.

01:11:56 screen changes to ComponentContainer.java

ComponentContainer.java. The component container implements the container interface. Because it implements this container interface it implements these methods: the inti method, the start method, and the stop method.

Notice it has some other methods that it uses for the work it does. These are basically various component loading things. So when this initializes, if necessary it can pull arguments from the ARGs array. These arguments are the startup arguments, so what that means is the command line arguments for the java/jarofbiz.jar. Those will basically be available to the container when it initializes here through the ARGs array, just like in any sort of a mini method that you have in a Java startup.

Then we have another name of the config file here, the config file location. With that config file location we can use the containerConfig to get the container class that represents that. This will be the container file, basically like the ofbiz-containers.xmkl file that was used for starting up OFBiz in this case.

01:13:28 screen changes to ofbiz-containers.xml

So when it gets this it will actually get a Java object that represents this ofbiz-containers.xml file.

01:13:32 screen changes to ComponentContainer.java

Then it can get properties related. Oh I should mention by the way it's not getting a class that represents the entire file.

01:13:53 screen changes to ofbiz-containers.xml

It's specifying the name of the container right here that we want the configuration for, so the component container.

01:14:01 screen changes to ComponentContainer.java

Notice that it's not passing it any properties in this one, but it does support two properties, the loader config and the update-classpath property, which by default is true.

01:14:20 screen changes to limited-containers.xml

We saw the use of this update-classpath property in the limited-containers.xml file, because when it loads the component definitions this will normally set up the classpath, based on the classpath entries on these component containers. But here we're telling it not to with this property. So other container implementations will have different properties, just name value pairs, the different ones that are needed or available will be dependent on what that container is doing.

01:14:48 screen changes to ComponentContainer.java

So basically this does the initialization then we do a startup. This one doesn't really do anything, this component container. Basically when it's initialized it loads all of the components then leaves it at that. For stop I imagine it doesn't do anything either.

01:15:11 screen changes to RmiServiceContainer.java

If we look at some of these other ones, like the RmiServiceContainer, it does a little bit on initialization, just keeps track of the config file, the start is where it really does all of its work.

To start up the RMI, let's see where does it actually do that. So here you can see it initializing the delegator, initializing the dispatcher, and then using that delegator for the dispatcher. Then it creates a remote dispatcher implementation, and it puts that into JNDI. So it uses an InitialContext to bind that into JNDI, and then it checks that, looks it up to make sure it got in successfully.

01:16:03 screen changes to ofbiz-containers.xml

So basically that's what it's doing with the RMI Dispatcher. It's setting it up and then dropping it into JNDI, so that the service engine or any remote application can get this remote dispatcher and then call methods on that remote dispatcher object. Basically be communicating with that server, executing services as if they were on the remote server. When we go over the service engine we'll talk about in more detail what this does.

01:16:34 screen changes to RmiServiceContainer.java

So the main things to remember right now are that this has, when implementing a container, or implementing this container interface, we're going to have an init method, a start method, and a stop method. Basically when it stops it de-registers it or shuts off the remote dispatcher, telling it to de-register itself.

01:17:12 screen changes to Advanced Framework Training Outline

Okay so that's the basics of implementing a container. You can implement a container, basically just implement that interface to do anything if you are using spring, or want to do things based on spring for example in your applications. You can implement a container that gets spring started. You can have a property in your ofbizcontainers.xml file to tell it where the spring XML file or possibly multiple of them are located to set up spring.

And for any other tools, any other things that you might want to run along with the rest of OFBiz, you can write a container class that implements the container interface and then in your ofbiz-containers.xml files just mount it right in there.

Okay that's it for part two, the OFBiz base. We've covered various things about containers and components, as well as some information about the utilities and OFBiz loader. In the next section we'll be talking about the webapp framework, widgets, and MiniLang, the simple-methods and the simple-map-processors.

Again you've already had an introduction to these in the OFBiz framework introduction videos. But we'll be going into these in significantly more detail as part of this course. It is of course important to have watched those other videos in order to have an important understanding of how these various things fit together between themselves along with the entity engine and service engine and other things. And in these next videos we will be looking at all of these topics in much more detail.

See you then.

Part 3: The Webapp Framework, Widgets, and MiniLang

Introduction

Starting Screen: Advanced Framework Training Outline

Welcome to part three of the Advanced Framework training course. In this part we'll be covering the webapp framework, the widget library, and the MiniLang component which has two miniature languages in it, the simple-method and the simple-map-processor. Let's start by looking at a high level view of the framework. We'll be using the same diagram that you saw in the framework introduction videos.

00:00:32 screen changes to ofbiz-reference-book.161copy.pdf

So here we are, you should recognize this. In this part three we'll be covering the Control Servlet, which basically handles the incoming browser requests. The Control Servlet is mounted in the Servlet Container in the web.xml fiel, the Control Servlet configuration file. The controller.xml file has mappings for incoming requests, optional events associated with those, and then how to respond to those requests, either by chaining back to another request or moving on to a view.

Also in this part three we'll talk about the Screen Widget, various parts of the Screen Widget, other widgets such as the Form Widget, the Menu and Tree Widgets, and we'll talk about using BeanShell scripts for data preparation and FreeMarker templates for lower level templating than what you would typically do with the Form Widget or other widgets.

This part three is mainly focusing on the web based user interface, so these two boxes on the right hand side, the blue bordered ones. But we'll also be talking about the simple-methods in here, the MiniLang, because even though the simple-methods are primarily used to implement services, they can also be used, as this somewhat difficult to follow arrow shows down here, to implement events.

So in other words a simple-method can be called as an event, or as a service, and there are specifically operations in each, depending on how it's called, so when you write a simple-method you can write it meaning to be called as an event or as a service or both.

00:02:36 screen changes to site/training/AdvancedFramework

Okay let's step back to your outline for a minute. We'll be talking about the control servlet, and about all the Screen Widgets. These are basically the primary best practices for view generation, some secondary best

practices are BeanShell. In cases where you have more complex data preparation logic and such, you can use the BeanShell scripting to prepare that information, and from a BeanShell script you can use the Form or other widgets for actually displaying the forms or tables or whatever, or you can do whatever you want in a FreeMarker template, where you'd actually be laying out the full html.

We'll also talk briefly about generating PDFs using XSL-FO. In other words we will create a screen that, instead of producing HTML, will produce XSL-FO, and then we'll use FOP to transform that XSL-FO into a PDF. You can also transform it into other outputs but this is the most typical scenario, the one we will be discussing here.

Then we'll also be talking about some other view generation things, such as JPublish based views and Region based views, the Region framework that's part of OFBiz. These are older things that have been used in the past. JPublish is what we used to separate data preparation from the templates and layout before the Screen Widget was implemented.

And now that we have the Screen Widget that is much more flexible and much easier to use with other parts of the OFBiz framework, we've pretty much deprecated the use of JPublish, although it is still supported in there. You may occasionally run into some older artifacts that still make use of JPublish, but they're not very common anymore.

The Region based view are mainly for JSPs. If you want to use JSPs for templating you'll have to use the Regions, and the Regions framework if you want to use something for composite views, because the JPublish and the Screen Widget do not support JSPs as templates. Because JPublish and the Screen Widget are based on the writer or text screen, which is much more flexible and easier to create tools for, and much more powerful tools as well.

But JSPs cannot be used in this way, they can only be used in a servlet context, so they have to have the HTTP servlet requests and HTTP servlet response objects in order to do what they do.

Okay so we'll talk a bit about the Region framework and the OFBiz tagLib that is, like the Regions, also used with JSPs. We'll also talk about the MiniLangs as I mentioned.

We will also talk about the concept of the simple-map-processor, which is basically for data validation and mapping from an input map to an output map. That's either used for an entity operation or preferably calling a service or that kind of thing.

We'll talk about the simple-method, the general concept with a context in the simple-method, how variable handling is done and so on. And we'll look at some of the operations that are available in the simple-method, and check out some examples.

So that's what will make up part three.

Part 3, Section 2.1

The Control Servlet

Let's start with the first part of the Webapp Framework here, namely the Control Servlet. A couple things to note about the Control Servlet, the configuration file for the Control Servlet is the controller.xml file. Each webapp will have one of these at least. The location of the controller.xml file, this is a component URL, so here's the name of the component and here's the path relative to the root of the component.

Typically the component will have a webapp directory that has all the webapps inside of it, so for the given webapp name, many components just have one webapp. But some components do have more than one.

For example we'll be looking at some things in the e-Commerce component here, so the component name would be e-Commerce. Then we would have /webapp, e-Commerce which is the name of the webapp, then inside the webapp directory is typically a web-inf directory like this.

This is a standard servlet convention, for servlet specification based webapps, and is used for internal files that are protected from outside access, so these are for use by the applications.

Inside the web-inf directory we have the controller.xml file. The schema for this XML file is the site-conf/xsd file which is located here. From a high level control servlet the controller.xml file has requests, optionally each request can have an event associated with it. We'll talk about what an event looks like. Then the parameters going in, the return values, and different event handlers and types.

We'll talk specifically about service events, simple-method events, and Java events, and designing a new event handler if you want to do that. Special case events such as first visit, the first hit in the visit, the pre-processor before each request, the post-processor after each request and after login, those events will run after a successful login.

We'll talk about request responses. We'll talk about views, different view handlers and types, and of course the main one we'll be talking about is the Screen Widget based view.

Going on in this next section we'll be talking about in detail the different views: Screen Widget, JPublish, and Region based views. We'll also talking about designing a new view handler, and what that looks like. So you

can plug in whatever type of view you want, just as with the event handler you can plug in whatever type of logic you want.

00:10:19 screen changes to Framework Quick Reference Book Page 8

Okay let's get started. This is page 8 of your framework quick reference book. This is the reference for the controller.xml file, and it has information about the request and view definitions, and other things in the controller.xml file. The controller.xml file will have a root element of site-comf like this. There's the handler element where you can specify two types of handlers; the request event handlers and view handlers.

Let's look at some of these briefly. The example here, it actually has a pretty complete list of the different event handlers that come out of the box in OFBiz. All of these type=request are the request event handlers, the different logic for processing input from request or Form submissions.

For each one we give a name that can be referred to in the event type attribute, like this. So type=java corresponds with the name=java up here. And we have a fully qualified class name for each one. So if you want to look at the source of those you can, to see exactly what they're doing.

Okay for view the same thing happens. Basically each one has a name. In the view map there's a type attribute that specifies one of these, that will refer back to one of these. Again the most common ones here are the screen, screen FOP for XSL-FO that's converted with FOP to a PDF.

And we'll also be talking about some of the older ones such as JPublish and Region, which you might even use if you want to do things with JSPs. There are also some types in here that can be of use, on occasion.

This will call in an FTL file directly, so there will be no data preparation before it, so it's not used very much. This will call a JSP or actually any servlet resource. So it can be a servlet or other things that are located. Basically it just does a call through the request dispatcher on the servlet API.

The FOP PDF view handler works along with JPublish so it's the older one that's been deprecated by the screen FOP view handler. The http view handler allows you to specify a URL, and what it will do is get the text from that http URL and return it as the view for the current page. It's kind of a funny thing but an interesting way to act as a kind of structured proxy.

DataVision is for the dataVision reporting tool JasperReportsPdf, for Jasper reports porting tool. For generating a PDF, and jasperreports.xml, like the PDF one except it returns XML, which is another way of rendering a JasperReports report.

Among the different request event types, the most common you'll typically see are Java Service, for calling a service as an event. When that is done, as we looked at in the framework introduction videos, it basically automatically maps the parameters, the request parameters, to the service input attributes.

ServiceMulti is a variation on that where the service is called multiple times. This is usually done if you have a table with a large form, and it's treated as if it was a form for each row of the table, as an example, and calls the service multiple times. This makes more flexible user interfaces and such.

Simple-methods can be used as I mentioned too; they can be called as an event. That's how this would be done, with the simple-event-handler. BSF would be for calling scripts through the BSF, the BeanScripting Framework. Originally from IBM, it's now An apache project. And this could be used to call BeanShell or any other type of script.

So those are the main request handlers you'll run into in OFBiz.

Okay let's move on through the file. Some other tags. The first visit preprocessor, postprocessor, and after-login tags are special tags that allow you to specify events to run at different life cycles of a request end session.

So first visit, basically all of the events under it will be run on the first request or first hit in a visit. Visit is synonymous with session in the servlet specification vernacular. If a first hit in the visit is the same as a first request in the session, preprocessor these events would run before each request. Postprocessor, these would run after each request. And after-login these would run after a successful login.

00:16:47 screen changes to controller.xml

Before we go on let's look at some examples of these. A good place to look at examples for these things is the e-Commerce application. There are a lot of things that we do automatically that we want to attach to these different parts of the life cycle.

So on the first visit we're running these different events, things like checking tracking code cookies, setting default store settings and so on. Before each request we're doing a number of things, checking an external-login-key, setting the association-id, these would be for affiliates and so on, checking the tracking code URL parameter and the partner tracking code URL parameter.

This is part of the tracking code stuff and the marketing component. And finally some order related things: shopping cart, keeping the cart updated, this is for the case where we have an auto-saved cart, and restore

auto-save list, which is a shopping list related thing if you have an auto-save list that is set up on the store.

Okay then these ones. We do have a common-to-doubt section here for post-processor. These aren't used very commonly, but if you need something to run after each request then you can do that here.

The after-long, these are the events that will run in after a successful login.

Update-associated-distributor, or affiliates and distributors. This is related to the setAssociationId up here. And basically just associates those with the user that's logged in, as soon as we know how it is. Also when they log in we do this keyCard update and restoreAutosave list, so that immediately after login once we know who it is we can restore an autosaved cart and keep it up to date. We also have this saveCartAutosave list, so if the autosave list is turned on for a store, this is configured on a per store basis.

Once we know who the user is we save the list and associate it with that user. So those are different things that happen after a successful login. These events are implemented the same as other events.

The thing to note about them as they run is that any error or other return codes that return from them will be ignored. If there is an error it will be logged in the console log, but it will not affect the flow of the program, or rather the flow of the request processing.

00:19:59 screen changes to ofbiz-reference-book.161.pdf

Okay let's go back to our reference page. The next tags we'll talk about are the request map, which as a few sub elements, security, event, and it will have zero to one (0..1) request events associated with it.

Response, you can have one response for each possible return code from the event typically, and that's the name for the response. Typically that will be success or error, as you can see over here. Those are the most common responses that you'll have, but an event can return any string, and then that string can be used to route to different responses.

Let's talk about the different response types for a minute. Actually let's wait for a second; we'll come back to that and go over those in detail.

The view-map tag is the last one, the view-map element, this is like the request-map. Once we get to the point of rendering a view, each view will have a name. A type which just as for the request corresponds to the name of a view handler up here. Screen-one is probably the most common one you'll run into, now.

Let's go over some of these in a little bit more detail. The security tag, under the request tag, basically has security related things for each request, and it will have one or zero security tags, so it will be under the request map. We have the HTTPS and off tags, these are the two most common you'll run into.

HTTPS is used to specify whether this request should be accessed through HTTPS or not, when a request is received for a certain request map, and say HTTPS is true. But the request is received using a non HTTPS, basically using an HTTP prefixed URL, then what it will do is send a redirect back down to the browser, so it will access it through the HTTPS location.

The one exception to that is if it is a form submit and you've already sent data in, then it will process that form input because it's already too late to protect it. So protecting these is typically done preemptively. The main point actually of having the HTTP set to true or false, here for the security tag for a request map, is that the OFBiz URL generation which is done in different ways in different places, there's an OFBiz URL tag in FreeMarker, for FTL, there is also one for JSPs.

In the Screen Widget, Form Widget, and other widgets, they have built in the functionality to generate an OFBiz URL. And basically an OFBiz URL is just a URL, either relative or full, that refers to a specific request map as defined in a controller.xml file.

When it's creating that URL it will look up that request to see if HTTPS is true or not. If HTTPS is true, and the current page was accessed through HTTP, it will create a full URL to point the browser to the HTTPS site. If the current page was retrieved through HTTPS, but HTTPS is false, which will happen down here on certain things, I guess this comes from one of the back end applications, so none of these will be false.

But if HTTPS is false, in other words the secure socket is not required, and the current page was retrieved through HTTPS, so it was secure, then it will create a URL that will push the browser over to the non secure site. Based on this HTTPS equal true or false it will do that automatically.

The auth tag here is also either true or false; it's a boolean value. If auth=false when you hit the request, even if you're not logged in, it will allow you to go through. If auth=true, when you hit the request if you're not logged in it will forward you over to the login page.

There are certain requests that are conventions for the control servlet, that should always be present: the check-login request, the login request, and the logout request. And there should also always be a login view. So this is how these are used. When we run into a request where auth=true but there is no user logged in, then it will forward the request processing to the check-login request. That will run whatever event is configured there.

On success it means the user is logged in, and so it will go back either to the main page or to the original page that was requested, such as the edit product page here. If there is an error it means that the user is not logged in, so it will send the user to the login view. So you can actually have other views to send it to. And that would be configured here.

Okay the login request is the request that is hit when the login for when the login view is submitted. So this actually processes the login of the user. On success it will go to main, or actually the control servlet will override that, it treats the login as a special case.

On a successful login it will recognize the original page that you came from, and the parameters are passed through as well, so you can get back to whatever you were originally going to before the login process tarted. On error it will take you back to the login view.

The logout request is similar to the login request except it does a logout. And when it is complete by default you can change this by changing this response definition right here. But by default it changed to another request, the check-login request which is up here, which will check to see if the user is logged in, and we just logged them out so obviously they will not be.

So it will take them back to the login view. You can change that flow to take them wherever, but that's the default flow you'll typically run into. On error from logging out it will take us back to the main page by default. And that will typically show a message about what happened, why the logout failed.

Okay so that's basically the flow related to the auth tag. Let's talk about some of the other tags that you don't run into very often.

This direct request tag, true or false. By default it is set to true, as you can see there. If you set it to false it means that the request cannot be accessed from a browser client. If it's set to false in other words it can only be used in a request chain, so it can only be used where a response has a type=request and goes to that request.

Okay external view by default is set to true. If you set this to false what it means is that the external view override pattern will be ignored. So let me explain what that pattern is real quick. There are some cases where we want to reuse a request, such as create-product for example. Let's use that as an example. Usually this will be done for a request that has an event associated with it.

00:29:57 screen changes to
ofbiz-reference-book.161.pdf

And when that request is complete it will eventually chain to a view, if it always goes through success. So this is for basically no error cases, or non variation cases. This for the view override or external view override is only for the primary success scenario, that's what it's designed for. So this is used quite a bit in e-Commerce for example, where we're doing an add to cart and we want to come back to the same page, like the product detail page or a category listing page.

00:30:45 screen changes to
ofbiz-referene-book.161copy.pdf

In the URL where normally we'd have /control/request, we'll have /control/request/externalviewoverride. So in the success scenario when it gets to the success view, instead of going to this default success view that's configured in the controller.xml file it will go to the view that's after the / after the request name, after the request URI.

00:31:16 screen changes to
ofbiz-reference-book.161.pdf

So by default that functionality is enabled, or is true, but if you set it to false then it will ignore that external view, or the view override.

00:31:48 screen changes to
http://localhost:8080/ecommerce/control/main

Let's look at an example real quick of how this looks. Let's just open up the e-Commerce application. Looking at the main page of the e-Commerce app, we'll just do an addToCart. An addToCart is designed to come back to the same page. That's why here it has addItem, which is the name of the request, /main, which is the name of the view it overrides to.

If I go to one of the product detail pages and do an add to cart, you'll see it has additem/product so it can come back to the same page.

00:32:30 screen changes to
ofbiz-reference-book.161.pdf

And that is the external view or the external view override. Okay let's look at the event tag in more detail. This is a fairly simple one so I guess we've pretty much already gone over everything already. Each event will have a type, that refers to one of these handler names, for any of the request event type handlers.

Path is the location of the event, and invoke is the name of the method to invoke at that location. So that path for a Java event will be the fully qualified class name, invoke will be the method name. For a service there is no path, so you can just leave it out, leave it blank like that, and invoke is the name of the service.

For a simple-method it will be the fully qualified classpath resource location, or other OFBiz specific location, and the invoke will be the name of the simple-method within the simple-methods XML file that is referred to in the path.

Okay the next tag, response. Each response will have a name. There are different types of responses so the name is the return string from the event, or success if there is no event. Like here, there just is a success response. The type of the response, the main types that you'll see and that we talked about are view and request, but there are other types as well that can be useful in certain circumstances. The none response basically means don't do anything else.

This is used in situations where the event will actually be generating the response. If you have an event that returns a binary download for example, you would use a response of type=none, so that the control servlet doesn't try to render anything and send it down to the client. The event will just take care of that.

So if you write an event in Java you have access to the HTTP servlet response object, and you can stream binary information down to the client. You can also set the content type or mime type, which would normally be specified in the view-map. You can also set the text encoding, which would normally be specified in the view-map. Typically default to encoding will default to UTF8 and content-type will typically default to text/html.

Okay some of these other ones. The request redirect is just like the request, except instead of internally chaining it will send a redirect down to the browser telling it to go to the new request. So if you have a bunch of request redirects then what the user will see is the browser basically flashing different URLs until it gets to one that generates a view. That can be nice in some applications, especially to help with back buttons and things.

The URL response type is kind of like the HTTP view type, it will retrieve the information at the specified URL, and return it as the response for this. The value of a response, depending on the type of response, will either be the view-name, or the view-map-name request map URI. Request redirect it will be a request map URI. URL will be either the full URL.

Okay we talked a little bit about the view map and about what some of these extra parameters mean that you'll run into on occasion. Especially the encoding and content type attributes. Let's look at one example real quick of where the encoding and content type are specified. When we use the screen FOP view handler, we'll be returning a PDF instead of an HTML file.

00:37:25 screen changes to controller.xml

This is done in the order manager, so let's look at the controller.xml there. Order PDF and return PDF are generated through the screen FOP view handler. So the page is a location, a full location of a screen definition. Content type we're specifying as application/pdf, since it's a PDF we'll be returning. And encoding is none. That's how you can tell the browser what you're returning back to it.

00:38:03 screen changes to ofbiz-reference-book.161.pdf

Okay that's basically it for the controller.xml file. Let's look now at some more specific things like how to implement an event in Java.

00:38:26 screen changes to site/training/AdvancedFramework

Actually stepping back a little bit with the outline, with service events you'll basically implement the service just as you would any service. The point of doing the automated mapping is that you would do nothing special in your service definition or implementation to make it callable from the control servlet. Simple-method events are slightly different.

A service won't know it's being called as an event because the mapping is done automatically for it. So it doesn't know anything about the webapp context, the request and response objects, and such.

The simple-method is a little bit different. When it is called as an event it does know about the session, the request, the response, and so on, and can do certain things based on that. We'll talk about that more when we cover the MiniLang and simple-methods in particular. Java events are very specialized for being called as events. Let's talk about how those are written.

00:39:33 screen changes to loginevents.java

We're looking at the login events file here. Let's look at an example of how these events would be specified.

00:39:39 screen changes to controller.xml

We basically have the fully qualified class name, type=Java, and the name of the method within that class to call.

00:39:52 screen changes to LoginEvents.java

So storeCheckLogin is going to be one of the methods in here somewhere. Here's storeCheckLogin. All events written in Java will take two parameters in, the HttpServletRequest and the HttpServletResponse, and they will all return a string.

So they can get information about the request from the request object, including OFBiz specific things that the control servlet puts there such as a delegator for the entity engine, a dispatcher for the service engine, and so one. These methods will always be public and static, so that the control servlet can call them without enstantiating the class that they are in.

Again they'll always return as string, they can have whatever name you want that's specified within the controller.xml file, and they'll always take the two pa-

rameters, the request and response. That's basically how you implement those. What you do inside these does therefore require some knowledge of the servlet API. There are lots of examples if you want to see common things done with the request and response. Usually the response is not used except in cases where you are doing something special with the response.

00:41:28 screen changes to ofbiz-reference-book.161.pdf

Like I mentioned in the scenario of what would go along with the response type=none, for example streaming binary content down tho the browser.

00:41:43 screen changes to site/training/AdvancedFramework

Okay the last bit we'll look at in here is a little bit more advanced. Designing a new event handler. We talked about the special event types already, so let's talk about designing a new event handler, and designing a new view handler. We'll be talking about some of these different view handler types in more detail in upcoming sections, as soon as we're done with the control servlet.

00:42:13 screen changes to ServiceEventHnadler.java

Okay so implementing a new event handler. Basically all you have to do is create a class that implements the event handler interface, which has two methods, and init method and an invoke method. When a web application is initialized it will initialize all of the servlets that are associated with it, including the control servlet. When a control servlet initializes it will initialize all of the event handler and view handlers.

00:42:47 screen changes to ScreenWidgetViewHnadler.java

So just as you implement an interface with the event handler, you do the same thing with the view handler. And the view handler also has an init method; it's basically the same, it just passes in the servlet context. So you can see with these events all they do in here is really just save off the servlet context to a local variable so that it can be used later on.

00:43:08 screen changes to ServiceEventHandler.java

Okay then for an event handler you have an invoke method. To the invoke method is passed information from the event tag, like the path and the method. The path is from the path attribute, method is from the invoke attribute, and of course you can call these parameter names coming into your method whatever you want.

It's just the first string is always the path, second string is always the invoke, and then it passes in the request and response objects for convenience, so you can do something with the event. Of course those have to be passed into the event when you're calling it, so you need those for an event handler.

So the Java event handler does very simple things. This is the service event handler which is a little more complex. If you want something to read that's entertaining you can go through this and see how the service call is done, doing all the parameter mapping, the return putting that into the attributes, and so on.

00:44:19 screen changes to site/training/AdvancedFramework

One thing I want to mention about this before we move on is that I will be talking about this more in the Screen Widget. But when a service is called as a event the return attributes, the output attributes from a service, will be put into the request attributes. So the request.set attribute method basically will be used for these.

In the general order of things the reason for that is so that other requests that are chained after this request or view that's eventually generated can have access to that information. For example if you have service that creates a new record in the database but automatically sequences the Id, that service will typically return the Id in an output attribute.

When called as a service that output attribute will be put in the request, and then down in the view you can use that request attribute to override the request parameter to display the proper information, such as showing and edit form instead of a create form. In the parameters map in the Screen Widget this pattern is automated, but this is typically what you'll see as the override pattern when you're looking for a variable.

You'll first look in the request attributes, then request parameters, that allows the attributes to override the parameters. Then you'll look in session attributes and application attributes. So it's from the most local to the most global, with the exception here the request attributes and parameters attributes being first so they can override the parameters.

Anyway we'll talk about that more a little more because that's the pattern used to populate the parameters map in the Screen Widget, for convenience and because that's the most commonly used pattern.

00:46:12 screen changes to ServiceEventHandler.java

Okay let's finish up real quick by looking at the view handler implementation. Just like in the event handler there's an init method that we talked about, and the render method. This render method has all the information passed in from the view-map tag: the name of the view map, the page to go to, the info is just an extra info string that can be used for extra purpose view handlers.

The content type or mime type that character encoding, and of course it's passed in the request and response objects so we can actually render the view. So if you want to use whatever type of view you can do so by implementing a new view handler. For whatever type of logic, for processing events, processing input, you can implement your own event handlers.

00:47:04 screen changes to site/training/AdvancedFramework.

Okay so that's basically it for the control servlet part of the webapp framework. In the next section we'll be talking about Screen Widget based views, and we'll be going into quite a bit of detail about the various widgets, and also about using BeanShell FreeMarker with them, and even generating PDFs.

We'll see you then.

Part 3 Section 2.2.1

The Screen Widget: Overview and Widgets

Starting screen: Advanced Framework Training Outline

Okay now we'll move on to the next section about Screen Widget based views. We'll talk about templates widgets and so on.

00:00:22 screen changes to Framework Quick Reference Book Page 18

Let's start this by looking at our artifact reference diagram again.

Once we get through the control servlet we talked about in the last video and move on to the point where we actually decided on a view to render, if that view is a type of screen or has the type screen, then it is rendered through the Screen Widget.

So we'll be looking at some of the particulars of the Screen Widget, sections in the Screen Widget for each section; the conditions, Actions Widgets, and in the case the condition fails the Fail Widgets, which are basically just like the widgets.

By the way widgets, we're taking about these in this context to mean a visual element. There is actually just one condition so they should all be combined using boolean logic. The actions will be run in the order they are underneath the actions tag, and the widgets are visual elements that will be displayed in the order they are in under the widgets element.

So here we're showing there are various other types of actions. Here in this diagram we show service action for calling a service, entity-one for finding an entity by primary key in the database, and Script for calling a script, typically a BeanShell script. There are also various different types of actions and we'll talk about the others as well.

This diagram also just shows a few of the different types of widgets including a screen that goes back to render another screen. There are various platform specific things such as the HTML ones; HTML include for including a template, which can be used for FTL template.

And the include form would include a form definition rendered by the Form Widget. There are similar tags for the Menu Widget and the Tree Widget. We'll be looking at all of the actions, all of the widgets, and the conditions. A lot of the elements by the way that are in the Screen Widget actions and conditions are shared with the simple-method and with the Menu Widget and Tree Widget, and even the Form Widget as well. So many of these actions you'll see in various places.

00:03:20 screen changes to Framework Quick Reference Book Page 13

In fact when we look at the framework quick reference book down here on page 13 are all of the shared elements. We have the conditions over here in the left column, general actions, and entity specific actions. We'll be looking at all of these in detail.

We'll also be talking about the flexible syntax, which can be used in any of the widget files as well as in the simple-methods. This is basically a string expansion syntax. It uses the ${ } delimiters just like many other templating languages and such including FreeMarker in fact.

00:04:16 screen changes to Framework Quick Reference Book Page 9

So let's go back over here on page 9 is the reference for the Screen Widget. The root element for a screen definition file is the screens element and contain one to many screen elements in it. Each screen element is fairly simple, it has a name and then it has a single section under it.

The reason we put a section here is because a section, not only is it included here, but a section can be included under the widgets or Fail Widgets, so as is denoted here it's used in all widgets, which is included here under the widgets and fail widgets. So any of these which say used in all widgets can be included, zero to many of them, underneath the widgets or Fail Widgets tags.

You'll see various of these mention that, used in all widgets, widgets, and so on. So these are all the visual elements that have a direct effect on what the user will see, and these are things that include other artifacts.

Okay so a section, in addition to being includable in the widgets, is the single tag that's under a screen. Each

section can have a zero or one condition element, zero or one action element, must always have a widgets element, and can optionally have a Fail Widgets element. The Fail Widgets element is only run if there is a condition and the condition evaluates to false.

If there is a condition, and it evaluates to true, then the actions and the widgets will be run. If there is one condition then by default it's true and so the actions and widgets are run.

So it's just in the case where there is a condition and it evaluates to false that the actions and widgets are not run, and instead the Fail Widgets is run. If you want to include actions or other follow-on conditions then you would simply put a section element underneath this fail-widgets element.

Okay here we have the definition of all widgets as well as lists all of the different widgets, section container image label link and so on. So you can see a list of those here. What we'll do is walk through each of these and talk about them in detail, and then we'll look at some screen widget examples. Such as this one which is included in your quick reference book that comes from the product screens.

We'll actually look at the product screens themselves so that we can look at the declaration pattern in more detail. And see other things about the edit-products screen effect. Since the main point of the Screen Widget is to present visual elements we'll talk about the various widgets first, or everything that can go under the widgets and Fail Widgets tags.

Then we'll talk about the other side of the Screen Widget, which is to help separate the data preparation, the actions from the visual elements, or the widgets, and the conditions that can be used to optionally include different parts or different sections of a screen.

The container element is the first one. So the section by the way is the one thing that can go in the widgets that's defined over here. Each section can optionally have a name, and then it will have the tags of the elements that we talked about before underneath it.

Okay inside of widgets or Fail Widgets the other ones that we have are container. Container is simply a wrapper around a number of other visual elements, so any of the widgets can be included inside of a container, just as any of them can be included under the widgets or Fail Widgets tag. Which is why they have this all-widgets reference here, which refers to this placeholder and all of the widgets that are available in the Screen Widgets.

A container basically wraps around a section of the screen. It can have an Id or a style associated with it. When a screen is rendered as html, the container will translate to a div. A div tag, the Id attribute here will translate into an Id attribute on the div tag, and the style attribute here will translate onto a class attribute on the div tag.

As I mentioned before, then intention of the Screen Widget is to be platform generic so that the same screen definitions can be used for different types of user interfaces. The intention of having a style here is that it can be used very easily with CSS and HTML, but also so that in a desktop application you could declare styles as part of a Screen Widget rendering framework that could change things like background colors, fonts, and so on.

Okay so the container is a very simple element. Image basically includes the display of an image. The source is the location of the image. It will typically be a URL, but it can be other things.

By default the source is an OFBiz content URL. You'll see the content underlined here for the URL mode. We can also use an OFBiz URL that corresponds with the OFBiz URL tag in JSP and FreeMarker and such. this would basically refer to to a request in a controller.xml file. The content URL in OFBiz is one that has a prefix, which is always pre-pended to it, and that's configured either in the url.properties file. If there is a content prefix there would actually be one for secure or non secure, or HTTPS and HTTP content, so typically have two locations or two prefixes, so depending on whether the screen is coming through a secure request or a normal non secure request one of those prefixes would be chosen.

00:12:38 screen changes to url.properties

And the prefix, we'll be looking at this later as well, because it's used in various places. But in the url.properties file we have a content.url.prefix.secure, and a standard, so secure for HTTPS, and a standard for HTTP. This is for static content and could be images, javascript, CSS files, even non-dynamic HTML files. There are corresponding fields to these. In the website entity in OFBiz a web application in its web.xml file can specify a website Id that will refer to a website record.

In a website entity record there are fields that correspond to these, so that these prefixes can be specified for each webapp if desired. There are also similar fields in that object for these things which control the OFBiz URL rendering. So it controls whether HTTPS is enabled or not, the HTTPS port to use, if you want to force a different host, the HTTPS host to use, the HTTP port to use, and the HTTP host to use.

00:14:15 screen changes to Framework Quick Reference Book Page 9

Okay stepping back to the Screen Widget. So we have the OFBiz URL that uses those HTTP host and port

settings to render a URL that will go to a specific request in the current webapp.

The content is basically considered to be part of the static content for OFBiz, and it will have pre-pended to the source that's specified here, the content prefixes. Raw in this case means just a plain old URL that would have an HTTP or HTTPS prefix, and that image would be included. This image tag basically translates to an image tag in HTML, and has some of the similar attributes: ID corresponding to the ID attribute, style corresponding to the class attribute on the image tag, width and height corresponding to the same, and border corresponding to the same name attributes on the image tag in HTML.

Okay moving on to the next one, the label tag. This will insert a bit of text, but does not put the text on its own line. If you want to put the text on its own line, the label tag should be surrounded by a container tag to draw a square around it and force it onto its own line. It has the text that you want to put in the label, an ID, and a style, which would correspond with a class in HTML.

The label tag is implemented using a span with the text inside the span tags in HTML. So the ID would end up being the ID on the span tag, and the style would end up being the class on the span tag. This is basically just to display a bit of text. Similarly the link element is to render a hyperlink that links to a resource or another request, another screen.

The text is the text that would be displayed on the label for the link, ID would be put on the ID tag. So the link tag here corresponds to the anchor tag in HTML. The ID would correspond to the ID attribute on the a tag, the style would correspond to the class attribute on the a tag.

Target corresponds to the target attribute. This is the location that the link would go to. The name will translate into the name attribute on the tag. Prefix will be put in front of the URL as needed. Target window can have any string just like the target in a actually.

Okay this target corresponds with the HREF in the a tag, so we'll clarify that here. The target window corresponds with the target attribute in the a tag, so if you want it to always open up in a new window you would use the _blank HTML convention. Again the target is the destination for the link, and translates to an HREF if this is being rendered in HTML.

The url-mode specifies intra-app which is the default, which means it will go to a request in the same application, which means the target here will be the name of a request. Inter-app means it's going to another application that is an OFBiz application running on the same server.

And the target will typically contain a / then the name of the webapp, then another / then control for the control servlet, and then another / and the name of the request. So basically it is a relative link to the request in the other app.

Another thing the inter-app does that is special is it will append an external login Id value for the current user if they are logged into the current application so that they will not have to re-authenticate when they get to the other application. Content is just as the content URL mode up here, and plain is for just like the raw up here. So you would basically specify a full http or other URL to link to.

The full-path attribute by default is false. If it's set to true then in the case of an intra-app or inter-app URL it will prefix the protocol port server URL and so on, so that it is a full URL rather than just a relative URL.

Secure true or false. You can force it to always be secure, by default that is false, so it will be secure automatically in the case of intra-app. It'll be secure or not automatically based on the request, the value of the HTTPS attribute on the security element under the request map element that we saw when we were looking at the control servlet.

That's kind of the standard OFBiz functionality. The encode here is by default false. This will encode the URL that is generated using the HTTP URL encoding where it spaces characters using the &; syntax. Now it's possible underneath the link, if you don't want just text to be displayed, you can put an image tag so that it'll be an image link.

Okay the content. Tag is for rendering a content from the content manager that is in the content component of OFBiz. Using the content tag you simply specify the content Id and it renders that piece of content. There are certain things you can do. These are not actually implemented right now, the edit-request and edit-container-style, but these would be for administrative cases where you had inline content editing inside your webapp.

These are placeholders that are partially implemented, so if you wanted that functionality in your webapp you would use these tags. The xml-escape attribute by default is false. If you set it to true it will take the content that is rendered according to the contentId you specified looking up a content record or a record in the database in the content entity.

And after it renders that content it will xml-escape that content. In other words it will use the &;escape character escaping so that it can be included in an HTML or XML file without being interpreted as HTML or XML tags. The cub-content tag here is every similar, but does a little bit different things. The content-id that you specify here is a parent content-id, and along with an

association name it looks up a relationship going from the parent-id to the content that we want to rendered using the assoc-name.

So when you create an association between two pieces of content, in this case it would be from the parent to the child, you can specify on that content an association name.

00:24:34 screen changes to entitymodel.xml

Let's look at the content data model real quick, so that what I'm talking about is a little bit more clear. Okay so here's the content-assoc entity that associates from one content record to another. So the contentId that's specified in the contentId attribute of the sub-content element will be specified here.

The content that we actually render its Id will be in the contentIdTo, and its association name will be in the map-key field right here. So this is used if you want to have more managed content such as if you have a root content element that has all of the content associated for an e-Commerce site. We actually have some demonstrations for this, some demo data.

When you want to render the content you specify this root-content-id and the name of the association. For example this is basically like saying find me the current policy content that is associated with this website, or find me the current footer content that is associated with it.

And you can take advantage of the full power of the content association entity with effective dating and so on, so that the associated content, the contentIdTo, can change automatically over time without you having to change your code. It can change on an administrative basis since the only thing you'll be putting in your code will be the contentId, the parent content or FormContentId, and the map-key.

00:26:34 screen changes to Framework Quick Reference Book Page 9

Okay so that's what the sub-content is used for. The widgets in this section are for including other artifacts in the current screen. The decorator-screen is used to decorate part of the current screen with the name screen in the given location.

By the way you'll notice the location here is optional. If the location is not set then it will assume it's in the current file, just looking at the name. So the name screen is rendered, and we can have one or more decorator-sections the are basically inserted into the decorator screen. So inside these decorator sections as defined here, we can basically put all widgets, zero to many of each, and of course in any order.

Underneath a decorator-section tag we would put the same things that we put inside a container, or that we put inside the widgets or Fail Widgets tags. The decorator-section-include tag is put inside the decorator-screen, the one we are referring to with the name and location.

And in that screen when it gets to a decorator-section-include with a given name, it will look up the decorator-section that corresponds with the same name from the screen that called the decorator, and it will insert that there.

So that's basically the decoration pattern. We have different decorator-sections, and the decorator will basically wrap around these different decorator-section by including them wherever the decorator-section-include tag is found inside the decorator. That might be somewhat confusing, so we will be looking at some examples of this so you can see more detail.

Before we do that I just want to mention real quick about the protected context in the Screen Widget when you render a screen it has its own context. When you include another screen or a form or a menu or whatever, the context of the current screen is kept protected from the included screen. So if the included screen creates a new variable, it will be in that screen's context which inherits from the including screen.

But once this included screen is finished, its context will disappear, unless the share-scope attribute here is used. So you'll notice these various things to include a form, include a menu, include a screen, and include a tree. They have this share-scope attribute but it's not used for menus, just form, screen, and tree.

You can tell it to share the scope or not. By default it is false, so that the scope of the included artifact will be protected, or the scope of the current screen will be protected from the scope of the included artifact. So the same thing is done with the decorator-screen.

When the decorator-screen is called it will be running in its own scope. But the decorator-sections are considered part of the calling screen so they will run in a different scope, they will run in the context of the calling screen, rather than in the context of the decorator so they're truly isolated.

So real quick before we look at the example that I keep on promising. Include-form, basically we have the name of the form and the location of the XML file. Typically the the way these locations will start with component:// so that they can be the location of a file relative to the root of the component, either the same component that the current Screen Widget XML file is in, or any other component that's deployed in OFBiz.

It is possible also in these locations to specify a classpath resource. If you do no prefix, and you use / to separate the different directories, then it will basically look for it as a classpath resource. You can also use

other protocols such as HTTP and such, although the utility of that is probably somewhat limited.

Okay so menu, name location, just like the others. Screen including a screen name location just like the others, including a tree name location just like the others. Now here's one the iterate-section that is very similar to the section element over here, except what it does is it wraps around the section and iterates over a list. For each entry in the list it will put it in a variable, specify it by the entry name.

So if we had a list like a product list and we wanted each entry in the list to be called product we'd put product for the entry name, and notice these two are required. We can specify a view-size paginate target and whether to paginate or not when we are iterating over the list, so if it's a very long list we can specify a view so we'd only view ten or twenty at a time. The default size I believe is twenty, and paginate by default is true.

So this will render previous and next links to go to the next chunk in the list. One little quirky thing here. The key-name, if you specify that, will be used along with the list, if the variable named by the list-name is actually a map instead of a key, so each entry has a name/value pair, or a key/value pair, instead of just an entry.

Then the key will be put into whatever variable is named here, and the value will be put into whatever variable is named as the entry-name. So the list actually can be one of various different types of collections, including an actual list, or a set, or a map. Those are all accommodated using this same tag.

Okay just one more little set of widgets to look at before we look at some examples. The platform-specific. This is for denoting that these different widgets are not generic but rather are platform specific. Now some of these other things, you might be wondering how generic they can be. A label can be fairly generic, a link can be fairly generic but is obviously HTTP related. For other things if we are actually going to include a template, for example, that would be platform-specific.

Typically with in a platform-specific tag you'd have an HTML tag underneath it. We do also have a placeholder for a swing platform specific elements. We've looked at certain technologies like XUI which is used in the POS, Point Of Sale component in OFBiz. So this swing element could be used for desktop application specific visual elements. But right now the only one that's really implemented is the HTML one.

So underneath HTML we have the HTML widgets, and there are really only two of those; the html-template and the html-template-decorator. This one will treat the template as just template, and include it at the point where this is in the Screen Widget, in the screen definition. The decorator will wrap around a subset of the screen.

Okay we looked at the HTML tag. It has the HTML widgets in it, basically one of these two, you can do zero to many of each. The html-template is very simple, you just specify the location of the template you want it to include. This will typically be an FTL file, FTL meaning the FreeMarker Template Language. And the location again will typically use the component:// syntax to specify a filename relative to a component directory.

The html-template-decorator class is slightly more complicated. It has a location just like the html-template, of the FTL file for example to use. And it also has one or more html-template-decorator-sections. Each of these html-template-decorator-sections will typically have a name, and widgets underneath it that will be rendered.

So just like a screen decorator, as this decorator is run it will go until it gets to a point where a section is included and then this will be called. All of the widgets in here will be rendered at the point where that section is included and put into the stream, and then it will carry on rendering the html-template-decorator. You can have multiple ones here just like you can with a screen-decorator, that by the way we refer to as the composite decorator pattern, which is a small variation on the decorator pattern where you can have multiple decorator sections for the decorator.

So that's used here in the html-template-decorator as well as up here in the decorator-screen where it can have multiple decorator sections. Okay now we're here. Let's look at some examples. The main thing I want to focus on in the example is actually this decorator-screen, because it can be somewhat confusing when you're doing decoration along with the protected context. Just a little warning.

00:39:30 screen changes to OFBiz: Catalog Manager: Edit Product

So when we're looking at the EditProduct screen, which corresponds to the catalog manager, there is a main part of the page that is just for this screen. Basically the Edit Product form, that starts there and goes all the way down to the bottom here, is the last element in it. And it also has a little FreeMarker file that it includes to render this duplicate product form, that will create a duplicate product and copy over various related records as well.

So these are the main parts, that form and this included FTL file are the main parts of the Edit Product screen. The rest of the stuff up here is a decoration that's shared between all of the se screens. So if we look at the prices screen it has this same decorator on top of the labels inserted there, and it has the same buttons, content screen, same thing.

00:40:54 screen changes to OFBiz: Catalog Manager: Edit Product Categories.

So that's obviously the Edit Product content screen. The Edit Product Category screen, same thing, it has a decorator here and this part down here in two forms in this case. This is a very common pattern to have an edit form with an add form below, like we talked about in the Framework Introduction looking at the example component. So all of this is a decorator.

00:41:08 screen changes to OFBiz: Catalog Manager: Edit Product

So back to the Edit Product screen. When we look at, and then of course everything outside of that: the side bar, the left bar over here, the application header, the global header with all of the links, the tab bar up here, and all of the stuff in the top header, and also down at the very bottom of the screen.

All of that is basically just a decorator, and in fact this part in the footer, and the very top here the header, is in a global decorator that's shared between all these applications; accounting catalog manager, content manager and so on. They all share the same one.

00:41:39 screen changes to ProductScreens

Let's look at how that is implemented. So in the editProductScreens when we look at the example screens we saw how it runs through these, running the actions and then rendering the widgets. The first widget it runs into here is a decorator screen, so it's going to render this decorator, the commonProductDecorator, that screen, as a decorator.

00:42:07 screen changes to CommonScreens.xml

Now that screen is defined over here in the CommonScreens.xml file. Or should be.

00:42:16 screen changes to ProductScreens.xml

Let's look at where it's actually done. This is a parameterized location that you'll see more and more in the various screen definitions in OFBiz. The reason we use this parameter, the main-decorator-location, which actually comes from the web.xml file...let's look at the web.xml file.

This makes it easier to customize the screen so we can reuse this same editProducts screen with a decorator in another application, so we can basically copy it over without having to use the same decorator that was used here. We can use a different decorator.

00:43:02 web.xml

So if we look over here in the product catalog webapp in the web.xml file, find this main decorator location, you can see that this is in the CatalogCommonScreens.xml file.

00:43:15 screen changes to CatalogCommonScreens.xml

Let's open that one up. So that's where we'll find the main decorator. It'll typically have a main-decorator like this which is the decorator for the entire application, and notice that even it includes a screen here, the globalDecorator. And then inside that globalDecorator it includes the other decorated artifacts, like the editProductScreen. Here's the commonProductDecorator.

00:43:54 screen changes to ProductScreens.xml

That's the same one referred to in the productScreens, the commonProductDecorator. It has a single section used here named body. So we're saying run this decorator, when you get to the point where the body section is included then run all this stuff, and then when we're done with that run the rest of this decorator-screen.

00:44:24 screen changes to CatalogCommonScreens.xml

So when we look at over here in the CatalogCommonScreens, it'll basically run down, run the actions. This has a decorator of its own; we'll skip over that for a second and look at the section in here. So it's going to run this section which is used because we have a conditional element here to check a permission.

Inside this section basically it'll just be running down here. This condition, if the user has permission then we carry on, render the widgets. If the product Id is not empty we do all this stuff, regardless of all that we get down here, and we see the decorator-section include. At this point, basically this line will be replaced by the widgets underneath the decorator section over here.

00:45:14 screen changes to ProductScreens.xml

These will basically just be inserted right here where that line is, and then it will finish rendering.

00:45:27 screen changes to CatalogCommonScreens.xml

So there's the decoration pattern. Now this is where it gets to be more fun, because the decoration pattern, as I mentioned in the decoration pattern as I mentioned before, along with the protected context for a screen, it means that when it's rendering all of this stuff, it will be in the context of the commonProductDecorator screen.

00:46:05 screen changes to ProductScreens.xml

Which because it is included from the editProduct screen, it will have a context that inherits from the edit product screen. But it will actually have its own context for the CommonProductDecorator.

00:46:19 screen changes to CatalogCommonScreens

Now what that inheritance means is that if we refer to a variable here that is not found in the context of the

commonProductDecorator then it will look to see if it can find that variable in the context of the editProduct screen.

00:46:35 screen changes to ProductScreens.xml

00:46:44 screen changes to CatalogCommonScreens

So here's where the trick comes into play. All of this stuff, like all of these widgets and such, are being run in the context of the commonProductDecorator screen where we are right now.

00:47:09 screen changes to ProductScreens.xml

But soon as we get to this and it includes the section from the editProduct screen over here. So when it's rendering this stuff, this still will be rendered in the context of the editProduct screen, and not the context over here of the commonProductDecorator here. In other words it's as if this stuff doesn't know anything about the decorator, about what the decorators' doing. So when we write code here we don't have to worry about what's going on in the decorator, we can just decorate it and it stays separated.

But what that means if you declare a variable over here in the commonProducteDecorator like this leftBar-ScreenName, that will not be available under here, or in any other part of this editProduct screen. Won't be available in these because they use the context of the editProduct screen. Now it may seem a little bit tricky and backwards to do this that way, protect the context that way, but it really helps a lot to make things less error prone so you're not writing things here, or writing things over in your other screen that will interfere with each other.

Now there is a way around that. Each of these set operations has a little attribute on it that's called global. If you do global=true, by default it is false, then instead of putting it in the context of the commonProductDecorator then it will put it in the global context, which is above all other contexts, including above the editProduct context.

So this is a context that's initialized when the very first screen is rendered and the context for that screen is set up. It will basically sit above the context of the first screen, in this case the editProducts screen. So if we do that the global=true over here, for the leftBar-ScreenName, then our editProducts screen could refer to that variable and see the value that was set, even though it was set in a file that was included, as long as it is after the actions in that file run. Which means it would be okay here, in the decorator section. It would also be okay down here, if there are any other widgets after the decorator.

00:49:37 screen changes to CatalogCommonScreens.xml

Okay you'll see some examples of how the global=true is used in other decorators. Like if we look at the main decorator here, it's setting up a UiLabelMap with the localized labels for different languages, and it sets global=true because we want that to be available to all of the children of this as well.

Okay now one other little confusing thing, or interesting and really cool thing I should say, about the protected context, is that when we have multiple layers of decoration. So the editProduct screen is being decorated by the commonProductDecorator, and then when we look at the commonProductDecorator we see that it is being decorated by the main-decorator up here. And in this case instead of decorating it explicitly, there's kind of an implicit decoration going on where this is being passed over to the globalDecorator. This is the decorator that's the very outside, that's used for all of the applications.

00:50:53, screen changes to OFBiz: Catalog Manager: Edit Product

So if we look at it visually we have this decorator here, the commonProductDecorator that adds this stuff, these links, and a placeholder for this label. Then we have the main decorator for the catalog manager that has all of this stuff; this application header and this side bar. And then we have the global decorator that has all this stuff up here, and all this stuff down here.

00:51:21 screen changes to CatalogCommonScreens/ ProductScreens

The interesting thing to notice is that in the editProduct screen we use the name-decorator-section of body, so that this screen knows when it runs into a decorator-section-include with the name of body, that's what it will include.

Here's the funny thing, and it's done intentionally so we can demonstrate this feature of the protected context. The decorator-section here is also called body, so when the main-decorator runs, and it runs into a decorator-section-include named body, it will actually know which body decorator-section to use.

These are put into the context, but they're put into the local context, so that when we refer to the decorator-section-include body here, it knows that it's this one because it's the next level up. And for this decorator-section of body, if this was implemented as a decorator instead of pushing the decoration to the globalDecorator, then it would be here. But instead it's in the globalDecorator so let's look at that real quick.

00:52:36 screen changes to CommonScreens.xml

This is the CommonScreens.xml file in the common component, and it has the globalDecorator in it. This as a decorator-section-include down here with the name of body, and it will basically look at the next layer up.

00:52:53 screen changes to CatalogCommonScreens

Which in this case, since this isn't being used as a decorator, just as a pass-through decorator, then it will know that it is this body that we're looking at. So that's another feature of the protected context that's being used for when another artifact is included from a screen.

Okay so some other things. There are lots of things in these examples that we could look at. When we were looking at the main screen over there we had an include-form for the editPproduct, with the name of the form, the location of the form file. This is all kind of a review from the Framework Introduction videos.

00:53:41 screen changes to OFBiz: Catalog Manager: Edit Product

And that was the first part up here, all of this form stuff, all the way down to here. Then this next section, starting with the little separator bar, this duplicate product form is actually in an FTL file.

00:53:58 screen changes to ProductScreens.xml

So you can see here how we are including, using the platform specific tag and the HTML tag underneath it, we're including that FTL file.

Okay that's basically it for the widgets part of the Screen Widget, and we've looked at some examples and some details about how the decorator screen stuff works.

Part 3 Section 2.2.1-B

The Screen Widget: Shared Conditionals

Starting screen ofbiz-reference-book.161.pdf Page 9

Let's look briefly at some of these shared things. The condition, elements, and action elements. Where it says see shared here it refers to seeing the shared page in the quick reference book, which is page thirteen over here.

00:00:27 screen changes to ofbiz-reference-book.161.pdf Page 13.

So we'll hop down to that. As we looked at this before, these are all the conditions, these are all the actions, general and entity actions. There are also these link image ones, these are actually shared in multiple places too. In the Form Widget we saw the link and image elements. We'll also see these same ones in the Menu Widget and the Tree Widget.

These are all the conditions. Basically inside a condition tag you'll have all conditionals. This is the full set of conditionals up here, all of these guys up here on the left side. We have some that do boolean combinations of conditions: <and/> <xor/> (exclusive or), <or/>, and <not/>. Here are the definitions for these, and, xor, or, and not.

You'll see for and, xor, and or, they all support one to many other conditionals. So you can have a big list of if-validate-methods, or if-emptys or whatever, sitting underneath it. Rather than just having two you can have many. So when you have many in an and situation, basically they all have to be true in order for the and condition to be true.

For xor only one can be true in order for the xor to be true, one and only one.

For an or, as long as one of them is true then the or is true.

For the not you can only have a single conditional under it, just a one, instead of a one to many as with those others. You can just have a single conditional under it and it will basically reverse that. So if you had an if-empty for example and it was true, the not would change it to a false.

Okay let's look at some of the details of these. These various conditions are the same as the conditions used in the simple-method, and as I mentioned before, as in the other widgets, Form Widget, Menu Widget, Tree Widget and so on. If-has-permission checks the permission using the OFBiz security model.

Basically what that means, and if-entity-permission. If-entity-permission is basically a variation on the has-permission, where we allow a prefix, an entity-name, and an entity-id. And then the target operation. We'll look at how that's used in a minute.

So this is a specific case for doing things, making sure that the current user is related to has another record in the database that relates it to this other entity. This isn't used very often, usually when you have custom permissions like that that are needed you'll do some data preparation, and then you may do an if-has-permission. Along with an if-empty for example, or with a <not/> if-empty, to see if the associating records were found.

So the if-has-permission we basically have the permission and the action to split it into two parts. The reason we split it into two parts is because the action will typically be something like _view, _create, _update, or _delete, and we have a convention in our permissions where if there is an _adminPermission, regardless of what the action is, if there is a permission base along with _adminPermission, they have that named permission, then they have the permission regardless of what the action is. In other words it's looking for permission + action, or permission + _admin.

00:05:11 screen changes to OFBiz: Catalog Manager: Edit Product

The permissions are managed in OFBiz. Let's talk about that for just a minute. Permissions are managed in the party-manager, so we'll hop over and look at that in the security tab here.

00:05:24 screen changes to OFBiz: Party Manager: Find Security Group

These are the security groups. The general concept here is that in order for a user to have a permission, they must be in a security group that has that permission associated with them.

00:05:46 screen changes to OFBiz: Party Manager: Edity Security Group.

So if we look at the master of all security groups, which is the fullAdmin security group, there are two sides of it, the permissions, and the user logins that are associated with the group.

00:05:56 screen changes to OFBiz: Party Manager: Edit Security Group Permissions

So these are the all of the permissions that it has, and these are pretty much all of the permissions that you can or ever want to have. And you notice for the most part that they have this _admin, so they can view, create, update, or delete, according to the admin convention separating the permission and operation. There are various other permissions and you don't have to have that suffix on it, like datafile_main does not, just you have this permission or you don't. There are no more granular operations under it. So these are basically all the permissions that the fullAdmin security group has.

00:06:33 screen changes to OFBiz: Party Manager: Edit Security Group UserLogins

And these are all of the users in the group. The admin, userLoginId, and the systemUserLoginId.

00:06:44 screen changes to OFBiz: Party Manager: Edit Security Group Permissions.

So if we wanted to check to see if the user had the accounting_admin permission

00:06:53 screen changes to ofbiz-reference-book.161.pdf Page 13

in our file permission would equal accounting, action would equal _admin, or typically you would say view to view it, and other such things.

In the Screen Widget obviously view would be a common one that comes up. If you want to conditionally display a button for example, you might use other actions such as create, update, or delete. So say we have permission=_accounting, and action=_view.

00:07:27 screen changes to OFBiz:Party Manager: Edit Security Group Permissions

And the currently logged in user is the admin user. So the admin user as we can see here is associated with this full admin group, and the full admin group has the accounting_admin permission which would qualify it for that, and in that case it would return true. After it has permission it would return.

00:07:45 screen changes to ofbiz-reference-book.161.pdf Page 13

Okay moving on to some of the other conditionals here. If-validate-method will call a method in a class by default you can see here the class that it uses is this org.ofbiz.base.util.utilvalidate. They have a whole bunch of little validation methods in there.

But it can be any method in any class. The pattern it has to follow is it has to take a string argument, a single parameter that is a string, and it has to return a boolean, a true or false.

So what gets passed into there is the value of a field here. If necessary it converts it to a string, if it's a number or some other value, so in that field if it's in the current context, it will take the value of that and pass it to the method, specify here, and if that method returns true then this is true, and if it returns false then this conditional evaluates to false.

Okay if-compare. This is used in all conditionals, just like these other ones. It's comparing one field to a value. The operator for the comparison can be any of these: less, greater, less-equals, greater-equals, equals, not-equals, contains, so it can use any of these operators. And then we also specify the type.

So by default it's a string, just as we noted there. For the comparison we can specify the format of the string. This can be used to specify the format in order to convert the value here, as well as the field that is named here if it is a string. And we have some type other than a string, like a long-for-example, which is an integer, a long integer.

If the format is specified it would use that format to convert the value and if necessary the name field to a long from a string. If there is no format specified it uses the default Java formatting, and of course the conversion would be done for if-compare in any time it is anything other than string or plain string.

At least for the value, because this is going to be specified as a string. For the field it will do the conversion between whatever type of object this field name is point to, to whatever type is specified here. As long as there is a valid conversion between those using the utilObject class in OFBiz.

Or utilObjectType. And the if-compare field is very similar, except instead of comparing between a field and a value, it compares between a field and a field. Again using the operator any of these operators, and as nec-

essary or if necessary converting the named fields into whatever type of object is specified here.

Okay if-regxp will do a regular expression validation on a string. The named field, if it is a string then it will do the regular expression directly on that. If it's not a string it will convert it to a string, and then apply the regular expression to it.

If it matches the regular expression it returns true, if not it returns false. We are using the ORO, regular expression valuator for this, that's one of the Apache projects.

If-empty is a fairly simple one. If the named field is null then it's empty. If the named field is a string of zero length then it's empty. If the named field is a list or a set or a map or any of those collections and it has no elements in it then it's empty. Otherwise basically it's not empty. So if it's empty it returns true, if it's not it returns false.

If you want something to be not empty, then you would surround the if-empty tag with the <not/> tag. Okay and those are the conditionals. All of the conditions that can be used in the Screen Widget, Form Widget, Menu and Tree Widgets, so all these various places.

In the simple-method the conditionals are very similar, there are just a few variations like in the if-compare, if-compare-field, actually for most of these basically for anywhere you see a field name in a simple-method you can also put a map name to go along with that. This is not so necessary anymore, which is why we do not have it in the Screen Widget and other newer things because of the flexible syntax.

Part 3 Section 2.2.1-C

The Screen Widget: Shared Flexible Syntax

Starting screen ofbiz-reference-book.161.pdf Page 13

The flexible syntax has a few features. First of all this ${} is used for variable expansion, which is probably the easiest way to put it. Or as it says here it inserts the value of the variable in the screen. For example we have value=${productId}, and it'll take the value of that and insert it here into the string.

The “.” (dot) here is used to access a sub value, or a value in a map. If we have a map in a variable called product, and the map has an entry in it and the key of that entry is the string productId, then product.productId would give us the value of that entry in the map, with the key of productId. One thing to notice about this is that the generic value object of the entity engine and various other things implement the map interface to make this . syntax more universally applicable.

The square braces “[ ]” are used to access a member of a list, so if you say product is the name of a variable that represents a list, and products with [0] is there, then that will be referring to the first element in the list, or element number zero. This does treat list as zero based and not one based for the indexing.

That's when you are getting something out of a list. When you're putting something into a list, if you want to put it before a value you can use something like +o0 to put it a the very beginning of the list. Or if you leave it empty like this products[], then it will leave it at the end of the list. If you include a number but don't put a + in front of it then it will put it after that entry in the list.

Now for some fun examples. Even if it's not used in the string expansion, you can use the dot syntax for accessing a map entry in other places, such as in ENV name or a field name. You can do product.productId to specify that you want the value of the productId entry in the product map.

You can also combine these so ${prodcut.productid}, will get the value of the productId keyed entry in the product map. You can also do funny little combinations like this: products being the list, we have an open [to denote that we want an entry in the list, then ${curidx} variable to specify the index we want in that list, and then we'll use a . after it to specify that the entry that comes after the list is actually a map. So we have a bunch of maps in the list, and the value we eventually want is the value of the entry in the map with this key ${curkey}. So basically we have a list sitting in the context in a variable called products. Each entry in that list is a map, and that map should have an entry with the key of whatever the curkey variable is pointing to. All of this together would get us the value of the entry in that map, in the products list. So we can do fun things like that.

Let's look at a fun application of this flexible syntax, don't remember exactly what page it was on. It would be up here in the simple-method area.

00:05:04 screen changes to ofbiz-reference-book.161.pdf Page 5

Okay in this Control Snippets Example, what we're doing is iterating over this list. We're iterating over the orderItemAndShipGrpAssocList. Typically by the way when we use a variable name like this, the convention is basically to use the entity name for the variable name. This is a variable name, so we'll lower case the first letter, and if it is a list we'll just add a suffix of List to the entity name.

Okay we have this list. We're going to iterate over it using the Iterate tag. Each entry in the list will be put in a variable with this name, orderItemAndShipGroupAssoc. Now what we're doing is iterating over this list, and we're going to put each entry in the list into a map that is keyed by one of the fields in this generic value object.

This is a genericValueObject standing for an entity. We can treat it as a map because the genericValueObject implements the map interface. So in other words a genericValueObject can be treated like any old map. It has a bunch of entries in it, and one of the entries for this entity that will always be there because it's part of the primary key, is the shipGroupSeqId.

So when we do this entry name, as you can see here, .shipGroupSwqId, we're going to pull the value of that field. What this field-to-list operation is doing is taking this field which is the entry from the list that we're iterating over. So we take this field and we're going to put it into a list.

It's a little bit tricky because the list is not just a list sitting out in memory somewhere with a static name. We have this map that we want to contain each entry in the map, identify the shipGroupSeqId, will be a list. So the name of the list we want to put it into, as specified by the list-name attribute will be orderItemListByShGrpMap.${orderItemAndShipGroupAssoc.shipGroupSeqId}. This can be kind of confusing, but it's actually something that you run into fairly commonly when you're doing data organization.

Now we have this list of generic value objects coming in, and we want to split it into a bunch of lists, one list for each shipGroupSequenceId. That's basically all this does, it splits this list into a bunch of lists, one list for each groupSequenceId, and every entry in this list will have a shipGroupSequenceId because it's part of the primary key, and we're basically going to be organizing them by that.

The result of this will be the variable that has this name, and it will be a map. Each entry in the map identified by a shipGroupSequenceId will have a list of OrderItemAndShipGrpAssoc genericValueObjects.

So that's the end result that we'll be getting. Let's run through what this is actually doing one more time just to make sure it's clear. If this is still confusing you might want to go back to the beginning of this section and listen to it again.

We have the OrderItemAndShipGrpAssocList., we want to split that into a bunch of lists, one for each shipGroupSeqId. We're going to iterate over this. For each entry we're going to take that entry, that field and put it in a list. The name of that list will be inside this map, in an entry, and we know it's a map because it has a . right here. The entry we'll be putting it into is whatever the value of the shipGroupSeqId is.

And that's basically it, that's how it works.

00:10:14 screen changes to Framework Quick Reference Book Page 13

Okay let's hop back to this. So this is what I wanted to cover for the flexible syntax. All of this stuff here. So again we have the ${}, we have the ., and we have the []. and they can be used in all sorts of fun combinations for string expansion or accessing entries in a map or entries in a list, for both getting and putting from.

Part 3 Section 2.2.1-D

The Screen Widget: Shared Actions

Starting screen Framework Quick Reference Book Page 13

Okay the next section we'll talk about are the actions. This is a fun one. We have quite a few actions that are available: get-related-one, get-related, property-map, property-to-field, these are for reading value from property files.

Get-related-one and get-related are entity operations, although actually they're here, because there wasn't enough room in the entity column. Anyway those are entity engine operations to get records and entities that are related to the entity of whatever generic value you have in memory in the context.

Property-map will read in a properties file and put it in a map. Property-to-field will take a specific property out of a properties file and put it into a field. Script will run a script like a BeanShell script. Service will call a service, set will set a value in a field, either from a string value that you specified or from another field, plus it can do tight conversions and other little things, you can have a default value associated with it.

Entity-and, entity-condition, and entity-one are all entity operations. By the way pretty much all of these, with the exception of script, are available in simple-methods as well. In a simple-method there are some other things for doing scripts like a callBsh operation and a call-simple-method for calling those different types of scripts.

Okay let's look at these. Let's start with the non entity related ones, so we'll skip over get-related-one and get-related for just a second. Property-map basically loads a properties file, specified by the resource here. It needs to be the full location of the resource on the classpath, and it'll put that into a map. And we just give it the name of the map and it shoves it all in there.

If the map already exists by the way, it will turn the map into a MapStack, and put this on the bottom of the stack so that basically you can have multiple properties files represented by the same map. If the key is not found in the first one in the stack, then it will look down the rest of the stack.

00:03:00 screen changes to CatalogCommonScreens.xml

We actually saw an example of that a little bit earlier, in the CatalogCommonScreens here. In the main decora-

tor, it's using this property-map action to load the productUiLabels.properties file. This is locale aware, by the way, so if we were in a spanish_es locale then it would look for productUiLabels_es.properties, and commonUiLables_es.properties.

So it's going to find the productUiLables.properties file or some variation on it and put that in the map. And since we're using the same map name here (uiLabelMap) it will also, when we say we want the commonUiLables properties file put in that map, it will put that in there as well. The first one here will override the later ones, so these are in direct order of override preference. If it can't find a label in productUiLabel then it will look for it in the common UI labels.

00:04:27 screen changes to Framework Quick Reference Book Page 13

Okay, that's an example of how the property map can be used more than one at a time. Property-to-field is very similar. The resource just the same as the resource up here in a property-map, the property within that properties file, the field to put it in. The default value if that property is not found.

If you don't want it to use the locale you can tell it not to with the no-locale tag. As I mentioned the resource up here will be locale sensitive, if you're in a non English locale it will look for the suffix basically. You can tell it not to do that by setting no-locale to true for the property-to-field action. The arg-list-name is used for substitution, this is kind of for legacy compatibility.

When you have a property here, you can use the ${ } syntax for interesting values, or you can use the arg-list-name, where you simply have a { } and a number within curly braces, and it will look for that numbered entry on the arg list and insert the value there. That's something that's been used with properties files historically, so we support that here.

Okay script as I mentioned just calls a script. Typically this would be a BeanShell script. The location is typically a component:// location, but it can also use other protocols such as file://, HTTP, HTTPS of course, and various other things, including if you put no protocol in front it will look for it on the classpath. That's pretty common for all of these location attributes.

Service will call as service the name of a service, the result-map-name, which is optional, if there is no result map then it will put the results into the context. Result-map-list-iterator-name is for a special case. Actually this has been simplified; this attribute no longer exists, so I'll have to take it out later.

Okay this result-map-list-name can be a list or a list iterator. This is actually a special case when used with the Form Widget, and it corresponds with the list name for the form that the form will iterate over. So it doesn't have to match, you just specify the name of the result that's coming back that should be used as the list. Or if it is an instance of a list iterator you don't have to put that in a separate attribute anymore, you just specify that in the result-map-list-name, and we'll cover that a little more when we talk about the Form Widget.

This attribute, auto-field-map, by default is true, but you can set it to false to get it to not automatically map the inputs from the context to the service incoming attributes. If you specify false you'll just use the field-map element here to map all the fields in manually. If you specify true it will do the automated mapping, but you can also do additional field-maps if you want to override or augment what the auto-field-map does.

Okay the set operation as I mentioned sets a field, puts something in a field. It can either set it from another field or from a value, so these are both marked as optional but you'll use either one or the other, either the from-field or the value.

Now the global tag, just like we've seen in other places. Typically it's false, and if it's false it'll put it in the current scope. If it's true it'll put it in the global scope. For Screen Widget only we have this from and to scope. Typically the from scope will be in the current screen, and the to scope will also be in the current screen, these are the defaults.

You can also put it in the user scope, which corresponds to a session if the screen is being rendered in a servlet context. Or application, for an applications scope which is the servlet context application for the entire webapp basically.

Those aren't used too much, but that flexibility is available if you need it. As I mentioned before the set operation can also do a type conversion. By default it actually does a string, but you can specify any of the other object types, any of these other object types as well. If you specify object it will do no conversion, just copy it over. If it's a value it will end up being a string that goes into the field. If you do object-from-field, basically whatever that field's object is, that's the same object type that will be here.

Okay let's look at the entity engine specific ones. Entity-and corresponds with the find-by-and in the entity engine. So we have the entity-name, and then underneath it we have a field-map that basically results in name/value pairs, that will be used for the query and will all be anded together.

Some of these other things; use-cache true or false, filter-by-date, take the results back. The default is that it will not filter it by date, the list of results coming back it will look for from date and through date fields, for the entries in that list, and it will filter by the current date and time if you set that to true. The result-set-type by default is scroll which is flexible so you can go forward

and backward through it, but you can also set it to forward which is more efficient if you only plan on moving through it in one direction.

Okay by the way this result-set-type, for forward or scroll, really only matters if you do a use-iterator tag for the return type. So some of the elements underneath the entity-and. By the way the entity-and, entity-condition, entity-one, and the get-relateds are all pretty much exactly the same in the simple-method and in other places.

The field-map as I mentioned basically just has name/value pairs. Each one has the field-name as well as we can see the actual definition down here. So each one has the field-name and the env-name that the value can come from.

You can also optionally specify a subset of the fields to select. If you don't specify any of these select fields it will just select all fields on the entity for the given entity name. You can specify one or more fields to order by. If you specify more than one of these, it will order first by the first field, second by the second field, and so on, just as in a normal SQL query or any other ordering situation.

You can have up to one of these three things, zero to one, limit range, limit-view, or use-iterator. If you don't specify any of them the results coming back from the entity-and, and the same pattern is followed for entity-condition, if you don't specify any of them then it will return the entire list of results.

If you specify limit range then it will use the limit-range tag down here. So you can specify a start, value, and a number of results to include, so you can say start at result number 20 and include 10 of them, for example.

Limit-view is similar, except instead of doing a start and a size, you have the view-index and the view-size, and what this is referring to is, say you have a view-index of 3 and a view-size of 10, then it would start at view-index-1 times the size. So view-index of 0, and a size of 20, would get you results 0 to 20. View index of 1 and a view size of 20 would get you results 21 through 40, and so on.

So there's just a slight variation that can be convenient in some cases, especially for pagination and such. Use-iterator, if you do that one it will simply tell entity-and or entity-condition to, instead of returning a list, return a list-iterator.

Now for the most part in the widgets you won't explicitly close the iterator, you'll simply iterate over it. And once the iteration is complete the iterate tag will close the iterator. So kind of automatically close it for you. That's not as flexible as if you were using the Java entity engine API and manually closed it, but that's basically the way it's taken care of here.

So the use-iterator can be used when you expect to have a very large resultset that you want to iterate over one at a time without pulling all of them over to the memory of the application server, which might cause an out of memory condition.

Okay entity-condition is kind of like entity-and, in that it results in getting a list of results back, a list of rows from the database basically. Generic value objects is how they will be represented. Same entity-name, same use-cache, same filter-by-date. Here the filter-by-date is a little bit different because for both of these rather than post-filtering the list when it comes back, it will actually add these to the condition when it's building the query.

Okay distinct true or false, this corresponds to the distinct keyword in SQL. It will basically filter the results making sure that all of them are unique. There are any duplicates that are not distinct basically. For the given fields that are selected, using the distinct-equals-true you'll want to limit the selected fields. So any records that are not distinct or unique will be eliminated.

Delegator-name. So there is an extra flexibility point the entity definitions has. By default it would use the delegator for the current context. For the Screen Widget that means the delegator associated with the current webapp. For a simple-method, for the service, a simple-method being called as a service for example, it would use the delegator associated with that instance of the service engine.

Result-set-type is the same, either forward-only or scrollable. By default scrollable for flexibility, forward-only would perform better.

Okay now this is where the conditions come in, it gets kind of fun. You have a choice of any of these three elements; condition-expression, condition-list, and condition-object. We'll talk about what each of these is. If you do an condition-expr that's the only condition. If you do a condition-list you can have multiple condition expressions, other condition-lists, and so on. If you do a condition-object that's also the only one. Condition-object will be an instance of the entity-condition, or an object that implements the entity-condition interface.

So for example if you call a service that returns the entity condition, then you can actually call, use the entity-condition tag to run a query with that condition, by just specifying the at-condition object. We also have a having-condition-list. This is something that, to use, requires some knowledge of how relational databases work.

When you're doing a query that is grouped, the condition, these are the where clause conditions, that's what these would result in, are done before the grouping. And to filter out the records that would be candidates

for grouping. And then having-condition runs after the grouping.

So having-condition-list corresponds with condition-list, they're very similar. Let's look at the definitions of these actual things. So down here we have condition-expression, condition-list, and condition-object. Condition-object is the simplest one; you just specify the field-name for the field in the current context where that condition object is that implements the entity condition interface.

Condition-expr is the good old staple. This basically has a field-name, this would be the field on the entity that we're specifying, whereas this is the field in the context. It can be somewhat confusing. To distinguish between those in the is operation, the field in the context is referred to as the env-name. So you'll always have a field-name, you'll typically specify either an env-name or a value. If you want to refer to a variable use the env-name. If you want to just point inline a string value, you can put the value here.

Okay so the conditional basically would be the field and operator. These are the various SQL operators or operators supported in the SQL. We can tell it to ignore this condition if the env-name or value is empty or null. By default those are both false. If you want to ignore it when it's null set that to true, if you want to ignore that when it's empty set that to true.

That can be helpful in situations where you're designing a more generic query where they could pass in a number of conditions, but if they don't pass in a value for that condition then you just want to ignore it, leave the condition out. So ignore-if-empty is especially helpful in that circumstance. Basically this will result in part of the expression, that says this field, such as the productId, equals a certain value, like a productId variable sitting in the context.

If you use the between then the evn-name must point to a collection that has two entries in it. If you use like, then it's just a single thing, as will all of the other ones; less, greater, less-equals, greater-equals, equals, not equals.

If you use in this special case like between, then the env-name must be a collection referring to the java.util collection API. So this could be a list or a set or these other various things.

Okay so a single condition expression can be used with entity-condition. Or an expression if you have a more complex condition, it will typically be underneath a condition-list. A condition-list basically has a list of conditions that are combined with either and or or. The default is and as you can see here.

So if you have a number of condition expressions in a condition list, they'll all be anded together unless you specify or, then they'll be ored together. Notice that you can have condition-lists under condition-list, for building fairly complex condition trees, and you can also drop in condition-objects at any point too.

So if you have a condition-object that was prepared by a service, or a script or some other thing, you can use it in your condition-list just like you can use it as a top level condition under the entity-condition tag.

So that's basically how you specify the conditions. Let's move on. Actually before I move on too quickly I should note that these other elements are the same as entity and select-field, the fields to select. If you specify none it will select all fields, the fields to order by, field or fields to order by, and what type of an object to return, how to handle the list that comes back, either the full list, limit-by-range, limit-by-view, or to return a list-iterator.

Okay entity-one just basically does a find-by-primary key, so it will always return a single generic value object if it's found, otherwise it will return null. By return I mean put something in the variable name like value-name here. So we specify the name of the entity use-cache, just like the other.

Auto-field-map is a special one, kind of like the auto-field-map over here for calling a service. By default it's true, so it will look for all primary key field names in the current context, and map those into the conditions that are used to find this, basically the name value pairs that are used to find this generic value.

It will look in the current context as well as in the parameters map. It's common enough to have things coming into a screen or a simple-method in the parameters map that it will look for them there as well, if you have a screen that comes from a request where a produtId was passed to the request.

When you get in here all you have to do is entity-one, entity-name = Product with a capital p, value-name = product with a lower case p, and you're done. It will automatically map the productId from the parameters map.

If you want to get it from a special place just say auto-field-map equals false, then use the field-map to specify where it can find the primary key fields. Select-field is just like these other ones. If you specify fields to select it will only select those fields, otherwise it will select all fields and pull them from the database.

Okay let's go back and look at the entity operations that we skipped, that are in the general thing over here. Get-related-one. If you have a generic value object sitting in the context and you want to get a related entity following one of the type one relationships in the data model, then you can specify the name of the value here: the relation-name, the name of the relationship

between the two entities, and the to-value where it should put the result, and you can tell it to use-cache or not, by default it doesn't.

Get-related is just like get-related-one, except that instead of getting a single value back to put in the to-value here, it gets a full list back. Same sort of thing; you start with the the value object, specify the name of the relationship. You can specify the name of a map that will restrain the query further, beyond the field mappings and their relationship.

You can also specify how you want to order it. This will be a list sitting in the context that has string entries in it for each field that you want it to order by, on oro more fields to order the results by. Use-cache true or false, list-name for the output, and that's pretty much it.

So that's it for the actions. Again these actions are shared in a number of places, including the Screen Widget, Form Widget, Tree and Menu Widgets, and the simple-methods also have things very similar to this with a few exceptions, but for the most part these entity operations, the set operations just about the same. We'll look a the simple-methods in a bit so you can see exactly what those look like. Also the flexible syntax and other things are the same.

Part 3 Section 2.2.1-E

The Screen Widget: Screen Widget Wrap Up

Starting screen: Advanced Framework Training Outline

That's it for this section, so we will be moving to the next.

In this sequence, we're done talking about the control servlet, we talked about the screen contexts that are protected inherited, we talked about the screen decoration pattern that relates to the screen context and how we can have some fun.

We talked about this a little bit before, let me mention it just again real quick. The parameters map is one that's use d in a lot of different tools in OFBiz. In the simple-methods, the parameters maps are the input attributes if it's being called as a service, or the parameters very similar to this if it's being called as an event.

In the screen widget the parameters map is made up like this. First we take all the request attributes and drop them in the map. Then we take all the request parameters and for any that do not collide with an attribute, have the same name as an attribute in other words, we put them in the map.

One thing to note about these request parameters, we use the ? And & syntax parameters that are standard in http.

00:01:25 screen changes to OFBiz E-Commerce Store: Tiny Chrome Widget.

We also use, you can see over in this URL in e-Commerce, this is a search engine friendly parameter basically. We use ~ and then the name of the parameter, and then a = sign, and then the value of the parameter. In the screen widget and any time it uses the UtilHttp class get-parameter method, it will look at all these types of parameters as well as the ? parameters, which could go right on the end here, we could do something like this; foo=bar, and all of these things would make it into the parameters map. To a search engine it basically looks like it's just another directory, so that's what we do to make things search engine friendly. And friendly to certain other things as well.

00:02:29 screen changes to Advanced Framework Training Outline

Okay so first priority is attribute, second priority is parameters, third priority is session attributes. If there are any session attributes that do not collide with request parameters, or request attributes basically anything in the map already, it puts those in the map. Same thing with application or servlet context attributes. Goes through them, if it doesn't find each one in the map, it drops them in the map.

That's basically it for the screen definition. By the way the typical location for screens, inside a component, the given name, will typically be a widgets directory, as we talked about when we were talking about the components structure. We may have a sub directory under there which is represented by this **, and then *Screens.xml is typically how the files are named.

00:03:27 screen changes to CommonScreens.xml, ProductScreens.xml, CatalogCommonScreens.xml,

So we looked at a few of these in this section, like the commonScreens, productScreens, catalogCommonScreens, all of these files basically follow that naming convention.

00:03:38 screen changes to Advanced Framework Training Outline

I should mention here this didn't make it into the outline, that the XSD file for the screens and all of the other widgets are in the framework directory, the implementation of all these are in the widget component.

00:03:56 screen changes to CatalogCommonScreens.xml

Inside there is a DTD directory, and we have the widget-screen schema file, widget-form schema file, widget-menu schema file, widget-tree schema file. That's where all those live.

00:04:14 screen changes to Advanced Framework Training Outline

Okay that's it for our coverage of the Screen Widget. In the next section we'll be talking about the Form Widget.

Part 3 Section 2.2.2-A

The Form Widget: Form Widget General

Starting screen: Advanced Framework Training Outline

The next topic for part three here among the webapp framework, widgets, and MiniLang, would be the in the widgets section, and that is the Form Widget.

We've already gone over framework review, control servlet, and the Screen Widget, now it's time to go over the form widget. The Form Widget is something that you're introduced to in the framework introduction videos as part of a data gathering and simple data interaction user interface that is so common in enterprise applications.

The Form Widget is the best practice tool for defining a form for use in an application. The Form Widget was actually the first of the widgets to be created, and helps significantly in reducing the code size basically for a form, especially compared to implementing it in an FTL file or JSP, where we are laying out explicitly every little field in the form and all of its layout and such. Using the Form Widget all of that can be more implicit and automatic, although you still have a lot of control over the style and layout and how it's presented.

Okay so the general idea of the Form Widget is you have a form definition, and the Form Widget can render that form in the user interface of choice, as long as it's HTML, for now anyways. The intention of course is, as with the Screen Widget as I mentioned before, is that form definitions for the Form Widget should be usable for other types of user interfaces or other types of applications, in addition to just HTML Java servlet based applications. Although the initial implementations, the interpreter for the Form Widget, is very specifically for HTML.

Okay the form definition files are generally located within a component in the widgets directory. Sometimes there will be a sub directory under the widgets directory. But in a sub directory under the widgets directory, or the widgets directory itself, you'll typically see form definition files with this pattern to the file name *forms.xml, such as productforms.sml, orderforms.xml, and so on.

That XML file, the schema for it is located here, in the widget component under the framework, and as with all XML format definition files you'd find it in the DTD directory, and the name of the file for this is widget-form.xsd, which follows the same pattern for the other ones, widget-screen.xsd, widget-menu.xsd, widget-tree.xsd. Those are the only four.

Okay so the form definitions go in a file like that. I should note that historically, before the Screen Widget was introduced and we started using this widgets directory, it was common to put the form definition files in the webapp directory, basically in the same place that you'd put an FTL template.

Before the Screen Widget we'd actually render forms by creating a form renderer object in BeanShell, and then in the FreeMarker file, as part of the data preparation in BeanShell, we'd have this form renderer object, and then in the FreeMarker template we would call that render method on that form renderer object, and it would basically just return the text to be returned right there in the FreeMarker template.

So in those days early on in the Form Widget, we would basically just put the forms.xml file in the same directory as the FreeMarker templates. You'll still see that in some cases. Of course wherever it's located you'll have a reference to the form in one of the widgets tags in the Screen Widget, the include-form tag there, or using the older style you might still run into that on occasion.

It's pretty rare if it exists at all anymore, where you have the form renderer object prepared in a BeanShell script, and then the render method on it being called in the FreeMarker template. So in that case you can look in the BeanShell script to see exactly where the forms.xml file is located. But more normally you'll use the include-form tag, you'll see the include-form tag in a screen definition that will specify the location and such of the form file.

Okay let's now look in detail in the framework introduction you saw some examples of usage of the Form Widget, and we will be looking at some examples of usage, especially in some more advanced things like the type-multi form. But let's start out with a review of the form element, and all of the form subfields.

00:06:22 screen changes to Framework Quick Reference Book Page 10

To do that let's look at our handy-dandy quick reference book. On page 10 is the first page with the Form Widget information. It has the root element over here on the left, of the XML file which is just forms, and then you can have one to many form elements in that file. Each form element can have a whole bunch of attributes on it, and we'll go over all of these in beautifully painful detail, and then. So a lot of these are really not necessary, but can be used to control more of the style and layout of certain behavioral forms.

00:07:24 screen changes to Framework Quick Reference Book Page 13

We'll look at the sub elements of forms, with the exception of the actions, which are on the shared page over here on page 13, the general actions and entity actions. We covered these under the screen widget details, so if you missed these or want to those you can go back and look in the shared elements section under the Screen Widget.

00:07:45 screen changes to Framework Quick Reference Book Page 10

Okay so we have a few different elements down here that are part of the Screen Widget that we'll talk about. This is an example that's taken from the product forms. And so we may refer to some things here but we'll mostly be looking at more advanced examples.

00:08:11 screen changes to Framework Quick Reference Book Page 11

So on page 11, we have more information about the individual fields of the forms. This is kind of the more important part of the Form Widget, and the Form Widget reference. These are all the different types of fields you can have in a form.

So each field element here has optionally one sub element. The type of the field does have to be defined somewhere. But if it's already been defined previously, and you're using a field with the same name, then you don't have to specify the field type element again, you can just use the second field element with the same name to override certain things like a title, or styles, or use-when, or whatever, to override that behavior for the previous definitions of that specific named field.

But generally you'll have a field element, and under it, in this allFields group, you'll have one of these field type elements. So here's the all fields group. These are all of the different types of fields. And over here in these two boxes, this box being the first of two, and this box being the second of two, are the details for all of the field types, so we'll go over those.

And then we have the field sub elements, so some of these field types have certain sub elements that they use, and these are shared between a few of them, so we'll talk about those in context of where they lie in here.

00:10:24 screen changes to Framework Quick Reference Book Page 10

Okay let's go back to page 10, the first page of the Form Widget, and let's start looking at all of the attributes for the form tag itself. The most common case for a form definition, you'll specify the name of the form, this is used for referring to the form externally, also when the form is rendered in HTML. For example, this is the name that will go in the name attribute on the form element in HTML. There are different types of forms, single form and a list form. We saw the single and list forms in the Framework Introduction videos, a single form being the style where you are editing a single record basically.

00:11:46 screen changes to OFBiz:Catalog Manger: Edit Product

So this edit-product form is an example of a single form, where we're basically just editing a single record. Let's get OFBiz started up again so we can look at some of these other things. A list form would have a table, with cells, and would basically have one record for each row.

00:12:08 screen changes to Framework Quick Reference Book Page 10

The multi form is very similar to a list form, except that in a list form you'll have a submit button for each row, like one form actually on each row, form in terms of an HTML form, so one form that can be submitted on each row.

With a multi form you'll have one form for the entire set of rows, with a single submit button that will submit all of them together. There are some special options to keep in mind when you're using the multi forms, and we'll look at some examples of those, especially as we get down here to some of the multi form specific attributes, like row-count and use-row-submit.

Okay the upload type form is not used a whole lot, but when you need to do an upload this is the type of form to use. There's a special case, and it's kind of like a special case of the single form. For every form you need to specify the type of the form and the name; most of these other elements are optional.

Target is optional, but typically you'll want to have a target so it knows where basically this would be usually the name of the request that the form would submit to, but what goes in the target depends on the target type. This is the same as you've seen in other places, where you can have intra-app, or the target would then be a request in the same application, or inter-app, where you are jumping from one application to another and usually specifically one OFBiz application to another.

This is the one that puts in the external loginId on the URL, so that if the current page was the last one that was requested for a given user, when they click on one of these inter-app targets or submit a form with an inter-app target, it will pass in that external login id to make sure the user is logged in in the other application so they don't have to re-authenticate. Content is kind of like the analog to the intra-app, it's basically the OFBiz URL transform that you'll see in FreeMarker, and in general the OFBiz URL generation for point to a re-

quest, and content will be the target for pointing to an OFBiz content URL with the content prefixes.

It's not very common you'd use this in a form submission. In fact I don't know if you even want to use it, but it is there for consistency with the other target types. And you'll see this sort of target type attribute in a lot of places, so it's there for consistency with that.

Plain tells the Form Widget not to do anything with this target, just to use it like plain HTML or whatever, like an HTTP address or other type of address. It could be a Javascript snippet or that type of thing. Obviously when using the plain target type it is more HTML specific and may not be so reusable on other platforms, if that's a concern to you for a given application.

The most common type is the default intra-app, or going to another request in the same application, so most of the time you don't even see the target-type attribute used, because it defaults to that. That's basically how the target-type attribute is used, how it determines what would go into the target attribute.

Target-window for HTML rendered forms translates into the target attribute, so if you wanted to do it in a new window all the time then you could do _blank, or you could give a new window a name so it's always open there.

Okay the title and tooltip. Title is just the title of the form, a tooltip is a little tip about the form. That is shown inside the title, typically for the case of the form. Each form field also has these attributes, title and tooltip, for information about the field.

Okay list-name is specific to list and multi type forms. This is the name of the list to iterate over when displaying the forms, so each entry in the list will typically correspond to one of the rows in the table for the list or multi form.

List-entry-name is optional. If you specify list-entry-name each entry in the list here will be put into that entry. If there is no list-entry-name then the list must contain map objects, and those maps will be expanded into the form context. So each name/value pair will then be a variable in the form context. One thing to note about this list name; the object referred to in it can be an actual Java util list, or it can be an other type of collection. It can also be a list iterator, or more specifically the entity list iterator from the entity engine, and the Form Widget will support iterating over that and closing in the case of the entity list iterator, closing it at the end as well.

Okay the default map name, default-entity-name, and default-service-name. These correspond to the three similarly named attributes on the field element: the map-name, entity-name, and service-name there. If you specify a default-map-name, default-entity-name, or default-service-name for the form, and there is not map-name, entity-name, or service-name, respectively, on a given field, then it will default to the value you specified on the form.

We'll talk about these in more detail when we're talking about the field. Each field can have a special map name, so rather than looking in context it looks in a specific map in the context, for its value to populate in the field. In default-entity-name, each field can correspond to an entity-field or a service-attribute. So each field has a field-name and an entity-name, and an attribute-name and a service-name, to specify a given field and an attribute.

When a field is automatically created, and we'll talk about that in a bit, it'll keep track of the entity or service info for the entity field or service attribute that it came from. And basically what the Form Widget does when you automatically define fields is do a best guess based on the entity-field definition or service-attribute definition, or preferably both. It can get a pretty good guess from those things.

Okay this form-title-area-style and the form-widget-area-style are similar to the default-title-area-style and default-widget-area-style, except that pretty much for all of these default tags there is a corresponding tag on the field elements, just like we talked about with the default-map-name, entity-name, and service-name. With the default-title-area-style there is a title-area-style on the field element, as there is a widget-area-style, title-style, widget-style, tooltip-style, and so on.

The title-area-style corresponds with the title-style, so each field will basically have two parts; it'll have a title and a widget. In the widget area the tooltip is also rendered, kind of more information about that widget. And by the title, if a field is set to required, a little star will be put in this required-field-style. And so that's what that style is used for, to denote a required field. Where it has area, in a form if it's a single form, you'll have the titles down the left side.

00:22:53 screen changes to OFBiz: Catalog Manager: Edit Product

Like this, so this product Id is the title. The actual value of the id here is the widget. And this inside this funny -[and]- is the tooltip, and you'll notice it's in the widget area, over here to the right. So this is the title, and around it is the title area, which right now is implemented as a table cell. If you put a title-are-style in there then it'll style that cell. So if you wanted to do a background color around this or whatever, then you could do that there.

That's opposed to changing the style of the title itself. If you put a background cover here it would just cover what is selected, this area of the actual text. But there is a table cell here that the title is in, that's the title-area,

just as there is a table cell over here that the widget is in, and the tooltip, and that is the widget-area.

On a field if you specify the widget-area-style, or the title-area-style, that's where it'll be used. It'll probably make me log in, because it's been more than a half hour. Let's try that one more time.

00:24:28 screen changes to OFBiz: Catalog Manager: Edit Product Prices

Okay so in a list form we have the titles up on top, and then the widget areas down here. This table cell up here would be the title area, so if you put a style on it that's what would be styled. This is the widget area, so if you put a style on it that's what would be styled there.

00:24:52 screen changes to Framework Quick Reference Book Page 10

And that's how the title-area-style, title-style, widget-area-style, widget-style and tooltip inside the widget area, and required-field-style in the title area are all used. These other elements up here, the form-title-area-style, and the form-widget-area-style, are similar to the normal title-area-style and widget-area-styles, but they're used for a list or multi form where separate columns is false, which is the default.

00:25:36 screen changes to OFBiz: Catalog Manager: Edit Product Prices

In a list form, the default behavior where separate columns is false is that it will put all of the editable fields, plus the update button for that, into a single cell, basically as is done with this form. So this is the form-title-area.

00:25:58 screen changes to Framework Quick Reference Book Page 10

And that corresponds to the form-title-area-style. If we want to specify the style for that we'd use this attribute, the form-title-area-style attribute.

00:26:08 screen changes to OFBiz: Catalog Manager: Edit Product Prices

And this is the form-widget-area. It's called the form-title-area and form-widget-area because this is where the actual editable part of the form is put, in this style of form.

00:26:39 screen changes to OFBiz: Catalog Manager: Edit Product Prices

If you do this separate-columns set to true then each of these elements, in this case To Date, Price, termUomId, and Update, those four fields would all go in separate columns, just like these other things. And they would each have their title-area, and therefore title-area-style, and widget-area and widget-area-style.

00:27:02 screen changes to Framework Quick Reference Book Page 10

So if you use separate-columns = true then the form-title-area-style and form-widget-area-styles would not be used.

Okay the next attribute on the list is the paginate attribute. By default it's set to true. This is also just for list and multi type forms. When you have a very large list to iterate over, either an actual list or an entity list iterator where you're looking at just part of it on a given string, then the default behavior of the Form Widget is to separate that into multiple pages.

When you do say paginate = true, you can specify a paginate target, but you don't have to use it; it will use the normal target for the form if you don't. I think that's the default, I'll have to check that. It may actually be the form name that it uses by default. But the paginate-target should be optional and it will kind of guess at what I believe is the form name, if the form name corresponds with the screen name, which is not often the case.

It's quite common that you'll have to specify the paginate-target if you want to support pagination, cause of course that would be different than where you actually submit the form to. The paginate-target-anchor is basically an anchor on the screen corresponding to an HTML anchor in the paginate target, so it can in a browser HTML rendering for example.

It would just be that # sign, the number sign, with the anchor name after it, and if that anchor is defined in the HTML it'll cause the browser to jump down to that anchor when it goes to the paginate-target. So that can be helpful if you have multiple forms on a page, and after clicking the next button to got to the next page want to jump down so they're in the same context.

Okay the paginate-size-field. You can specify the name of the field where the size is, which will be passed through and looked at, same with the index-field. Paginate-previous-label, which will by default be previous, paginate-next-label which will by default be next, but you can change those to be whatever you want. You can also specify the style of the previous and next labels. And that is it for the pagination attributes.

The item-index-separator. This attribute is specific to the multi type form, and usually you'll just want to use the default value here of the _o_, because that's the default value that the service multi event handler looks at. Let's talk a little more about using a multi service now.

00:31:06 screen changes to OFBiz: Catalog Manager: Edit Product Prices

When you're using a multi service, as I mentioned before, instead of having a form for each row, as we have here, you get a form for all rows together.

00:31:16 screen changes to OFBiz: Catalog Manager: Edit Product Features

If you look over at the features here, if this has any features...let's hop to a product that does have some features. Giant Widget I believe does.

00:31:35 screen changes to OFBiz Features For: Giant Widget [ID:WG-9943]

Okay so if we look at this form here in the features section, instead of having an update button in each row in the form, we have an update button down here at the bottom. So this is a multi form.

00:31:59 screen changes to Framework Quick Reference Book Page 10

If we look at the definition of this form, we'll see that the type = multi, and it has a few helpful things on it. Typically it will not specify the item-index-separator because that's a fairly standard thing, but you can override it and specify whatever you want there. You just have to pass it in so that the service-multi-event-handler knows what to expect.

00:32:28 screen changes to OFBiz: Features For: Giant Widget [ID:WG-9943]

So what will happen when you a submit a form like this is it will call the service to process the form, instead of doing it once it will call it for each row. So each field here in the form will have its name, just as it would in a normal form, but it will also have an appendage sitting on the end. Which will consist of this item-index-separator, the _o_ thing.

00:32:52 screen changes to Framework Quick Reference Book Page 10

00:33:01 screen changes to OFBiz: Features For: Giant Widget [ID:WG-9943]

And the number of the row and the form. I think it's zero based so zero, one, two, and three for these different rows.

00:33:18 screen changes to Framework Quick Reference Book Page 10

Also if-selected. This is configurable with the use-row-submit attribute. A couple of other attributes that are multi form specific are row-count, and it's usually best to not specify this and let the Form Widget figure it out automatically, but you can specify the number of rows that are in the multi form. So that's not used very much but it can be used, if your code is somehow maintaining that.

00:33:57 screen changes to OFBiz: Features For: Giant Widget [ID:WG-9943]

Use-row-submit by default is false, but if you set it to true then you'll have a top level check box like this. If you check it it will check all of them, and if you uncheck it it will uncheck all of them. Or you can just check individual ones. If you don't want to submit all of the rows you can just check the rows you want to submit. And that's the use-row-submit feature.

00:34:29 screen changes to FeatureForms.xml

If you set use-row-submit to true then there's a special field that you need to specify in your form definition. Let's look at those real quick. Here in the feature forms we have a type-multi and use-row-submit = true.

00:34:51 screen changes to OFBiz: Features For: Giant Widget [ID:WG-9943]

So we have this _rowsubmit field that is used along with this. If we looked at the form definition, I think that's a different form definition than we're looking at in the user interface, it would have the same sort of thing.

00:35:15 screen changes to FeatureForms.xml

So it has a type = multi, use-row-submit = true, and it will have an _rowsubmit field. So _rowsubmit is also a convention for the multi type form, and something that the service-mutli-event-handler looks at.

00:35:41 screen changes to OFBiz: Features For: Giant Widget [ID:WG-9943]

On the topic of the service-multi-event-handler, when we submit a form like this there's a special type of event that we need to use in the controller.xml file.

00:36:02 screen changes to controller.xml

Let's look at the controller.xml file for the Catalog Manager. Product catalog...here we are. And the type of event handler we need to use is this guy right here, service-multi. So let's find an example of that. Basically we just say service-multi is the event type, and we just specify a service name. And the service-multi-event-handler will automatically take care of splitting all of the different form fields that come in.

00:36:30 screen changes to OFBiz: Features For: Giant Widget [ID:WG-9943]

It knows which row they're on because of the suffix that's put on each field name, with the _o_ and then the row number, so it'll split all the fields into rows.

00:36:44 screen changes to controller.xml

And it will call the service you specify here, for the service-multi-event-handler. So this'll be the name of the service that it will call for each row in the service-multi form.

00:36:57 screen changes to OFBiz: Features For: Giant Widget [ID:WG-9943]

That can create very nice user interfaces where if they want to edit a bunch of this stuff all at once they can just go through and edit, like changing the ordering of these, for example if I wanted to reverse the order, and therefore change all of them. Let's go ahead and check all over here. 4, 3, 2, 1. So if I want to do a change to multiple rows like that, we had black first and four-wheel last. I click update, and it flips the order so that 4-wheel is first and black is last.

If you want to edit more than one row at a time, that can be a powerful feature to enhance the user experience. And that's basically how it would be used with the Form Widget and the service-multi-event-handler.

00:37:56 screen changes to Framework Quick Reference Book Page 10

So those are the multi-type form specific attributes. For the next, let's carry on the with next attributes. The extends attribute is so a form can extend from another form. It basically inherits the general information, form attributes, but most importantly it inherits all of the field definitions from that form; they're basically copied over into the current form. This can be helpful when you want a form that's similar to a stock form, like an out of the box form in OFBiz, but you want some variations on it.

You might want to remove some fields, in which case you would use the ignore-field-type to override the field-type from the form that you extended from. Or you can change fields, basically you can override anything you want on them. And it's fairly granular, so you can change it from a lookup window to a dropdown box. Or you can just change the title on it and leave the widget side of it, whether it be a text entry box or a drowpdown or a lookup window or whatever, and leave that the same, just override the title.

Various things like that can be done with the Form Widget, which is kind of how it works out of the box. When it runs into a field with the same name as an already existing field it just overrides whatever is specified on the new field element.

We have extends, and going along with that extends-resource. In the extends attribute you'd put the name of the form. If there is no extends-resource it assumes it's in the same file, otherwise it looks in the location specified in the extends-resource, which will typically be a component:// type location to find the named form to extend.

Okay we talked about the separate-columns = true/false. This is for type lists and multi forms, if you want the editable fields to all be put into a single column or be separated into multiple columns. By default this is false, so they're all put into a single column, which we call the form column, corresponding with this form-title-area, form-widget-area. Just to review that real quick.

But you can separate them to individual columns by selecting true. That can end up resulting in very wide forms, but if you don't have such busy forms then it can be a very nice option to make the user interface look a little bit cleaner.

Okay the view-size, this goes along with the pagination stuff up there, it should be organized in with those, and basically specifies the number of rows to be put in each page. Row-count and use-row-submit are the multi form specific attributes that we talked about before.

Skip-start and skip-end, and also hide-header, these go together. By default they are false. They can be used if you want to have a single form split into multiple forms, if that makes sense, so a single form in the user interface split into multiple form definitions.

In your screen definition you can just include the forms back to back. On the second one you would skip-start = true so it doesn't render the opening form element. On the first one you would say skip-end = true so it doesn't render the close form element. The first form will render the open form element only and not the close one, and the second one will just render the close form element and not the open one. So they can be submitted together. That way you can create forms that are more complex.

In between them you can do other forms where skip-end and skip-start are both true, or you can insert custom templates or that sort of thing, like you could insert the html specific like a FreeMarker template in between them to get special forms fields. The hide-header goes along with that. This is for type-list and multi forms that have a header across the top.

00:42:41 screen changes to OFBiz: Features For: Giant Widget [ID:WG-9943]

Basically this guy up here, this header row. If you set hide-header = true then it will just not render that row.

00:42:52 screen changes to Framework Quick Reference Book Page 10

This is typically used along with the skip-start and skip-end attributes, but you can use it whenever you want, if you do not want it to display a header row.

Okay odd-row-style and even-row-style. These are nice things if you want to have different background colors, like alternating background colors. If you do then you can specify an odd-row-style and an even-row-style to set those colors, and to go along with those a header-row-style so that the header can have a different background color or whatever.

The default-table-style more accurately could be called the table-style, because it's explicitly the table-style. This would basically be the table that wraps around the entire thing.

00:43:45 screen changes to OFBiz: Features For: Giant Widget [ID:WG-9943]

So the table that wraps around all of this for example, from corner to corner.

00:43:53 screen changes to Framework Quick Reference Book Page 10

So that's it for the form element attributes. That's not so bad is it?

As you can see, different attributes in many cases are only applicable for different types of forms, depending on whether it's a single, list, or multi, specifically with many of them being for either the list-type form or the multi-type form.

Okay no sub elements. You can have actions for a form that run before the form is rendered to prepare data and such. Alt-target for using alternate targets in different conditions. This is one of the things we use that's helpful for this pattern of in an edit form supporting both the create and the update, so we'll look at that in detail.

Automatically creating fields from a service, automatically creating fields from an entity, explicitly specifying fields or overriding already defined fields, and then sort-order allows you to, regardless of where the fields came from, the default-sort-order would be the order the fields are defined in. But you can specify the sort order for the fields using the sort-order element.

One of the cool things you can do with the sort-order is you can put banners in between them, labels for different sections in the sort-order. Let's talk about some of these high level tags in more detail, and then we'll go and look at the field definition tags in more detail. That's the more powerful and rich part of the Form Widget.

Okay so alt-target is here, you simply specify a use-when tag. This is basically a BeanShell snippet, that's basically true for the use-when tag here and everywhere else you see it; each field has a use-when tag that you can optionally put on it. So you can have different fields, even with the same name, that are used in different circumstances.

Alt-target is just a use-when and a target. If the use-when is true, then this target will replace the target attribute, the default target basically, and the target attribute up here on the form element. If no alt-targets are found that apply then it just uses the target up there on the form element.

So that's fairly simple. While we're looking at this let's look at an example where we have a create and update in the same form.

00:46:57 screen changes to ProductForms.xml

This is actually fairly common. Let's look at the editProduct form for example. This is a pretty big form that is fairly manually specified. There are no auto fields used in this case, we just manually specify every field and what it should be, a dropdown, a text entry box, or whatever.

Here's the pattern for doing a create and an update in the same form. In the target we'll typically have the update target there. For an alt-target you'll have some sort of a primary map, and this corresponds to the default map name on the form. This is commonly the case.

When that map is null, instead of doing an updateProduct we'll go to the createProduct target, because we know we're not updating an existing product we're creating a new one. This is a special field in here that's passed around because there are certain complications with this screen, in the flow. So we tell it explicitly that this is a create, in the case that the product is null. This field will only be included in the form if the product is null, that's the is-create field.

Here's another field that has different alternatives based on what's going on, what data is passed into the context. So we have 3 different options for this productId field. The first one is if the product is not null, we know it's an update update. So we display the productId and we have a tooltip here that's an internationalized label that basically says we can't change it without recreating the product.

In the case where the product is null, in other words we're doing a create, and the productId is null, in other words no productId was passed in, then we show a text box that allows them to enter a productId. Now with the way the product services work and such for creating and updating a product, or in this case creating a product, if you specify an Id it will use that Id. It will validate that Id first, and then use it if it's valid. If you don't specify an Id it will sequence one automatically. This basically allows you to enter an Id with this pattern in the form.

This is the third scenario if the product is null. So we know we're doing a create, but the productId is not equal to null, so we know there's something strange going on. In other words a productId was specified but not found. So we show a little tooltip that tells them that the productId they specified was not found, and then basically the user interface will show a form so they can create a new productId. It will fill that in to text box so they can use that productId to create a new product, or they can change it to whatever Id they want and such.

So that's the typical pattern that's used. The rest of the fields can be exactly the same when doing a combined create and update form. It's just the primary key or identifier field or fields that need to have these special cases, depending on some sort of object like this, that you can use to determine whether you're doing a create or update. And then we'll also have an alt-target that goes along with it, so we can tell it to do a create instead of an update. That's where that's used.

00:51:03 screen changes to Framework Quick Reference Book Page 10

Okay moving on, the auto-fields-service and auto-fields-entity attributes are very similar. They automatically declare fields according to, in this case a service definition, and in this case an entity definition. If you specify a map-name here on either of these then it will use that map-name for each of the fields that it created. So this is basically an alternate to explicitly declaring each field.

There's also a default-field-type that's basically the same for both of these. So when you do it based on a service it's going to go through all of the incoming service attributes only. And based on the field type it will create a field for those, either for editing them, for finding them, for displaying them, or they can all be hidden fields.

When you're doing an auto-field-service the most common one to run into is the edit, which is the default. Because usually the intention of the form that will be submitted to a specific service is to edit it. And when you have an edit form, you'll typically want to use the auto-fields-service and not the auto-fields-entity, because what you want in an edit form is all of the fields that correspond to the input fields in a service.

So it's very natural to use that. Auto-fields-entity will go through the entity definition and create a field for each entity-field. With one exception; there's some automatically created entity fields that the entity engine uses for internal information purposes, namely the created-timestamp, the created-transaction-timestamp, the updated-timestamp, and the updated-transaction-timestamp. Those will be excluded in the auto-fields-entity because those are automatically maintained by the entity engine and never really user writeable.

Okay so with an auto-fields-entity it's more common that you'll use a find or display type, though with either one you can use a hidden type. Of course with either one you can use any of the types. But it's more common for the auto-field-service to use the edit type, as we talked about, and the auto-fields-entity to use a find or display type.

So if you're displaying the results of a query you can just use auto-fields-entity with a display type and a list form to lay them out, can automatically paginate them for you and all sorts of nice things like that.

The find type. When we look at the field types we'll see a number of field types that are explicitly meant for finding, or doing finds, so they have things like begins-with or contains or ends-with. They also ignore-case, those are for text fields, and for dates you can do a date range like before or after. Or within a date range, for numbers you can do various options like that.

So this is kind of similar to the old QBE or Query by Example thing, where you're just specifying the constraints for the query in the form. So find-forms will typically use the find-fieldtypes, and you can do that with auto-fields-entity or auto-fields-service, using this find thing.

00:55:24 screen changes to Category Members For: Giant Widget [ID:WG-9943]

Let's look at a quick example of that. In these lookup windows, it's very common that at the top we'll have a number of find fields. This is a find form, and there at the bottom we'll have a display form. If I say the productId begins with WG, and we'll ignore case, and search for it, then I get all the WG- products, which are the widget products.

So this is a display form, with the exception over here where we have a hyperlink. Over here, and this is a conditionally used field only when the product in the list is a virtual product, we have a link to go to the variants.

So as is the pattern with this lookup window, we'll be looking at this lookup field types. So with the lookup window pattern if we select one of these it will typically pop us back to the field, and insert the Id that we selected there. If I go select a different one here, if you hit lookup then it will often do a complete find.

And here we can see an example of the pagination of the Form Widget; 1-20 of 33, and it will go to 21-33 of 33. If I do one of these other things like a standard room then it inserts that product Id here. So this is the little product jump form that hops over to the quick admin page or whatever for a product.

00:57:22 screen changes to Framework Quick Reference Book Page 10

Okay so those are the find fields, and we'll be looking at those in more detail in a bit. Okay so the sort-order, there's only one type right now of sort-order, the explicit-sort-order. And that's just a placeholder for possible future functionality. Basically what you do is you just specify a number of these sort-field elements, and the name of the field for each one. I think there's something missing for this. And that's the banner.

00:58:09 screen changes to WorkEffortForms.xml

I remember one that I was looking at recently. Let's look at an example, workEffortForms. Okay so here's an example of where this can be useful, we have a large form, the editWorkEffort form. It is automatically pulling in fields from a service definition. When you do that basically the order of the fields will be whatever they were in that service definition; how the attributes were defined there. So it does preserve that order.

But if you want to change that order you can do that using the is-sort-order thing, so we just have a whole bunch of fields. In this case it's just doing fields. I believe there are also some down here...here we go.

This is another nice thing you can do with the sort-order, specify fields and also specify banners that go in between the fields. This will basically just be a text, like a text row for displaying things. It has left-text, and left-text-style. Okay so this has left-text and left-text-style, or right-text and right-text-style, or general-text and text-style.

So these banners can be used in a few different ways, different options for where to put the text by these, and the style to use on them. Let's look real quick and I'll show you how this form renders.

01:00:24 screen changes to OFBiz: WorkEffort Manager: Edit Work Effort

This is over in the WorkEffort Manager, and when we have a WorkEffort, let's see the Test Parent WorkEffort for example, and we look at the children, we have our little tree here. If we add a new child, we get this sort of thing where we have the fields that are sorted here. Now the reason the sort order is valuable in this case is because it's pulling these fields from multiple forms, all of these fields down here are just reused from the editWorkEffort form.

And then we're adding these other fields and then putting in some banners to describe how these should be used. So these are left-text, and if you did right-text it would be over in the right column here, or just text would go all the way across. That's how the banner works with the sort order.

01:01:27 screen changes to Framework Quick Reference Book Page 10

Okay back to the reference. We've now finished these high level fields for the Form Widget.

Part 3 Section 2.2.2-B

The Form Widget: Form Widget Fields

Starting screen: Framework Quick Reference Book Page 11

Let's hop down to the next page, page 11, and let's look at the field element and all of the field attributes. Okay so these are all of the different types of fields. The type of the field is basically represented by a sub element that goes under the field element. Here's the field, a sub element in this group, in the all-fields group, should go underneath it.

We'll talk about each of the types in a bit, but first let's talk about the field elements. We talked about a number of these actually in the context of the form element, because it has default values for some of these things. Each field will have a name, and as I mentioned before if you define a field for a given name that has already been defined in that form, then it just does an override behavior. Whatever information you specify will override the information on the previous field definition.

So that can be useful for when you are extending another form, or if you are doing an auto-fields-service or auto-fields entity and you have an automatically created field. Let's say you want to override things on it like the title, or the type of field to make it a dropdown instead of just a text entry box, which is the normal default for the autofields, depending on the entity-field-type or the service-attribute-type and such.

Okay then there were some that we looked at on the form element, like the default-map-name, default-entity-name, and entity-service name. I mentioned that each field can have a map name to look in for its value instead of looking in the global context, and also an entity name, a field name in that entity that it corresponds to, an attribute name, and a service name so that you can specify a service attribute that it corresponds to.

These are mostly used for auto-fields-entity and auto-fields-service. They'll be populated by those automatically, and then those are used when it's figuring out the type of field to create by default for it. The entity-name will basically be used instead of a field name if you're doing a map name or that sort of thing. The parameter name will be used instead of the field-name for the actual form-parameter name.

These are not used very commonly, entry-name and parameter-name, you just specify the field-name and that's used for both of those situations. But you can override them, the entry-name and parameter-name, as needed. So these will be the parameter-name in the form that's actually defined, so the HTML form name.

Okay the title, which is basically just like we talked about there. A title and a tooltip go together to describe the field to the user. The title will be in the left column if it's a single form, or if it's a list or multi form it will be in the header row. The tooltip will be in the widget area to describe hints about how to use that.

The title-area-style. We looked at the default-title-area-style element, the default-widget-area-style, default-title-style, default-widget-style, and default-tooltip-style. These all correspond to those attributes, along with the

required-field-style. So they're attributes with a default prefix that correspond to all of these on the form element we talked about.

This style is the area the title is in, and styles the area that the widget's in, styles the title itself, styles the widget itself, the widget being whatever the form element is, check box, display, dropdown, radiobutton, lookup box, submit button, whatever.

Tooltip-style will style the tooltip, the little comment that goes to the side that we looked at. If required-field is true it'll put a little star by the field so you can specify for each field whether it's required or not, then the required-field-style will be the style used on that. So those are the styling attributes here, let's go back and get the ones we missed.

Header-link and header-link-style go together. We specify a header-link, this is basically the header-link target, and the header-link-style would be the visual style that's applied to the header-link, usually something like link-text or something that's one of the common OFBiz conventions.

00:06:07 screen changes to OFBiz: Catalog Manager: Edit Product Categories

So the header-link if specified, especially for list forms, is mostly used to put a link up here in the header, and this would be used for things like sorting buttons or that sort of thing. That's the most common purpose for those.

00:06:28 screen changes to Framework Quick Reference Book Page 11

So that's how that sort of thing can be done. Position attribute by default is one, how this behaves will depend on whether it is a single form or a list form.

00:06:51 screen changes to OFBiz: Catalog Manager: Edit Product

In a single form we actually have some examples here like the Edit Product, where position 1 is basically the left column, position 2 is basically the right column. It will go through and find the number of positions and then create columns corresponding to those. If there's something in a position 1 and there's no position 2, then it will allow the position one field to stretch across the other columns, like this comments field here that's a bit wider.

If there is something in a position 1 and a position 2 then basically the position 2 stuff just goes into a second column over here. So we have a lot of little fields in the product form, and those are split out into two columns so that it's not one huge long list of fields. That makes it more user friendly, and though it is a little busier, it's much easier to use.

You can have fields that are related to one another like the product-height and the height-uom next to each other. That enhances usability and in general allows you to better organize the form. So that's what the position attribute does.

00:08:11 screen changes to Framework Quick Reference Book Page 11

Okay red-when will turn the widget red in different circumstances. Before-now is commonly used for dates, actually mostly use for dates, so before-now or after-now you can tell it to be red in those cases. By default it's by-name, so the Form Widget will, unless you specify never, automatically turn the widget red like a date or even a date entry box, according to the name.

There are a few special cases based on a few common conventions in OFBiz like the from-date and through-date, which are the most common ones. If from-date is the name of the field then it will turn it red when the from-date is after-now. If the from-date is in the future, hasn't been reached yet, then it will turn it red. If a thru-date is before-now, in other words if a thru-date is in the past, then it will turn it red there too.

00:09:37 screen changes to OFBiz: Catalog Manager: Edit Product Prices

So let me show you what that looks like. In the user interface, just hop to any of these screens. We'll specify a from-date here, which is basically now, so as soon as this refreshes it will turn it red because it's in the past. This is a thru-date and it's in the past, so it will turn it red like that. So that's the red-when attribute.

00:10:01 screen changes to Framework Quick Reference Book Page 11

Okay the use-when attribute. We saw this in the context of the pattern for using a single form definition for both create and update, so just to reiterate what this means. It's basically just a snippet. We actually use BeanShell to evaluate these snippets; it needs to be a boolean expression that evaluates to a true or false. And it will basically determine whether this field should be used or not.

You can have multiple fields with the same names that have different use-when attributes, and of course those will be only used when their corresponding logical expression is true. So it's good when you have multiple fields with the same name with different use-when expressions, to have the use-when expressions to be mutually exclusive so that only one of them will evaluate to true. If you don't then the last one on the list I believe is the one that will be used.

Okay event and action. These can correspond to events and actions on the widget tag in HTML that is eventually generated. So the event would be things like on-click or on-focus or on-blur, and action would be

the method to run. In the case of an HTML user interface this would typically be a Javascript method call, or it could even be a little Javascript snippet, so that you can have additional logic associated with these form fields more easily. Okay and that is pretty much it for the form field attributes.

In the next section we'll be talking about all of the form field types. This is the first set of field types; check box is very simple, has no attributes or additional information associated with it, it is just a check box. When you specify this it will just create a check box. I don't know if there's really a whole lot more to say about that, it's pretty simple.

Okay date-find corresponds with date-time. You'll see a number of these elements that have the find suffix, and we looked at an example find form when we looked at the product lookup. So basically with the date-find it will do a date range type thing.

The attributes here are actually the same as with the date-time; you can specify the type, by default it's a timestamp, which means both date and time, or you can specify just date or just time. You can default-value hard code a default value for this. It can be just a hard coded string, or you can use the flexible syntax, the ${ }, to insert the value of a variable there.

Okay display just causes the value of a field to be displayed. You can tell it to also hide it (also-hidden) or not. By default it does also create a hidden form field so that the value of the field will also be submitted to the server when the form is submitted, in addition to being displayed to the user. It's still submitted to the server when the form is submitted, but the user can't change it. that's basically the effect you get with that. If you don't want it to be sent to the server just set also-hidden to false.

The description attribute is optional. If you don't use the description attribute it will just display the value of the field. If you do use the description attribute you can have it display other information; string constants, internationalized labels, as well as the value of the field. Or in fact you could just throw out the value of the field and have it display or whatever. So that's basically what you can do with the description. Its' very common to use the ${ } and such for expanding variables with the description there.

Okay display-entity is a variation on display. This is kind of a nice thing where, if you have an Id field, rather than displaying the value of the Id, we can actually look up, like if it's a type field, like a price-type for example. Instead of showing the price-type-id, which we typically don't want to do, we want to show the description on the price type.

00:16:01 screen changes to OFBiz Catalog Manager: Edit Product Prices

This is the product prices screen and the price type column. We're using display-entity to look up the description for each of these price types. We're doing the same thing for each of the purchases and for each of the currencies, and actually even for the store groups.

00:16:29 screen changes to ProductForms.xml

If we look at this form definition, and we can throw in some examples to make this a little less boring for you. Here in update product price we have a whole bunch of these display-entities. You can see using this, especially if the field name corresponds with the primary key field name on the entity you want to display the description of, and the text you want to display is on the description field. That's the default for the display-entity, to show what is in the description field.

So the productPriceType entity has a productPriceTypeId to identify each type, and it has a description field to with the text to describe each type. All we have to do is say display-entity and the name of the entity, and it will figure the rest out. Now if we want to display something a little in addition to the description on it, or as an alternative to it, here's a little more complicated case.

In this case currencyUomId, this is the currency unit of measure, is different from the key field on the entity we want to display, so we need to specify the key-field-name as well as the entity-name. And then in the description, this does have a description field we want to display, but in addition to that we want to display the uomId, the Id of the unit of measure. It's common when we're displaying an Id like that to put the [ ] around it, that's just one of the many conventions that you'll run into in the OFBiz user interfaces. So that's how we get the description and the uomId.

00:18:15 screen changes to OFBiz: Catalog Manager: Edit Product Prices

Out here when we're looking at this we have the description of the currency and the uomId, euros in this case. So that can be kind of a nice little feature, a shortcut for getting those sorts of descriptions in place.

You'll notice that when we're using display-entity it also has an also-hidden tag that by default is true. Typically when you have an Id that you're displaying a description of from another entity, it is an Id that you'll want to submit with a form. Especially if it's one of the primary key Ids, then you definitely want to submit it.

Cache by default is true, it will cache the lookup. Most of the time when you want to display a description that corresponds to an Id like a typeId, or one of the enumerations or statusItems or that sort of thing, the same record will be used over and over and over. So it makes good sense; it will increase performance without a whole lot of memory overhead to cache those values.

And it does that by default, so if you don't want it to cache it just set that to false. One of the cool things you can do with the display entity is sub-hyperlink. In addition to displaying the stuff you can also drop a hyperlink into there. You can do that with a lot of the other field types, and you'll see this in various places; drop-down, sub-hyperlink-file, and sub-hyperlink. Hyperlink doesn't have a sub-hyperlink, that might be interesting. Image has a sub-hyperlink, lookup has a sub-hyperlink, and so on.

00:20:11 screen changes to OFBiz: Catalog Manager: Edit Product Prices

We saw this actually in the example application in the Framework Introduction. Maybe we should just go back there and look at it. If you have an example, and we look at features...let's go ahead and create a new feature.

Okay if we look at the examples related to this we can add a test example. Here's an example of a display entity. This example just has the exampleId, and it's looking up the description of the example using the display-entity tag. Then it has a sub-hyperlink to point us over to edit that example if we want to.

So that's a fairly common pattern to do that sort of thing. And going in the other direction we have the same thing, display entity with the sub-hyperlink.

00:21:17 screen changes to Framework Quick Reference Book Page 11

And that's a common thing to run into. With the sub-hyperlink it's very much like the hyperlink field over here. You can put a conditional use-when on it if you want, or a link-style, this will typically be something like button text, and in the example forms we just saw it as button text. And you can also put a little wrapper around it, or whatever style you're using it's a custom style in your application. And then we have this common pattern where we have a target that goes along with the target-type.

These two attributes are commonly found together, target and target-type, and even target-window is often found in there too. I guess we've gone over this a few times.

Default with intra-app, target is just the name of the request. We can go to another app, to an OFBiz content URL or just a general URL. The description in the case of the sub-hyperlink is the text that we use in the link itself.

The target-type and target, the description is the same as on the sub-hyperlink, that's the text that will be shown there. And the target-window, if you want to do the _blank thing or whatever you can use that. This pattern with the target, target-type and target-window, that's similar to what we saw on the Form Widget, links in the screen widget, and all sorts of stuff. It's a very, very common pattern.

Okay going back down through our natural, or unnatural but alphabetical progression, dropdown is next. Dropdown will create just a dropdown box. There are different ways to populate the dropdown box, and we'll look at those in a sec.

Okay while we're going over that we might as well do the hyperlink one real quick, since we're here. This has an also-hidden flag like many of these different field types do. Any of them that displays something, that doesn't have something that would be edited and naturally submitted, would allow you to have it passed in as hidden or not.

By default we do not allow it to be empty; if you set allow-empty to be true, basically it will just add an empty element at the top of the list. There are two styles that are supported here for telling the user which is the currently selected element. This would be for a field if it already has a value. By default it does this first-in-list thing. This is kind of a nice pattern so even if they change the selection and come back to the dropdown, they can still see which was the originally selected item. That's why we use that one by default. You can also have it just be selected, rather than copying the currently selected item to the first-in-list.

So let's look at this first-in-list pattern real quick, just to make sure it's clear what we're doing. These are all create ones. Let's go back to the catalog manager and the product we were looking at, the large widget here.

00:24:45 screen changes to OFBiz: Product For: Giant Widget [ID:WG-9943

So this for example is a first-in-list. The product type is finished good, so it puts that first in list, even though that's found here as well. And it puts this little divider between the current one and the other options, so that it's very easy to see which was the current or previously selected option.

The nice thing is if I select another one I can still go and hit the dropdown and say oh yeah, that was a finished good, maybe we'll change it back before submitting a form. The divider if that one is selected will basically keep it at this value, the original value which is finished good in this case.

One of the cool things that it can do with this first-in-list style is it will automatically, based on the current productTypeId for example in this case, look through the list of options, find the one that matches the current id, and then use that description up here at the top. And again it'll do that for you automatically.

Now there are certain cases like the status one that we looked at with the Framework Introduction videos, and the example where the first in the list, with the first-in-

list the current value. If it's not down there then you need to do a little bit extra to get it there. I won't cover that again; you've already seen it I guess.

00:26:15 screen changes to Framework Quick Reference Book Page 11

Okay other things about this. No-current-selected-key, if there is no current value by default it will select the first of the options, or you can explicitly specify which of the options will be the default. Or in other words the one to have selected when there's no current selected, the key to use to select one of them.

Current-description is for that case where the current item is not in the choice to change to, which is the case for the statusValidChange pattern. We can specify the description of the current element to use, and this will usually be the flexible string expander, the ${ }, to insert a variable value. Okay these are the different ways of getting options into the dropdwon. You can specify any combinations of these tags, and any number of each of them.

So entity-options pull the options from the database, list-options pull the options from a list that's already in memory, and the option tag basically explicitly specifies an option. Let's look at the entity-options; it's right here in the field sub-elements box.

We'll specify the entity name, that's always required. So this is the name to query on to get the records. Key-field-name is optional. If you're using a dropdown, and the field-name matches the key-field-name on the entity, then you don't have to specify this. Otherwise if they're different then you do need to specify the key-field-name. If this is a product type dropdown, and the field somehow is not product-type-id, you'll specify the key-field-name as product-type-id.

The description will be the description of each element in the dropdown, and you do have to specify what that will be. Usually this will use the flexible string expander, and the context for this will be each entity record that's looked up, as well as the context of the form. If you specify a ${ } string expansion variable here, and it is not one of the fields in the entity, then it will look it up in the larger context of the form.

You can put internationalized labels or any other information here in the dropdown as well. You can turn off the cache by saying cache=false. Filter-by-date, it will do that by name. In other words if the entity you are doing the dropdown options from has a from-date in the thru-date, it will filter by those. So it will filter out any records that have a from-date in the future or a thru-date in the past. It will do that automatically if it sees the from-date or thru-date fields. So that's the by-name, just like the red-when we saw over here, how it does the by-name thing, the from-date, or the thru-date. So if you don't want it to do that you can turn it off.

When doing the entity options you have an entity-constraint, zero to many of those. If there are no constraints it will return all the records in the table. This is very similar when doing the entity-condition, that action that we looked at a while back, the condition-expression, that's very similar to that.

So we have name, this is the field-name. Operator, how to compare it to whatever we're comparing it to. We have an env-name and a value, so if you want to put in a string constant you can drop it into the value here. If you want to use a variable you can use the env-name to specify the variable to prepare the field to in the query.

And just like with the condition expression we have ignore-if-null and ignore-if-empty. So if the env-name is pointing to a variable that is null or empty, depending on which one of these you specify, then it will just ignore the constraint. If you don't specify the ignore-if-null and ignore-if-empty, and it does point to an empty value or a null value, then it will put that into the condition. If that's what you want then that's exactly what it should do. If you want to do an equals-null or an equals-empty-string or something, then you can certainly do that. Okay so that's the entity constraint.

Entity-order-by will order the options, which is very helpful in dropdowns. Usually you will order by the same thing that's in the description, typically the first field that's in the description. If there are multiple fields then perhaps each of those you'll order by, so when the user looks at it it's in a natural order. For the order-by is specify the field name to order by. You can specify zero to many of those. Okay so that's it for the entity options.

The list-options. This is pulling in options for the dropdown from a list. You have a list name, which is the name of the object in the context that has the list in it. List-entry-name is the name for each entry in the list to put in the context, or if nothing is specified here then it will assume it's a map and put that in the context. The context in this case is what we'll be using to expand the description string, just like the context for the entity-options is the entity generic value that was looked up.

That's the main context here, the list entry will be expanded into the form context or put into the entry that you specify. If the list is actually a map, this is another case where we'll use the same element for lists and maps, and you can use it for other collection things as well like sets and stuff. If it is a map then you can also put the value of the key for each entry into the context by specifying a key name.

So then the description will basically, by default if you don't specify it, will just take the list-entry-name and drop that in. Otherwise you should specify exactly what the format is that you want to be displayed in the drop-

down. And that typically uses the the flexible string expander.

Okay so the last way of getting options into a dropdown is to explicitly specify them. You can do that with the option tag, basically just have a key which is required, and a description which is optional. If you don't specify the description the key will be used, that's the text that will show up in the dropdown. The description will show up in the dropdown if specified, if not the key will show up in the dropdown. And the key will always be used to pass back to the server. Okay that's it for the dropdown element.

The file element is used for uploading a file, and corresponds to that type of field element in a HTML. You can specify the size of the text entry box, which by default is 25. You may want more than that, and chances are you wouldn't want less. Max length would be the maximum length of the field. You don't have to specify this, it's very common not to actually. And the default value. You can put in the default value, but it's not terribly common. You'll have a default file-name where you'll predict the name of the file on the client computer but you could do that.

Okay sub-hyperlink. Just like in all other cases where we have a sub-hyperlink, you can put one by the File Widget. So hidden will create a hidden element that is not shown to the user but is passed to the server. With the hidden type field you can specify a value. This is optional, so if you don't specify a value it will just use the value of the field for the value, which is fairly natural. In some cases you will want to do that, and that's of course the value of the field that's sitting in the context, or sitting in the map if you have a map name associated with the field.

Now if there is nothing, if you just want to pass a static value, a constant to the server, you can do a hidden with a name on the field, and the value right here in this thing. You can also use the flexible string expander to drop in variables and such. This is basically a way of passing things to the server without the user seeing them.

The hyperlink we discussed when talking about the sub-hyperlink.

Okay ignored basically means do nothing with the field. If you're declaring a field initially it's fairly silly to say here's the field, here's the name, here's the information about it, and oh by the way ignore it. But it's very useful if you're overriding an existing field. Say you're extending another form, or you have an auto-fields-service or an auto-fields-entity you, can just override the field by doing the field, and make sure you have the name in there, and then ignore it. And then it will not be included in the form when its' rendered.

Okay image will display an image. You can put a border on it and specify the width and height, and the value would be the image URL. You can do a default-value if it's not there. As with many other fields you can do a sub-hyperlink by it.

Okay lookup. This is an interesting one. You basically specify the target-form-name. Which is really the name of the request to use to pop up the lookup window. The size will be the size of the text box to enter the Id value in. Usually a lookup window is used to look up an Id value, like a look up a productId or look up an orderId. We saw an example just a bit ago where we had a lookup window for the productId.

Size maxlength, you can do a maximum length on the lookup box, and a default-value. The sub-hyperlink here, just like in many places. One of the cool things you can do with a lookup; since it does represent a round trip to the server, you can have multiple fields that are looked up where one lookup depends on another one. This is a little bit funky but it's something that can be useful.

The default-value, maybe I shouldn't mention it there, since it's not really very easy to do with the Form Widget. You have to throw in some javascript stuff, so maybe just forget I said that.

00:39:14 screen changes to OFBiz: Catalog Manager: Edit Product

It is possible to do that. But this is most commonly used in cases like this where we have a little text box, for the productId in this case, and we have this little icon that represents the lookup window. So we click on that and it pops up the lookup window and so you can query and find the thing you're looking for.

It's very common to run into these; you'll see them all over the place, over in the content page if you want to add a content by Id, use this lookup window to look up a contentId by name, description, or whatever. Click on it it inserts the Id back in here. That's the common pattern, and it's a very handy thing to have in a user interface.

00:39:51 screen changes to Framework Quick Reference Book Page 11

Okay the password field type is just like the password field type in HTML, you can do a size and a maxLenth in a text entry box. You can specify a default-value; assuming that the entry in the context is null or empty, then a default-value can be used. Typically though for a password you don't want to have a default-value but you might at some point.

With a password you can also have a sub-hyperlink, maybe to change the password or for whatever reason. So the password basically is when you type onto it, it's just a text entry box, but when you type into it it shows

stars instead of showing the actual characters. That's a common sort of thing.

Okay radio-button is actually a lot like the dropdown. It has a no-current-selected-key just like the dropdown does. You can have a bunch of different options, use any combination of these just like a dropdown. They're the exact same tags used in the exact same way; entity-options, list-options, and manual explicit options. So basically when you do a radio-button instead of a dropdown it will just put a bunch of radio-button items into the user interface instead of a dropdown.

That can be very helpful for cases where you have a smaller number of options in the dropdown, and want to display them all at once without the user having to click, or you can save a click basically when they're going through a form.

Okay range-find is another one of these find-field-types, like the date-find and other things. You'll see some other ones like the text-find. Range-find is for finding something within a range, so it will show like a maximum and a minimum-value. Other than that it's just a normal text entry box.

As far as these attributes go you can drop a default-value and put a hyperlink next to it. Fairly simple things. Reset will create a reset button that clears out the form, just like the reset-type button. Type input in html.

Submit is a button. There are a few types; you can do a button, a text-link, or an image to submit the form. If it's an image you specify the image-location, which is just the URL with the image. I should note by the way, when doing a button or a text-link , or for any of these things when you want to put the style on it, the styles are not over here in the widgets because they pretty much always have styles, so you just use the widget style over here. That's how you'd style the button, that's how you would style the text box, that's how you would style everything.

Okay so the text box is the next one. It basically has a size, a maxlength, a default-value like some of the others that we've looked at, and a sub-hyperlink that you can put next to the text box. TextArea is like text box, but it basically corresponds, so this would be a text input in HTML. This is a textArea in HTML. They'll typically be wider and have a number of rows, so by default we're saying we have sixty columns and three rows.

Of course you can specify any size of box there, and you can put a default-value in it. It can be manually entered string, just a constant sitting there, or it can be variable, as a flexible string expander to drop it in. Okay text-find is like the other find ones. This is a text specific find.

00:44:16 screen changes to https://localhost:843-Lookup Content

This is the one that we were seeing over here. Let's see what we have in this one. So here's a text find that has the equals, begins-with, contains, and ignore-case. This is kind of fun. We have a popup window in a popup window. I don't know if that works, it's kind of silly. In fact I don't know if that's intentional, it might be a bug, so just ignore it. Anyway that's what the text find box looks like.

00:44:45 screen changes to Framework Quick Reference Book Page 11

And that's it for the Form Widget field types. And that also means that, sadly, we've come to the close of Form Widget coverage here.

00:45:01 screen changes to Advanced Framework Training Outline

We talked about the general concept of form definitions and form rendering. This is the schema file. One thing to note about the schema file is that it has a fair number of inline comments, and we're working on increasing those in various areas here.

00:45:19 screen changes to ProductForms.xml

So it's very helpful to have an XML editor that can do auto-completion. Like here for example. This is the Oxygen XML Editor, and I can popup things like this. If I do a field, we look at other attributes, we have annotations in place that will pop up as you're looking at attributes or elements, tags that have these annotations, basically inline documentation associated with it. Quite a few of these, especially in the Form Widget, have this kind of stuff. In your editor you can see descriptions, basically a lot of the information that we covered here. You can see it to remind you inline of what these different things mean. And those are all things that are in the schema file, so it's very helpful to have a schema aware XML editor. That can be extremely valuable, in fact.

00:46:34 screen changes to Advanced Framework Training Outline

Not only can it do auto-completion and validation on the fly of the XML that you're iterating, which is very helpful for these complex XML files, but they do have pretty good schemas that will simplify your task in dealing with those a lot.

Okay we looked at the form element, the different actions, those are in the shared elements, so again if you want to review those you can look at the shared actions of the Screen Widget video. This is just a little bit earlier of this part of 3 up here, Section Actions.

We talked about auto-field, when and how those can be used for different circumstances. The form fields and all the different types of form fields, and the pretty good set of options that's available with those. And the form sort order in the special case that you want to insert banners, or if you're extending a form or doing an auto-fields or something, then you can manually specify the sort order by just specifying the names of the fields in the order you want them to appear.

Okay so that's it for the Form Widget. Next we'll go onto Menu Tree Widgets and then we'll get into the fun stuff, BeanShell for data preparation, FreeMarker for templates, generating PDFs, XSL-FO, all sorts of fun coming up.

Part 3 Section 2.2.3

The Menu Widget

Starting screen: Advanced Framework Training Outline

Okay now continuing part three of the Advanced Framework training course. In this part we will be looking at the Menu Widget, and the Tree Widget. Both of these are done in XML files, just like the Screen Widget and the Form Widget, but have their particular purposes just like those other tools.

The Menu Widget is used for horizontal and can potentially be used for vertical or even in theory hierarchical menus. Although that has not been developed yet, you can see how the structure in the XML files supports that, and I'll talk about that as we get into the details.

The Tree Widget is typically database driven, although it can be driven by other things and we'll see how that works, and it renders an hierarchical display of elements, the definition of nodes and sub-nodes in a tree. Just like the other widgets they have XSD files.

Here's the widget-menu.xsd file and the widget-tree xsd file. These are in the same directory if you want to look at the actual source files, in the framework/widget/dtd directory. We have widget-form, widget-screen, widget-menu, and widget-tree.

The actual menu and tree definitions will typically go inside a given component in the widgets directory, either directly in that directory or in some sub directory of it, and the name of the file will be *menus.xml, and for trees *trees.xml. This also follows the same pattern as the Screen and Form Widgets where we have *screens.xml and *forms.xml.

Okay now the most important items for the Menu Widget are typically the menu-items, and for the Tree Widget the tree-nodes. Let's look at some examples of where these fit in.

00:02:57 screen changes to OFBiz: WorkEffort Manager: Edit Work Effort

In the work effort manager application, we're in the Work Effort tab over here, and looking at the edit work effort page for a given work effort. So we'll recognize some things that we've seen in other places. We have a form here for editing the work effort. We have the header, the various other things that are in the decorator of the screen. There are actually two places here where the Menu Widget is used.

00:03:29 screen changes to OFBiz: WorkEffort Manager: Work Efforts

And one place is this children tab, where the Tree Widget is used to show an hierarchy. We'll look more at that in just a minute.

00:03:35 screen changes to OFBiz: WorkEffort Manager: Edit Work Effort

So let's look at the examples of the Menu Widget first. So this is one place right up here, in this application header, a Menu Widget is used for that. In some of the applications you'll still see an FTL or FreeMarker file used for this application header. Another place the Menu Widget is used here is in this tab bar, to show different things related to a work effort.

Let's look at the definitions of these two menus. In both of these cases, by the way, the menu is included in the screen using the include-menu element that we've already gone over in the Screen Widget.

00:04:29 screen changes to WorkEffortMenus.xml

These are both defined in the same file, the workeffortmenus.xml file. That's in the work effort component, and of course in the widgets directory inside that component, which is the standard place to put it. These are the two menus that I mentioned, the workEffortAppBar and the workEffortTabBar. This one is used for the entire application, this one is just used for the work effort area.

The one for the application can be in any file. In this case it's combined with this other one, and these are the only two in this field. But that can, and often is, put in a separate file as well.

Okay, so some different things to note here. Of course we've seen that each menu has a name. When a menu is rendered, just like a screen or a form or a tree or anything, we refer to the file it is in, and then in this case the menu. So each menu will have a number of menu items, each menu item also has a name, and then various other things.

These ones are fairly minimal; each menu item as a name and a title. These titles are internationalized, so we see the ${ } and uiLabelMap pattern that is used for internationalization in various widget XML files, as well as FreeMarker files and other places.

This represents the label in the internationalization properties file that will be shown in the user interface. Each of these menu items are links, that's probably the most common type of menu item you'll see, and each one has a target. The target here is just like the target we saw in other places, where you can have a target type associated with it and such. Actually maybe you can't have a target type with these ones, you can have it point to a window. The URL mode of course.

So the default is intra-app inside the same application, that you can go into other applications: point to content, static content, or just a plain URL that won't be modified. So in the default intra-app mode this is just a name of a request in the controller.xml file, as are the rest of these.

When we look at this we'll see these different menu items, in the application bar in this case, which is what we're looking at right now. So these will be the different items in the header bar for the application.

We also have a logout and login items, and notice that these are used unconditionally. So this is saying that if user-login is not empty, in other words if there is a user-login object, or if a user is currently logged in, then we will show the logout tab. If user-login is empty, if there is no user-login object, so no user is logged in, we'll show the login tag. So logout is a just a constant request we can go to to log the user out.

The checklogin URL is a special variable that is populated by the screen widget to handle this sort of scenario, where you might want to do a login.

So let's look at how this actually ends up. Actually let's draw some parallels real quick; we have task, calender, going down to the login and logout on the other side.

00:08:51 screen changes to OFBiz: WorkEffort Manager: Edit Work Effort

So when we look at this here we have the tasklist, the calender, my time and so on, with the logout on the other side. And you'll notice this has an element highlighted; this is one of the features of the Menu Widget, whether you're using it for the application bar here or a tab bar like this. This has the edit work effort one highlighted, this has the work effort for the general work effort area of the application highlighted. That is done by setting variables in your screen definitions.

00:09:33 screen changes to WorkEffortMenus.xml

Here this selected-context-field-name is the name of the field to use in order that it will look at to see the name of the item that should be selected.

HeaderItem. If there's a variable called headerItem and it has the value task, that's the one that will be highlighted. In that screen we were just looking at work effort was highlighted, which means this headeritem variable has the string workeffort in it.

00:10:16 screen changes to WorkEffortScreens.xml

If we look at the WorkEffort screens we'll see that it corresponds with this. So editWorkEffort has a decorator, the common workeffort decorator. This is the more common place to define that sort of a variable so it can be shared by all of these screens that use the same decorator.

00:10:54 screen changes to CommonScreens.xml

Which file this is in. So the common work effort decorator, as we would expect, has a variable called headerItem that is set to WorkEffort.

00:11:09 screen changes to OFBiz: WorkEffort Manager: Edit Work Effort

So that's how we know which one up here to highlight.

00:11:20 screen changes to WorkEffortScreens.xml

The one here to highlight will use a different variable name, obviously, so they can be independent of each other. If we look at the workeffort screens we see a tab-button item, and the name is WorkEffort.

00:11:31 screen changes to WorkEffortMenus.xml

If we go look at the work effort menus down in the WorkEffort tab bar, the menu item with the name WorkEffort is the one that's highlighted. Notice that if there is no selected-menu-item-context-field-name attribute specified it defaults to this value, TabButtonItem. That's the default one to use. The reason for that is the most common application of this; the bulk of the menus that you will see are used for the tab bars.

00:12:08 screen changes to OFBiz: WorkEffort Manager: Edit Work Effort

For these guys, this kind of thing right here. Okay when you're declaring a menu and you're defining a menu each one of these has a style. Or you can have a default style for the elements and you can override the style per element. Typically you'll have a default style and a default selected style for each one. You want to have a different selected style so that the selected element will appear different from the rest of them.

00:12:40 screen changes to WorkEffortMenus.xml

So looking at the menu definitions, we see here the default widget style headerButtonLeft, and the selected style headerButtonLeftSelected. So all of these that do not have an explicit style specified will share or default to these two styles.

These ones, the login and logout buttons that are over on the right, have a different style. They'll have a line

on the left side of the button, instead of the right side of the button, so they can go on the right side of the bar. These have the headerButtonRight style, and when selected the headerButtonRightSelected style. So they both have those specified. They also have an align style, column right instead of left.

Okay other things to note. Here's the default align style, just call. These are some old styles that have been used in OFBiz for some time; they're called call and row. They're kind of very generic names but these are the styles that are used, the CSS styles is what they translate to in this web application.

00:14:09 screen changes to OFBiz: WorkEffort Manager: Edit Work Effort

These are the styles that are used for these different header tab or header links. These other tab button bars have a different set of styles.

00:14:24 screen changes to WorkEffortMenus.xml

So you'll see where we have tabButton as the default widget style and tabButtonSelected as the default-selected-style. Okay some other ones that are important to the orientation. Here is horizontal, as well as here. That is the default so you don't have to specify it. The type is simple for both of them, and again you don't have to specify that because that's the default.

00:14:59 screen changes to Framework Quick Reference Book Page 12

Now that we've kind of looked at the overview let's look at some of the details in the reference. One thing to note about the Menu and Tree widgets. As I mentioned before they were designed with a fair amount of flexibility, not all of which has been implemented.

So we have the simple style of menu and the cascading style, which is one that has not been implemented. You'll typically just use the simple type. Again the orientation; typically you'll just use the default horizontal because the vertical ones I do not believe have been implemented.

Another thing that hasn't really been implemented in here but is there for future use is this idea in a menu-item to have menu-items underneath it. This would be for hierarchical menus, like dropdown menus in an application and that sort of thing.

Okay the main elements that we've looked at that we need are title, we'll have the default-title-style and the default-widget-style. The widget is basically in this case wrapping the tile. These names are brought over from the Form Widget so you'll recognize them from there, although the context of how they are used in the Menu Widget can be somewhat confusing because the naming came from the Form Widget.

00:16:24 screen changes to WorkEffortMenus.xml

So let's look at the actual examples again real quick. We have the default widget style being the more important one. We don't really need a default title style for the tile of each menu item, unless the title is basically inside the widget so you can override the text or whatever, for the tile of each emnu item. So typically you'll see the widget-style and the selected-style specified, or correspondingly for the defaults the default-widget-style and default-selected-style.

Okay the menu-container-style is the one that goes around the entire menu. Default-align-style is the one used on each cell in the container within the container, basically a table cell I believe is how.

00:17:26 screen changes to OFBiz: WorkEffort Manager: Edit Work Effort

Actually no, these are all implemented with CSS, correct that. So this one, this type of menu, as well as this type of menu are both one hundred percent CSS, or Cascading Style Sheets. And that of course is specific to the HTML implementation or HTML rendering of the Menu Widget.

00:17:54 screen changes to Framework Quick Reference Book Page 12

Okay it's possible for a menu to inherit from another menu, either in the same file, if you just specify the extends or extends-resource to inherit from, or extend a menu in another file. The default-menu-item-name is important; if no other menu item is selected in the selected-menuitem-context-field-name, then it will look at whatever is in the default-menuitem-name.

00:18:34 screen changes to WorkEffortMenus.xml

This one did actually have one, they both have one actually. So if nothing else is selected then it will highlight the work effort; that's the default one it will go to.

00:18:56 screen changes to Framework Quick Reference Book Page 12

Okay so that pretty much covers it for the menu and the menu item. Most of the other things are not used so we don't want to cover them in too much detail. Like the position and the content-id, I think were in there for an integration with the content management stuff, but i don't know if those were implemented.

So this would be able to associate a content with a menu item. Hide-if-selected, and there's a corresponding default-hide-if-selected. By default this is false. What that will do is not render the selected menu item, so if you want to exclude it it's not very common, that's why it's false by default. But if you want to exclude instead of highlighting the selected menu item you can

do that; on a general basis for the whole menu, or on a per item basis.

Disable-if-empty will disable if the variable name there is empty.

Disabled-title-style. If you're going to use the disable then you can specify a style there, so it looks different when it's rendered and disabled.

Target-window basically goes along with a link for jumping over to a different window.

Sub-menu kind of goes along with these menu-items in here, to basically include another menu underneath this menu. So it's kind of going back to this hierarchical menu idea. That's anoher way to affect that sort of concept, but again nothing has been implemented for that right now. In other words the renderer for the Menu Widget for HTML just does flat menus right now.

Okay now as we saw with the login and logout links, a menu-item can have conditions in it. It can also have actions, although most commonly you'll just see conditions associated with them. And they'll use just about the same element from the shared elements for continues and actions, along with some of these additional attributes, this subset of the actions. That can be included there.

Conditions, in addition to the normal thing you can specify on a condition, you can have a pass-style and disable-style. So it will typically be hidden if disabled, but you can also change the style.

00:22:20 screen changes to WorkEffortMenus.xml

So as we saw in the example over here, we just had a condition without those extra elements, so that if the condition evaluates to true then it shows the menu item. If the condition evaluates to false then it does not.

00:22:44 screen changes to OFBiz: WorkEffort Manager: Edit Work Effort

Okay so to sum up again we have here two menus defined that are used in these different places. In this case for the appHeader and for the tabBar, in the tab button bar here. These are meant to be treated like tabs typically, so that the menu is always the same. It just just highlights different ones as you hop between different screens.

00:23:09 screen changes to WorkEffortMenus.xml

Part 3 Section 2.2.4

The Tree Widget

Okay now that we've looked at the menu let's look in some examples of the Tree Widget, and definitions of the tree as well.

00:23:22 screen changes to OFBiz: WorkEffort Manager: Work Efforts

I put some test records into the database. We have a work effort root, and it has three children; testWorkEffort 1, 2, and 3. TestWorkEffort 1 and 2 also have children. So this is one of the features of the widget tree, how it renders with the HTML renderer.

The expanding/collapsing does do a round trip to the server. It's not dynamic right now, so this is an area where something like DHTML in Ajax may be introduced in the future but is not right now. This does a round trip to the server...I guess the link on that one is not set up correctly.

Expanding these ones you can see how this is rendered. We have the root element, it has a child there, a child there, and a child there. This child testWorkEffort 1 has two children: testWorkEffort 1-a and 1-b. And testWorkeffort has one child, testWorkEffort 2-a.

So for each item for, each row basically, in these tree displays, you can include a screen or just piece together tables and links and such. In this case we do have a screen, and we'll look at the implement page in just a second to verify that, where we have a label here along with a number of links to do various things.

This user interface is kind of a nice thing. You can at any point in the tree add a child with a new workEffort, or add an existing workEffort as a child, or edit the association between this element and its parent, or edit with with some additional details. This is just part of this specific user interface implementation of course, an example of what can be done with the Tree Widget.

00:25:33 screen changes to WorkEffortTrees.xml

Let's look at the definition of this workEffort tree. This is in the workefforttrees.xml file. Following the conventions that's where we'd expect to find it.

00:25:44 screen changes to WorkEffortScreens.xml

There is the name of the tree, it would be included in the screen definition just within an include tree. If we search for that in here, there we are, we can find the very line that includes that in the screen. As usual we have the location and the name of the tree, and the file at that location.

00:26:02 screen changes to WorkEffortTrees.xml

So the treeWorkEffort, and the workeffortrees.xml file. Okay the general idea with a Tree Widget or tree definition for the Tree Widget, is that you'll have a number of nodes. Each node can have different sub nodes, or different sets of sub nodes.

This is a fairly simple one: we have a root node and then child nodes, and the child nodes are hierarchical.

So this has a sub node which refers to node list. And this also has a sub node that refers to node list.

Basically the way these are defined is a flat list of nodes, each with a name. And then each node can have various sub nodes, including referring back to itself. This node list has a sub node definition that refers back to itself.

So when we define a sub node we'll use an entity-condition entity and/or call a service to find the records that should be used to render that sub node. Each of these nodes basically has just the visual element in it, or is just including a screen, and they're both using the same one, workEffortTreeLine.

00:27:27 screen changes to WorkEffortScreens.xml

So we go look at the definition of that screen. Here it is: workEffortTreeLine. This is where we see the actual actions to prepare data for that particular line. And this is where we see the label, the various links, and then we have a couple of conditional links.

00:27:52 screen changes to OFBiz: WorkEffort Manager: Work Efforts

That condition is what makes these two buttons appear here, but not on the root. Because there's no association between this and a root element, those links won't appear there.

00:28:03 screen changes to WorkEffortScreens.xml

Which is what this condition is checking for, this workEffortAssoc record.

00:28:15 screen changes to OFBiz: WorkEffort Manager: Work Efforts

Okay each line is rendered using a screen, so these are just little screenlets. Each one of these lines, and the Screen Widget itself, will have a style in it that defines the area for each node. So that's how this line is drawn around here, and around each one of these, to kind of encapsulate the node in the tree and its children.

00:28:40 screen changes to WorkEffortTrees.xml

Okay let's look at some other things in the definition. Each one, when we get to it, we're going to look up the workEffort element. Then we're going to render the tree line, then the sub nodes, doing a look up on the entity engine on the entity-name workEffortAssoc, where workEffortIdFrom equals the current workEffortId.

So we want to find all these associations from this workEffort to other workEfforts, and those will be considered the children. This doesn't distinguish between different association types or any other constraints, it just shows all of the other workEfforts that are associated with this one. It's a fairly simple data model in that case.

When we're in the node list, the entry name we're looking at is workEffortAssoc, and we're going to go ahead and tell it which entity that is. The join-field-name is used I believe for the trail, or to figure out which element in this the parent is identified by.

Now to look up the current workEffort for this guy, we're doing an entity-one here, with this current node in the node list. And we're saying the workEffortId to look up is in workeffortassoc.workeffortidto. So both of these are looking up workEffortAssoc records, and for each one that is found it's going to render this given sub node. This is always going to render the node list sub node as is this one.

So this is basically just a top level node, the root node, and it renders children. And then each of those children can render the same thing under them. Again as each children is reached, what it will be defined by, each children will be rendered for a given workEffortAssoc record. So that's what we have being passed in here.

And this entry name is the name in the list that's looked up here under the sub node element that the variable in the list will be put in, in the environment. So we'll have a workEffortAssoc variable, and we'll use that variable to get the workEffortIdTo, which is what we want to render for the current line.

So this will vary the different entities used here and how they're associated. In this case as those structures change how these will be defined will change. In this case we have workEffortAssoc which implements a min to min relationship. If you had a one to many, like if workEffort had a parent workEffortId on it, has a similar notion in it.

Just for example, if it had a parent workeffortid then what we would be doing in the node list in the entity condition is we would find all of the workEfforts instead of workEffortAssoc. So we would be finding all of the workEfforts where the field-name parentWorkEffortId equals, like up here which is a little bit closer to reality, this equals currentWorkEffortId.

And then the entry neme would just call workEffort here, or perhaps just childWorkEffort, but it's in its scope so it doesn't name. So we could just call it workEffort. The entity-name would be workEffort, rather than workEffortAssoc.

Join-field-name would be the parent workEffortId. We would not have to do an entity-one to look up a detail because we would already have the workEffort record, so we could just render the workEffort tree line. And then here, where we're doing the sub node, it would just be the same pattern again. We want to look up

workEffortRecords where parentWorkEffortId equals workEffortId.

Actually in this case there may not be any need to distinguish between a root node and a child listing node because we are looking for the same thing every time, rather than something different at the root than for the children. Because at the root we do not have an association to look this up from. That's the basic idea there.

When we do have a join entity we'd need both a root element and a listing element, so that it can handle the extra lookup necessary for finding the related record and such.

Okay so that's some of the general stuff there.

00:34:21 screen changes to Framework Quick Reference Book Page 12

The render style, expand-collapse. I believe that is the default, and I don't know which other ones are actually implemented right now. I think simple, so some of these different render styles simple will just render the entire tree rather than rendering the expand-collapse buttons.

00:34:47 screen changes to WorkEffortTrees.xml

Rather than rendering these, we can play with this a little bit if we want now, and we might as well. Let's look at the node list; we'll just change what this one does and do simple. Save that.

00:35:06 screen changes to OFBiz: WorkEffort Manager: Work Efforts

Go over here. Let's pop back. So now rather than doing the expand/collapse it's just showing the whole tree.

00:35:14 screen changes to Framework Quick Reference Book Page 12

Another one of the options, show-peers can be somewhat visually confusing.

00:35:20 screen changes to WorkEffortTrees.xml

Let's take a look at that real quick. I'm pretty sure this one was implemented as well.

00:35:31 screen changes to OFBiz: WorkEffort Manager: Work Efforts

Okay this is basically just showing the current level, not going down to show all children. That's fairly limited in its usefulness because there's no way, unless you put a link in here to go down and see the children, there's no way to show the children.

00:35:55 screen changes to WorkEffortTrees.xml

Follow-trial; let's see that one does real quick.

00:36:03 screen changes to OFBiz: WorkEffort Manager: Work Efforts

As you can see these are reloaded on the fly so we can do this sort of thing. Show-trial is also another special case where it does not have anything to show the entire tree.

00:36:12 screen changes to Framework Quick Reference Book Page 12

And those two, the show-peers and follow-trial, may not be fully implemented. I'm not totally sure about those. The main ones you'll typically use are simple, which will just render the entire tree, or expand-collapse, which will show the little plus and minus buttons so you can expand and collapse the tree.

00:36:40 screen changes to OFBiz: WorkEffort Manager: Work Efforts

And let's go ahead and change this back to expand-collapse. For potentially large trees this is probably the best way to go, the normal expand collapse tree so you can expand to the whole thing.

00:36:54 screen changes to Framework Quick Reference Book Page 12

Okay let's see some other things in here. We saw how first of all each node can have a condition associated with it, and in the example we looked at it did not. If the condition evaluates to true for that current record then the node will be shown, if not the node will not be shown.

These are the possible visual elements, either including a screen or doing just a single label or a single link. If you want to do a composite line, like we had, you'd typically include a screen. And that screen would be basically included over and over for each one.

00:37:41 screen changes to OFBiz: WorkEffort Manager: Work Efforts

One thing to note about this; don't worry about the performance. Including screens, and other nodes in a tree for example in this case, all happens very fast because it's all cached in memory, and it's doing basically hash-map lookups and such, which are very efficient.

00:38:01 screen changes to WorkEffortTrees.xml

As we saw here it's including a screen, so it has that extra flexibility.

00:38:11 screen changes to Framework Quick Reference Book Page 12

But you can also do simpler trees, where it just has a label or a link, and then of course sub nodes. Each sub node will have a node-name that will be rendered for each record that's looked up in the logical state.

So we will have one of these three options: entity-and, entity-condition, or service. These are exactly as defined in the shared that we looked at before, when we were looking at the Screen Widget. So you can use one of these three to pull in data, to look up a list basically, of sub-node elements to render.

Out-field-map. This is kind of a special case, I guess I should mention it. If you want to do some data mapping in between the lookup and actually calling the node or the sub-node to render, then you can do that with zero more of these out-field-map elements.

00:39:29 screen changes to WorkEffortTrees.xml

And that's basically it for the Tree Widget.

So the most important thing to keep in mind is that you define a number of different nodes, one of which will be the root node, which is identified here with this attribute. The root node name, which is conveniently called node-root. And then each node can have various sub nodes, and it will render one of those sub nodes for each element in the list that's looked up.

In this case it's using an entity condition to look up a list of these elements from the database. You can also call up service to have a list of whatever type of elements you want; they will typically be maps or GenericValue objects.

Of course if it's coming back from the entity engine they will be GenericValue objects that you can treat as maps, since they implement the map interface.

00:40:35 screen changes to Advanced Framework Training Outline

Okay and that is the general idea for Menu and Tree Widgets.

And that also wraps up our coverage of the widgets in general, so we'll be going on to some of what I consider the secondary best practice tools: using BeanShell for data preparation and FreeMarker for templates. And we'll talk about this special case generating PDFs.

Part 3 Section 2.2.5

BeanShell

Starting screen Advanced Framework Training Outline

Next section of part 3 of the Advanced Framework, now that we've covered all of the widgets, is to talk about BeanShell and FreeMarker. These tools are some that we consider secondary best practices right now. So the preferred approach where possible is to use the Screen Widget to coordinate things, and then within the Screen Widget to use actions such as the entity operation, service calls and that sort of thing to prepare data. And then the visual elements, preferably platform neutral visual elements like the Form, Menu, and Tree Widgets, or labels and links and such that are right there inside the Screen Widget.

When you need to get down to actual HTML, where you want that level of control, the recommended tool to use, and the tool we use throughout OFBiz, is FreeMarker, the FreeMarker templating language or FTL. And for data preparation when we want something that is a little bit more immediate and close to the user interface, rather than using calling entity operations or calling services from the screen definition, we use a BeanShell data preparation script.

So these scripts are typically called with a screen action and FreeMarker, these are typically called with the platform specific HTML include tag in the screen definition.

00:01:57 screen changes to Framework Quick Reference Book Page 18

So we can see where these fit in to the bigger picture, here's the overall diagram that we've looked at a fair amount, the web application specific stuff, the control servlet and this sort of thing.

In the Screen Widget, when we get down to the actions, there's a script tag, and that script tag will typically point to a BeanShell script as an alternative to calling a service, going over to the service engine, or doing entity operations and going over to the entity engine. Likewise under visual elements instead of including another screen, a form, or other types of widgets or doing labels and links and such, you can also use this platform specific HTML include tag to point to an FTL template.

The tag we use is platform specific because the intention of the Screen Widget and other widgets in general is that they will be mostly applicable, with little variation. A given definition in these various tools should be usable in a different type of user interface platform. But once you get down to using a FreeMarker template and either use HTML or XSL-FO, or some other way of formating the information, then you are to a very platform specific level.

Okay this is where BeanShell and FreeMarker fall in as secondary best practices. Now one place where we use these the most is inside the e-Commerce applications because typically one of the priorities for e-Commerce is that you want a very high level of control over exactly how the site looks and feel is, and also over how the site flows.

So that's the reason for using the FTL or FreeMarker templates. BeanShell scripts and that sort of context help because those who will typically create the FTL files would like some sort of simple scripting language that's similar to other things they're familiar with.

BeanShell is fairly similar to Javascript, and is basically a scripting language based on the Java programming

language, so for people either familiar with Java or Javascript, picking up the basics of BeanShell is generally not difficult. So they can make modifications to those scripts or create new scripts, along with their FreeMarker templates to accomplish various things in their custom sites. So for ease of customization in the e-Commerce application we use these tools rather than using the widgets and service and other things as much for data preparation and presentation.

00:05:25 screen changes to Advanced Framework Training Outline

A little bit later on in these videos we'll be talking about JPublish. We originally started using BeanShell and FreeMarker when we started doing JPublish based views. Initially we had JSPs and used the Region framework for doing a composite view then we moved over to JPublish, which is a more dynamic tool which is based mostly on the decorator pattern.

That same decorator pattern is what we use in the Screen Widget, though we use a little bit more flexible and complicated version of it, the composite decoration pattern. So JPublish is a fairly simple tool but you'll see a lot of the patterns that were established there still used in OFBiz and a lot of the ideas for the Screen Widgets actually came from JPublish.

Thanks to Anthony Aaden for all his good work there. So the main part of that is that we still use BeanShell in FreeMarker, although we call it from a screen now typically instead of from JPublish Page Definition.

00:06:56 screen changes to OrderScreens.xml

So let's continue looking real quick at how these fit into the context and then we'll look in detail at some examples and some of the general principles and ideas in BeanShell and FreeMarker. This is the order history screen in the e-Commerce application, and this is what we'll be using for examples since it's fairly straightforward.

Like other screens we have some set actions. We basically get down here to a script. We have the full location of the script relative to the e-Commerce component. This is the typical place that BeanShell scripts will be located, in the webinf directory in the actions folder under that directory.

Under there they will typically be in the same path or use the same structure that is used for the FreeMarker templates directly under the webapp root directory, which is here. Here we have actions/order, just as down here we have ecommerce/order for the order history .ftl file. So the same pattern is used to organize both of them.

Here we're using the script tag under the actions to call this BeanShell script, an here we're using this platform specific HTML template tag to render an HTML template. There are also some other variations on this you can use to use FTL files, namely the HTML decorator.

00:08:48 screen changes to Framework Quick Reference Book Page 12

We should look at the reference real quick. Let's get down to the screen widget platform specific. So we have html-templates which does a plain include, and then html-template-decorator. The template-decorator is basically just like a screen decorator, it allows you to include sections from other screens in an FTL file.

We'll look at some examples as we go through these of how to include another screen in an FTL file, whether it's being called as a decorator or not, and then what the decorators look like as well.

00:09:33 screen changes to OrderScreens.xml

Okay so that's basically how typical BeanShell script and FreeMarker templates are called from a screen definition.

00:09:43 screen changes to Advanced Framework Training Outline

Now let's step back and look at a little bit more detail about what these tools are all about. The main site for BeanShell is this beanshell.org.

00:09:56 screen changes to www.beanshell.org

Here's the site. This is a project that's been around for quite a while. It has a number of uses; it can be used for real time scripting as well as for prepared scripts, which is the main context we'll be discussing. But you can use this for real time scripting as well, so either as a replacement for something like Pearl for other command line scripting types of things. In fact in OFBiz we have a little BeanShell container mounted so you can go to a web page or let it into a certain port and get a BeanShell client.

00:10:55 screen changes to ofbiz-containers.xml

If we look at the ofbiz-containers.xml file, down at the bottom we load this BeanShell container. Telnet port is specified here and that port minus one is one you can go into with a web browser.

00:11:15 screen changes to http://localhost:9989/

Let's just look at that real quick, show you what it looks like. This is something you'll naturally want to protect from outside access, or disable altogether in a production environment.

00:11:28 screen changes to BeanShell Remote Session

So if we click on the link for this example here at this console basically right inside here we can execute...I guess that applet's not working. Let's do the awt one.

00:11:42 screen changes Beanshell Remote Session-AWT console

Okay so right in here inside this console we can execute things, basically Java code written, or BeanShell script that is basically a more flexible variation on the Java syntax. So that is what we can do.

This is helpful for testing and for other things like that, or for getting into a live server and doing very low level demonstration or that sort of task. It doesn't come up a whole lot; it's mostly useful probably for testing, but it can be useful for even inspecting live instances of OFBiz that are running.

00:12:45 screen changes to Framework Quick Reference Book Page 18

Okay let's go ahead and let that go. The main thing again that we use BeanShell for in OFBiz is as a scripting language for data preparation for something to display.

00:12:57 screen changes to orderhistory.bsh

Looking at a BeanShell script, let's look at this orderhistory.bsh. It's kind of like a Java class, class definition or a .java file, except you don't actually have a class in here. You can define classes as well as inner classes, basically when you write a script like this you have an explicit class with the simple-method.

so you have to do import statements so that as BeanShell is interpreting it, it knows which objects to look at, just like in a Java file. Then the rest of it is basically as if it was just the body of a method in a Java file, with a few variations. It is a scripting language meant to have some of the flexibility of scripting languages, and so type declarations and type castings is not necessary.

This orderRoleCollection is a list, but it's implicitly recognized as a list rather than me having to declare it explicitly like this. By the way just a side note here in Eclipse, which is the editor I'm using for recording this video. I'm using the Java Scrapbook Editor for the BeanShell files. I have my editor set up to associate .bsh files with the Java Scrapbook Editor, and it can do some fairly helpful things.

It does light syntax highlighting, And I guess if you want auto-completion and that sort of thing there's a pretty good BeanShell editor in NetBeans that I believe was kind of deprecated for a while but is available as a plugin. Anyway when looking at one of these scripts basically that's the main difference you'll notice between this and Java code.

There's typically no type casting, although you can put in type casting and it will understand it just fine and even enforce the typecast. But if you want to leave that open you can. In a BeanShell script called from the control servlet there are various variables that are always available, like the context variable.

00:15:47 screen changes to OrderScreens.xml

When you're doing an action in a screen definition like this set action it's putting this value, rightBar, into the rightBarScreenName, so it's populating that field there.

00:16:05 screen changes to orderhistory.bsh

If we want to do the same sort of thing in a BeanShell script we use the context variable and just treat it like a map. Context.put, and then here we do the name of the variable and the value. And it's that simple. So it's very common you'll do some sort of a query like this.

Here it's doing an ordering and filtering on the results of that, and then putting that in the context and doing another thing for a different entity. This is for the downloadable products list, and then it takes that and also puts that in the context. And then these variables that are put in the context can be used in the FreeMarker template, so this orderHeaderList.

00:16:48 screen changes to orderhistory.ftl

Let's look over at our FreeMarker template. We'll typically find that used in something like this, where we are using the list operation here to iterate over the orderHeaderList, each element of this being called an orderHeader. And then we can do things with that.

00:17:09 screen changes to orderhistory.bsh

So we'll get into FreeMarker in more detail in a bit.

Other details about BeanShell. When used in the context of the Screen Widget there are various things available such as the delegator object. There's also a dispatcher object for calling services, and you also have a parameters map that has all of the incoming parameters. And it is the parameters map that's discussed in the Screen Widget that we're talking about up here.

00:17:39 screen changes to Advanced Framework Training Outline

It follows this pattern in the parameters map, where the first priority is request attributes, then request parameters, then session attributes, then application attributes.

So basically when you are in an BeanShell script it is inheriting everything from the context of the screen, and the same is true for a FreeMarker template. Typically the FreeMarker template will not modify the context that it's running in, it will simply use the data that's already there to pull things from, in order to render the information that needs to be presented.

So we talked about BeanShell, some of the general concepts of it. We use it mostly for data preparation actions. You'll find more details about it here. This is

where it's usually located, in the component structure. It uses the Java syntax with some scripting features like dynamic types, which is the main one.

00:19:07 screen changes to ScreenRenderer.java

Another file that I want to point you to here that can be helpful is the screenrenderer.java file. There's some methods here to populate a context for the screen Widget, so these are variables that you'll always have available in a screen.

A screen can be rendered in a service context or a request context. This is where you'll see it most of the time, but you can render a screen from a service and pass in the dispatch context and the service context. Most commonly, though, we'll be rendering a screen in an HTTP request from the control servlet, and so we'll have the HTTP request response and servlet context objects coming in. From those it populates quite a number of variables.

Here you can see where it's putting together the parameters map. And then if you want to see everything that's available this is one place to look, in the screenrenderer.java. The main method to look at here is this one, populateContextForRequest. And you'll notice that it populates the basic context down here.

This is just a shared method used for both screens and for request or HTTP request context rendering. So this populateBasicContext, these things will always be available. Screens. This is just a screen renderer so that you can render other screens. Actually in an FTL file you can call screens.render, and we'll see some examples of that in a bit.

You'll always have a globalContext as well as context so you can put variables in each of them. Null-field can be handy for some things, especially when doing entity operations. Then we have our parameters map, entity engine delegator. service engine dispatcher, the security object for checking permissions, the current locale and the current logged in user. Some basic things that are always available.

And then there are some additional things that are available in the http request context like the request object, response object, the session object, and the application, also known as the servlet context object.

So there are really all sorts of things. The root directory for the current webapp, the websiteId, whether we're using HTTPS or not. We've got all sorts of things in here, so it's a nice exhaustive list of it that can be a good thing to refer to.

00:22:00 screen changes to orderhistory.bsh

But again the typical things. You'll do some entity engine operations using the entity engine API from the generic delegator object, which is always there being called delegator. And we'll go over some of these in detail when we talk about the entity engine API later on. And you can also call services from the service engine dispatcher object and all of that sort of thing.

00:22:33 screen changes to Advanced Framework Training Outline

Okay so that's basically it for BeanShell. For more information about the BeanShell tool itself there is some documentation here in www.beanshell.org. But in general if you understand about Java, or even about the language like Javascript, the syntax should be fairly familiar and it shouldn't be too difficult to create or modify existing BeanShell scripts.

Part 3 Section 2.2.6

FreeMarker

Starting screen: Advanced Framework Training Outline

Okay moving on to FreeMarker. Let's look at some detail here. So the main thing that we use FreeMarker for is data presentation, data formatting, and layout. The site for FreeMarker is freemarker.org.

As we talked about when we were looking at the screen definition, a typical place for FTL files under the webapp directory in a component, you'll have a directory that follows the name of the webapp, and then somewhere under the directory you will have FTL files immediately in the webapp root. They'll typically be in a directory under the webapp root.

00:00:51 screen changes to Orderscreens.xml

The one we were looking at here with orderHistory is in the directory. Here's the webapp root directory e-Commmerce, and it's in a directory under that called Order, and then we have our orderhistory.ftl.

00:01:07 screen changes to Advanced Framework Training Outline

Okay so when looking at FreeMarker for the first time, you'll notice that if you are familiar with php and certain other tools, that it has a PHP like syntax for certain things.

But it uses Java, is built in Java, and for calling methods you can call Java methods typically as long as they return something. So you can use them in if or assign statements, and it also supports the . syntax for referring to entries in a map.

That can be very handy with OFBiz given that we use maps so much, even with the generic value objects from the entity engine. Those implement the map interface, and you can use the . syntax in FreeMarker for convenience. Okay some of these concepts like the control construct, built ins, and we'll talk about this exis-

tence concept and failfast, and then some of the OFBiz specific transforms.

00:02:33 screen changes to http://www.fremarker.org/

So let's look at the FreeMarker site. Here's the home page www.freemarker.org. They have pretty good online documentation right here, so I highly recommend looking at that. They also have a page here for some editor and IDE plugins. They have a pretty good plugin for clips that I use, and that's very handy.

00:02:59 screen changes to http://www.freemarker.org/docs/index.html

Okay in the FreeMarker manual they have a number of helpful things, and this might be good to go over and learn about, to just kind of read through, especially some of this initial stuff in the designer's guide. And that's mostly what we'll be talking about.

The prospective of the programmer's guide. Most of this stuff is taken care of for you by the Screen Widget when you are including FreeMarker templates in the Screen Widget, because it sets up the context, does the actual call to FreeMarker, gives it the template to render, and so on.

The Screen Widget for both BeanShell scripts and FreeMarker templates will even do pre-parsing and caching of the parsed form of these scripts and templates, so that they can run as fast as possible. Okay so let's talk about some of the basics for FreeMarker.

00:04:10 screen changes to FreeMarker Manual-Directives

Some of the main things we'll be looking at are directives, which are these guys that have the # or number sign in front of them. We'll also do user design directives and macros, and we'll use the @ sign in front.

00:04:30 screen changes to FreeMarker Manual

So directives are important. Another thing that's important to understand is this concept of built-ins.

00:04:36 screen changes to FreeMarker Manual-Expressions

So in expressions you'll often see built-ins. Let's see if they have a section on those in here somewhere. They do introduce the concept somewhere in here, but let's go ahead and look at the reference.

00:04:56 screen changes to FreeMarker Manual

There's a built-in reference down here. So built-ins for different things. There's also a directive reference here by the way. Ones you'll run into most often are these if, else and elseif directives, and the list break directives, and assing, that's a big one to run into a swell. There are a number of others, though, that you can use for different things. As well as user-defined directives and such we'll be looking at those, and that's basically custom tags in JSP and other things.

00:05:32 screen changes to FreeMarker Manual-Built-ins for strings

So built-ins for strings. Basically a built-in refers to this thing where you have a ?, and then something following it.

00:05:47 screen changes to FreeMarker Manual

So there are a number of these, and you can do different things with strings, numbers, dates, booleans, sequences, and hashes. All these sorts of things, with sequences being a list and hash being a map.

00:05:59 screen changes to FreeMarker Manual-Built-in Reference

One of the important ones to know about is the existence built-in. There are a few of these. One of them is if-exists, another one is exists, and those are here. Another one is has_content, which requires it to not only be not null, but depending on the type of object like a string will need to be a non-zero length string. A list will need to have more than zero elements in it and so on in order to meet the has_content.

00:06:52 screen changes to FreeMarker Manual

So these are used in different places in expressions in FreeMarker, either for assignments or if expressions or these sorts of things. Let's look at some examples to see these in action.

00:07:05 screen changes to OFBiz E-Commerce Store:

So what we're looking at is the order history screen, in the e-Commerce application. And basically it's just showing a list of all of the orders, with a link to view the detail for each one. It also shows a list of the download files available, depending on the orders that have been placed for downloadable digital goods, this digital gizmo is an example of that. So we have orders here of different amounts, different status, they have different things on them.

00:07:39 screen changes to orderhistory.bsh

So basically in the BeanShell script that we saw it was getting a list of all of these orders where the current party that's logged in using the user login object, which is always populated for the Screen Widget. So we want the partyId to match that, the partyId from the userLoginObject. And the roleTypeId will always be placing customers.

So we want to find all of the roles where the current party is the placing customer. And then we're taking this, we are doing a few things to it like going through it, getting all of the orderHeader records. So we have orderHeader records instead of orderRole records, fil-

tering those, ordering them, and then we take this orderHeaderList and drop it in the context.

00:08:36 screen changes to orderhistory.ftl

Over here in FreeMarker template, just as with any template, all of the text that you drop in here will simply be output until it finds something special like a $#{ } which means insert the value of this variable right there. You'll see that's used all over the place for internationalization, and other things of course. But in this case specifically, especially where you're seeing a uiLabelMap, that will typically refer to a localized label.

Okay all of these html things are simply going to be there. This comment right here is a special type of comment for FreeMarker, and it is a server side comment. In other words everything between this comment opener and the comment closer will be excluded on the server side and not sent down to the client.

If you want to send something down to the client then you would use a normal html content like this. This editor by the way, this Eclipse plugin that's a FreeMarker editor that does the syntax highlighting, is very handy to give you an idea of what's going on with these different things, and how differnt parts of the template are being treated.

Okay so we get down here and we see this list directive. We're iterating over the orderHeaderList that we were just looking at the preparation of in the BeanShell script. And for each element in that list we're going to call an orderHeader and treat it like an orderHeader.

Now we're not displaying a whole lot about each order. We want to display the orderDate, the Id. So here's the order date, the orderId, the grand total of the order, and we'll talk about what all this stuff is in just a minute.

The status, and then we have a link to go to the orderStatus in the order detail page. Alternatively if the orderHeaderList does not have content, or fails the has-content built-in check with the NOT, then we'll show a message that says there were no orders found. The next part of this page does that same sort of thing for the downloadable products that we saw in the second section there, so it's the same pattern.

Let's look at some of the details that are up here in this. This is assign-directive; it's going to create another variable in the FreeMarker file only. When the FreeMarker file finishes it will not be in the screens context, so it's just in the FreeMarker file context.

OrderHeader is a generic value object. We're calling a method on that called getRelatedOneCache and passing it as string. So we're getting the statusItem related to the currentOrderHeader. That status item is being used down here so that we can show the description of the current status instead of the ID of the status.

00:12:03 screen changes to OFBiz E0Commerce Store:

As we can see here we're getting the description, not just a statusId. So that's how that is implemented.

00:12:09 screen changes to *orderhistory.ftl

Now you'll notice a funny thing here. We can treat status like a map and just do something like we did here. We can do status.description, like here we have order.orderid. So it's getting the orderId entry in the orderHeader map, but here we want to do something special.

In some places it will automatically internationalize what you're referring to, in a FreeMarker template going through the map interface to get the orderId for example. If we wanted to internationalize this code it's not passing in the locale, so the entity engine through the map interface is not getting the locale at all.

However, the generic value object implements a special interface that we call the localized map that's part of OFBiz. It has a get-method on it, just like a normal map, get. Except instead of just passing in the key you also pass in the locale. So in order to internationalize the display of entity fields in an FTL file, you need to do this sort of thing where you do, .get the key or the name of the field that you want to get, and pass in the locale object.

The locale object is one of these that the Screen Widget always has in place, so you'll always have a locale sitting there that you can use for whatever.

Okay another one that's a little bit different here is this ofbizCurrency. This has the @ sign, which means it's typically either a macro, or in this case it's a user defined directive. Now you'll notice by the way that FreeMarker in the directives and such does not follow strict XML syntax.

List orderHeaderList as orderHead. These three are not not valid XML because they're not attribute name-value pairs. We're simply specifying these things in a certain order, with this a keyword to clarify the syntax, and with these other two being variable names. So the same sort of thing happens here, except we do actually kind of have for these user defined directive/built-ins? name value pairs.

A user defined directive can have parameters that are passed into it. So the ofbizCurrency directive has an amount parameter, and an isocode. The isocode is the currency unit of measure that's used in OFBiz; the UomId for a currency unit of measure is actually the iso code itself. So it makes it very easy to get that iso code for showing the currency, and dealing with the currency in whatever way. All this user defined directive will do is take the amount and format it as currency, according to the isocode that's specified.

So here's one question; how does FreeMarker know about this user defined directive? In OFBiz this is not an extremely flexible thing. We have a class that's called the FreeMarker Worker.

00:16:04 screen changes to FreeMarkerWorker.java

And it has a little static section in here that loads all of the FTL transforms. It loads them by the full classname, so this is obviously something that's kind of begging to be put in a configuration file. But it has a big enough dela in the project to date so it's kind of stayed here.

So there are a number of different user defined directives that are available in OFBiz. The most common ones you'll see are these ones here: ofbizUrl and ofbizContentUrl. OfbizUrl basically does the URL transformation, so in an ofbizUrl you'll just specify the name of the request to go to.

00:16:59 screen changes to *orderhistory.ftl

So for example down here we have an a HREF ofbizUrl. OrderStatus is the name of the request, and enclose the directive over here. So it's going to transform this, this is just the same as any OFBiz URL transformation when we are creating a URL to go to a request that's in the same webapp. It'll look up this request name, make it https if necessary, or http, a full URL if necessary, otherwise it's a relative URL.

Those various things to add the session id if needed, all this kind of stuff. So that's what the ofbizUrl tag is for, and we already looked at the ofbizCurrency one.

00:17:50 screen changes to FreeMarkerWorker.java

Others in here, the ofbizContentUrl is kind of like the ofbizUrl but simpler, and it's simply what we use to centralize static content. By default out of the box in OFBiz that is in the framework images component, in a webapp that's under there, that gets mounted as /images. But you can define any static content prefix you want in the url.properties file or in the website record, as we have looked at before.

OfbizCurrency just formats a currency. OfbizAmount just formats a number according to the locale. SetRequestAttribute is kind of a cheat one that allows us to set a request attribute, modifying the context in essence, even though its not really a best practice to do that. Sometimes before including another screen we need to put information somewhere so that that screen will know about it.

Okay we have various things in here for the Menu Widget, renderingContent and subContent, looping through or traversing content, checking permissions, all sorts of little things. But those you typically won't run into as often. If you want to write your own transforms to plug in you can simply add them to this list in this file, and recompile and run. And they will be available for your FreeMarker files.

00:19:36 screen changes to OfbizurlTransform.java

And if we look at something like the OFBiz URL transform class there's simply this template transform model that you need to implement, and the main method that it does is this getWriter. With this getWriter basically you're returning a writer, and what you do is its basically creating a dynamic class in here that's an instance of the writer, that inherits from the out. And it does something special for writing; flushing it and closing it.

The way these transforms work in FreeMarker is, you in essence get passed in what is between the open and close tags, but in a stream, and then you can transform that into whatever you want. And that is what will actually be written out into the FTL file.

00:20:42 screen changes to *orderhistory.ftl

So it will take all of this and replace it with whatever is specified by the implementation of the transform. Same thing here, it will just replace all of that with what is in there.

And it does do the variable expansion before it passes those in, and other handy things like that so you just have to worry about the straight up string.

Okay and those are the basics of FreeMarker. You'll see a lot of these directives like if; you'll have a boolean expression and then a close if. Assign and list; those are probably the main ones that you'll run into frequently.

00:21:42 screen changes to www.freemaker.org

For information on those and others in fair detail again the FreeMarker documentation on freemarker.org is really pretty well done.

00:22:03 screen changes to *orderhistory.ftl

Okay some other things that are related to FreeMarker.

Let's talk about the existence real quick, I forgot about that. One thing FreeMarker does, part of the design that they've chosen, is a fail-fast paradigm. That means if you do something like orderheader.orderid and it does not exist, that will cause an error in the rendering of this FTL file, and it will not successfully render. It will instead just show a big error message.

So if I wanted to make it possible for orderId to be null then I need to do this. By the way this new plugin for FreeMarker does have some auto-completion, so it has the built-ins here which is very nice.

Basically we want the if-exists built-in, and that will tell FreeMarker that even if this variable, even if there is no orderIdEntry in the orderHeaderMap, it's okay, just render nothing and continue on with the tempalte. Now if it

was like this and the orderHeader didn't exist, it would still complain.

Basically a built-in would just apply to the one that it's sitting right next to. If you want it to apply to more than that you put parenthesis around what you want it to apply to. And now the efiexits will apply to not just the orderId but the orderHeader as well. So even if there is no orderHeader variable if it's null it won't complain, it will just insert nothing and carry on.

In this case we know there will always be an orderHeader because we're iterating over this list and creating it as we go, so we won't get into this section of the template unless there is an orderHeaderObject. And for orderHeader orderId is a primary key, so we know that will always be there. This is making an assumption that the orderDate will always be there even though it's not part of the primary key, which is a safe assumption for the orderHeader because that is always populated.

Okay so that's the failfast and how you get around it with the efexitsst built-in. Next thing I wanted to look at is some stuff that is more specific to interacting with the Screen Widget inside an FTL file. Let's look at an example of the categorydetail.ftl file.

00:25:08 screen changes to categorydetail.ftl

If we're running along through an FTL file and we want to include a screen somewhere in there we can do this sort of thing. If the screens object will always be available, as we saw in the screen renderer where it populates the context, it always puts a screens variable in the context. And this screens variable has a render method and you can simply pass it the full name of the screen. Now in this case that's in a variable, and that variable is populated in the BeanShell script that prepares data for this.

I thought it was populated here; it's populated somewhere, but not in this file. Let's look at the category BSH which wraps around this. It's not there either. It's most likely in testing, in the screen definition.

00:26:30 screen changes to categorydetail.ftl

So anyway inside here we have this variable that has the screen name. If we looked at the full screen name it would be just the standard component:// whatever, like /ecommerce/widgets/catalogscreens.xml, and then the # or number sign, and then the name of the screen in that file to render.

The render method will return a string. We get that string into the FTL template by treating it like any other string variable. So we're treating this just like a method that returns a string, that's really all it is, and it'll pop the results of that screen rendering right there into the FTL file. So that's really fairly simple.

00:27:27 screen changes to Framework Quick Reference Book Page 9

Now another thing I wanted to cover was the HTML template decorator.

I'm not sure if that's actually used anywhere, so let's get a search going for that real quick. And we just want to find that in the screen definition XML files. Looks like there are actually a few.

00:28:09 screen changes to ConfigScreens.xml

Okay here's an example of the HTML template decorator, and it's very much like the screen decorator. We have the template decorator, where we tell it which template we want it to render and treat as a decorator, and then we have decorator sections.

In this case we have two sections where it's going to insert whatever is inside these sections. And here we have just a single include tag in each one, wherever that section is identified in the FTL file. So let's look at the FTL file to see how that section is identified.

00:29:06 screen changes to EditProductConfigOptions.ftl

Okay when an FTL file is rendered as a decorator, in addition to the screens object we'll also have this sections object. And all we have to do is do sections and call the render method on it. So sections.render, and then the name of the section to render. So where we're gong to be including the createConfigOptions form.

00;29:34 screen changes to ConfigScreens.xml

Which over here, createConfigOptions form is just going to include that form.

00:29:44 screen changes to EditProductConfigOptions.ftl

And it will just render everything that's inside that section and drop it into the FTL file right there. It'll basically return it as a string, so we get FreeMarker to display it and send it down to the client by doing a ${ }, treated as just a string variable or any other method that generates a string and then returns it.

Then down here we do the same thing; we're rendering another of the decorator sections, createConfigProductForm.

00:30:08 screen changes to ConfigScreens.xml

Which corresponds with this one, and so it'll render that and drop it in. So that's how the HTML decorator stuff works.

00:30:20 screen changes to EditProductConfigOptions.ftl

So again to kind of recap. We have the sections.render method we can use, when the FreeMarker file is being treated as a decorator.

00:30:32 screen changes to categorydetail.ftl

And in any point in any FreeMarker file we can always call the screens.render method. Let's see if I can find another example real quick of that where it's not a variable it's actually listing out the full thing. Let's search for screens.render in start.ftl this time; and it looks like we found a few of them.

00:31:05 screen changes to viewinvoice.fo.ftl

Here's an example. As I mentioned, basically it's just a full location of a screen, and it'll typically be component based. With this URL if you don't put a prefix on it like this, if you just have a / on the front, then it'll treat it as a classpath resource. That's kind of standard for any place you would put this type of location. But the most common is to use the ofbiz specific component:// name of the component, and then the path of the resource relative to the component root.

So this is where that orderPrintForms file is. And the name of the screen within that XML file to render. By the way this one that we just happened to chance upon is an FTL file that's rendered through the Screen Widget that's generating XSL-FO, hence all these funny tags, rather than generating HTML. We'll look at examples of that in a little bit more detail; how that fits into the control servlet, what the screen definition looks like, and so on. But that's for the next section, the very next section actaully.

00:32:23 screen changes to Advanced Framework Training Outline

And that's pretty much it. We've talked about the basic concepts of FreeMarker, where the FreeMarker files are typically located, and of course the FreeMarker website where you can get more information.

And we talked about general syntax with the ${ }, and the pseudo XML tags for these control constructs or directives. As well as the built-ins, all of the ? Things.

We talked about the existence concept and fail-fast, especially with the if_exists, and some of the ofbiz-specific transforms, also referred to as user defined directives. The most important being ofbizUrl and ofbizContentUrl. We also talked about some of the other important ones like ofbizCurrency and ofbizAmount.

I guess I should note real quick that BeanShell script can be called pretty much anywhere. We have a BeanShell worker, just like the FreeMarker worker, that facilitates calling of BeanShell scripts. We use little BeanShell scriptlets all over the place, including the use-when conditions in the Form Widget. You could actually put BeanShell scripts inline in a simple-method file, a MiniLang simple-method file. And we'll talk about that in more detail in a little bit. So BeanShell is just a general all-around good scripting language.

FreeMarker is, like BeanShell, a good all-around general tool. Except in this case its for templating, for taking a file with a bunch of static stuff in it, inserting dynamic things, variable expansion looping, control constructs, ifs and such, and generating some kind of dynamic output.

FreeMarker can be used not only in generating XML, but as we mentioned for generating XSL-FO to eventually turn into a PD, and we'll talk about that in more detail.

But we can also use it to generate any type of text. We can use it to generate Java files, and it's very handy for generating XML files. And we actually use that as part of the entity engine, you'll see later on, to transform incoming XML files into another XML structure that the entity engine can understand.

So FreeMarker has some very nice built-in stuff for handling XML files; parsing as well as basically transforming XML files from one format to another, whether it be to XML or whatever other format.

In general FreeMarker is just a good templating language, and we can use it for quite a number of things. It can also be used on the fly when rendering content in the content manager in OFBiz, to transform content based on templates that are also stored in manageContent records.

So there's a lot of flexibility available with FreeMarker, and we do a lot of things with that.

Okay and that's it for BeanShell and FreeMarker.

Part 3 Section 2.2.7

PDF Generation with XSL:FO and FOP

Starting screen: Advanced Framework Training Outline

Okay let's talk for a bit about generating PDFs using the Screen Widget and FreeMarker and whatever tool you prefer for preparing data, BeanShell or calling a service or using the entity gets as actions in the Screen Widget. In order to generate an XSL-FO file, which is then transformed into a PDF using FOP, which is an apache product. So first let['s talk about all these tools in detail, and get an idea of how these will work, and what it's good for.

00:00:50 screen changes to OFBiz: Order Manager: View Order

So in the Order Manager when we're looking at the detail of an order, this is one that I just placed for testing purposes, there's a little link up here to generate a

PDF of this order. Now using this style of XSL-FO and FOP to generate a PDF is very good for printed documents where you need a lot of control over the layout, formatting and such.

But it is not necessarily the best option for reports per se, where the primary objective is data gathering and summation. So when you do need a printed document then this is the way to go. With XSL-FO and FOP it is possible to do things like bar codes and graphs and other such things, because you can inbed XML snippets for graphing languages like SVG right inside your XSL-FO file.

00:02:06 screen changes to order.pdf

So let's look at when we click on this PDF link it will generate something like this. You can include images in it, text of different size and fonts, and you can have pretty good control over layout and how things are formatted on the page.

This is a pretty simple example of a sales order slip , just a printed page that represents the sales order. You can see each item is kept very concise, kept to a single line with a short description, quantity, list price and the total. And then adjustments if there are any on a second line. This is a lot less information than you'll have on the order detail page in the order manager.

For example it shows all of this information about order items, but that's typically what you want in a form. All of that information that is available on the order detail page is also available for this. In fact it's very easy to get at because the same data preparation script is used for the order details script as is used for the order PDF.

00:03:28 screen changes to The Extensible Stylesheet Language Family (XSL)

Let's first talk about the tools. The first tool that we're using is the XSL Formatting Objects, or for short XSL-FO. This is part of the extensible status sheet language family, and is one of the W3C standards. Right here we're just looking at w3.org/style/xsl. This is the home page for XSL, and for XSL-FO.

00:04:01 screen changes to Extensible Stylesheet Language (XSL)

There's a link here for the actual XSL-FO specification that has a general description here.

00:04:15 screen changes to
www.renderx.com/tutorial.html

In general XSL-FO is meant for laying out printed documents. It's a lot like html. The W3C is the same standards body that is responsible for the html standard, the XML standard, and various others. And it's a lot like html and CSS except applied to a print layout instead of a web page layout that you'd see in a web browser. This introduction is a good one, that's linked to this XSL formatting object tutorial.

00:04:51 screen changes to www.w3.org/style/xsl/

And there's a link to that from the XSL page here. There are various other helpful resources here.

00:05:06 screen changes to
http://incubator.apache.org/ofbiz/documents.html

There are also some good books on the topic. There is one that I referred to on the ofbiz.org site, XSL-FO, making XML look good in print. This is one of the better ones that I found that talks in detail about the XSL-FO standard itself, and not just the application of XSL to transform an XML document into an XSL-FO file. So this goes into detail with the information that we need for creating the type of template that we want in order to generate our XSL-FO files.

00:05:41 screen changes to entitymodel.xml

Because we're not transforming them from XML, we're simply creating an XSL-FO document like this, in a FreeMarker template.

00:05:56 screen changes to controller.xml

So staring from the beginning, this is how it works. The view, eventually you can have a request, you can have an input processing associated with the request just like any request map, eventually when it gets to a view, the view-map, we'll use the type-screen FOP. When we look up at the top of this file we'll see this is one of the declared view handlers; screenFopPdfViewHandler is the full name of the class that's used there.

So the type is screenFop, which means it's going to run it through the Screen Widget. We're going to render a screen, and then we're going to run it through FOP. That's basically what it does.

00:06:46 screen changes to
xmlgraphics.apache.org/fop/

Now FOP is a project from Apache. It's part of the XML graphics project, and you can find links to it also in just xml.apache.org, under the XML project. So basically this allows rendering of an XSL-FO file. You can kind of see in that magnifying glass some of those fo: tags, from an XSL-FO XML file to a printed page.

It also supports other outputs, so FOP supports PDF, as well as PCL, postscript, SVG, XML, different XML standards, and the area tree representation, the actual print using the Java printing stuff: AWT, MIF and TXT files. But the main one that's targeted, the main purpose of the project, is to output a PDF. If you wanted to output to one of these other formats you could do so.

00:07:49 screen changes to controller.xml

This is more of an advanced thing, but what you would do is...we talked about about this screenFop view handler, and that refers to this class up here. So you would just create a variation of this class this screen FOP PDF view handler, to generate something else like PCL or postscript file. And then you could have it return that.

00:08:19 screen changes to http://xmlgraphics.apache.org/fop/

Okay so the Apache FOP library is included with OFBiz, so you don't have to do anything special to install it. If you want more information about FOP there is some good information on this site.

00:08:37 screen changes to controller.xml

Let's continue on with our example. We're going to render a screen. This is the screen it will render; it will output an XSL-FO formatted XML file, which will then run through FOP, which is what this view handler does, then output a PDF. When we do that in a view we need to tell it which content type we're returning.

We explicitly say this is going to be an application/pdf, so that's the Mime type for a PDF file. And the encoding is none, because there is no text encoding to be done since it's a binary type and not a text type. When you're using screen FOP or any other PDF generator you'll want to specify the content-type in the encoding so the browser knows what kind of document it's dealing with.

If you do it that way you don't even have to put the PDF extension on it, and the browser should figure out automatically that it's a PDF file, although the request we use for the order PDF, let's look at that, is actually called order.pdf.

In a request map URI, or a request name, you can use the . in an extension all you want. This is also helpful for certain browsers that don't look at the Mime type, but instead look at the filename extension to figure out what type of file it is.

So it's helpful to name the request according to the type of file that's coming out. You notice we don't do that in a lot of cases. We could call all of these for this one for example, lookUpPerson.html, but we don't just to shorten it down. It's kind of redundant, and most browsers will handle a default HTML just fine.

00:10:35 screen changes to OrderPrintForms.xml

Okay in this screen, in the OrderPrintForms.xml file, we have the order PDF file. This is fairly similar to any other screens that we've seen, we set a title for it that it can be passed in. This would just be whatever parameterized field the script or the template is looking for, might be looking for.

Notice that we have all the property maps right here for internationalization. These are in the screen definition where we usually don't see them in the high level screen definitions because the decorators do this. But in this case there is no decorator, so all we're doing is just rendering the FTL file, so we have it setting up those.

Then we have it calling this script, the orderview.bsh script. This is one handy thing about this pattern, you can use a BeanShell script for data preparation that's been used elsewhere.

00:11:33 screen changes to OrderViewScreens.xml

For example in the orderviewscreens.xml file, the orderHeaderViewScreen right here.

00:11:43 screen changes to OFBiz: Order Manager: View Order

This is the screen that results in displaying all of this, all of the order details in the orderManager application.

00:11:49 screen changes to OrderViewScreens.xml

So the same script that prepares data for that, or the bulk of the data for that, there is some other stuff that's prepared in addition.

00:12:03 screen changes to OrderPrintForms.xml

The same script that prepares the information needed for that, we also use over here in our order print forms to prepare information for the PDF file. So that reduces redundancy in the code, just sharing the data preparation.

Okay the next step in the screen, now that the data is prepared, is just to format it, generate an XML file. And we'll do this with an FTL template, and notice that it is using HTML and HTML template to render the template, even though it's not actually outputting HTML. There's nothing special about these tags, they just denote that we are doing something web specific.

We may introduce at some point some sort of a text and text template instead of this, or XML instead of HTML, but for now this works just fine. So you can use the HTML and HTML template tags for that.

00:13:04 screen changes to orderView.fo.ftl

Okay in the template itself, basically the output of this template is going to be an XSL-FO file, so most of what you'll see here is just the fo: tags, that are part of that. Here's the name space for an XSL-FO file, all of the prefix tags. I won't talk in detail about what these are, all of these tags, you can look at the reference to see what all of these mean.

One thing you'll notice in here is that a lot of the attributes on these different elements, some of the elements are fairly similar to HTML tags. A lot of the attributes

are actually similar to CSS, margin bottom, margin right, a lot of these same sorts of things, font-weight.

In a file like this we use the same FreeMarker markup as we would use in any FTL file; the assigned directive, the #{ } syntax for printing the value of a variable. The same pattern here, for generic value objects, calling a get and passing in the locale for internationalized entity fields.

You can do the same sorts of list directives, if directives, and so on. So the only real difference here is that we're producing XSL-FO instead of HTML for the output. So the result of all this when it runs, you can see for example up here at the top, we're importing another screen, the same screens.render that we've seen in other places works here just fine.

So this is a screen that is shared to put the company logo in different forms for the order, the return, and other such things. So you can even share different parts of the PDF source of the XSL-FO files.

00:15:26 screen changes to order.pdf

So that little include right there basically ends up rendering this guy up here. Over on the right we have the text sales order, the date order, the order number, and the current status.

00:15:32 screen changes to orderView.fo.ftl

Those are these things right here. So this is where it prints the sales order. It will print either sales or purchase depending on the order type. It will have a label for the date ordered, and we're printing out the order date, and you can see it's just formatting it inline right here.

So here's something that's helpful to know about FTL if you want to call a static method on a class. You can just do static and [the syntax is somewhat complicated and], so surrounded by square braces and quotes, you have the full name of the class, and then a ., and then the method name, and any parameters you want to pass to it. If it returns something you can call a method on whatever it returns. Like the date formatting object here that is created by calling this static method. Okay and that's pretty much it for XSL-FO.

00:16:51 screen changes to OFBiz: Order Manager: View Order

We use this for a number of different things in OFBiz. Of course one of the things we use this for is the order PDF here. We also use it for the picking and packing sheets, in the facility manager. I didn't prepare anything here so let's...I guess there aren't any orders that are ready to pick.

00:17:29 screen changes to OFBiz: Facility Manager: Pick List Options

00:17:34 screen chagnes to OFBiz: Facility Manager: Pack Order

So anyway the pick list here is another example, the pick list is I believe this one, or maybe the packing sheet, that has a bar code on it. This user interface is made so that if you scan the barcode with the barcode scanner it will enter it there, hit enter and go into the packing screens.

So the picked sheet with the barcode on it can be used to feed into the packing process, for those that are actually doing the box packing. So certain things like that can be done with this type of printed document.

00:18:15 screen changes to Advanced Framework Training Outline

Okay let's hop back to our outline. That's pretty much it for how to generate a PDF using the Screen Widget, how to generate an XSL-FO file for a printed page oriented layout. And then using the FOP project, the FOP library from Apache, to change that XSL-FO file into a PDF.

And that's it for this second section, the Screen Widget based views, so we've no gone over everything related to that. Next we'll talk a little bit about JPublish and the Region framework. These are both deprecated things for the most part in OFBiz, abut we'll talk about those because there are some resources that are still there and can be helpful.

Part 3 Section 2.3

JPublish Based Views

Starting screen Advanced Framework Training Outline

Okay in this section we will be discussing JPublish based views. JPublish is one of the tools that was used in OFBiz for a while, in fact for a while it was our primary tool we used for composing pages for multiple parts, using includes as well as the decorator pattern. One of the main features for it for the top level page is it facilitates the decorator pattern very well, and it also allows you to separate data preparation logic from the template itself.

The general idea is each page has a page definition XML file, which specifies which template to use to decorate the page, and also actions to run, properties to set, and that sort of thing.

00:01:15 screen changes to http://sourceforge.net/projects/jpublish

JPublish is a project that was run by Anthony Aeden, and it's been a while since he has been very active in it. This is the home page of the sourceforge.net site for JPublish, and you'll notice it's been a while since they've done a release. OFBiz started actively using it

quite a while back, during the process of the development of 3.0, so OFBiz mostly uses features that were around in the 2.0 release.

There is a link to a home page, JPublish.org, but that typically doesn't work. I don't think that site has been alive for quite some time. So it's been a while since JPublish has been actively developed, and that was one of the reasons that we decided to move away from it in OFBiz. But that was only one of various reasons. The Screen Widget has various advantages over JPublish in that it is more flexible, it fits in better with other tools in the Open for Business Framework.

00:02:36 screen changes to Advanced Framework Training Outline

In general it has more features, it also has a bit more of a simple semantic, and so there are less concepts that can be confusing at times. But all around JPublish is a very good piece of software, and a lot of the features in it were very helpful for moving OFBiz forward and inspired parts of the design of the Screen Widget.

So in general JPublish itself was configured for each webapp in the jpublish.xml file, that is in the web-inf directory under the webapp root directory. The page definitions, the convention that we use in OFBiz, you can specify different locations for the page definitions. But the location we use in OFBiz is a directory called page-defs that is inside the web-inf directory. And so this is where, in that directory and in sub directories under that directory, is where you'll find the page definition XML files. Okay we'll talk about specifying page variables, the decoration templates, and some of the different types of actions.

00:04:20 screen changes to controller.xml

Okay's go look at, kind of walk through the flow of how this works, let's start with the controller.xml file. This is the controller.xml file for the webtools webapp, and it has a few JPublish files in it, although most of those that were using JPublish have been converted to use the Screen Widget. But most of the screens here still use the Region framework and JSPs, which was the original view technology in OFBiz, so many of these have not been converted yet.

Let's look at the login page here, which is a JPublish page. Just like any view handler there is a view handler class for JPublish that implements the view handler interface, and is declared at the top of the XML file. When using JPublish you specify the page that you want to render, but the first thing that JPublish does rather than looking at that page is it finds a page definition XML file that corresponds with that page.

As I mentioned before there is a page-defs directory where all of these live, so it will look for a file with the same path and same name, but different extension. Everything would be the same except for the extension, so in this case it would change .ftl to .xml. It's going to look in the page-def directory for a login.xml file.

00:05:55 screen changes to login.xml

And this is what it will find. The XML file is pretty simple. You've got page, you can specify a template to wrap around the current page, you can specify properties, the name is an attribute of the property element. And then what is inside of the property element, the text inside of it, is the value of the property.

This is kind of like the set operation in the Screen Widget, and so a lot of things that you can do in here are similar. This is a pretty simple example.

00:06:40 screen changes to EditDataResourceType.xml

Let's look at a different example. This is the EditDataResourceType page definition file from the content manager in the content component in OFBiz, and the content component is pretty much the only other place where JPublish is still used. Some of these older pages have not been converted yet. You can see here it's declaring the template to use to wrap the page, it's declaring various properties, and then it's calling the scripts.

Now this is calling two different types of action, the page-action and the content-action, and there is one other type of action, the template-action. This is one of the parts of JPublish that can be somewhat confusing initially, but it does make sense and can be useful in certain scenarios, so in general the original page that is rendered, the top level page, is considered a page.

When another page is used as a template, a decorator basically to wrap around the page. Only a page can have a template decorating, so only the top level page can have a template decorating it. The template is rendered differently.

The page definition XML file if it saw a page, like for this main page if it saw a page action there and it was being rendered as a decorator template, it would ignore the page action, and it would also ignore the content action. When a page is rendered the top level page it will run all page actions as well as all content actions.

Content actions are also used when a page is included inside another page. When a page is included in another page it is handled differently by JPublish than when it is the top level page. An included page can still have a page definition XML file, but the page actions in that file will not run, only the content actions.

So that's the difference between the three types of actions in JPublish, the template-action, page-action, and content-action.

00:09:35 screen changes to login.xml

So let's look at the concept of the template a little bit more, and what that does. Looking at the login.xml file we're telling it that we want to use the main template. JPublish knows where to find the main template by a setting in the jpublish.xml file.

00:09:48 screen changes to jpublish.xml

There are also various other settings in the jpublish.xml file. One of the important ones for finding the template is the template root. All of these templates are relative to the root of the web application, or also known as the webapproot directory, so the page root where it expects to find the page definitions.

As I mentioned before, this is the convention we use in OFBiz, and this is making it explicit so that JPublish knows where to find it. In the web-inf directory there is a page-defs directory, and that is the page definition root for JPublish. The template root is the templates directory, right inside in the root of the webapp. The action-root where the actions will be located is web-inf/actions. So that's the directory it will look in when we specify that we want to run an action.

00:10:46 screen changes to EditDataResourceType.xml

Like over here we say content-action or page-action. The name here of the action finds the location of the script to run by iterating this as a relative path to the action root.

00:11:04 screen changes to jpulbish.xml

As specified here. Inside the web-inf directory we have these two important directories: the actions directory and the page-defs directory. And then for templates, the decorator templates that wrap around a page, those go in the templates directory.

So if we look over here in the webtools component in the webapp directory we have the webtools webapp. Here is the root directory of the webapp. We have a templates directory here, and it has a main.xml file and a main.ftl file and those two together constitute the main decorating template. This main.xml file is a page-def XML file. This is the one exception to the location of the page definition XML files. For templates that will go in the templates directory with the templates rather than in the page-defs directory.

00:12:17 screen changes to main.xml

Here we have the main.xml file. This one is extremely simple. In fact, it's just an empty page definition file because there's nothing special we want to do with it.

00:12:23 screen changes to main.ftl

The FTL file, on the other hand, has some interesting things in it. When its rendering the decorator template, basically like a decorator in the Screen Widget or anywhere else, it means that it renders the decorator template and then when it gets to a point in that template where it calls back to the calling page, it includes the calling page.

A few things in here to note that are used in FreeMarker templates or other types of templates that are being rendered through JPublish, there are some objects that are available. There is a page object and a common object. These objects are not always there, we actually explicitly declare these objects in the jpublish.xml fiel. But what these objects allow you to do is render certain pages.

So if I say ${pages.get("/includes/appheader.ftl")}, then it's going to render that FreeMarker file through JPublish. It will look for a page definition XML file in the page-defs directory in the includes sub directory called appHeader, instead of .ftl .xml. And that page definition XML file will be treated as a content page definition XML file. So content actions will run, page actions will not, template actions will not.

The common object here is just like the pages object, except instead of pointing to the current webapp it points to the common webapp. In OFBiz we have various FTL files and other resources that are shared between the different applications, and this common object points to that directory instead of to the current web application root.

00:14:23 screen changes to jpublish.xml

Also these are declared as I mentioned here. They're called repositories in JPublish, so we have a pages repository and a common repository, and you can see the config-dir. They'll typically always be web-inf/padgedefs, for both of these. And then daunt is the root of the pages, so it's pointing to the current web application. This one (<root>../../..common/webcommon</root>) is a relative path that points to the webcommon directory in the common component, which is in the OFBiz framework.

This is a clue to kind of a little workaround that we use so that we can share resources between web applications when we're using JPublish. It's now much easier to do with the Screen Widget because we have the Component:// URL and other such things to refer more generically to the locations of different resources.

Okay other things to note about what you can do in here. You can specify global actions, basically scripts that will be run on every page request. The env-setup.beanshell file sets up all sorts of things for the OFBiz environment for screens that are rendered through JPublish.

Most of the things that are set up there are set up by the Screen Widget when you're using that so there's no need for this sort of a script. That's another way that the Screen Widget is more convenient and comes from the fact that the Screen Widget is not mean to be a generic tool like JPublish; it's very specific to OFBiz and is meant to be used specifically with the other tools in the framework.

So we have some character-encoding we can specify here, also mine-mapping for extensions and mine-types. This is used in the OFBiz user JPublish when it's going through the control servlet, but the control servlet will override this with its own content type or mine-type definitions and character-end-codes. Especially as far as the information that is sent down to the client for what the current page consists of.

Okay a few last couple of things that are in the jpulbish.xml file. We have a view-renderer and a page-renderer. These are things that are specific to the JPublish and allow us, instead of going through the when a FreeMarker file is rendered, rather than going through the default rendering that JPublish uses, we will instead always go through the OFBiz specific FreeMarker view renderer to render the FTL files. So this is kind of a plugin feature that allows us to render templates in the way that we want to.

00:17:45 screen changes to controller.xml

And that is the basic idea behind JPublish, as well as some of how it is implemented and used. Occasionally you'll still run across these views that are implemented with JPublish in OFBiz. They're fairly rare now because we replaced most of them with the Screen Widget, and new ones have been written using the Screen Widget. The utility of this information, the stuff you just learned in this section, is most useful for these areas where you find an older page like that.

It may also be useful if you ever decide to do something with JPublish, although we would typically not recommend it going forward partly because of some of the limitations of using JPublish in OFBiz, and also because the JPublish project has not been maintained in some time. But it is a good tool, and as I mentioned before it inspired many of the features in the Screen Widget, and making that a better tool in general.

00:18:56 screen changes to Advanced Framework Training Outline

Okay that's it for our coverage of JPublish. One thing to note here with JPublish; we always use BeanShell for data preparation and always use FreeMarker for the templates. In the screen widget those are optional. You can use other things to generate output, either for the data preparation side, the actions, or for the templates side for the data presentation, but with JPublish you'll always use BeanShell for data preparation and always use FreeMarker for templates.

In general for a web application JPublish can only use one template language at a time. If we were to use Velocity we could only use Velocity, if we were using FreeMarker we could only use FreeMarker, and so on. I think that was made more flexible in later versions of the alpha efforts of JPublish, but not something we were ever able to take advantage of.

Okay that's it for JPublish. Next we'll talk about the Region framework.

Part 3 Section 2.4

Region Based Views and JSPs

Starting screen Advanced Framework Training Outline

Okay the next section is on Region based views. We talked about JPublish, how it's one of the older types of views that we don't really use anymore, and the Region based views also fit into that category. This is actually the original view technology that we used in OFBiz, JSPs and then combining JSP into a composite view using this Region tool. As opposed to the JPublish tool, which uses the decorator pattern, the Region tool uses what is called a composite view pattern.

00:00:55 screen changes to Region.java

A lot of the ideas for the Region stuff came from this book, Advanced Java Server Pages, by David M. Geary. This is a good book to look at for information about JSPs, although of course it's a few years old now. There may be a new edition of it, and the JSP standard has of course progressed since we kind of moved on from it.

00:01:29 screen changes to Advanced Framework Training Outline

Rather than using the more flexible pattern of JPublish and the Screen Widget, where we are assembling text on the fly and combining different blocks of text together and such, the general idea though with the Region framework is that the Region framework uses the servlet API, which it has to do in order to work with JSPs.

JSPs cannot be rendered as a normal template, they have to be run in a servlet environment. So the Region framework basically allows piecing together of multiple JSPs, passing around as needed the objects for the servlet environment. Just as with the JSP, when you're using a Region based view you cannot get the text out of it without going through the HTTP server.

That's one of the reasons that we've moved on to more flexible technologies, so that we an use the same tools for different types of templates. For example even this

generating PDFs could not be done using the regions and JSPs. This is because we wouldn't be able to get the text out of it and run it through FOP, and then return that client without playing some tricks. Either doing a request from server back to itself, an HTTP request to get the page, and then there are some security issues with that because the client is the server, so it doesn't have the authentication tokens, the cookie or HTTPS session in order to validate the client...

So it leaves a bit of a problem. And that's one of the reasons that we moved on to these more flexible technologies that use a stream or writer based approach, rather than a servlet environment using the request dispatcher object to include or forward to different pages.

Okay in the Region framework itself we'll talk about defining different types of regions, template based regions, or you an define a region that inherits from another region. And then there are sections of regions; this is where the composite view comes in, basically injecting different sections into a template.

We'll also talk about the OFBiz taglib, which is the JSP tag library that is part of OFBiz. There are various different tags available. These are for use in JSP files. You'll note that we have some generic things like conditional tags, flow control tags, internationalization tags, and so on.

That would typically be part of a more generic tag library, but keep in mind that at the time that this was written the JSP Standard Tag Library was a standard that was not yet finished. In fact about the time that we moved away from JSPs was about the time that that was introduced.

Of course even if you're using that sort of a tag library there are certain data presentation tags, service engine tags, and URL tags of course that are very OFBiz specific, that interact with other parts of the framework and are really just part of the Open for Business project itself.

00:05:23 screen changes to controller.xml

Okay let's start by looking at the controller.xml file. This is the example that we'll use, in the webtools webapp, and we'll look at the findUtilCache screen, which is a fairly simple screen. You can see with these that it will, just as with the others where we have different types of view handlers, we have type = region for the view handler.

Now the interesting thing with the Region type view handler is that you don't have to specify a page as you do with other types. You can specify the name of a page that is defined in the regions.xml file for the current application, but if you don't it will just default to the name of the view right here. So the way the Region stuff works is that we're basically saying findUtilCache is a region.

00:06:25 screen changes to region.xml

So we can find the definition of that region by looking in the regions.xml file. The regions.xml file lives in the webinf directory, so you can see the path there. Webtools is the name of the webapp, so that's through a directory of it. We have the webinf in the standard place, and have the regions.xml inside that directory.

So this is where the region is defined. You can see that this inherits from the main region, and overrides two sections of that inherited region; the title section, this is the title that will be used, and the contents section. So you can see with these other regions it's pretty much the same; they're overriding title and content.

So this is how they are injecting the specifics for that page into the more general composite view that they're inheriting from, which is the main region one. The main region is just another region that's defined here at the top. This one is a template based region, so rather than inheriting from another region it just points to a template.

Now the way the Region stuff works is we say this is a region and this is the template for it, so when we render that region it's going to render the template, and as it goes along we'll run into spots where it's going to render sections of the region.

Each one of these sections that is defined here will be rendered inside this main template. Now if there's a section defined here that's not in the template it's just ignored. If there is a section in the template included in the template that's not defined here, it's also just ignored, it inserts nothing.

Let's look at this file, the main-template.JSP. Notice some of these that we have here: the title, header, app bar, all the way down to footer, the content being the important one; kind of the main body of the region.

00:08:28 screen changes to main_template.JSP

So this is the standard JSP file. In order to include those sections we're using part of the Regions tag library, the region:render tag. And we just specify the name of the section that we want to render. You can see up here we have the header, the appbar, the appheader, leftbar if there is one, error messages, content, this is the main body of it, the rightbar, and the footer.

Now a section can be another region, so this can be hierarchical as well, but in most of the examples you see here basically it's just flat. Okay so let's review this again real quick.

00:09:31 screen changes to regions.xml

So when we say to render the header section it's going to look up in the region definition what to use in the header. Now this is the region that we're rendering, findUtilCache. It inherits from main region. There is no header here among these sections that are defined, so we go and look up here in the region it inherits from, and see that there is the header.

So the net effect is this main template basically becomes a decorator of sorts, and is kind of like the composite decoration pattern that we use in the Screen Widget. So you'll see some of the evolution of the view technology in OFBiz.

JPublish is largely based on the decorator pattern. The decorator pattern itself is an independent thing that has been around a lot longer. Same with the composite view pattern; it's been around for a long time and it's used in various different technologies, and this Region framework is one application of that. The pattern that we use for composing a view in the Screen Widget is a combination of the two. It's the composite decorator pattern, is what we call it.

Whether there's actually an official pattern to find anywhere in any of the patterns books for that I'm not sure, I'm not aware of any. But basically it's a combination of the composite view and the decorator patterns.

00:11:08 screen changes to main_template.JSP

Okay another thing to note in this file. Let's just start from the top; we're including this envsetup.JSP. When we looked at the JPublish stuff we noted there was this envsetup.bsh file that's basically derived from this. Another thing to keep in mind with the JSPs and regions is that there was no separation to different files of the data preparation logic and the presentation part. So we used the same stuff, or the data preparation logic, namely the JSP files.

One thing you'll still see in many areas of OFBiz is the use of what we call worker or helper methods, and these are basically where we would rewrite code in Java to prepare the data, then that would be called from the JSP files. And that was an early effort with no enforcement in the framework to separate the data preparation from the presentation of that information.

Okay this is a fairly standard thing to have in a JSP file, the taglib tag here. We have some OFBiz tags, and the prefix here is OFBiz. There aren't any in this file, though we'll see some in the other JSP files we'll be looking at when we go over the OFBiz tags. The Regions taglib has a prefix of region, the main one here is the render tag.

00:12:38 screen changes to regions.tld

There are some other ones. Here's the TLD file, the Tag Library Description file for the Regions tags. The render one is the main one that we have here. Actually that is the only one, just the render tag, and it has various different things you can do. Many of these were actually not implemented, they were just part of the pattern that we were doing, so it's basically just the section that you'll use here.

00:13:06 screen changes to main_template.JSP

Okay so also in here we notice we have the OFBiz tags.

00:13:25 screen changes to web.xml

By the way if you're not familiar with the tag library definitions in JSP, those two strings, those URIs, refer to tag library definitions in the web.xml file. Each web application has a web.xml file. We talked about this a little bit before, and if you're familiar with the servlet API and web applications in general this should be very familiar.

So a tag library will be declared there in the web.xml file. It's given a RUI and then the location of the TLD file. So here's the regions.tld file that we were looking at. We typically put these in the webinf directory. So there's the Regions one and the OFBiz one.

00:14:11 screen changes to Regions.xml

Okay so once we get to the point where it's rendering the regions we're walking through, and the main template says render the content. This is the default content to put in the body area of the page, but we're overriding that for our page because we really want it to put in the FindUtilCache.JSP file.

00:14:30 screen changes to FindUtilCache.JSP

So then we're going to hop into the FindUtilCache.JSP file and run that. So this has a taglib definition to use the OFBiz tags with the prefix OFBiz, we pretty much always use that prefix. We have some imports up here.

So here you can see some of the start. This page is one that actually has very heavy code intermingled with the HTML. So in case you're not familiar with JSPs, this type of tag using the % sign and the close % sign basically just means treat this as Java code.

JSPs are actually interpreted into Java files, and then those Java files are compiled into class files. And it's those class files that are executed at run time when the page is rendered. So it's a fair bit different from other technologies.

In a way it can run very fast because of that, but because of some of the other design decisions in the JSP, especially related to tag libraries, they actually do tend to perform very poorly, even compared to something that is interpreted like FreeMarker, especially when we cache the parsed template so that all it has to do is run through the objects and memory and produce the output. So FreeMarker can actually compete very well

even with the compiled Java class that results from a JSP.

Okay some things to note in here. Going on to the topic of the OFBiz tag library. These are basically just Java code; there's nothing super special about them. If statements as well can be put in here. In a JSP file this is just Java code that'll be inserted.

Now when we get down here to the a HREF, this is one of our custom tags. This is the OFBiz URL tag, it's just like the at-OFBizUrl custom tag or transform that we use in FreeMarker; it calls back to the same thing. It does the same thing in various widgets: Screen Widget, Form Widget, and Menu and Tree Widgets when we are specifying a location for a link, when it is an intra-app, So within the app, style link.

They all do the same thing, they all call the same background logic. Basically to transform the name of a request into the full URL of that request. So that's how you do it in the JSP; use this custom tags library, the OFBiz tags tag librarty, and ofbiz:url. There's also a content URL, just like there is an OFBiz content URL in FreeMarker.

And for the content type links in the various widgets. Okay you'll see it quite a bit in the OFBiz code JSPs where we're using just while loops and other standard constructs. There's also an iterator tag that we have as part of the custom JSP tag library for OFBiz.

You don't necessarily have to use that. If you're using something like the standard tag library then that's better to use than a while loop, which we're basically just using to iterate over this tree set, and also better than using the OFBiz iterator tag. But that does still exist for backwards compatibility and such.

Okay so the result of this is very much like any other template. We have some information that's shown up at the top, we have a table, here's the header of the table, and then we're displaying all these lines. And then at the bottom we have some links for various things as well.

00:19:02 screen changes to OFBiz: Web Tools:

So here's the page that it produces. This is in the webtools webapp. Here is the header and other sections from the Region that this is defined in, so you can treat that very similarly to a decoration, header and footer and all these other things. Here are the links we were looking at, the table, the links down at the bottom we were looking at. So that's basically it, that's how the JSP is rendered.

00:19:35 screen changes to Advanced Framework Training Outline

So there are a number of other tags.

00:19:47 screen changes to ofbiz.tld

You can see the definitions of all the tags in the ofbiz.tld file. So each of these; the URL tag and the content URL tag, and some of the other ones: if, unless, iterator, iterate-next, iterate-has-next, format for formatting different things, prints an attribute from the page context, displaying a field, a field in an entity (entityfield). So these have some descriptions in them, and finally creating an input value.

A lot of these are shortcut things that we did. One thing you might see in some of the JSPs, we found that a lot of these shortcuts that were very handy for coding JSP ended up running very slow.

And so we have what is called the pseudo tags, and they're basically just static methods that do the same things as these tags, but would run significantly faster. We had pages running fifty times as fast because of using that sort of pseudo tag calling a static method, instead of calling one of these custom tags.

So stuff to call a service, parameter for calling a service, making an object available. A lot of this stuff is very old and we found much better ways of doing things since then. But if you are using JSPs then some of these things can be very handy.

So some other examples of JSPs. Let's see if we can find one that's a little bit more complex. Let's look in the controller.xml file.

00:21:56 screen changes to EditEntity.JSP

EditEntity.JSP is a very interesting one. We had some stuff in here a long time ago as part of the user interface based on the entity engine, to be able to edit entities on the fly through the web browser, and then write those out to the entity definition files. This is actually a very bad idea, especially for production systems, and we found in time that it was a lot easier to just edit the XML files, so it was somewhat of a failed experiment.

Although with an IDE-based user interface like something here in clips, that could be very handy, and we may do something like that again in the future. And it's also certainly an area where a person could create a commercial product based on OFBiz that would be a good value add for OFBiz developers.

So here are more examples of the OFBiz URL tag. The OFBiz URLs you can see are by far the most common used. I was hoping to find the iterator tag, or some of those. But it's not that important.

Those were used in some of the more customer facing pages, not as much in these administrative pages. And the reason they were is because those pages did tend to get more complex, and we wanted to make them easier to customize, especially in the e-Commerce application.

So that was the motivation behind a lot of these tags, to simplify the code in the JSPs.

00:24:01 screen changes to EditCustomTimePeriod.JSP

But many of them don't exist like that anymore because we now have the other technologies in place. Especially the Screen Widget and the other widgets.

Okay so here's an example of an OFBiz if tag. You just specify the name of the variable. It needs to exist, and if it's a boolean be true. There's some other funny things like this; needs to not be empty, that kind of stuff. And you'll see in here a few of these OFBiz if tags. Another comment about that; it may turn out to be much better to use the standard tag libraries than using some of these things.

Here's another interesting one. In this inputValue tag you could specify the name of the variable for the generic value object the entity was in. And then the name of the field to display. And then it would actually generate an input value, basically one of the form field HTML elements, and just pop it right into the page. So that was used for convenience obviously. Those are one of the more verbose things to create.

Entityfield. This is displaying a field of an entity. Attribute specifies the attribute in the current context, where the generic value object is, and this is the field of the generic value object to display. Another example of the inputValue tag.

So there are a few more of the tags in action. As you can see this is far less refined and less efficient then what we are currently using. So converting a lot of these things to FreeMarker resulted in much simpler and smaller pages, and in general a very good all around experience, and has been much easier to maintain and such. But if you have to use JSPs for some reason then these tags could be of use.

00:26:19 screen changes to regions.xml

If you want to have something, if you have to use JSPs and there's a possibility of using some tool to combine different pages into a single page using an external configuration, then these Region definitions can be very handy.

00:26:25 screen changes to Advanced Framework Training Outline

Okay but chances are for the most part the only way you'd be using this is to understand better and possibly make modifications to some of the old pages, especially in the webtools webapps where most of them have not been modernized, and they're sill using the JSPs and Region-based views.

Okay and that is it for our web application framework, the web application framework of OFBiz. So the next section will be the overview of the MiniLang, which is often used with the user interface. And so we've grouped that in with these, although one of the more common uses of the simple-method now is to implement services, and so it also ties in very well with the service engine, which we'll be discussing in detail later in part 4 the entity and service engines.

Okay, see you next time.

Part 3 Section 3.1-3.2

MiniLang Introduction

Starting Screen: Advanced Framework Training Outline

Okay now we are into section three of part three, and talking about the MiniLang or Miniature Languages that are in OFBiz. There are actually two MiniLangs or Miniature Languages in the MiniLang component. One of them is the simple-map-processor, and the other one is the simple-method.

Now we wanted to make it possible to, instead of just calling a simple-map-processor externally, actually be able to call it, or define a simple-map-processor inline in a simple-method XML file. So there's actually one schema, just a simple XSD file, that defines the XML structure for both the simple-map-processor and the simple-method.

The general idea behind both of these is that we have a particular set of operations that we want to do over and over and over in certain parts of a business application. Namely mapping of inputs to outputs, validating inputs, doing operations interacting with the database, writing data to it, reading data from it, calling services, and other types of logic.

In general these sorts of things are fairly high level, and we piece together a bunch of these things to make more compacted processes. In order to make it easier to deal with these kinds of common operations, these MiniLang things are basically like an API that you use in a special context. So rather than being like a Java API, it's like an API that's defined in an XML format and it runs a little bit differently than a Java method so that it's a little bit easier to read, easier to customize, and easier to write as well.

Typically a method that's implemented in the simple-method will have significantly fewer lines than a comparable method implemented in Java because we've narrowed the scope and made certain assumptions that we can do as part of the framework of OFBiz. These are significantly more efficient. The error handling is more automated and more reliable because of that, many things are more reliable because while you're writing the code you don't have to worry about them.

Looking at services written in Java I often see all sorts of problems with the way that error messages and other things are handled, with the way that exceptions are handled, and lots of things like that. Basically writing a service in Java is complicated, and so unless you need very complicated logic or to interact fairly intensely with the Java API it's typically much more efficient and less error-prone to write it in a simple-method.

A simple-map-processor can also have significant efficiency gains over writing validation, parsing, and mapping code in Java. This was actually the first tool that was written, the simple-map-processor, and later the simple-method was written. The simple-map-processor the motivating factor behind it was that we had all this logic around to process input, especially from custom forms that needed to be fairly dynamic so that as the business need to perform change, the processing logic and validation logic also needed to change.

Doing that in Java was a lot of work, so we had in some cases around a five to one difference in the lines of code that were necessary for doing this. In some cases even more than that, closer to ten to one. So this is one of the biggest efficiency gains, the simple-map-processor, especially when you have a custom form that needs custom validation and mapping.

00:04:46 screen changes to
http://incubator.apache.org/ofbiz/docs/minilang.html

Okay there is a document that describes the Mini Languages, the Mini Language guide here, and this is available on the ofbiz.org website, which as you see is currently hosted on incubator.apache.org/ofbiz. Eventually that URL will change as OFBiz graduates form the incubator. So you're seeing kind of an interim thing.

This is kind of an older document and so not everything in here is one hundred percent correct. For example this is still referring to simple-methods.dtd, and in a very old location, core/docs/xmldefs, which hasn't been used for a very long time. It is now simple-methods.xsd,

00:05:30 screen chagnes to Advanced Framework Training Outline

and it is the location listed here, framework/minilang/dtd/simple-methods.xsd.

00:05:38 screen changes to
http://incubator.apache.org/ofbiz/docs/minilang.html

But some of the general information here is very good for the background, for instance some of the patterns that it is based on like the Gang of Four interpreter pattern. And Mark Grand also has a set of patterns that he worked on; one of these is the little language pattern. This is actually the one that was most inspiring for working on this, along with this book called Building Parsers with Java, by Steven John Metsker. That book was originally used as the basis for the OFBiz rule engine, which has now been deprecated in favor using other Open Source projects that have matured to fill that need.

One of the big differences though with the way some of these patterns are applied is that rather than using a FreeForm BNF, the Baccus Now Form type syntax, or an English-like syntax, we use XML documents in OFBiz. Some people may say that is a real pain because they feel XML is not a very natural flowing text structure, especially when you're used to speaking English or any other language, although if you're a big fan of reverse Polish notation on a calculator you might find this much more natural.

But in general basing it on XML, once you understand the way that XML files work and have a good XML editor, then these sorts of files are much easier to use. The auto-completing editors and other things, visual editors built on top of these languages and such, are much more friendly in general. Also most of the other tools in Open for Buisness are driven by custom XML files like this, so it fits very naturally into the rest of the framework.

Okay we're starting with an overview of the simple-method and the simple-map-processor, so let's look at a couple of quick examples. Simple-map-processor basically will have an input map and an output map, so when you call it you pass in those two maps, and input map and an output map. Then it takes things from the input map and it processes them, so it's basically processing different things from the input map.

Here for example to process the workEffortId, and we want to copy it to the output map. If we wanted to use the same name then all we have to do is say copy and it will copy it to the output map. Basically we process one field at a time; we name the field and then we can do various operations on it. For the currentStatusId for example, we can copy that, and then we can also put a not-empty constraint on it, with if it fails the fail message “status is missing”.

This fail message tag by the way is there rather than just having a message on this element, because there are to options; you can do a fail-message or a fail-property. This is something we'll see in other places in the simple-method, as well as here in the simple-map-processor.

Rather than doing just a copy, another option is to convert, specify the top we want to convert to. If the conversion fails then this is the error message that will be shown. Again we can do a fail property instead of a fail message which will read a property from a properties file, which is typically used for internationalized messages. You'll see that quite a bit in different parts of OFBiz, since there has been a lot of effort put into internationalization.

That is the general idea for the simple-map-processor. You can also do things like make-in-string. This is kind of a special case where if you have a bunch of strings coming in and you want to format them in a certain way, string them together in a certain way, or combine them to make another one, then you can do this.

So we have in-field, constant, other things we can string together to make a string. And then this string that we've created can be used as a normal incoming field when we're doing things like a comparator here, and then a conversion. That's the general idea for simple-map-processors. This is especially useful along with the simple-method in cases where you have a single big form that's coming in.

00:10:39 screen changes to newcustomer.ftl

Let's see the newcustomer.ftl file. This is a big custom form that has data that we want to send to a whole bunch of different services for persistence in the database.

00:11:01 screen changes to CustomerEvents.xml

In a special simple-method as that's called as an event we basically process all this incoming stuff using a bunch of these map processors, so this is how you call a map processor inline in a simple-method, just use the call-map-processor tag. And then if everything underneath here the simple-map-processor definition is exactly the same here as if the simple-map-processor was in an independent or separate file.

So when we call a map-processor, as I mentioned before basically it operates with an in-map and an out-map. So we just tell it which maps those are and then we go along and do it.

You'll see a whole bunch of these. This is a very common pattern, where we have a bunch of map processors pulling things from the parameters map, which is the map that comes into a simple-method that has all of the parameters. When it's called as an event, through the control servlet in essence without going through the service engine, this is the parameters map made up of the request parameters and such. When it is a service, the parameters map has the incoming context that the service passes in.

Okay the out-map. Basically with these map processors typically be pulling things from the parameters map and creating a whole bunch of these out-maps. So there's the personcontext, the addresscontext, homephonecontext, workphonecontext, and so on. And this is another common pattern for these methods. At the beginning we have a whole bunch of validations going on, and as we're going along we're just adding error messages to a list so that we can report as many or as complete a set of error messages as possible back to the user.

And then we get to the point where we're done with the validation and mapping and we run the check-errors tag. And if there are any error messages in the list then it will return right here. Then if there are not any errors we go on and do all of this processing.

So each of these maps that we were preparing up there is then typically used to call a service. We're calling createPartyPostalAddress with the address context, createPartyTelecomNumber with the phone context, createPartyTelecomNumber with the work phone context and so on to get all of the information from this one big form into the appropriate places.

So there's an example of a common application of the simple-methods and simple-map-processors of processing input for a custom form.

00:13:46 screen changes to http://incubator.apache.org/ofbiz/docs/minilang.html

Now we'll be looking into, especially the simple-method, quite a lot more in a bit. You'll see what the different categories of operations are and what we can do with them.

Going back and continuing with our overview of the Mini Languages. A simple-method can be called in two ways, as I mentioned, either as an event in the controller.xml file, referred to here with a type=simple. The full path of the XML file that the simple-method is in, and the name of the simple-method in that file to run.

Alternatively you can call it through a service definition, so you can use a simple-method to implement a service in other words. We saw a bit of an example of this in the framework introduction videos. Where we have the createPartyRole service we tell it the engine is the simple-method engine, that's how it's implemented.

Again the full location on the classpath of the service.xml file that has the simple-method defined in it, and the name of the method in that file to run. That's how they're referred to in those different contexts.

The location of these simple-method files is typically on the classpath. This is pretty much true throughout OFBiz right now. We are considering a move to use the same component:// URL as the standard. One thing to note even in places where we use the component:// URL in a location attribute, you can just leave off the protocol of that URI and then it will interpret it as a classpath location and look it up on the classpath.

So that will still be possible of course, but just a note that we are considering making that the standard, using the component:// because it's a little bit easier to understand. It's also a little bit easier to narrow down exactly where the file is without having to search through all the resources that are on the classpath. This for example

could be in a JAR file, could be in a directory, that sort of thing.

00:16:11 screen changes to Advanced Framework Training Outline

The current convention right now, as we discussed up here in the structure of a component in the component construction convention, in the script directory is where we put all of these sorts of things. The script directory will go on the classpath, so everything under there will be available on the classpath. That's where we typically put simple-method files, simple-map-processor files, and other things that are used like even BeanShell scripts that are used for implementing services or being called as an event will typically just go right here.

If you put them in the source file then it'll be copied over to the JAR file when this is jarred up, and be available on the classpath. But you won't be able to change it on the fly, so that's one of the reasons that we have this exploded directory sitting on the classpath with all of the files individually accessible. So you can modify these files and see your modifications take effect immediately depending on cache clearing and such.

But again out of the box the default cache settings for OFBiz and other settings for OFBiz are for development and so out of the box or out of subversion. Out of SVN, the code repository, everything will operate in the development friendly way, and so these things are reloadable on the fly without having to change anything.

Okay going back down to the MiniLang section here.

00:17:48 screen changes to http://incubator.apache.org/ofbiz/docs/minilang.html

Some of the other concepts for the simple-method. We'll go over all the elements and attributes in a bit, and talk about the operations in more detail and look at some examples of how they're used. These are in this document so this is one way of looking at it. These are the different attributes on the simple-method element itself. Typically the ones you'll use are just these, the method-name and short-description.

00:18:26 screen changes to Framework Quick Reference Book Page 4

As I mentioned this is older, so a better reference is the quick reference book.

00:18:34 screen changes to http://incubator.apache.org/ofbiz/docs/minilang.html

One thing I should note about the future of this document and other documents like it that have descriptions like this for different attributes and elements.

00:18:44 screen changes to CustomerEvents.xml

We will be putting those into annotations in the XSD files so that they will pop up automatically in your XML editor when you're editing things. We saw some of that when we were looking at the Form Widget in other places.

00:18:59 screen changes to simple-methods.xsd

Right now there are not a whole lot of annotations in here. There are a number of these like this on the simple-method, there's a nice big annotation. This is what the XSD file looks like by the way. You don't need to know a whole lot about this. Typically we won't work a whole lot with XSDs, but just so you can see what it looks like. (screen flashes to http://incubator.paache.org/ofbiz/docs/minilang.html)

00:19:25 screen changes to CustomerEvents.xml

So the plan is to basically move all of this reference stuff into annotations in the schema file so that as you are editing the XML file you can see those popup annotations without having to jump to a different document and such.

For example if I look at the simple-method here in the auto-complete, it'll pop up the annotation that came from the file there. This is just one of many XML editors you can sue. This is the Oxygen XML editor running as a plugin in the clips, but there are many others that support that sort of on the fly annotation documentation viewing. So that's what we want to do with all of this stuff.

00:20:06 screen changes to Framework Quick Reference Book Page 4

As a quick reference, for a quick overview, that's basically what the reference book is for. Starting on page 4 we have the first of four pages on the simple-methods (Pages shift through 4, 5, 6, 7, and back to 4). These are the other pages. There are some examples in here as well as a reference of all of the elements.

I guess I should mention the way this reference is really intended to be used. Just like for everywhere else this does not have full detail about what all of these things mean. The intention for using this quick reference book is to jog your memory basically. And it's especially helpful in a lot of these XML files with a good auto-completing XML editor you don't need to flip back to this so much.

I find this very useful when I'm in conversations with other people talking about how something might be implemented. I also find it very useful for something like the simple-methods, where I'm not sure of which operation I might need. Because they are categorized here I can look at the different types of operations; all the control operations are summarized here, all of the call operations here, the entity operations here.

00:21:27 screen changes to Page 5

And then there are details for all of these here. Here are the if operation details, all the control operation details, and so on.

So I can look at a summary on page 4 of all of the operations, including all the if operations and such, and then on 5, 6, and 7 (flips through these screens) I can look at examples.

All of the dotted line ones are examples, and the solid line ones are the details. These are the entity find ops, the misc entity ops, entity change ops, write values to the database, here's a summary of all the simple-map-processor elements.

00:22:05 screen changes to Page 4

That's a good way to use this. Okay so again typically with the simple-method when you're defining it every method has to have a name and a short description, so often those are the only two elements there.

00:22:17 screen changes to
http://incubator.apache.org/ofbiz/docs/minilang.html

Okay we've talked about the special context access syntax. This same syntax was originated for use in the simple-methods, but it has proven a very valuable part of the design of the various widgets; the Screen, Form, Menu, and List Widgets, as well as certain other things in OFBiz.

This is ${ } for variable expansion, . for accessing a map, [ ] for accessing list elements. And we talked about these in detail in the Screen Widget section. So in the video about the flexible syntax, let me look at the name of that real quick. In the Screen Widget section there is a shared flexible syntax video that covers that in more detail and has some examples.

00:23:19 screen changes to Page 13

And that by the way uses this page for reference, the shared page in here. There's a little reference block in here that shows a quick summary of how this works.

00;23:29 screen changes to
http://incubator.apache.org/ofbiz/docs/minilang.html

There's also some more verbose stuff in this document by the way if you want to read through that.

Okay call operations. This is one of the may categories and goes through and shows all the operations. Basically this document is organized much in the same way as the pages in the reference book over here (screen jumps to reference book Page 4, then back). So we have all the call operations for example here, quick reference to them. This document goes over the call operations in more detail.

We will be talking about these in the training videos to look at some of the details, but this can be a good reference. And here again note that all of this reference stuff will be planned to be stripped out of this document and embedded in XSD files so you don't have to jump over to this document, you can just have the annotations pop up in your XML editor.

And then of course as part of that effort we plan to flesh out some more of this as there are some newer elements that are not documented here. So the rest of this document basically just goes over the references and that sort of thing.

00:24:41 screen changes to CustomerEvents.xml

So some other things to keep in mind about simple-methods and the way they operate. When you're running through simple-method the operations, basically each element is an operation that's how we refer to them, and each one of them is executed in order. There are some control structures like if elements. There's an iterate element, iterate-map element, iterate over a map. You can see lots of ifs being used around in different places. I don't see any iterates being used but this is a fairly flat file.

But basically these operations are just executed in order as I said. This was a scripting language, and also being very much like a scripting language variables are not declared. Instead the first time a variable is referred to the simple-method execution engine would look at what type of variable that should be given the context that's referred to in and it will create it automatically.

For example right here we're telling it to copy the allowPassword field from the product store map, or generic value object, to the allowPassword field in the context. As soon as we tell it to put it there it's going to create this variable, and it's going to have the type of whatever came through here.

One thing I should note. This is a little bit older process that hasn't been updated yet. The env-to-env oeprations, field-to-field, env-to-field and field-to-env were needed before we had the flexible screen expansion stuff with that ${} syntax and the . syntax for maps. These were needed before that in order to get into maps and other such things, even though in a very limited way. But now they are not needed anymore and so the set operations is the preferred one.

In fact you'll notice if you're looking you'll still see these in some older scripts, these env-to-env, field-to-field, string-to-field is another one, and field-to-env and env-to-field.

00:27:06 screen changes to Page 4

Those operations you'll notice aren't even in the quick reference book anymore. We just have the set opera-

tion, and that can do what all of those could do plus a great deal more; it's significantly more flexible and powerful, and a bit more concise too so it's nice.

00:27:23 screen changes to CustomerEvents.xml

But anyway that's basically how variables are created as soon as, in this case for example I say I want to put the string customer in the roleTypeId field in the parameters map. And then it's going to create the roleTypeId field and put it in the parameters map. By the way this with the . syntax would be equivalent to this Parameters.roletypeid.

Okay and that's true of everything even when you're referring to a map. If I was doing something like telling this to put it in, the parameters map will already exist it always exists, but if I told it to put it in foo.roleTypeId, then it would recognize that map foo does not exist but it needs to be a map. That's how we're treating it as, a map because of the . here and so it will automatically create that.

We'll see that quite a bit more as we're looking through examples and things, but that's one thing to keep in mind. Another thing to keep in mind is that within a simple-method it has its own context, so calling a simple-method from a service or an event or any other way it will have its own context. But there is no separate context inside of if blocks or iterate blocks or things like this, so a variable declared inside an if block will be available outside it.

Sometimes that causes problems in more complex methods with iterations, so the thing to keep in mind is instead of relying on a variable you declare and define inside an iterate block, the best thing to do is...

00:29:25 screen changes to InventroyServices.xml

Usually it's best to clear the field at the beginning of the block instead of the end so that you're sure regardless of the circumstance that that variable in, so there's no old garbage sitting in the variable from a previous iteration or from outside that block.

So that's one of the consequences of this more loose scripting like variable scope that is used in the simple-methods. But what that gives you on the other hand is it saves you the trouble of trying to figure out which scope things are. The trouble of declaring a variable and setting it to a dummy value before you get into a loop just so you can put a value in it inside the loop, and then use the value afterward outside the loop.

So the clear-field element is something you'll see frequently used, especially in these iterations. These are the only examples of this I've seen in here.

So that is the purpose of that operation, to basically handle variables and because of the non protected scopes inside these code blocks, which is something you'll normally find in a language like Java or other structured programming languages. So to avoid that kind of thing you just clear the field if you're going to use it again, or use the same name again, which is basically what you're doing in any iteration.

00:31:00 screen changes to Advanced Framework Training Outline

Okay so that's the introduction to the MiniLang, the simple-methods and the simple-map-processors. We've talked about some of the elements of the simple-map-processors, and we'll go into those in a little bit more detail in the end.

We've talked about the general concept of the simple-method, so now is the fun part, we'll actually walk through and look at a bunch of these operations, and kind of talk through what each operation does, typical scenarios where they're used, and what the different attributes on each operation element mean. And then we'll look at various examples.

We've already looked at some examples of these as part of the framework introduction and certain other parts, but we'll look at some more in depth examples of the MiniLang as well, since mastery of the MiniLang is very important for working effectively in OFBiz. And I'm sure you'll find, as many others have, that working with the simple-methods is significantly easier and less error-prone and will allow you to implement the business processes you need with less code and less time and also make them easier to maintain.

One of the biggest complaints, I should note, about the simple-method is that debugging at times can be a little bit annoying, because there is no IDE right now that you can use to step through one line at a time to inspect variables and that sort of thing.

If you're used to using logging for debugging then you're in luck, because we have some good log operations that can be used in the simple-method and that can be a very effective way in general to debug and/or figure out what's going on in a simple-method. Especially in a context like OFBiz where we have an enterprise application that has multiple threads and such that are going on, and real time debugging can be difficult. Another problem with real time debugging is you tend to cause transactions to time out and such while you're looking around through the code and looking around at the values of different variables. And so it can be very time consuming at times, even more time consuming than throwing in a few log operations and seeing what happens as you run different scenarios. Okay so that's it for the general overview of the MiniLang.

00:33:40 screen changes to Page 4

And next we will start going through these simple-method operations and what these different things mean.

Part 3 Section 3.3

Simple Map Processor

Starting screen: Framework Quick Reference Book Page 7

Okay now in the last video we were looking at some of the general information about the MiniLang component, the simple-map-processor simple-methods. In this section I'd like to go over some of the details of the simple-map-processor.

This is page 7 of the Quick Reference book, and this is the last page about the simple-methods. The previous pages, 6, 5, and 4, go over the rest of the simple-methods, and we'll go over those in the next segment. For now let's talk about the simple-map-processor and look at the details there.

There are two ways you can set this tag, the simple-map-processor that's right here. One of them, as it mentions here, can be under the simple-map-processors tag, or under the simple-methods tag.

When it's under the simple-map-processors tag, this can be the root element in an XML file. With the Mini-Lang stuff you have two options for the root element, either simple-map-processors or simple-methods.

When you do simple-map-processors as the root element, you simply include a list of the simple-map-processor elements themselves underneath that. And each simple-map-processor has a name that's required, so when they're in a separate file you simply refer to the location of the file and the name of the simple-map-processor to run, and pass in the input and output maps when you run it inline, inside a simple-method file.

By the way when you're calling it externally you can do that from a Java method or a simple-method. When you are doing it inline, have a simple-map-processor inline inside a simple-method file, that's what we were looking at in the last video, where the simple-map-processor element is simply called includedDirectoy under the call-map-processor element.

00:02:41 screen changes to Framework Quick Reference Book Page 6

Let's look briefly at the call operations. So the call-map-processor operation right here, you can either specify the resource and the processor name, but those are both optional, so an alternative to doing that is just including the simple-map-processor underneath, which you can see is also optional.

When you call it externally you specify the resource, the name, the in-map and the out-map. And the error-list-name if you want it to be something other than error_list, which is the default. When you're calling it inline then you leave off the xml-resource in the processor-name, and you just include the simple-map-processor tag underneath the call-map-processor tag. And it's that simple.

00:03:32 screen changes to Framework Quick Reference Book Page 7

Okay so in the simple-map-processors, just like in the examples we looked at before, we have a series of these make-in-string operations to assemble incoming strings or other variables into a single string that can be processed along with the other input variables. The main purpose of this is to combine various input parameters before beginning the processing, so to do that before conversions or validations and such.

Then the main tag, the real work for the simple-map-processor is this process tag. So basically you process one field at a time from the in-map, and tell the simple-map-processor what you want to do. So all of these sub-elements of the process tag, with the process tag you just specify the field, and then you can do various things.

So these are the validation ones: validate-method, compare, compare-field, regexp, and not-empty, and to move it to the output map you have the copy and convert elements. You can have as many of these as you want and in whatever order you want.

Okay before we go into the details about what each of those is for, let's talk about the make-in-string. We saw an example of this but basically with make-in-string we tell it the string that we want it to put the string in. And this will become an incoming field that we can refer to as a field in the process tag, as soon as the make-in-strings are done.

An in-string can have any combination of these underneath it: the in-field, the property, and the constant. An in field you just name one of the incoming fields, or one of the fields in the in-map to add to the string.

Property will load the value of a property from a resource bundle a properties file, with the given resource on the Classpath, and the name of the property in that file.

And constant you just put a string value right inside it. It doesn't have any attributes or anything, you just put the string value in there. It can be a space, it can be any character, whatever. So with a combination of these three things, and in-field, property, and constant, it puts them all together, pins them all together, and puts them in as a string in the named field here. And that just becomes an input field, an incoming field like

any of the other incoming fields in the in map to be used with the process element.

Okay so with the process element, as we talked about before we're basically processing one input field at a time, and we can do multiple operations on that input field with a single process tag by having any combination in any order of these various tags.

Let's talk about these ones in detail. The first one we're looking at is the validate-method tag. With this it takes the field that is being currently processed and converts it to a string and passes it to the named method in the given class. There is a validation class-util-validate in OFBiz, and this is the location: org.ofbiz.base.util.utilvalidate.

If you don't specify a class name, then it will default to that class name so you can more conveniently call methods in that class. And then basically the method that is in the class, whether it be this class or another one, will take it, have a string coming in, and will return a boolean, a true or false. If that method returns false, then the fail message or fail property will be used to add an error to the error list.

The compare element allows you to compare a variable, a field, to a value or a constant. So a value, the string value goes here in the value element. You can use the flexible string expander and such here. The format attribute is used in cases where you have something like a date string, and you want to do a comparison as a date or a number or some other thing that is not a string that needs to be converted.

This format will typically be used in combination with the type element here. And you'll have an operator, use any of these operators. This is a common list, same as you'll see elsewhere. Notice there is no field here like there is in the if-compare tag in the simple-method, because we're comparing this value with the field that is currently being processed, the field that is named in the process tag up here.

And again if that fails, if the comparison depending on the operator and such returns false or fails, then it will add the fail message or fail property to the error message list.

Compare-field is very similar to compare, except instead of comparing it to a value, to a constant that's specified in here, it compares it to a field name. The format, operator, type attributes, fail-message, and fail-property work the same way.

Regexp is for a regular expression, so it will validate the current field, convert it to a stirng if necessary, and then validate it against the specified regular expression. We use the Apache oro, regular expression evaluator to do this.

If it matches the regular expression then it will be true, if it does not match the regular expression then it will fail, and either the fail message or fail-property will be added to the error-message list.

Not-empty is just an assertion that the current field being processed should not be empty. If it is empty you get the fail-message or fail-property. Copy just copies the field currently being processed from the input map to the output map, and the name of the field it will put it in in the output map is specified here in the to-field attribute. Notice that this is optional with the 0..1, so you can have zero or one of these.

Basically if you do not specify a to-field name, then the name of the current field being processed in the input map is the same name that will be used for that field in the output map.

The replace attribute specifies it can be true or false, by default it is true. If you set it to false then it will look in the out-map, and if it finds a value of the same name already there it will not copy over top of that value.

Set-if-null by default is true. If the field coming in is null, and you set this to false, then it won't copy it over to the output map. By default it is true, so regardless of whether it's null or not it will be set in the output map.

The convert operation, for processing a field. Again you have the to-field that works the same way. It's optional, so if you specify a field name then that's the field name that will be used in the output map. If you don't specify a field name then it will use the name of the field in the incoming map as the name of the field in the outgoing map.

This is doing a conversion, so you specify the data type that you want it to be converted to and you have all of these options. Just like with the copy we have the replace and set-if-null attributes.

And then one additional attribute that goes along with the type attribute is the format attribute, and this is optional. If you specify a format it will use that format to convert the incoming field to whatever outgoing field it is. If you're converting to a string then it will be the format of the string going out.

Plain-string by the way just does a to-string on the incoming object to format it the default way that that object is formatted, based on how the object is implemented. The format is also used going from a string to a date or time or any of these numbers, timestamp, boolean and so on, to determine how the incoming string should be formatted. If you're using a nonstandard date-time or timestamp format for example, or some sort of a nonstandard number format, then you can specify the format that you're using here.

And again we have the fail, just like with the conditions on the convert, even though it's not doing a validation

it's doing a conversion and that can fail. If it does than you get a fail-message or fail-property.

Or you can use these tags to specify. Actually you have to use one of these tags to specify what the error message will be if the operation fails, and that is the error message that will be put on the error message list.

Okay and that is it for the details of the different tags in the simple-map-processor.

Part 3 Section 3.4-A

Simple Method General Details

Starting screen: Advanced Framework Training Outline

Okay in this next section we'll start with details about the simple-method. We'll talk about exactly what different operations are used for, and what the different attributes and such on them mean. We'll start out by going through a couple of quick examples to remind you of the context these operations fit in, and then we'll go over the details.

00:00:36 screen changes to Framework Quick Reference Book Page 4

So let's hop over to the Quick Reference book. This is on page 4, which is the first of four pages about the simple-methods. Let's hop down to the bottom, look at the simple-method example.

These are actually some simplified Create Update and Delete methods from the example-services.xml file. We have a create example, update example, and delete example.

There's some permission checking going on, which checks the permission and adds a message to the error message list if the permission fails, which would be caught here. The same thing is done very commonly at the beginning of a simple-method, so you see the same thing here for the update, and here for the delete.

Now for a create. The typical pattern for a sequenced id, where the id is, rather than being passed into the method, generated on the fly and then returned.

First we make the value object. This is basically just a factory method to make the value object, and the variable name will be newEntity. We'll put it in; this is for the entity-name example, that's the name of the entity the generic-value-object will represent.

Then we do a sequenced-id-to-env, and tell it the sequence name, and where to put the sequenced id, just newEntity.exampleId. Then we take the field and put it into result. This is an operation that is only used when a simple-method is called as a service.

We specify the field name where the value will come from, and the name of the result to put it in. Then we set all non primary key fields (set-nonpk-fields) from the parameters map on this new entity, this generic-value-object. We call the create-value operation, which actually does the create of the insert in the database to pass it which value we want to do that on. Then these last couple of fields are part of the status maintenance pattern, and all we're doing is basically saving status history for this new example record.

So in this first operation, set-service-fields, we tell it the name of the service, the incoming map that has the field to copy from, and the map to copy them to. It's going to take all the incoming files for this service, or all the incoming service attributes, and look for fields with the same name in the incoming map and copy those onto the outgoing map. Then that same map is used as the in-map here for calling the service with that same service name that will create an example-status-record.

The update is fairly similar. It does an entity-one to look up the examples, and saves it here. This field-to-result is part of the status-pattern, returning the old statusId so you can trigger on it. If the status that was passed in in the parameters is different form the old statusId you can use the ECA, Event Condition Action, rules which we talked a little bit about in the Framework Introduction. We'll talk about those in more detail in the service engine section.

But basically that allows you to create an ECA rule that will execute only when the status is being changed. So that's the reason we return the old statusId. There's some more code in here that takes care of status management that you can look at in the ExampleService.xml file.

But the important parts of the pattern for update are grab it from the database, set the non primary key fields, and then store it back in the database. And for a delete grab it from the database and then remove it.

Now the reason we have these; it would be possible to roll all of these into a single operation, like the reading from the database and removal, the reading from the database updating fields and removal. But this is a good example of why we don't; there are often different things that we want to do in between.

Even if that's not the case for the initial implementation, these operations are designed this way to be in separate steps so that customization requires very little change to existing code, just adding new code if needed.

Okay so there's a basic overview of that certain type of simple-method, and we'll look at some examples later. But now let's start looking at some of the details, now that you have an little refresher on how these things flow.

On this page we have details about the simple-method itself; all of the attributes on it, and all of the categories of operation, are right here. Then we have some of the general elements that are reused in various places down here.

Over in these two columns, in these four boxes, we have the different groups of operations: the general operations, the environment related operations, the event operations that are only called when the method is called as a control servlet event, the service operations that are only called when the method is called as an OFBiz service, and other general operations.

Call operations are for calling other methods, whether they be simple-methods, map-processors, BeanShell scripts, Java methods, class methods, or object methods. The last create an object related to the call-object method, then call a service or do it asynchronously, and also set service fields to facilitate calling the service.

Control operations. These are all for the iterate error checking operations, so the explicit return operations.

Various if operations. These are the basic If operations that are used in different places. These are some of the higher level operations: if, search, check-permission, check-id, these sorts of things.

These are the entity operations. There are typically quite a few things you do in a simple-method related to a database, so these are all of the database related operations, everything from the sequencedId and the subSequencedId operations down through different query operations. These are different things to make, create, store, and remove other GenericValue related operations, and list entity-list related operations and some transaction related ones.

We'll look at all of those in detail in upcoming parts of this section, but first let's go over the attributes on the simple-method and talk about each of them in detail.

We know the method name is required for all simple-methods, and this is how we refer to simple-methods, through its name. This is just a short description that's used in error messages and such things.

This is an attribute to specify whether a login is required, or in other words if it is required that a user be logged in in order to call this simple-method. Notice that by default it is true, so if you do not want to require a login for a user calling a method you need to explicitly set it to false.

Same with use-transaction; by default if there is no transaction the simple-method-engine will begin a transaction. If you do not want it to do that, if you do not want your simple-method to run in a transaction, or do anything with a transaction, you can explicitly set that to false.

The default-error-code. This is for when you return with an error, which is basically down here like the check-errors tag. When it returns you can explicitly specify the return code for check-errors as well as for return, but if you don't specify a return code here it will use the default error code, which you can see the default value for that is error.

There's also a default success code. If it gets to the end of the simple-method, or you use the return operation without specifying a return code, then it will use the value of this attribute, or if none is specified the default here is success.

This is the name of the incoming map that will contain all parameters. This is not typically changed but can be. The default here is parameters with an S, as you can see down here with the example, the event-request-object name.

When this is called as an event there are various objects that are put into the context, so these are the names to give to those objects. There's the http-request object, the http-response object, and the response-code, when it's called as an event. And finally the name of the context-field or variable for the error message, for an event message which is like the success message.

These are things that you will typically not override; you'll just use the defaults. Actually most of the intent of having these attributes in here is to document some of these different things that are available in the context for your use.

Like those event ones there are some service ones.

The service-response-message; this is like the response code here for the event. Error-message; this would be like this error message for the event. Error-message-list is just a variation on the error message for multiple messages.

The error-message-map is so that the messages can have a name value pair. This is actually part of the error-message general management for service with the OFBiz service engine. It can have a single error message along with or these can be independent or used together; a single error message or list of error messages, and a map of messages which basically has for exceptions for examples you could have the key be the name of the exception, in the value could be the exception message.

Success message for service goes here. This is very similar to the event message for an event. Success-message-list is kind of like the error-message-list, you just have multiple of them.

So these are other things again that you will typically not override, but they're there for your convenience

when using these things, kind of lower level things that are available for simple-methods.

And some other ones. These are a little bit more standard; the name of the variable where the locale is put, the delegator for the entity engine generic-delegator-object, the security object that can be used for checking permissions, the name of the dispatcher, the service engine dispatcher, and the user login, the name of the variable where the currently logged in user login generic value object will be put.

Okay and that is it for the attributes of the simple-method. Again basically what this is representing here for all the sub elements of the simple-method, is that you can have zero to many of all operations. All operations consists of all of these different groups of operations here.

In the next section we will talk about these operations, basically go through them in order here, as they are in the Reference Book. We'll start by talking about the If and Control operations.

Part 3 Section 3.4-B

Simple-Method: If and Control Operations

Starting screen Framework Quick Reference Book Page 5

Okay we're now continuing on with the details of the simple-methods. This is page 2 of 4 on the simple-methods in the quick reference book, page 5 overall. In this page it covers the various if-operations and the conditionals, as well as the control operations.

So let's talk about first the big master if-operation. While it's possible to use all of these if-operations individually, for more flexible and powerful conditionals we have this very simply named if-operation that can wrap around them.

So if will consist of a condition, a then, if the condition evaluates to true. If it does not evaluate to true we can have a series of else-ifs, followed by zero or one else, to be executed if none of the other resulting blocks are executed, so if the then block is not executed and none of the else-if blocks are executed.

The condition will have underneath it just a single element, either one of the combinedConditions or a basicOperation, a basic if-operation. The basic if-operations are things like this validate-method, instance-of, if-compare, if-regexp and so on.

The combine operation or combine conditions are these ones: OR, XOR, AND, and NOT. So the point of the condition here is that whatever is under it will reduce down to a single true/false value, and it will evaluate to either a true or a false.

So as we talked about these other ones, the then can have any other operation under it. The else is the same way; it can have any operations under it. The else-if is different; it has a condition and, which is just the same as the condition under the if tag, as defined here. And it has a then, which is also the same as the then under the if tag as defined here, just a container for a bunch of operations.

Okay then for the combine conditions that will combine other conditions together the or, xor, and and are all very similar. They can have one to many of any of the if conditions. The if conditions are a group that consist of either the if-combine conditions or the if-basic operations.

The if-combine conditions; this is the same as up here. These two, these are all of the if conditions together. So the if-combine condition is just this guy defined right here, the ifBasicOperations. For the actual definition of that we can hop back to this previous page (flips to Page 4) and here are the ifBasicOperations. They're basically these guys over here, starting up here, and going down through the if-empty and if-has-permission.

The funny thing is they do not include the if-not-empty. The reason we don't have the if-not-empty is because the if-empty combined with the not will make an if-not-empty. So that's just there to simplify it a little bit.

The not tag is like the and, xor, and or, in that it modifies what it contains, except it will just have a single one of the if conditions. And again the if conditions shown here, all of these are basically defined this way; the ifCombineConditions and the ifBasicOperations, as we just went over.

So the OR, in order for it to be true, just one of the elements underneath it needs to be true. In xor only one underneath it can be true. For an and all of them underneath it will be true.

Now as for the order these are evaluated in. For the or it actually follows the process of going through and evaluating each condition until it finds a true one, and then it will return true. If it finds a true one it will not evaluate the remaining ones after that. So you can code things to count on that way of evaluating conditions. If it does not find any true it will go all the way through, and if they're all false the or will be false.

XOR is a little bit different. It will evaluate all of the conditions underneath it because it needs to make sure that one and only one is true. Actually I believe if it finds two that are true then it will return immediately after that. But if it goes through and finds them all false then it will continue to evaluate all the way through.

The AND is similar, except that as soon as it finds a false then it will evaluate to false. Otherwise it will go

through all of them, and if they are all true then it will evaluate to true.

The NOT as we discussed can only have one condition underneath it, and it will simply reverse the boolean status of that condition. So that's basically how the if element is structured. Let's look at an example of that.

This guy over here, and I'll zoom in a little bit so the text is more clear. So we start out with the wrapping if element, which opens there and closes way down at the bottom. We have a condition. Because we want this condition to be more complex, the top level of the condition is an OR, which ends here.

Inside of that we have 2 and blocks; there's the opening and closing of that one, and the opening and closing of that one. Inside the first and block we have a not if-empty, so we're saying that this field parameters.estimatedshipdate should not be empty. And then we have an if-compare-field which means that this field, parameters.estimatedshipdate, should not be equal to lkdupval.estimatedshipdate.

Down here in the second and block we have a similar set of conditions. So we're saying this field in this map, parameters.originfacilityid, should not be empty. And another if-compare-field, so parameters.originfacilityid should not be equal to lookedupvalue.originfacilityid.

Again this is kind of like the calculate operations that we'll be going over in a bit, and other things in general with the simple-methods. But the way these tags are designed, it removes a lot of ambiguity that comes into play with programming and scripting languages like Java and Javascript and C, and various things like that where you have to use parenthesis to clarify the order that the conditions should be applied in.

With this sort of thing it's very explicit; we see an or tag and we know that everything under that will be ored together. We seen an and tag and we know that everything under it will be anded together. Logically this is a very natural progression, with tags like this.

But to read it in English we can start from the bottom and move up, which is how it will be phrased in English. We're going to say if parameter.estimatedshipdate is not empty, and parameters.estimatedshipdate is not equal to lkdupval.estimatedshipdate, or parameters.originfacilityid is not empty and parameters.originfacilityid is not equal to lookedupvalue.originfacilityid, then the condition is true.

So stating it in English you basically have to have the parenthesis around the ands to clarify that they should be ored together. But in this format how the if tag is organized, that is explicit in the way the tags are structured.

Okay so that's the way the condition tag fits together. It will have a single conditional or combine operation underneath it, so that we get a single positive or true or false value out of it.

Okay and then the then. This is what will be executed if the value is true, and this can be any sequence of operations just as normal. So it's doing some entity-one lookup in the database. And it sets a few things, set-service-fields, to prepare for a service call, and then calls a service for updating the work effort.

So this is basically saying if the estimatedShipDate has changed OR the originFacilityId has changed, then we need to update the estimatedShipWorkEffort. That's basically what that code does, and that's an actual example from the shipmentservice.xml file.

Okay going back to these operations. We talked about the if operation, how it is structured.

The next one is the assert operation. This is mainly used for testing, and for writing simple-methods that are meant to be used as part of a test suite. So each assert operation will have a title that can be used in the report for the testing. These can be used in the normal code.

By the way if you want to build assertions into your code this one way of doing checks on various things, because you can have oncediotnison, arbitrary conditions underneath it, and in case of failure it will simply report all of the conditions that failed. And it will add messages for those to the error list.

So that is the assert operation there. The basic idea is that this has conditions under it just like any of these operations, like the and or xor, and it has one to many of these if operations underneath it. So you can put any of the combined conditions or any of these if operations under it, and it will simply assert that all of them are true. It will just run one of them at a time, assert that each one is true, and if it is not then it will show a message to that effect. This is mostly useful for testing because the messages are targeted at a programmer, and not really at an end user.

Okay check-permission is often used to compliment the if-has-permission operation. So rather than treating it like an if that has an else or that can be run when run as a top level operation, or that can be used inside of the main if, the guy up here, check-permission is just run as an operation on its own. It does the same sorts of things, such as do the permission and action.

So this would be catalog_create, or catalog would be the permission and create would be the action, and the error-list-name to put the message on if the permission fails.

Accept-userlogin-party. Basically if that tag is present then it will accept-userlogin-party, rather than requiring that the user have the permission. If the userlogin object has a partyId that matches the partyId in the envi-

ronment with this name. Here's the details of the accept-user-login party tag.

If userlogin.partyid matches the name of the variable defined here (if not specified it defaults to just partyId), matches if this tag is present and that partyId matches, then it will not require that the user have the permission. It will allow them to run the operation even without it. So that's often used in cases where you want to allow a user to for example see their own order, or update their own contact information.

Okay alt-permission allows you to specify a number of alternate permissions. This tag basically has the same attributes as the check-permission tag, permission and action, which is the suffix to permission basically. And with those alt-permission tags if the user has any of the alt-permissions, even if they don't have the primary permission, then they have permission.

And then no error message will be added to the error list named in the error-list-name, which usually defaults to error_List. Unless you want to have multiple error_lists, the best thing to do is just let that default there. The error message itself will come either explicitly specified here in the fail=message tag, or loaded from a properties file with a fail-property tag.

Okay check-id is kind of like check-permission. It is a top level operation that is simply used to check an ID value, following a pattern of many things. It has a field-name and map-name, and map-name again is usually not used because of the . syntax that's available now. So the named field will be checked to see if it is a valid ID, and then if it is not then the fail message or fail-property value will be added to the error_list.

A valid-ID means a number of things. This checks it for validity, or most likely for being able to be stored in the database and passed in an http URL parameter and those sorts of things. So it checks to make sure that certain characters are not present in the ID string, things like spaces and % signs and those sorts of things.

Now we've talked about some of these general if operations, let's talk about the specific if operations. And again these can be used, these are basically single conditionals that check certain things. And any of these can be used, any of these that attr wih if- can be used independently as top level operations. So you can just drop them in your code anywhere that operations can be put, and they'll evaluate it in that context.

If they evaluate to true then the operations under that element will be run. If it evaluates to false and there is an if tag, then the operations under the else tag will run. And that's true of all of these: instance-of with this else tag, or allOperations if-compare with its allOperations or else tag, and so on.

Now when it's used underneath this if tag, or underneath the condition element of the if tag, then they're a little bit different. They will not have an else tag and they will not have any sub elements. They will simply have in-validate-method, for example, and it will close at the end of the tag. So it will have a />, rather than other elements underneath it and a separate closing tag. And that is basically what we saw here, where we have if-compare-field that is self closing.

Okay if-validate-method. A lot of these actually are very similar to the simple-map-processor process validation things that we looked at. So if-validate-method is very similar to that, except of course each of these need to specify the field that they'll be operating on.

Here's our typical field-name/map-name pair. So for a given field it's going to run a method. This is going to be a Java method that will run on a certain class, same as the simple-map-processor. If you don't specify the class it will be org.ofbiz.base.util.utilvalidate, but you can specify your own class there and it will call the name to method in that class.

This method will take a string that's coming in. So if the field is not a string it will be converted to a string and passed to the method, and then that will return a boolean, a true or false. If it returns true it runs the operations under here, or returns true back to the if condition.

If it returns false it will run the else operations if there is an else tag, or if this is part of the master if over here it will evaluate to false as part of that condition.

If-instance-of basically checks to see if the field is an instance of the name class. This is just like the instance-of operator in Java.

If-compare compares the named field to a certain value. You can specify whichever operation you want for that. You must specify an operator, of course, so that it know whether it's an equal, greater or less, or whatever.

You can specify a type to convert both the field and the value to if needed. Before the comparison is done that's what the type will do. And the format goes along with the type, just in case you have a string format that is different from the default for a comparison to or from one of these objects.

If-compare-field is very similar to if-compare, except instead of comparing the named field to a value, it compares the named field to another named field. It has the same operator attribute and the same type and format attributes. So if a type attribute is specified then it will convert both the field and the to-field to that type. If not it will convert them both to a string and use that for the comparison.

Okay if-regexp will run a regular expression, specified in the expr attribute on the given field name. If the vari-

able in that field, converted to a string if necessary, matches the expression then it will be true, if not then it will be false.

If-empty just checks to see if the given field is empty. If-not-empty is the opposite. To be empty the field can either be null or it can be a non-null empty object; like a zero length string, a list with zero entries, or a map with zero entires. And any of those sorts of things will constitute being empty.

If-has-permission checks to see if the current user for the method has the permission and action. These are the same permission and action pair we saw over here in the check-permission tag.

Okay and as an example we saw one just a bit ago where you'd have a prefix like example_create for the action. Or _update, _view, _admin, or you can have just a permission without an action if there are no extensions to the permission.

And that's it for the if operations. Those are the conditionals in the simple-methods, and they're that simple.

Now for the control operations. These are basically for flow control. Technically an if is also a flow control operation, but these are the non-if control operations.

So we have an iterate to iterate over a list, specify the name of the list, and each entry and the name of the variable to use for each entry in the list. As it iterates it will run all of the operations underneath the iterate tag for each of the entries in the list, and each of those of course will be placed in the variable named in the entry-name attribute right here.

Iterate-map is very similar, except instead of iterating over a list it iterates over a map. Now a map has a number of entries just like a list, but each entry will have a key and a name-value pair. So you can specify the name of the variable to put the key in and the name of the variable to put the value in. And it will run all of the operations underneath the iterate-map-element for each of the entries in the given map, setting the key for that entry and the key-name variable, and the value for that entry and the value-name variable.

Loop, rather than iterating over some sort of a structure, will simply loop a certain number of times. The number of times to loop is specified in the count attribute, and it will put the current count in the field attribute, or in the variable named by the field attribute. So if you want to loop ten times you say count=ten, and field=count for example. First time through count will be zero; last time through count will be nine.

Okay check-errors. This is the tag that we've talked about in a number of cases where we have a general pattern of doing validations and such at the beginning of a method. And then we get to the check-errors tag, and if there are any errors in the error list then it will return immediately.

If it does return it will use the error, the value of the error code element for the return code, and the default for this is error. This goes back to the service engine, and other things in OFBiz where we have the main return codes of success and error. So this just returns error by default, but you can specify alternatives to that in this attribute. Okay these elements, error-prefix, error-suffix, message-prefix, and message-suffix.

00:28:22 screen changes to Framework Quick Reference Book Page 4

These are actually defined I believe on the previous page over here. Error-prefix, suffix, but also corresponding things like success-prefix and suffix, and message-prefix and suffix which we just saw there, the default-message we've seen already, and the usages of fail-message and fail-property.

But anyway basically these error-prefixes and suffixes specify a resource and a property. So it loads a property for these prefixes and suffixes.

00:28:54 screen changes to Framework Quick Reference Book Page 5

And what it does in effect when there is an error message, or a list of error messages, it will add a prefix and a suffix to each, same with the message ones will be considered here as well. Okay the add-error tag is meant to add an error message to the list.

So it has this same sort of error-list-name attribute as many others, with the default value of error_list. And it has a fail-message and fail-property, just like many of the conditionals, like check-permission for example. If they do not have the permission then it will add the fail-message or fail-property to the named error-list. So this just explicitly adds a message to the list. This is often used inside conditionals, or the master if-conditional if you need more flexibility.

Okay when it reaches this tag it will just immediately return, and it will return using the response tag, which by default is success. You can return any string you want from a simple-method, which is how things work in both service and events. If this is called as a control servlet event or a service engine service, they both expect a return response code string.

So in check-errors, if there are errors by ECA default the response-code or the error-code, is error. And for return it by default is success, although you could set this to error or any other string to use as a return code. Okay and that's it for the control operations and if opeartions for the simple-methods.

00:30:57 screen changes to Framework Quick Reference Book Page 6

Next we'll move on to other things that actually get stuff done, general operations, call operations, and so on. The other things, event and service specific things, and for the more complicated ones like calculate we'll look at an actual example.

00:31:28 screen changes to Framework Quick Reference Book Page 7

And then we move on finally to the last page which has all of the entity operations on it.

Part 3 Section 3.4-C

Simple-Method: Call, General, and Other Operations

Starting screen Framework Quick Reference Book Page 6

We are now continuing with the next set of groups of operations for the simple-methods. This is page 3 of 4 of the simple-methods reference in the quick reference book. (change to page 5, then back to 6) In the last video we covered the If and Control operations, So now we're getting in to some more of the specific operations that actually do things in a simple-method.

Let's start with the call operations that call out to other types of logic, and then we'll go over the detail things that do things right inside the simple-methods with these other types of operations.

Okay so the call-map-processor we've already talked about in the context of the simple-map-processor you can call out to an external simple-map-processor. This is specifying the XML resource and processor name, or you can have alternatively an inline simple-map-processor right underneath the tag. It has an in-map and out-map and for any errors that happen it can specify the error-list-name to use.

Call BSH will call a BeanShell script. This has two options as well. You can specify the resource where the BeanShell script is located. Typically it will be a Classpath resource, but it can be other in other places. There's also an error-list-name so you can specify the where to put error messages that are generated during the evaluation of the BeanShell script.

This tag is the call-bsh tag. In addition to being able to refer to an external resource you can actually put the BSH script right underneath the call-bsh tag. So rather than self-closing the tag you would just use the > and then have a /call-bsh tag to close it off. It's very common to use the seed data construct in XML in this so that you can use characters flexibly without worrying about whether they are valid in an XML file, like < and & and all of these sorts of things.

Okay the next one, call-simple-method. This calls another simple-method in either the same file if you just specify the method name, which is required, or in another file. You just specify the resource, the location of the simple-method XML file. That is optional, so if you don't specify it it calls the method in the same file.

Now with the call-simple-method this will do an inline call, or in other words it uses the same context. The simple-method has read and write access to the same context so it does not protect the context. If the simple-method that you call creates a new variable it will be available in the calling method when the call-simple-method is finished. In other words it's just like taking the operations in the other simple-method and inserting them verbatim in the current simple-method. So it is very literally an inline simple-method call.

Now there are a few here that are used for calling methods on Java classes on object, call-context-method, call-class-method, and create-object. Call-object-method will call a method on an existing object, so you specify the field-name and optionally the map-name for the field where that object is located in the current context, the name of the method to call in that object, the name of the field to put the return in, and optionally the map to put the return in.

Again these are field/map pairs, with the . Syntax. The map-name is typically no longer necessary because you can do map-name.field-name in the field-name attribute, but those are still there for legacy purposes. But the main important ones are obj-field-name, the method-name, and if you want to do something with the return value then the ret-field-name. If you leave off the ret-field-name it will just ignore the return value. To specify the parameters to pass to the method that you're calling you use these two tags; the string tag and the field tag.

00:05:27 screen changes to Framework Quick Reference Book Page 4

These are defined up here on the first page of the simple-methods. Down here in the shared area we have the string tag and the field tag. With the string tag you simply specify a value. With the field tag you specify the name of the field, optionally the name of the map that it's in, and the type of the field.

00:05:53 screen changes to Framework Quick Reference Book Page 6

Now the type for the field is optional in general for the field tag. But when used in the context of a call-class-method or call-object-method, when the field is used in these places or in the create-object for passing it to the constructor of the object then the type is required right now. So the type needs to be explicit. The reason the type needs to be explicit there is because it will use these string and field tags to find a method that map with the given name and the given parameters coming in, that matches these requirements so it knows which

method to call. That's true of the object-methods which call a method on an object.

The class-method is only different from an object-method because you specify a class name instead of a field name for the object in memory. This will call a static method on that class rather than creating an instance of it. So you specify the name of the class which is required, the name of the method on that class, which needs to be a static method or a class-method in Java terms. Then you have the same return-field and ret-map and the same string and field tags that can go underneath it, and they need to be in the proper order for all of these: call-object-method, call-class-method, and create-object.

The string and field tags can be intermixed, but they need to represent the actual order of the parameters for the given method. The create-object operation is kind of meant to go along with this call-object-method, although it can be used more generally just to create an object at any time you specify the class-name of the object, the field-name to put it in, and optionally the map-name.

By the way I should mention here, with the field-name it is optional. If you do not specify the field-name it will create the object but not put it in the current context. Sometimes there are things that the constructor of an object will do that method that you want done without keeping a tab on the object itself that's created. So you can just leave the field-name blank.

Class-name is the name of the class to create. Typically the class will have a constructor. If you don't pass any string or field values in here it will use the default constructor with no arguments. Otherwise you can have these string and field tags intermix just like these other Java call-methods, and it will use those to pass those parameters to the constructor of the object or class being created.

Okay so that's it for these three Java related methods for operations. Now for the other operations here. These ones are specifically for calling a service. The main one here is actually call-service right here, where you typically specify the service-name and an in-map-name.

The in-map-name is optional. If you're not going to pass any parameters to the service than you can just leave off the in-map name, although typically in a service tag you will see a service-name and the in-map-name passed in. There are other options that are helpful here when calling a service.

Include-user-login by default will include the user login, so if there is a user login for the current simple-method it will pass that in to the service. If you don't want it to pass that in you can just set this to false.

Break-on-error. If there's an error in the service by default it will stop the current simple-method and return an error message that came from the service it called. If you don't want it to when there's an error you can just set that to false. Interpreting the return of the service that you're calling. When the service returns you can specify which code to interpret as an error, and which to interpret as success. The error-code. These are just the default OFBiz ones, error and success.

Okay error-prefix and suffix, success-prefix and suffix, message-prefix and suffix, and the default-message. These are all used to kind of decorate messages that come back from the service. So if the service returns error, success, messages, then the error prefix and suffix will be added to the error ones, the success prefix and suffix will be added to the success ones, and the message prefix and suffix will be added to all of them.

There's also a default message that you can specify for the case where the service does not return a message, you can just specify a default-message to use as if the service had returned that default-message. Okay so you can have zero or one of all of these different message related tags. These ones you can do in any order, zero to many as is specified there.

Results-to-map will take all of the results of the service, the outgoing maps from the service and put them in a map of the given map-name.

Result-to-field is defined down here, specify the name of the field in the result and then the name of the field in the context you want to put it in, and optionally the name in the map. There's a field-map there. If you don't specify a field-name then the result-name will be used for the field-name, that's the name of the variable that will be created in the current context for the value of that result.

Result-to-request is an event specific one, so if the simple-method is called as an event then you'll get result-to-request, then this will run, same for the result-to-session. It takes the result with the given name and puts it in a request attribute with the given name here. Again the request-name is optional. If you leave it off then it will put it in an attribute with the name of the result-name.

Result-to-session is similar. We specify the name of the session attribute that you want it to put the value in. If you don't specify one it will use the result-name. Result-to-result is used when the ucretn service is being called as a service. So it will take the result of the service you're calling with the call service operation and it will put it in with the result of the current service. So result-name is the name of the result in the service that was called using the call-service tag.

And the service-result name is optional. If you specify it that's the name of the result of the current simple-

method being called as the service where it will put the value. If you leave that empty or do not specify a service-result-name it will use the value of the result-name for what it puts in the current service results.

Okay so those are the different call operations for calling out to other simple-methods, BeanShell scripts, methods on Java classes or objects and calling other services.

Let me talk about this. This is call-service-asynch, a variation on the call-service-operation that will call the service asynchronously. This one specifically ignores the results, so it'll call the service asynchronously and immediately continue in the simple-method. Just specify the service-name, the in-map-name or to include the user login or not. It doesn't have any of these things related to the return because it just sets up the service to be called asynchronously and then moves on.

Set-service-field is a convenience operation. You pass in the name of the service required, the map to pull the field from, and the map to put the fields in. It will look at all of the in attributes for the service with the given name and it will copy those fields if present from the incoming map to the outgoing map.

Okay that's it for the call operations. Let's move on to the general operations. These general operations here are mostly for moving data around granularly in the environment and working with variables and such. The most common one you'll use is the set operation.

If you used simple-methods in the past you'll probably be aware of the predecessors to the set operation, operations like env-to-env env-to-field field-to-env, and field-to-field. Those are all now deprecated and replaced by the set operation, which is more concise, more consistent all around, one operation instead of four, and more flexible. This is because in addition to just moving a value from one field to another field you can also take a value, just a string constant or a string that is made up of a mixture of constant and flexible string expansion variables, the ${} things, that will be put in the field.

You can also specify a default value in the case that the value evaluates to an empty string or the from field is null or empty. Then the default-value will be used. Again you can use the flexible string expander here, the ${} syntax and such. It can also do a type conversion, so going from whatever type the source data is in, which would be a string value or whatever the variable type is for a from field, it can convert that to any of these types before setting it in the target field.

It can also specify set-if-null and set-if-empty. Set-if-null means if the source variable is null it should set it on the new field, and the default for that is false. If the source variable is null it will leave the target field as it was.

Set-if-empty tells whether to do a set if the source value, either from a value or from a field, is empty, and empty-string an empty list or a null value. In this case it's set to true. If you don't want to set, if you want it to leave the target field alone when the source is empty, then you need to set this to false, right there.

Okay the field-to-list operation basically takes a field, given a field-name, and a corresponding map-name which is optional. This is another file/name pair. So it takes the named field and it puts it in the given list. Note that we've left this tag in place because it can be a little bit more clear, but with the set operation you can do the same thing by having the target field use the flexible syntax with the [ ] braces to denote that the target field is a list.

00:19:48 screen changes to Framework Quick Reference Book Page 13

For what that looks like we can look at the shared elements page. This is page 13, and down here in the flexible syntax we have these square braces [ ] for denoting that it is a list. If we wanted to put it at the end of the list we could do something like this, products with empty [ ]s.

00:20:13 screen changes to Framework Quick Reference Book Page 6

Basically you'd just have a field here that had products with [ ] braces. It would treat that products variable as a list instead of a single value, and it would just append the value to that rather then replacing the variable in the context. So the same thing that you do with field-to-list can be done with set.

List-to-list is a little bit different. It will simply take all of the elements of the source list and add them to the target list. If the target list already exists it will just append them all to the end. If not then the target list, the to-list, will simply be a copy of the from-list.

Map-to-map is very similar, copies all of the elements from the source-map to the target-map. If the target map exists then it will add the elements one at a time to that list, since the elements in a map have key value pairs. If there are any elements in the target map with the same key as an element in the source map, the element in the target map will be replaced by the entry with the same name in the source map.

Okay the string-append operation tells it the field that you want it to operate on. This is the target field where the value will be put, and this is the string to append to that field. So if the field does not exist then it will create a new field with this string value. If it does exist then it will append this string value to the end.

This has an ARG-list-name as well as a prefix and suffix, so the prefix and suffix are just strings that will be used a a prefix and suffix around the string, and the

ARG-list-name is used to insert values from a list into the string using the object in the standard Java library that does this sort of string expresison with a { } brackets and a number, no dollar sign.

This pattern of the arg-list-name with the prefix and suffix is something form the early days which is still supported, but the best thing to do here is just use the flexible string expander which is far more flexible and powerful. So you can have the prefix variables to expand and everything all mixed into one string.

Okay string-to-list will take a string literally that can also have a flexible string expander and such in it, and it will add it to a list. Note that you can have an arg-list-name here just like you can here, for using the standard Java style argument list where you have in the source string numbers inside of { } brackets that represent the number the index in the argument list to insert at that point.

Okay the to-string basically takes the given field and converts it to a string. You can specify a format so based on the type of object that it's coming from, if it's a date or a number or whatever you can specify the format there. If it's a number you can also specify padding to use for numeric padding.

So if you want to create a number that will always be at least five digits then you could use a number-padding=five, and then it will take the field. If it's an integer number like the number eleven and numeric padding is five then the resulting string that will be put in the field would be 00011. So it just pads the number with zero, that's what the numeric-padding dose.

Okay clear-field will clear the given field name, basically just remove the variable. If there's a map-name corresponding with the field-name then it will remove it from that map. The file can also use the . syntax, the list syntax with the [ ], and all of those sorts of things.

If you remove something like parameters.productid, it will just remove the given field, and not the map that it was in. So it will just remove the productId from the parameters map, but will not remove the parameters map itself.

Okay first-from-list is kind of an alternative for an iterator. All it does is take the first entry from the given list and put it in the variable named by the entry-name attribute here.

And that's it for the general operations.

So moving on to the other operations. The main difference by the way between the General Operations and the other operations is these are mainly concerned with moving variables around in the environment and dealing with variables. The other operations are just miscellaneous things that are handy to have and the types of logic that are typically implemented with simple-methods.

The log operation is very helpful. This will basically send a message out to the the log file so you can specify the log level, which is basically required. And you can either specify a message as a string, or a combination of field and string elements.

These are the same field and string elements that were used in the call-class-method, call-object-method, and create-object things up here. They have the same structure, they're just used in a different context here. If you do use field and string elements underneath the log operation those will just be combined in the order they're presented to create the message string.

Those aren't used as much anymore, because now with the flexible string expander just having a message here with variables surrounded by a ${ }, and that's typically much more convenient for specifying the message that you want to generate. This is very helpful for debugging or for inspecting variables and such to understand what is happening in the simple-method.

Okay now-timestamp-to-env will create a timestamp representing the current date and time and put it in the named env variable.

Now-date-to-env is the same sort of thing, except instead of being the date and time it's just the date that it puts in there, an the data will be a JavaUtil data object. This one will be a java.sql.timestamp object, which is actually the more common one typically used in OFBiz.

Property-to-field will take a property from a properties file. The location of the properties file on the Classpath will be specified on the resource attribute here. The name of the property within that file to use is specified with the property attribute here.

The field to put that property value in is specified in the field name, optionally complemented by the map-name. You can specify it default so that if the property is empty or not found then it will use the default value instead.

No-locale you can tell it not to pass in the locale. This is false by default, otherwise it's double negative so by default it will pass in the locale. If you do not want it to pass in the locale just specify no-locale=true.

And this is the good old arg-list-name. If the property coming from the properties file has the in-text numbers surrounded by the { } syntax that is part of that standard Java string expansion, then you can use an arg-list-name to specify the list where it can find the arguments to insert there. You can also use flexible string expansion syntax here, as well as the default attribute.

Okay set-current-user-login. The value-name should point to a user login generic value object, and this will set that user login as the current user for this simple-

method. That user will be the current user only for the current simple-method, and of course anything that the simple-method calls, such as other simple-methods, other services, or that sort of thing. It will not affect whatever called that current simple-method.

The calculate operation. This one is a little bit more complicated, which is why it's conveniently at the end here. Calculate will do a calculation and put the result in the named field, in the named map if specified. Into the named field goes a value, the given type, and you can see the type by default here is double.

This was changed a little while back, it originally operated on a double internally.

00:31:25 screen changes to simple-methods.xsd

But it was changed a little while back so that it also supports a bigDecimal, and the internal operations are done with the bigDecimal.

00:31:37 screen changes to Framework Quick Reference Book Page 6

But the default value is still a double, just as documented here, so by default it will convert the number to a double. But it does also support the bigDecimal, and it uses bigDecimal internally for the calculations.

Decimal-format is just a format string that can be used to format, especially if you're converting it to a string. The value is just a string that will determine the format of the outgoing string.

Okay inside a calculate you basically have calcop and number, and zero to many of each of these, if you specify a calcop it can do. When you specify a calculate operation or a calcop, you specify the operation that it will do on the sub elements of the calcop. All of the intermixed calcop and number elements here will all be added together by default, that's the default operation for combining these. Any that are directly under the calculate tag.

Okay so the calcop tag basically has an operator: get, add, subtract, multiply, divide, and negative. So add, subtract, multiply, and divide are just the basic arithmetic operations. Get is basically just an add; it gets the positive value. Negative will just negate the value; it's kind of like an opposite get, in a way. Or in other words is a subtract, except rather than subtracting all of the sub elements from the first element it will just add them all together and negate the result.

Okay so just like the calculate operation up here that has any zero to many combinations of calcop and number elements, this has the same thing, any combination of calcop and number elements, zero to many of them. Number you just specify a literal or flexible string expander number here, and it will convert it to a number.

This might be somewhat confusing so let's look at an example. We have another operations example over here that's very helpful. By the way here's an example of the log. Let's zoom in on this a little bit. Here's a typical use of the log; level=info, message = all this stuff. So you have text, and then typically somewhere in the text you'll have ${ }, which is usually together by the way; they cannot have a space between them. This just spilled over to the next line because of the formatting here.

Anyway, inside the ${ } demarcation is the name of the variable to insert into the string. And an example of the return operation where we specify a successful response code. So this is kind of a giving up block of code; it just logs an error message and then returns.

Okay here's a calculate example. One thing to note about the calculate is that just like other things, just like the if tag and such in the simple-emthods, it's much closer to a reverse Polish way of denoting things than the typical English spoken way of denoting things.

Here's an example formula using parenthesis, basically in the typical English speaking way of things. A=B+(((C+X+2) all together * -D, all together /e. So in order to clarify that of course you need to have parenthesis in order to clarify the order of operations.

With a calculate tag, because the operations are all hierarchical, the order of operations will be explicit by how the calcop tags are organized. So this same formula, this B + all of this, can be represented in a reverse Polish notation like this. And this notation we have the operation and then in parenthesis comma separated list of the elements to be operated on there.

So this is going to add B plus a divider, and the divider has all of that divided by this. And that first part is just a multiplication of this, times -d, and then this plus is just adding these three things together.

That's the equivalent of this formula, so that's what we will put in our calculate operation. It's easier to do it when it's structured that way because this is the same hierarchical representation that the calculate operation uses.

So first to do the top level add. As I mentioned before any elements you put directly under the calculate element will be added together. So this calcop will be added to this calcop, and that's where it closes so those are the only two that are under that calculate element. So those will be added together, that's the default way of what it does.

Okay so all we're doing is getting B to be added. The next calculate operation is doing a divide, representing this guy here, and inside that divide the things we want to divide are a multiplication which starts there and

ends there. And we just want to get the variable E, so the result of this multiplication will be divided by E.

So the multiplication itself contains a calcop here and a calcop here, the first one adding these three things together, so we want to add C to X and the number 2. Those three will be added together in that calcop, and then in this one it will simply get the negative of the variable D. So we have that part of the formula.

And that's pretty much it. It will evaluate this formula, just as it would here.

Now this is a good place to note that obviously the simple-method syntax in an XML file is a bit more verbose than you would have on a string there. And so one thing to keep in mind is that it's very valuable to have an XML editor. This is always the case with the simple-methods with all of these operations, that it's valuable to have an XML editor that does syntax highlighting as well as automatic completion.

The one we looked at before, the Oxygen XML editor that I use as an Eclipse plugin, has not only auto-completion but will show annotations during the auto-completion. So you have built in live documentation to remind you of what things mean.

Okay that's the calculate operation. Let's finish this section off with the event and service operations. As I mentioned with these ones up here, result-to-request, result-to-sessions, and result-to-result, there are certain things in the simple-methods that will only run when they are called as an event or as a service. Those are the two ways of calling a simple-method, as a control servlet event where you have the HTTP request and HTTP response objects, and those make up the context by the way, and the alternative where the simple-method is called as a service or used to implement the service. In that case you have the context of the service coming in as a map, and the context a result map being returned back.

These first few, actually all of them except the last one, are in the event operations. These are operations will only be called if the simple-method is called as an event.

And this last operation will only be called if the simple-method is called as a service in the service operations.

Okay field-to-request will take the named field and put it in a request attribute with a given name. This is optional, so if you don't specify a request name it will put it in the request attribute with the same value as the field name.

Field-to-session is very similar except instead of a request attribute named here we have a session attribute that will name here, default the same; this is not specified because it's optional. But the field-name will be used for the session attribute name.

Okay request-to-field does the opposite of a field-to-request; it takes the given request attribute name and puts it in the field. If the request attribute doesn't exist or is empty then you can specify a default string to be put into the field.

Request-parameters-to-list. The request-name and the list-name are the two attributes coming in. Now the name of this element is somewhat confusing, because it actually just takes the value of a single request parameter. And in an http request a request parameter can have more than one value, so it will take those values, put them in a list, and put that list in the context specified in the list name. If you don't specify a list name it will default to the value of the request-name, which is the name of the request parameter that the data is pulled from.

Session-to-field is like request-to-field, except instead of taking a request attribute and putting it into a field it takes a session attribute and puts it into a field. So it takes the session attribute, specifies the session name, and puts it into the named field. If the session attribute is empty or null then you can specify a default string that will be put into the field.

Okay webapp-property-to-field is very similar to the property-to-field operation over here, where it pulls a property from a resource and puts it into a field. This one is very similar except that instead of the resource pointing to a classpath resource it's pointing to a webapp resource.

Now a webapp resource basically means that it is a properties file that is under the webapp root directory. So the path it goes into the resource here will be relative to the root of the webapp, as opposed to the path of the resource here that is relative to the root of the classpath, or one of the directories on the classpath.

Okay this also has two overrides. If the property is not found or is empty you can specify a default value to put into the field. You can also specify the name of a session attribute to be put into the field if it's not there. The default will be used if the property is not found and the session attribute named here is not found. And that's how the named field will be populated.

Okay field-to-result is the one that is only run when this current simple-method is being called as a service, so all of the return information from the service will be put into the result map as is standard for the way a service operates. You specify the name of the field, optionally the map-name. That's where the value will come from, and it will put that in the result-name, in a field in the result-map with the result-name is perhaps a better way to put it

The result-name is optional as denoted here. If it's not specified then the field-name will be used as the name of the field in the result-map, or as the result-name.

Okay and that is it for this third page of the simple-methods: the call operations, general environment manipulation or context manipulation operations, other operations that do miscellaneous things, and the veneg service operations.

00:47:07 screen changes to Framework Quick Reference Book Page 7

And on the next page we will be talking about all of the entity specific operations. We already reviewed the simple-map-processor, so we'll just be going over the different entity engine related operations.

Part 3 Section 3.4-D

Simple-Method: Entity Operations

Starting screen Framework Quick Reference Book Page 7

Okay now we're on the fourth of 4 pages of the simple-methods. In this section we'll be talking about the entity engine specific operations. (screen changes to 4, 5, 6, and 7 through this, then back to 4) We've already gone over the other ones, the general operations in the general list, all of the if and control operations, the call and other miscellaneous operations, and general operations environment and such, and now we're on the entity ones.

On the first page in these four boxes we have a summary of all the simple-method operations, organized by different groups, and in this box right here we have all of the entity operations. You can see they're separated into the groups of the miscellaneous, find, value, list, and transaction operations.

00:01:03 screen changes to Framework Quick Reference Book Page 7

Now we'll look at these in detail. The details are on page 7 of the book, which is page 4 of 4 of the simple-methods. The most basic operation for the entity engine is the find-by-primary-key. Note that most of these operations correspond with an operation on the entity engine API, and in the entity engine section we talk about that in significantly more detail.

But a find-by-primary-key is a pretty simple operation; you just specify the name of the entity that you want to look up. The map name is a map containing name/value pairs that needs to have all of the primary key fields represented in it, so the name of each primary key with a corresponding value on the map. Value name is the name of the variable to put the resulting generic value object in, that represents the record that's found in the database.

Use-cache specifies where you want to use the entity engine cache or not. By default it is false, so if you want it to cache the operation then just set use-cache to true. You can also specify a list that contains the fields to select. By default it will select all fields of the entity, but you can tell it to just select certain fields. That is optional of course.

And then the delegator-name is the name of the delegator to use. We talked about the entity engine configuration. You can have multiple delegators configured in a given entityentine.xml file, if you want this operation to use a delegator other than the delegator for the context the simple-method is being run in.

For the service engine or for the web applications preconfigured delegator that would typically be the places that they come from. But if you want it to be a certain one, like for integration code or that sort of thing when you're using a simple-method to move data from one system to another, you can specify a different delegator name here.

Find-by-and is like find-by-primary-key, except instead of finding a single value by the primary key the map will represent a non primary key set of fields for the entity named in entity name. And it will return a result that is a list of generic value objects instead of a single value object. Then that will be put in the variable named here on the list-name.

So since we are finding a list, or will have multiple records coming back, we can use this order-by-list-name to specify the fields to order the list by. Each field in the list, or each entry in the list, will just be a string with a field name. It can be preceded by a plus or a minus to specify an ascending or descending sort for that. The default is ascending sort, so you just put a minus in front of the field-name if you want it to be descending.

Delegator-name is the same as above, for if you want to override the current context delegator.

Use-cache same thing as in the find-by-primary-key. It defaults to false, so if you want it to use the cache set it to true and it will cache it to memory on the application server.

Use-iterator by default is also false. If you set it to true then instead of getting all of the results back and putting them in a list, and putting that in the list name, it will put an entity-list-iterator in here. The iterate tag, for iterating over a list, will recognize the entity-list-iterator and will close it when it's finished.

It's important to iterate over the list if you do specify use-iterator=true, so that that list iterator will be closed and the connection to the database is also closed. We've already seen other places where there are options to do this in the shared methods, and the main advantages if you expect a very large result is instead of pulling them all into memory and putting them into a list, this will use a database cursor to pull over results

one at a time. Or the JDBC driver will pull them over one hundred at a time, or however it's configured.

Okay and that's it for the find-by-and. Entity-count is very similar to entity-condition, so let's step back here for just a second.

00:06:41 screen changes to Framework Quick Reference Book Page 4

Looking at all of the entity operations you see some of these entity-one, entity-and, entity-condition, get-related-one, and get-related. These are all used in the Screen Widget and in other places, so we covered them before when we were discussing the widgets. The details for them are on the shared elements page here (screen changes to page 13). So the detailed overview of all of these elements would be in the Screen Widgets section of these training videos.

00:07:13 screen changes to Framework Quick Reference Book Page 7

So hopping back to page 7 here the entity-count is very similar to the entity-condition. Specify the entity-name, optionally the delgator-name if you want to override that, and then the name of the variable to put the count in.

You can do the same condition-expr (condition expression) or condition-list which can have condition-expr and other condition-lists underneath it for a tree of conditions that can be arbitrarily complex. You can also use the have-in-condition-list, this is the same as on the entity-condition.

What this will do basically is, rather than doing a query and getting the results back, it will just count the results and put that number in the context in the variable named by the count-name here.

Okay order-value-list does not actually do anything with the database. It takes an exiting list that has been returned and sorts it. So we specify the name of the list of generic value objects that we want to sort, the name of the output list. If it is empty, as it is optional, it will simply use the list-name. In other words it will take the ordered list and put it over top of the resource list.

The order-by-list-name is the name of the list of strings that specify the fields to order the list by. This is the same sort of structure that would be used here with the find-by-and.

Okay filter-list-by-and will take an existing list just like order-value-list. So given a list name and an optional to-list-name to put the result in, it takes in a map of name/value pairs where the name is the field name and the value is the value of the field. And it will filter the list by these name/value pairs such that the only elements that will remain in the output in the to-list are the ones that match the name/value pairs in the given map.

Okay filter-list-by-update is similar to filter-list-by-and in that it takes the incoming list and filters the entries in that list, and puts the results in an outgoing list. Or, if there is no to-list-name, back in the same list in the same variable.

Valid-date-name is the name of a variable in the context that should be used for the current date and time. If this is empty the actual current date and time will be used, but if you wanted to do it sometime in the future or the past you can specify the date to use for validation for the filter.

The from-field-name is specified here. It defaults to from-date, which is the common convention used in the OFBiz entities or in the OFBiz data model.

Thru-field-name you can optionally specify here. It will default to thru-date, again the convention used in the OFBiz data model. And all-same means that all of the generic value objects in the list are of the same type of entity, and by default that is true.

When that is true the performance is a little bit higher because it doesn't have to look up the field meta-data for each record based on the entity, it just has to look it up once for the entire list. So if that's not true, if there are mixed entities in the list, then you need to set that to false for it to work properly. But that's not a common case so it is true by default.

Okay some other miscellaneous entity operations. Sequenced-id-to-env is the primary key sequencer. Make-next-seq-id is the secondary key sequencer. So this would be something like an orderId for example, where we're sequencing an orderId automatically. And this would be something like an orderItemSequenceId, where we have a sub-sequence that varies for order item records related back to an orderHeader. So all of them will have the same orderId.

So to use these you just specify the name of the entity for a sequenced-id-to-env preparing the primarySequencedId. The name of the entity is typically what we use for the sequenced name, but you can use anything you want if you want to have different sets of sequences.

The risk of course of many different sets of sequences for the same entity is unless you somehow prefix or suffix the value, you could have a key conflict. So we just use the entity name for these primary sequences.

Env-name is the name of the variable in the environment or the context where it will put the sequenced Id. By default it puts a string value there.

Get-long-only will get a numerical long value and put it there, so it does not do that by default. By default get-long-only is false. If you want it to just get a long number then you can set that to true. That's in there for supporting lower level functionality, but that's not the

typical pattern used in OFBiz as we do use strings for sequencing.

Okay stagger-max, by default this is one. But if you want to have sequenced Ids that are staggered, instead of consecutive, then you can set this to something like twenty. And then it will do a random staggering for each sequenced id; instead of picking the next value all the time it will pick something between the next value and twenty away from the next value, if stagger-max is set to twenty. So that can be used to make the sequenced Ids more difficult to guess.

Okay so now we're on to make-next-seq-id. This is for creating a sub-sequence like I mentioned. The orderItemSequenceIdShipments, for example have a sequencedShipmentId, and then the shipmentItems have a sub-sequenced shipmentItemSequenceId.

So typically what you'll do is you'll create a generic value object, and populate everything or make sure the rest of the primary key is populated. Typically with this you'll have a two part primary key. One part will be the Id that is the same for all of the shipment items for example, would be the shipmentId.

And then the seq-field-name, the field that will have the sub-sequenced value, and in the case of the shipment item example that would be shipmentItemSeqId. We use this seqId suffix on the field names in the OFBiz data model to denote that that field is a secondary sequenced ID and should therefore be maintained for this sort of operation.

The numeric-padding on the Id. Since these are eventually strings we do numeric-padding so that the sort is consistent, and by default we pad it with five positions. So it will typically have 4 zeros in front of a single digit number, 3 zeros in front of a 2 digit number, and so on.

Increment-by. Typically we'll just increment by 1, but if you want to leave some space in the sub-sequence you can increment-by 10 or 5 or that sort of thing.

Okay those are the primary sequenced and sub-sequenced Id operations. The different transaction operations here, begin, commit, and rollback, are not very commonly used, but they can be used for more granular control. We talked about some of the details of transaction management in the Just Enough Java before at the very beginning of this Advanced Framework course. For the most part you won't use these operations because the simple-method engine itself will start a transaction around a simple-method by default.

00:16:45 screen changes to Framework Quick Reference Book Page 4

Unless you set this use-transaciton flag to false it is true by default. That's the use-transaction attribute on the simple-method element for each simple-method.

00:16:59 screen changes to Framework Quick Reference Book Page 7

And then the other place that will typically manage the transactions for you, instead of requiring you to use these operations, is the service engine. It is very common to use simple-methods to implement a service, and in that case the service engine will typically manage the transactions for you, unless you tell it not to.

So you could tell the service engine not to start a trasnction for you, and you could also tell the simple-method engine not to start a transaction for you, and you could control them with these flags. But typically you'll just tell the service engine to, for example, manage the transaction for you. And it has more options as well, as we've talked about briefly and will talk about in more detail as well in the service engine part of this course.

Okay when you're using these operations they each have this began-transaction-name that is optional because it has a default value of began-transaction. This is basically just the boolean variable that keeps track of whether or not you actually began a transaction. Like if you call transaction-begin, it will put a boolean variable value in here that specifies whether or not it began a transaction.

So if there is one already in place this will be false, because it will not begin another transaction. If there's not a transaction in place and it did begin one, then began-transaction will be true. When you commit it looks at this flag to see if it should actually do a commit or should go through successfully.

When you rollback it will look at this flag to see if you should do an actual rollback if it's true. Or if it's false, if we didn't actually begin the transaction, then it will do a setrollbackonly, or it will set the rollbackonly flag on the transaction. And that's just following the pattern that we discussed in the Just Enough Java, which was at the beginning of this Advanced Framework course.

Okay moving on to the entity change operations. We talked about these different things for querying, these operations in addition to the shared ones, like: entity-condition, entity-and, entity-one, get-related, and get-related-one. When we want to modify a generic value object these are the operations that we use.

Make-value is a factory operation that will create a generic value object. It will put it in the context in a variable with the value-name here, and it will create a generic value object representing the entity-name here. And you can set initial values on that operation by passing in a map of the name/value pairs.

The field name is the key or the name in the map, and the value the value to set for each field. And that's how we can create a generic value object. This is very

common to do preceding a create-value operation, which takes in a generic value object by name and sets it, and actually does a create or insert in the database.

This do-cache-clear attribute by default will always clear the cache, but you can tell it not to clear the cache if you want. But that's not a very commonly used operation, because typically you'll want to have it clear the cache when you insert something into the database. Because if there's a list of results that was cached that should include that value but does not, you want it to clear out that result so that it will in the future.

Clone-value we skipped over when talking about the make and create. Clone-value will simply copy a generic value object. You give it a name and a new value name and it will copy it. This basically just does a shallow copy, which is typically all that's need because the value in a generic value object like strings and various number objects are immutable. So copying them is not a problem, or to point to the same one in two different generic value objects is not a problem.

Okay store-value is like create-value except instead of doing an insert in the database it does an update in the database. So when you do a create, if a value with the same primary key already exists in the database this will return an error. If you do a store and the value does not yet exist in the database then this will return an error. So when you're initially creating a record you should use create-value, and when you're updating you can use store-value.

Refresh takes a value name and just rereads it from the database. So basically does a find-by-primary-key, and replaces the variable in the context with the new information. Actually the entity engine does this inline, so its the same generic value object, and it just refreshes the field values and that object from the database.

Remove-value goes along with create and store, and actually does a remove from the database. So it will remove, or delete is the actual SWL term, delete the record that corresponds to the value from the database.

Remove-related is kind of like get-related; you tell it a value name and the relationship name, and given that value it will follow the relationship and remove all related records, whether they be a type one or type many relationship. For a type one relationship it will remove a single record if it exists, and for a type many relationship it will remove all the records that are related to it.

This of course can be dangerous, for example if you have a product-type entity and you do a remove-related with a certain product type value object here, and the relation name product here, it will remove all products of that type. This is of more value when you're doing something like removing an order or removing a product, and you want to remove all related elements before removing the product to resolve foreign key issues.

Usually the best practice for that, for removing a product for example, is to do a remove-related on certain types of information related to the product, and then try to remove the product but not remove all related tables. In many cases, if the product has been ordered for example, then you do not want to remove the product, and so you can do all these remove-relateds.

When you get down to removing the product, you not do a remove-related on the OrderItem. And then if it sees an OrderItem that the product is related to then it will not be able able to remove the product record, because of the foreign key which will cause the transaction to rollback and all the product information to be restored in essence. So this is a good operation to use with that pattern for safe removes.

Remove-by-and is similar to removing a value, except instead of removing a particular value by primary key, we just specify an entity name and then the name of a map with the name-value pairs in it that will constrain the remove-by-and. So any record in the database that matches those name-value pairs will be removed when this is done.

Okay clear-cache-line is related to these do-cache-clear things. This is something that, because the automatic cache clearing is very good in the entity engine now, doing explicit cache line clears is typically no longer necessary. But you can do that by specifying the entity-name and name-value pairs, and it will clear all caches, all records from caches, whether they be primary key or by-and or by-condition caches that match those name-value pairs.

Clear-entity-caches will clear all of the entity caches. That's not used very much; it's fairly extreme, but it's there if you need it.

Set-pk-fields and set-nonpk-fields are operations for convenience. If you have a map coming in, usually the parameters map, and you want to set all the primary key or non primary key fields from a certain map to a generic value object that exists, then you just call this set-pk-fields or set-nonpk-fields method. They both have the same attributes working largely the same way; they copy the fields, the primary-key or non-primary key fields, from the map to the generic value object, based on the definition of fields in the generic value object.

Set-if-null by default is true. If you do not want it to replace values in the generic value object with null values then you would set this to false.

Store-list will take a list of generic value objects and store them in the database. This is a little bit different from storing a single value, because if you store a single value, and it does not already exist in the database, then it will return an error. With the store-list it'll go through each generic value object in the list. If it does

not exist it will create it, if it does exist it will update it with the value that's in the list.

Remove-list goes through each generic value object in the given list and removes them from the database. This is the same as using an iterate to go over a list and doing a remove-value on each one, except that the entity engine takes care of it on a lower level.

Okay and that is it for the entity engine simple-method operations. There are some examples right here of some different patterns. Here's a make-next-seq-id example.

This is the sub-sequence-id, and this is actually the example that I was talking about before with the shipmentItem. So we make the value, specify the value-name as newEntity and the entity-name is shipmentItem. We're going to set all the primary key fields from the parameters map to the new entity, and the non primary key fields from the parameters map to the new entity.

Then we call the name-next-seq-id on the new entity and tell it that the field that will contain this sub-sequence-id is the shipmentItemSeqId field. And if it is empty, if no ID was passed in, the set-pk-fields will typically just populate the shipmentId and not the shipmentItemSeqId, leaving it blank for the make-next-seq-id operation to populate.

So what it will do is it will look up the other shipment items from the database, find the highest value and add one to that, and off we go, we have our new unique sub-sequence-id. Typically when you generate a sub-sequence-id you want to take that field and put it in a result, so we have newentity.shipmentitemseqid, and we put it in the result as shipmentItemSeqId. Now we have the full primary key plus we populated all the non primary key fields, so we can call the create-value to insert that record into the database.

Sequenced-id-to-env is the primary sequence generator. We make a value, this is the factory method to create a generic value object, which we will put in the newEntity variable. The entity-name for this generic value object will be shipment, so we'll have that metadata.

We'll set all the non primary key fields from the parameters map to the newEntity, then sequence an ID and drop it in. Sequence name is the name of the entity shipment we'll put as a temporary variable called sequencedId, then we'll copy sequencedId into the newentity.shipmentid.

And then as before when we're generating an ID we typically want to return it, and so we return this sequencedId there. By the way a more common pattern you'll see these days, rather than putting it in a temporary variable like this, the env-name will simply be newentity.shipmentid, which makes this env-to-field not required anymore. By the way this is also deprecated in favor of using a set.

So this is the more common pattern you'd see in the code, and all you'd have to do is this sequenced-id-to-env, and then a field-to-result. Field-name would be newentity.shipmentid, and the result-name would be shipmentId.

Okay there's some examples of how to use the primary sequencing operation and the secondary sequence ID operation. One more little thing before we move onto the simple-method example, as part of the entity operations. There's one that's a little bit newer, that was part of the development for the testing efforts but has other uses as well, which is this entity-data tag. Let's look at that real quick.

What you can do with the entity-data tag, and you'll notice that this is now inserted into the quick reference book, is basically you specify the location of an XML file. This XML file should be in the same format as any of the entity engine XML files, or the entity engine transformed XML files. And what this will do is either load or assert that file depending on the mode.

The default mode is to load the file into the database, but you can also assert it. What it means to assert is it'll go through each row in the file, each record, and it will look that up by primary key. If it does not find it, it'll add a message to the error list about not having found it.

If it finds the record but any of the fields are different, this is in the assert mode still, it will report any field that's different in a message on the error list. So that's basically what you can in essence assert, that a certain set of data exists in the database. This is a very helpful part of test suites where you might set some operation through service calls or through a user interface, and then you can assert that a certain set of data is in the database. This can go along with when you're using simple-methods for writing testing code with the assert operation.

00:34:36 screen changes to Framework Quick Reference Book Page 5

Which is over here in the if-conditions that allows you to specify a certain set of conditions, and each of those must be true. Otherwise each of those is checked, and if they're not true then an error list for that condition is added, or an error message is added to the list, along with the title so we know which assert block it came from. So these are two things that can be used for testing purposes, and that's their main goal.

00:35:03 screen changes to Framework Quick Reference Book Page 7

Of course the entity-data can also be used to load data into the database. It's not as often that you'll do that through code, but if you have some sort of an integration piece or something like that then it can certainly be used there. You can override the delegator or default context by specifying a delegator-name. And you can also override the timeout to start a new transaction and load the data with a longer timeout. And of course the error-list-name you can override the default value of if you'd like; other wise it just uses the default error_list. Okay and that's it for the entity data.

00:35:50 screen changes to Advanced Framework Training Outline

The next step in the coverage of the MiniLang of simple-methods is to look at some examples.

Part 3 Section 3.5

Simple-Method: Examples

Starting Screen Advanced Framework Training Outline

Okay now we've finished the overview of all of the simple-method operations and simple-map-processor operations, and general concepts and an overview for both of them. We've looked a some brief examples, now let's look at some more involved examples.

00:00:30 screen changes CustomerEvents.xml

So for the first one I would like to look at this is in the customer events file, in the applications order component. And this is in some ways like the create-customer method that we saw earlier, this one just has a few more features and does a few more things. It also shares some methods.

So we have a createCustomer method here and an updateCustomuer method here, and they both share this validateCustomerInfo method, as well as the createUpdateCustomerInfo method. Those are both called inline using the call-simple-method that we looked at before. That's actually the first thing this createCustomer method does, it calls this simple-method of validateCustomerInfo. This method is defined at the top of this file here, validateCustomerInfo.

The first thing we do is create a now-timestamp and put it in the variable called nowStamp, and then we use a simple-map-processor to go from parameters to person-map.

We copy over the person title: the first name we copy, and we validate that it's not empty, middle name copy, last name copy and verify that it's not empty, suffix copy, birth-date we convert to a date and put it in the field birth-date. If it fails there's our message over there, these are all failures from properties file. And finally gender we copy. So these are some of the basic things coming in that might be on a newcustomer form or on an updateCustomer form.

So next is simple-map-processor going from parameters to email map. Take the email address, copy it, we do a not-empty validation, and we do a validate method there is an is-email method in the utilValidateClass in OFBiz. We're not specifying the class, which would default to utilValidate as you can see there. So we can just leave it off, and if that validation fails then we tell it the email address is not formatted correctly, and we copy the emails to allow solicitations.

Here's an example of where we're copying from one field-name to another. The next one is doing. First of all we compare to see if we want to use the address or not. So in your form you could use the same method on the back end by saying use-address=false and leaving out the address, or use-address=true and throwing in all your addresses.

Here's an example of the make-in-string. If we want to make a string that has the full name, which is the combination of a first name, a space, a middle name, space, and a last name, and then the full name here would be put in the to-name field of the address.

The address 1 that we copy, make sure it's not empty, address 2 copy, city we copy and make sure it's not empty. StateProvinceGeoId we copy. PostalCode we copy and make sure it's not empty. CountryGeoId we copy and make sure it's not empty. AllowSolicitation we copy.

Now here are some special cases. If USA is the countryGeoId that's being passed in and if the stateProvinceGeoId is empty, then we're going to complain that the state is missing, which is required in the United States. Same thing with Canada; if Can=countryGeoId and stateProvinceGeoId is empty, then we complain that the state is missing and is required for an address in Canada. These are some custom validations that are outside the simple-map-processors, and there's an example of that.

We have some other simple-map-processors here for populating the homePhoneMap, the workPhoneMap, and then that's where our simple-method finishes, right down there. So going back to the method that called it, the createCustomer method, which called validateCustomerInfo right here.

Now that we've done all the validation and populating of the maps, we call check-error to see if there are any problems. If so it returns there, if not we carry on. So the next thing we'd do is in the createCustomer. We'd create person, and we'd get the partyId back, put it in the tempmap.partyid, and also put it in a request attribute. So this is obviously meant to be called as an event.

If the userlogin is empty then we fake it with the anonymous userlogin, doing a quick find-by-primary-key, setting the partyId on our userlogin object. This userlogin object we're setting the partyId but it will never be saved to the database; this is just a dummy object that will be passed around to satisfy the login requirements for the service and such. And we set the current userlogin for this message to this, our dummy userlogin.

Now if it's not empty then we go on, just set up the same thing in a different way basically. Here's an example of logging on to an info level. We show the userlogin object that we ended up with.

Okay now we want to make sure that the party is in the customer role, so we do a set of this constant-string value, another set of the partyId from the tempmap.partyid, and call the create-party-role service with the role map going into it. Then we set up from the tempmap.partyid we do a global-field of partyId, right in the context itself, and we call the createUpdateCustomerInfo service. Or a simple-method rather, which goes up here, to here. First we see if there's an email contactMechId.

So this is designed to handle both creates and updates for each of these different structures. So the first one we check to see if we have a contactMechId for the email address. If it's not empty, in other words if we have one, then we set things up in the email map and do an updatePartyEmailAddress.

We call that service. If it is empty, so if the not-empty is false, then we're going to create one. So again we're setting up the partyId in the email map, telling it we want it to create a primary_email, and then we're going to create the partyEmailAddress. We'll call that service instead of the update.

These other ones are similar; we check to see if we have a homePhoneContactMechId. If we do then we do an updatePartyTelecomNumber. If not we do a createPartyTelecomNumber. Same thing with the work phone. Same thing with the shipping address, and that's pretty much that method.

Again that method is used here in the createCustomer as well as here in the updateCustomer. So this is an example of where we're using these inline methods where they can be used generically. The nice thing about doing them in the create is we could we know we've just created the party.

We could just do a create-only, and in here in certain circumstances we might be able to get away with an update-only. But really it needs to be able to create or update, depending on what is being submitted to the new form for the update. Because of that we need a createUpdate method for the updateCustomer method, so we might as well take advantage of the create part of that and use it here as well.

Okay so finishing off the create one. If the userlogin being passed in is not empty, we can set that, this is just some some stuff actually that's not quite finished. These simple-methods by the way are intended for replacing some of the current code and are actually used in certain places for the checkout process, especially the anonymousCheckout process, to make it more flexible.

By anonymous by the way it doesn't mean we don't know who the customer is, it's not an accurate word. But typically an anonymous checkout process in e-Commerce is one where the user does not create an account.

Okay the update customer one is very similar. There's an example of that style of service.

00:11:07 screen changes to PicklistService.xml

Another one I wanted to show you is this findOrderStopPickMove. Now this has a number of fairly complicated restraints, as you can see here, in different structures.

We have a couple nice big comments at the top to show what some of the different fields should be, so we're only finding sales orders that are approved, items need to be approved this isRushOrder is something we will take into consideration, although it's not a requirement. Grouping by that, ordering by that, grouping by that, is null or less than now.

In other words we have all of these constraints here, and then this is a list of data to create. These are the output parameters basically of this method. When it's finished, this is what it will return.

So we have this one, the ordernnedsstockmoveinfolist. Each list contains a map, so each entry in the list will be a map with an orderHeader, or a shipGroup, or a shipGroupAssocList. That list, that list, each entry in that list there has this map with this generic value object, this generic value object, and that generic value object, and so on.

Basically we're creating a fairly complex thing here. Let's look at some of the general patterns of how this is being done. First thing we're doing is looking up the orderHeaders, there are sales orders that are approved.

If the isRushOrder parameter was passed in we'll want to use that as a constraint as well, so we'll either find all of them that are, or are not, rush orders. But if it's not passed in, if it's empty, then we will ignore it here and we order by +orderDate.

So the oldest is first, otherwise if the orderheaderlist. Another way to use this method, it supports passing in the orderHeaderList rather than looking it up through these constraints. Here by the way is an example of an entity-condition where we have an entity-condition combined by and, and a few expressions. So if an orderHeaderList is passed in we just copy that to the orderHeaderList, login info message and move on.

Now we're keeping track of the number of orders. We'll tell it it's a long, comes from maxNumberOfOrders. And number so far we just set that to a zero, tell it to make it a long instead of a string, so that's what that type will do there.

Then we iterate over the orderHeaderList, looking at each orderHeader, do a quick log, find all orderItemShipGroups related to the orderHeader or with an orderId that's the same as the orderHeaderId. Sorted by shipGroupSeqId, and those go into the orderItemShipGroupList, then we take that list and iterate over it.

For each orderItemShipGroup in the list, here's an example of a nice big condition, to see if a shipmentMethodTypeId was passed in or was not passed in, or if passed in then is equal to the shipmentMethodTypeId of the orderItemShipGroup.

Here's another flexibility built into this method so you can pass in a shipmentMethodTypeId or not. So if you want to constrain it by that then you pass it in, if not then you leave it.

Same thing of the shipAfterDate; check to see if the namtimestamp is greater than or equal to the shipAfterDate. Or the orderItemShipGroup shipAfterDate can be empty. If it's not empty then we'll check the second constraints to see if.

If it passes those things then we will go ahead and process this orderItemShipGroup. It's not qualifying so far for either needing a stock move or being ready to pick. So now we're looking up other things. I won't go into all the details of this because it is a fairly involved process, as you can see, but this is the basic pattern.

So we're looking things up, we're testing other constraints as we're going along to see if we should continue for a given record or not, same sort of thing here. Now we're iterating over the inventory reservations for the shipGroup. We're trying to figure out if we should pick this order, or if it needs a stock move, or allpickstarted.

These are the default values, and we'll adjust these as we're going through, as we do various tests. Eventually we get to the end, we're setting things up, this new object we just created we're dropping it into this list. And here's an example of clear-field, because this is going to be used again as soon as we iterate around.

What else, here's a cand we're looking at the numbersofar, and this is just a very simple thing to add one to it. So there's an example of the calculate operation and action.

And on we go. Eventually we get down here to all the shipmentMethodTypes. So we do a quick lookup here, for there are not conditions for this entity-condition. We can see that it's just going to look up all records in the shipmentMethodType entities, table put them in this list, order them by the sequence number, and then we iterate over that shipmentMethod type.

Now this is similar to an example that was in the framework quick reference book. Where we're iterating over these types, we've got this typemap. If it's not empty, in other words if there is an entry in that map with the key value that shipmentTypeId, then we're going to take that that value from that map and put it in the list here.

These are just some examples to basically get them in this list. This is basically just ordering these map entries by the shipmentMethodType as we go through that list, and of course that list is ordered by the sequenceNum for the shipmentMethodType. So what we'll get is a nice little list here that's ordered by the shipmentMethodType. That basically just accomplishes that.

Anyway there's just a very brief overview. We skipped quite a bit of it, but his is an example of a simple-method that is gathering data for other things to use, reading a whole bunch of stuff from the database, doing a validation on it, and doing different things or supporting various different input options and such. Which could be seen in the service definition by the way, typically documented and commented there. And then we're creating this sort of output, as documented here in this comment.

And there's an example of the pickListServices of using a simple-method to assemble that sort of more complex information. And that's basically just reading the data.

00:20:17 screen changes to InventoryReserveServices.xml

The next example I want to look at is this inventory reservation method. So reserveProductInventory. Here's an example of a more complex method that's not just gathering data that will be used in a user interface or business process, but is an actual business process itself.

There are a few things passed into this. First of all we'll be reserving inventory for a given orderId, so we'll look up the orderHeader here. A few things that we'll know here; productId, facilityId, and containerId. These things are actually optional, but if present will be put in this lookupFieldmap. ProductId will always be there,

facilityId and containerId as mentioned are optional here.

Okay then this is looking at the reserveOrderEnumId that's being passed in, comparing it to these various things and setting up an orderByString. So we have unitCost in the direction, expireDate in both directions, and dateTimeReceived in both directions.

And if none of these is true, if we get else else else else else, we eventually get down to a -dateTimeReceived. And in other words that's default to this first in first out (FIFO) by received date, because the default orderEnumId.

Okay and then this orderByString, here's an example of using set to take a variable, a field, and put it into a list. It's going to put it into a new list called orderByList that will have one value in it, whatever was in the orderByString field. And from the quantity, this is the quantity to reserve. So we're initializing this quantityNotReserved, and this will be decremented over time as we reserve different quantities.

Okay now this next thing we're doing, looking at inventory time and location, doing a few queries in this, iterating over these, and comparing to see if it's a pick-location. So the reason it's doing this is we always want to go by the pick-locations first, reserve there, and then reserve in the bulk locations. So it's going through those, and for each of them it's calling this reserveForInventoryItemInline.

So first it'll go through all of the pick-locations. If those are exhausted it will go through the bulk locations, if the quantityNotYetReserved, that thing we initialized up here, is still greater than zero. And if all the bulk locations are exhausted then we'll use all of the locations that do not have any particular locations actually. They're just inventory in a facility, rather than in a specific location in a facility.

Okay so if the quantityNotreserved by the way here is zero, then we're going through this other process where we are going to find the last non serialized inventory item and subtract it from the availableToPromise. In other words we'll have a -availableToPromise somewhere. So we weren't able to reserve everything, and that's what this latter part of the method is doing going through all of that.

There's some examples in here of some interesting code that gets more complicated, for example some BeanShell. Here's an example of how you can use the call-bsh tag with the CDATA element underneath it. Now when you use this construct in XML, this CDATA construct, it treats everything inside this [ ]] and > sign have to be together, to closet this off as plain text.

So things that would normally not be valid, like > and < and $ and stuff like that, are okay inside a CDATA block like this. So we'll typically use that for a BeanShell snippet. Here's an example of doing a call-bsh where the BeanShell script is sitting inline right here. And the reason it's doing this is because there is not a simple-method operation, right now anyway, that's meant for doing calender operations.

We're using the JavaUtil calendar object to add a certain number of days, the number of days to shop, to the orderDate, so we can get the promisedDateTime. So it returns that, and when you return from a BeanShell script inside a simple-method and return a map, it will expand that map into the current context.

What we get is a promisedDateTime sitting in the context, that can then be used down here, for example here. So it's set in the reserve order item ship group inventory reservation map (reserveOISGIRMap), which is used to call the reserveOrderItemInventory service.

Anyways as you can see with these sorts of processes, what we're doing in essence is just gathering data from different places, parameters coming in, evaluating for defaults, and setting up lower level things like an order buy based on different parameters coming in, like a reserve order. Iterating over things, doing queries in the database, calling simple-methods inline, calling services inline, blocks of setting up maps for calling a service; sometimes these can be done in more or less automated ways.

This actually might be older code that has been somewhat modernized so it does it one at a time. A lot of this could actually be replaced in its coming from the parameters map and going into the reserveOISGIRMap. We could do the setServIdfield and use that operation to trim down some of this code.

But anyway the idea is basically the same. We're setting up a map to pass into a service, doing the service call, and so in business systems like those that are built on top of the Open for Business Framework you can do most of what you need to do in business logic, in something like this where it's a very constrained set of operations that is tuned to this type of logic.

And it ends up being significantly more concise, especially in terms of number of lines, as well as the complexity of the lines. Now it's not necessarily less verbose; because it's implemented in an XML structure there is a fair amount of verboseness in it. But that can make it easier to read, especially for those that are new to it, because it is more self explanatory.

When you're writing or modifying the code it's not such a big deal, because there are lots of good auto-completing editors that have built in documentation display with the annotations and all of these sorts of things. So there are all sorts of varieties of processes and such that you can implement using simple-methods.

And then those cases where there is not an operation that does the convenient things you want, like in this case. This is actually a case doing these sort of calender operations where we may add some new simple-method operations at some point. But even without them you can still just drop down to a little scriptlet and run them.

As far as performance goes, not only do we use the simple-methods for inventory reservations here but we also use them for inventory checking, which involves a lot of looping over the values that come back from the query to the database and these sorts of things. So performance might be a concern for the simple-methods.

But in general these need to perform extremely well, because the simple-method XML file is only parsed once, and it's parsed into an object model in memory. And then we iterate over those operations, and perform the operations, and compile Java code. And it runs nearly as fast as if this was transformed into Java code, like we use some sort of a Java code generator based on the simple-method, and then it was executed from there.

So it's nearly as fast as doing that, especially with the database operations considered, and the time it takes to do database operations compared to the time it takes to run this sort of code on an applications server. The network communications serialization and deserialization is significantly slower than any of this sort of thing.

So for typical business processes there will be no noticeable difference, even with a performance tool. Very little measurable different between writing it in a simple-method and writing it in Java, and because it's more efficient to code this way from a performance perspective. It also allows you to write something quickly and then spend time on performance as needed, by evaluating spots that are slower, doing query optimization, or changing how the lookup is done from the database. And for tuning indexes and such in the database. And so for those sorts of things, doing optimization based on performance testing and profiling is significantly more successful than doing it here.

Now BeanShell snippets like this are admittedly a bit slower, because they are interpreted and so use the Java introspection API, the which the functionality can be a bit slower. The rest of the simple-methods do not use the introspection API, with the exception of the call-class method and call-object method and create-object operations.

So those are the only three where it will use the introspection API, so the slowdown from that only comes in there.

There are other parts in OFBiz like the service engine, the control servlet, and various other things where we're calling static methods and doing other things that are written in Java. And it does use the introspection API for those, and that results in a little bit of a slowdown. But we minimize that by using a cached-calls loader, so that loading a class that's in the cache will simply take the time of a hash-map lookup rather than requiring a full search through the classpath, which can take a few milliseconds.

And so calling methods is actually one of the performance problems that we had in the early days with the OFBiz framework written in java, with all this dynamic code going on. Especially dynamic code that interfaces with Java classes and such. So when we introduced the cached classloader it sped things up significantly.

It was interesting to note by the way that certain applications used a cached classloader, like Weblogic for example, so in the early days of OFBiz we actually found that some of those ran significantly faster. Even though certain things like their web applications containers and such were not particularly fast, they ran in general faster just because they had some performance tricks like the cached classloader.

So now that's available in all of OFBiz; it's loaded all the time. But in general the effect on performance in using something like the simple-methods, even these fields where we're using the . Syntax, the variable expansion in a string, all of these sorts of things are very bassed because they are all pre-parsed. Like this sort of thing, this sort of variable expansion with the ${ }. It's all pre-parsed and separated into structures. So there's a no object sitting there representing that the parameters, an actual variable called parameters, will be looked up as soon as this string is expanded. So all it has to do is do a hash-map lookup in the current context, it's basically just as fast as running it in Java.

00:34:47 screen changes to Advanced Framework Training Outline.

Okay so that is the basics those are some examples of using the simple-methods and MiniLang in general. We saw a number of examples of a simple-map-processor, and as you can see this is mostly using inline underneath the call-map-processor operation. And we looked at various combinations of different of these other operations.

And that is the world of the MiniLang component in OFBiz, or the simple-methods and simple-map-processor.

Part 4 Section 1.1-1.3

Entity Engine Introduction

Starting screen Advanced Framework Training Outline

Welcome to part 4 of the advanced training course of the Open for Business framework. In this part we'll be covering the entity and service engines, so we'll be starting first with the entity engine. Let's real quick check out a couple of these documents so you know where to get some more information. These are not totally complete but can be nice quick reference information, along with the transcript of this video of course. But these are some documents that exist on the OFBiz site, in the Docs and Books tab.

00:00:42 screen changes to OFBiz.org Docs & Books tab

And down in the main documents we have the entity engine guide. Up in the configuration documents there's also an entity engine configuration guide.

00:01:02 screen changes to Entity Engine Guide

These have general information about the entity engine and related things. Some of these patterns that the entity engine is based on and some background information, is here in the introduction. Some general information about entity modeling, example entity definitions, more references on entity and field definitions and whatnot, relationships, some comments on the seed data, view entities, a little bit of information about the entity engine API, and the transaction support, and then some webtools things there.

So that can be a helpful document or reference. We'll be covering all of that information here in these training videos, but that's another place to go for information.

By the way these documents are slated to be moved to the docs.ofbiz.org site, along with end user documentation and various other things, to assemble a more complete documentation set. But when that'll be done is a good question. As of now, July of 2006, that is planned but not yet complete.

00:02:03 screen changes to Entity Engine Configuration Guide

Okay the entity engine configuration guide basically covers the entityengine.xml file. You can see it's a little bit dated, as it's referring to the old location of the entityengine.xml file. It talks about various elements in it: the resource loader JTA elements, delegator declarations, entity model reader, entity group reader, entity ECA reader, elements, field-type XML files, and the datasource elements.

And when we talk about the entity engine file we'll be covering that as well.

00:03:02 screen changes to Advanced Framework Training Outline

Anyway those are some things that can be helpful for learning about the entity engine. Let's start with the section here, looking at the architecture of the entity engine and some of the design goals.

The first point made here is that for code using the entity engine, there are a few goals that we want to enable for that code. One of them is this: being database or dataSource agnostic. So you perform operations in your code without caring about which database or what sort of dataSource is running underneath. As long as the code does not know about the database or does everything through the entity engine API and so is not database specific, then you can change the underlying database without changing the code, or even point to multiple databases at the same time without causing problems.

Dynamic API behaves according to entity definitions. We talked a little bit in the Framework Introduction about this concept of a more general view of the model view controller pattern, where you have some sort of an internal resource. In the case of the entity engine it's the database and an external API, or the view. And the controller sits in between those and maps that external view to the internal model.

00:04:41 screen changes to Framework Quick Reference Book Page 18

And the thing that maps the two is the controller. So in our artifact reference diagram we have the database sitting on one side, and simple-methods or Java code or other things sitting on the other side and talking to the entity engine.

The entity engine looks at entity definitions and other information it knows about in the entityengine.xml file for configuration and whatnot. And based on that information, and the operations that you request of it, it generates the necessary SQL, sends it to the database, and does other things through JDBC, using a JDBC driver for each database, to accomplish the operations that you need in your program or in your code.

00:05:32 screen changes to Advanced Framework Training Outline

So basically it will do the same operations, like the find-by-primary-key will behave differently according to how the entity is defined that it's operating on.

Okay one of the other goals of the entity engine is to allow for a minimal redundancy in the data layer. So especially compared to other technologies like EJB, even Container Managed Beans, JDO, hibernate, and other such things that typically do have some data layer objects. Or in other words objects that are specific to a table or certain set of tables in the database.

Rather than doing that with the entity engine, we use a more generic approach where we have just entity definitions, and based on those entity definitions the entity engine knows about what is in the database and what

to expect. And then going through the API we can do generic operations without having to have any code that is specific to each entity.

Now we do typically have some services that provide lower level things like create, update, and delete methods, that are in the logic tier. But these introduce a process, a very low level process but a process nonetheless, that is related to the entities or the data layer artifacts. So in the data layer itself in OFBiz all we have basically are the entity definitions, and everything else is driven dynamically through the entity engine.

So we have very little code there and no need for redundancy. Even in the higher layers like the CRUD service implemented through the service engine, and with simple-methods, there are various techniques that we looked at in the Framework Introduction so that you can treat sets of fields more generically, and this also minimizes redundancy.

And it makes it possible to add a field in the entity definitions and have it immediately reflect, without any other changes, through the service layer and the certain user interface elements as well. And then all you have to do is customize exactly what sort of processes that will affect, and how you want the user to interact with that information.

Okay the entity engine is based on JDBC, so it uses JDBC drivers to talk to the database and JTA, the Java Transaction API, which is of course built on JTS, the Java Transaction services. And so these two lower level parts of the J2EE standards are what we use.

Because of this we can operate in the same transaction with other tools like EJB, JDO, Hibernate, or even just raw JDBC calls. As mentioned here you can drop down to JDBC when necessary to execute custom SQL or call methods that are in the database, and these sorts of things. So you have a choice of tools for ease and efficiency when you're using the entity engine API, or as much flexibility as you need dropping down to the JDBC API.

Now the entity engine API is good enough these days that you can do quite a variety of queries, handle very large data sets with the entity list iterator, and other such things. So it's very rarely necessary to drop down to JDBC. If you have stored procedures or methods in the database and other such things that you need to call, then using JDBC may still be necessary. But when you're doing general data interaction then that's not a problem.

If you do have stored procedures and such, using JDBC to access those is not a problem, and won't really affect your database independence. Because once you have stored procedures, your code is dependent on something that is dependent on the database, and so separating that part of your code anyway from the database would be difficult.

Part 4 Section 1.4

Entity Engine Configuration

Starting screen Advanced Framework Training Outline

Okay looking at configuration of the entity engine. The first file to be concerned with is the schema definition for the entity configuration file. The name of the netity configuration files is entity-engine.xml, and this is where it sits in the OFBiz directory structure.

There are various things you can control in there: the transaction manager, JTA setup, delegators, and datasources. Delegators can work with groups of entities, and the different entity groups can be assigned to different datasources as desired. You can configure JDBC drivers; typically the datasources in the entity-engine.xml file will point to he JDBC drivers, but the JDBC drivers themselves just have to be on the classpath somewhere. The ones that come with OFBiz like the Derby JDBC driver, which is actually the entire Derby database because we use it as an embedded database, are located in this directory here; ofbiz/framework/entity/lib/jdbc.

Another thing you can configure in the entity-engine.xml file is the distributed cache clearing. So before w look at that let's talk real briefly about entity groups and the data type dictionary, as these fit into the configuration. So as I mentioned with the delegator, basically it can support multiple groups, and each group would point to a different datasource.

Now the groups themselves, each entity is put into a group using an entity-group file. This is the location of the schema file that specifies the format of the entity-group XML file, and this is whee you'll typically find the entity-group XML files in the component, in the Entity-Def directory entity-groups*.xml. Each entity needs to be assigned to a group, so if you look at the entity-group files then you'll see them all assigned that way. When we look at the entity definitions in more detail we'll look at some more examples of that as well.

Each entity is assigned to a group, so that when you tell the delegator that you want to do a certain operation on the entity it can look up the group through the entity-group file, which is of course loaded into memory and cached. And based on that group in the delegator definition, see which datasource it needs to communicate with to perform operations for that entity.

Another concept in the entity engine is the data type dictionary. What we do here is have a set of predefined types, and each of those types is a more abstract type that corresponds to a Java type and an SQL type, and the SQL type will vary for different databases.

So this is the location of the schema that describes the field-type XML files. Each database has its own field-type XML file to specify the types that are used in that database; the SQL types and the corresponding Java types and such. So as we're going through the configuration we'll look at an example of those as well.

Okay let's step back here to the entity-engine.xml file and look at the file itself. You can see it's located in the framework entity config directory, at the very top of the file where we have a resource-loader, calling it FieldFile

00:04:18 screen changes to entityengine.xml

This is the resource-loader that the field-type files will be loaded through, that's a file-loader. It will pre-pend an environment variable to it, the OFBiz home, which is always present as OFBiz is running. And then this is the relative location within OFBiz home where you'll find the directory that has the field-type definition files.

This loader is used down here with the field-type entries. Here's the one for Hypersonic SQL (HSQL) for example. So they're all referring to FieldFile; you see them all referring to that loader. And the location within that loader, because it specifies the full path except the file name, is just the file name for each of the field-type files.

So that's where the loader is used. This is very similar to the same sort of resource loader and such that we use within the ofbiz-component.xml files, although the way it's specified here with the different classes is a little bit different.

I guess I should note I'll go through this file to show an example of how the entityengine.xml file is set up, and this is actually the current version of the file from the OFBiz source code repository from the subversion.

00:06:11 screen changes to Framework Quick Reference Book Page 14

But the entity reference book has some configuration pages at the end. And let's see where are we.

On page 14 we have details for the entity configuration. So here for example is the entity config stuff that goes into the entity-engine.xml file. It consists of this column, this column, and this last column here (Entity Config columns 1-3). And then we have an Entity Config Example right here.

00:06:59 screen changes to entityengine.xml

We also have on this page the field-type files, and the field-type example. So we'll go over this in detail in a minute, but let's real quick walk through the actual entityengine.xml file so you'll be familiar with it when you look at it.

Transaction-factory is just a class that's used for getting transactions. and this class needs to know how to communicate with the transaction manager and such. The default one out of the box in OFBiz right now is the Geronimo transaction-factory. If you're running in an application server, typically the application server will put the transaction object in JNDI, and so you'll use this one, the JNDI transaction factory.

And when you're using that one there are two sub-elements: the user transaction location in JNDI (user-transaction-jndi), and the transaction manager location in JNDI (transaction-manager-jndi). These are the two objects that make up the JTA interface, and there's some comments here about where to find the user-transaction and transaction objects in JNDI for different application servers.

Many of them use the same object to implement both interfaces, and they'll be in somewhere like this java:comp/usertransaction. Some however do use two different objects like JBoss, that's java:comp/usertransaction, and java:comp/transactionmanager. So that's the transaction-factory.

Delegator declaration. Each delegator has a name, you specify the entity-model-reader, entity-group-reader, and entity-eca-reader, and there's some distributed-cache-clear settings on this as well. And we'll talk about those in a minute, when we look at the reference book to look at a complete set of the attributes and such there.

So in the data model, as we talked about you can put different entities in different groups. These are two groups that come out of the box with OFBiz, and we point to two different datasources for those different groups, which is the typically the best way to do it. They can technically go to the same datasource, but it does cause some validation hiccups when it's starting up, so the best way to do it is to have different groups in different databases.

These are both running through Derby, but we have two LocalDerby databases basically that these are used for. This ODBC group by the way is for integration; it will typically point to an ODBC database that another application points to for data sharing. I believe there is currently an implementation in Worldship, which is a desktop application, that can point to an ODBC database. And OFBiz can point to the same database for integration with that shipping package.

So as you can see, you can have a number of different delegators to find all at once. And they can all point to the same or different datasources. We also have quite a variety of datasources defined down here for different databases and such: Hypersonic, Derby, there's the DerbyODBC, Cloudscape, which is what Derby used to be, so if you're using older Cloudspace you can still use this. And then there's Daffodil, Axion, some smaller

open source databases, then MySQL and ODBCMySQL to go with that.

Postgres is a commonly a good database to use. Oracle, this is Oracle with the data-direct drivers (OracleDD). Sybase, SAP DB, which is now not used so much anymore since most of it has been merged with MySQL in MySQL 5. Localfirebird, Firebird is another small commercial database, and MSSQL. P6Spy is basically a stand-in that goes in between the entity engine and the database to allow you to look at the communication going back and forth, kind of a special logging utility, and Advantage is another small commercial database.

So we have sample configurations for those types of things. With the entity-model-reader, entity-group-reader, and entity-eca-reader, you can put these inline, and you can specify files in here. Like a resource; specify a loader-name, which the loader needs to be specified at the top, and then the name of the file, like entitymodel.xml. So you can actually specify these right in the entityengine.xml file. Especially if you're using the standalone and not along with the component architecture in OFBiz.

But by default out of the box with OFBiz we don't have any of these here in the entityengine.xml file, because all of the model-readers, the model files, the group files, and the ECA files, the seed data, seed demo, and EXT data files are pointed to in the ofbiz-component.xml files. So in the entityengine.xml file we just declare that there is a main model-reader, a main group-reader, a main eca-reader, and there are three different data readers: seed, demo, and ext, for those different purposes of seed data, demo data, and external custom data. For your applications that are built on top of OFBiz.

Okay the datasource element. We'll look at all of those attributes in detail when we look at the reference for this. But the datasource element itself basically is used to point to a given database, and each one will have a name. The helper-class is always set to this, and this is something that's part of the early on design of the entity engine that is not really needed. You can implement your own helpers to go between the entity engine API and the datasource, for testing purposes and such, but typically you'll just use this generic one that comes out of the box.

The field-type-name HSQL in this case corresponds with the field-type-name up here, that's what that is referring to, and all these other options determine the entity engine behavior for that specific database. As well as certain startup options, whether you want it to add-missing things, check-on-start, use-foreign-keys, and so on.

Okay and the inline-jdbc element which is the most common one you'll see. They specify the driver class, the URI to specify the location of it. Maybe a better example is the derby one.

So here's the URI that we use for the local embedded Derby database. And we tell it to create it if it's missing, which is why it does that on start; it's one of the Derby features. Very simple username and password. Read-committed which is the recommended transaction isolation level, as we discussed in the just enough J2EE relative to transaction isolation. And for the connection pool we can specify the minimum size and maximum size. There's some other attributes here as well, and we'll look at those when we look at the details. Okay so that's basically it for the entityengine.xml file.

00:15:14 screen changes to Advanced Framework Training Outline

Now let's look at some examples. We looked at the entity groups and we're looking at the entity definitions, so now let's look at the field-type files. These are also very much a part of the database configuration for the entity engine.

00:15:44 screen changes to fieldtypederby.xml

So let's look at the field-type. You can see there are a quite a number of these. Let's look at the fieldTypeDerby, since this is the one most commonly used, or the one that's used out of the box actually, with embedded Derby databases. The type names here are the same type names that are referred to for each field in the entity definitions. And this collection of types is what makes up the data type dictionary for OFBiz.

These are the standard types that all of the fields in the OFB data model fit into. So we have all of the basics represented: large binary objects (blob), date and time (date-time), date only, time only, currency-amount, a precise currency amount (currency-precise), this is for interim cucrency values that are part of the process of the calculation, floating-point, and integer (numeric) with no decimal. And we have different standard IDs: a 20 character ID, a long ID (id-long) which is 60 characters, and a very long (id-vlong) which is 255.

We use character strings for IDs for flexibility. This makes it easier when integrating with other systems and such, and these days the database would perform somewhat better with an integer. For most operations the performance difference is minimal enough that it's not an issue for different lengths of strings.

Length one strings, what we use for an indicator, with capital Y and capital N for true and false. You can put other values in here, all you need is a single character.

Very-short is 10 characters, short (short-varchar) is 60 characters, long (long-varchar) is 255, and very-long is a CLOB, or Character Large Object. Which is generally meant to be for most application purposes a nearly limitless character array. Most databases will support

in many cases up to even multiple gigabytes of text in a single record for these.

Some special purpose strings: comment, description, name and value. And then some very specialized types like: credit-card-number, credit-card-date, email address (email), a URL, and a not empty ED (id-ne). id-ne this is just an ID with a constraint not empty. By the way these validate constraints, while here for documentation and enforced by the entity data maintenance pages and webtools, are not currently enforced by the entity engine. But they are part of the convention that we use in the data model to denote ID fields that should not be empty. Id-long-ne and id-vlong-ne, these correspond with these three ID values. And finally telephone number (tel-number).

Notice by the way that this credit-card-number is very much longer than the credit card needs to be. The reason for that is that we do have some two way encryption built into the entity engine. So the value that actually ends up in the database for a credit card number is typically encrypted, and will be significantly longer than just the 15 or 16 digits of a normal credit card number.

Okay so that's an example of the field-type file; it's just a list of these field-type-def elements. Each as a type name, sql-type, java-type, and they can optionally have a validate method underneath them. The method here is a method on the UtilValidate class. This is kind of like the validate method operations in the simple-method; you can specify the class, although it defaults to UtilValidate like so.

Some other types that you will see here that are useful every so often, I believe the postGres field-type has a good example of this.

00:20:11 screen changes to fieldtypepostgres.xml

Perhaps not anymore; I guess that's not necessary with the new version of PostGres. With some databases there's an issue with specifying one type for a field when it's being created, and then the database will actually use a different type internally, so when you request the metadata back it's different.

00:20:42 screen changes to fieldtypemysql.xml

I'm not seeing any. I think most databases have resolved that sort of problem, so it's not so much of an issue anymore. But if it does come up you can specify an sql-type-alias. For example here, if we told the database to create a decimal 18,2 and then we asked it what type it was, it was some internal thing like numeric 18,2. But we couldn't just specify numeric originally because it's not a legal external type, then we would have to use this sql-type-alias, so that the validation of the column types when the database is being checked will not result in an error.

Now as I said most databases have solved these sorts of problems, so that any type they use internally can be specified explicitly externally when you are creating a table, so this is typically not an issue. But if you do run into it that attribute is there to address that sort of issue.

00:22:03 screen changes to Framework Quick Reference Book Page 14

Okay let's look now at the details of the entity configuration. So we've looked at the field-type file, which just has a fieldtypemodel tag, and a bunch of field-type-def tags underneath it. Each one has the type sql-type, sql-type-alias, and we just talked about what that is used for, java-type, and optionally validate tags underneath it.

The validations here are very simple; they're just a class name that is optional, and the name of the method on that class to call. Each of these methods will accept a string argument and return a boolean. If they return true it validates, if they return false it does not. So it's a very simple validation scheme.

Okay and we looked at an example of the fieldtype.xml file. We also looked at an example of the entity configuration or entityengine.xml file. But here's another one right here, kind of a more brief example that you can use for looking at when you're looking at the reference book. Let's talk about what all of these elements mean; there's the root element.

So let's start out with resource-loaders, one to many. Transaction-factory, we'll just have one and only one of those. One to many delegators, one to many model and group readers, zero to many eca-readers and data-readers, as you can run the entity engine without those technically, although typically that's not done. Then the field-type definition files, one to many of those. And finally one to many datasources.

Okay so the resource-loader as we discussed. Basically each has a name, a class that represents what they are. The default class for the file-loader is this one. In this same package, base.config, you'll see other loader classes that you can use such as URL loader and such. But most commonly we use a field-loader.

You can pre-pend something from the environment (prepend-env), from the Java environment, like the ofbiz.home is pre-pended here. And you can put a prefix on it that will be prefixed before each resource is loaded, like this one.

Okay the resource tag is used as a sub element, and follows this pattern that we've seen elsewhere of having a loader and a location. The loader is the name of the loader, referring to the resoruce-loader name, and the location is the location within that loader of the actual resource we want. So we see the resource tag in vari-

ous places here: entity-model-reader, entity-eca, entity-group, entity-data, and so on. These all use the resource tag.

Okay we've looked at the transaction-factory. We always specify the name of the class that will be used for the transaction-factory. And if it is the JNDI one then we can specify the location of the user-transaction object in JNDI, and the location of the transaction manager object in JNDI. And we specify these: the jndi-server-name, and the name within JNDI where that's located (jndi-name).

00:25:55 screen changes to Framework Quick Reference Book Page 16

The JNDI servers I should mention are actually on the next page here, or next few pages. There is a jndi-servers.xml file, and here's an example of that. The most common one you'll see is the default JNDI server. And that basically uses the default values for the current application server, which are usually in a jndi.properties file that is somewhere on the classpath.

Here's the full specification for the jndiservers.xml file, it's pretty simple. Of course each JNDI server has a name. If you are specifying the full details they would look something like this; context-provider-url would be for local-host JNDI mounted on port 1099, which is a very common port for JNDI.

The initial-context-factory. Url-pkg-prefixes are some standard JNDI or RMI settings as well. Security-principal is the username and security-credential is the password, if necessary, to access that JNDI server.

00:27:05 screen changes to Framework Quick Reference Book Page 14

So over here when we're looking at the entity configuration the jndi-server-name corresponds to the name in the jndiservers.xml file.

Okay the delegator name. We've talked about the fact that each delegator has a name. Each delegator needs to point to a model-reader, a group-reader, and optionally an eca-reader. The data-readers are not pointed to by the delegator, they are pointed to instead by the dataSource, so under the dataSource here we have read-data.

There's also this older thing for data reading, while we're on the topic, this sql-load-path for each data-Source. You can specify different paths the path and pre-pending from the environment, like the ofbiz.home. The read-data by the way just has a reader-name, which corresponds to the data-reader name here.

The sql-load-path would just be a path pointing to a directory that has SQL files in it that would be run to load data. So they typically have a bunch of insert statements in them and that sort of thing. That was used early on and has not been used for a long time. The best practice now which is far better, less database dependent and has various other benefits like triggering ECAs as needed and such, is using the entity engine XML files. Which is what this read-data does, along with the entity-data-reader.

And the different data files, as we talked about a minute ago, are specified usually in the ofbiz-component.xml files rather than right here in the entityengine.xml file.

Okay the field-type files. Each field-type will have a name, a loader and a location. So we already saw examples of these. There's some examples down here as well, like the derby load for the field-file loader, which refers to this thing up here. So it will be within this resource-loader, and the location of the file within that loader. And there's the fieldtypederby.xml file, it just tells it where to find that. Okay now with the data-Source element here, this is where it start to get fun.

Actually before that let's cover this. In the delegator element here we talked about the different readers; when we looked at the distributed-cache-clear we just saw that the enabled flag was set to false. But if you want to use the distributed-cache-clear then you would set this to true. The reason you would use distributed cache clearing is if you have multiple servers in a pool that are sharing a single database. So when one entity engine has some sort of an update, either a create update or a delete operation that goes through it, it will clear its own caches. But it can also sent out a message to the other servers in the pool to clear their caches.

This is a feature that runs through the service engine which operates on top of the entity engine, and so the actual distributed-cache class is implemented in the entityExt component. You can see here the full name of that class, org.ofbiz.entityext.cache.entitycacheservices, which is where it will find the high level methods for that. And those methods in turn just call services, and then through the service engine the calls are sent remotely to the other machines in a cluster or a pool of servers in a farm.

So when you're doing a distributed-cache-clear it will result in service calls, and typically you'll send along userLogin credentials. You can tell it the name of the user to use, the default is the admin user, although often that's disabled so the better user to use is the system user. And then off the topic of distributed-cache-clear and on to other things.

This attribute here, the sequenced-id-prefix, allows you to specify a prefix for a sequenced ID, and you can actually have any string to put on the front. You'll typically want to keep it short, 2 or 3 characters, that sort of thing. We talked about some of the sequenced ID operations in the simple-methods. The main one that

gives the primary sequenced ID is what this would affect. If you specify a sequenced-id-prefix for a given delegator, then when that method is called this prefix would be put on the beginning of that.

This is especially useful if you have a number of systems running against their own databases, that will have all of the data sent to a central system. So this can be helpful for avoiding ID conflict and that sort of thing when all of the data is pushed into a single system.

Okay we looked at the group-map in the example. It can have one or may group-maps, and each group-map just has the name of the group, which corresponds to the group-names that entity is in, in the entity-group.xml files, and the datasource-name to use for that group. So that's how the delegator knows which datasource to use, and the datasource-name corresponds with the datasource-name over here.

Each dataSource must have a name and must have a helper class, this may change, as the helper class should really just use the default one right now, the one that we saw in the example which is this GenericHelperDAO. We'll also specify a field-type-name, which refers back to a field-type file here.

Okay use-schemas by default is true. This specifies whether you want to use the schemas in the database. Some databases do no support schemas although most do, these days anyways. And so when you're using a schema typically you'll specify a schema-name, although it is optional.

Many databases if you don't specify a schema-name they will use a schema by default. But sometimes that can cause problems because different operations will assume the schema name where others won't, and then you'll get an error saying that your table doesn't exist, because it can't find it when there is no schema. So typically when you're using a schema you'll want to specify the schema-name.

Okay check-on-start by default is true. And what this will do is when OFBiz starts up it will check the database metadata for each dataSource. And it does this when the delgator that refers to the dataSource through the group-map and datasource-name, so when the delegator is initialized then it goes through and does these dataSource initializations. When a dataSource is initialized for a delegator, check-on-start, if check-on-start is true, will check the database metadata, which includes the table definitions and such, against the entity definitions, the entity model and data model that the entity engine knows about.

Add-missing-on-start by default is false, although we usually set it to true so that if a table or a column is found to be missing then it will create that table or column. Now there are a few other things that it can check on start, and add-missing is needed. And we'll see those options further down.

Okay use-pk-constraint-names by default is true, or you can set it to false. Typically you'll want to do this, unless your database has some sort of a problem with it. One thing to note about the primary key constraint name is that the entity engine automatically generates them. And while they will typically be unique, sometimes you'll run into an issue where you need to use a slight variation on an entity-name in order to avoid that. Because it does have to shorten the names automatically, as these PK constraint names are sometimes even smaller than the table names. Usually they are the same length, but will have some sort of a pk_ prefix, and so they need to be shortened by 3 letters to handle that.

Okay the constraint-name-clip-length is where that comes into play, so for the PK constraint names, as well as foreign key constraint names and so on. For most databases 30 is a safe value; some databases support significantly more. But because many databases only support 30 characters for table-name, column-name, constraint-names and such, we usually clip these, or make sure that these are only a maximum of 30 characters long.

When a delegator is initialized it will always do an internal consistency check on the entity model, to make sure that no table-names or column-names are too long, and also to make sure that all relationship key mappings or key references match up correctly.

Okay user-proxy-cursor. These next 3 attributes go together. The proxy-cursor-name and the result-fetch-size will typically be used together. Most databases will not need to use a proxy-cursor in order to support the entityListIterator in the entity engine, which for the results of large queries will use a cursor in the database to manage those.

In some cases certain databases have little quirks with them, and you need to use a proxy-cursor to get that to work effectively. So if that's necessary for the database then set use-proxy-cursor to true, and specify a name for the proxy cursor. The default name that we use is p_cursor, which is typically fine.

And then the result-fetch-size. You can actually use this independently of the proxy-cursor settings. usually setting this to zero is just fine, and it will then use the default value for the JDBC driver, but you can specify other sizes to force a certain fetch-size. With certain JDBC drivers this can have side effects, so you'll want to check on those before making any firm decisions in production and such.

Okay use-foreign-keys by default is true, and use-foreign-key-indices is also true by default. If the database doesn't support foreign keys, or for some reason

you don't want to use them, you can set this to false, and then you'd typically also want to set use-foreign-key-indices to false.

What foreign-key-indices is referring to is when you have a type one foreign key. It usually means that a field on one table is referring to a primary key on the other table, and so it's common to follow it in that direction, but it's also common to follow it in the reverse direction. And so the entity engine, if this is set to true, will automatically create indexes on all of the fields that have a foreign key.

For example, on the userLogin entity there's a partyId that points to the Party entity. And so the partyId field on the userLogin entity is a foreign key to the Party entity, or has a foreign key going from it to the Party entity. And so the partyId column in the userLogin table, or the table for the userLogin entity if this is set to true, will automatically have an index put on it, so that if userLogins are found by partyId then that index will exist. So this is a fairly easy optimization that gets out a pretty good majority of the indexes that are needed for most applications. And that is following foreign keys int eh reverse direction.

Okay check-fks-on-start. By default this is false because many JDBC drivers have problems with it, but of course that's improving over time. So you'll see that many of the dataSource elements have this set to true.

Check-fk-indices-on-start, same thing for the automatically create indices, just like up here.

Okay the fk-style. There are two styles here, name_constraint and name_fk. These simply affect the SQL that's used, mainly for creating the foreign keys when tables are created, or foreign keys are added to existing tables; there are some SQL syntax variations. So you can see the values for these that are used in the entityengine.xml file.

Name_constraint is the most common one, but every once in a while certain databases have this variation where you'll name the foreign key directly. It's a little bit less generic, but certain databases do that in their syntax.

Okay use-fk-initially-deferred. By default is set to false. Note that this attribute only has an effect when the foreign key is first created; initially-deferred is not done on the fly, it's just done when the foreign key is created. So if you set initially-deferred to true, this is just a database term, but what that means is that the foreign key will not be checked right when an operation is done. Instead it will wait and do all of the foreign key checks before a transaction is committed.

So this allows you to insert data out of order, which is especially helpful where you have circular dependencies and such. And so for a given operation, when a single SQL line is complete it may actually violate a foreign key. But as the rest of the transaction is performed, or the rest of the operations in the transaction are performed, then it may satisfy that foreign key constraint again. And so if you want to have the flexibility to do that, then you'd set your foreign keys to be initially-deferred.

Okay use-indices by default is set to true. These are for the explicitly declared indexes, which are part of the entity definitions, and we'll talk about those in detail in a bit. If you don't want to use those indexes just set that to false.

You can tell it to check-indices-on-start. Note that by default this is false, again because many JDBC drivers have troubles with checking this metadata. But if your database does not it can be very helpful to check these on start, to make sure that all of the indices declared in your entity model are the same as those in the database. And then you'll know if you need to add any or change any.

Okay the join-style for view-entities and dynamic-view-entities, those are implemented in the database. When it generates SQL it just assembles join statements on the fly. Different databases will support different styles of join syntax. The most common is just the ANSI join style, and even databases that did not classically support that do now.

So actually, when OFBiz was just getting started with Oracle it pretty much had to use the Oracle theta (theta-oracle) join-style. And there's a similar one, but a bit different, for Microsoft Sequel Server (theta-mssql). But both of those now support the ANSI join-style just fine.

Some databases will only support a variation on the ANSI style that does not have parenthesis, so if you have complaints about parenthesis in your join statements then you'll know that that's the case. But for the most part you won't have to worry about this if you're using one of the databases that has an example dataSource in the entityengine.xml file, because we'll have already set the join style as well as many other default settings that are helpful.

Alias-view-columns by default is true. Most databases not only support this but require this, in order for the join statements to be used correctly. What this does is, in the SQL when you have a view-entity field-alias, it will give a name to that field-alias in addition to the full name of it, that generally consists of the table that's part of the join and the field on that table, and sometimes these can be other things like an operation. We'll see more of how these are applied when we look at the entity definitions, especially the details of the view-entities.

Always-use-constraint-keyword by default is false. This is just another SQL syntax variation that some databases need.

Okay now for a dataSource you'll typically have one of these three different elements: jndi-jdbc to get your JDBC dataSource from JNDI, and inline-jdbc where you specify all of the parameters inline, and this is what you'll see out of the box typically. So jndi-jdbc is what you'd use if you were running OFBiz in an external application or that sort of thing.

And finally tyrex-dataSource is still around, but not really supported anymore, because we do not use Tyrex for the transaction manager anymore. This kind of had some special custom things on it, like when we were using Tyrex it had its own dataSource configuration, so you'd specify the name of the Tyrex-dataSource and the transaction isolation-level

So the ones you'll see these days are typically these two: the jndi-jdbc and inline-jdbc. The jndi-jdbc is simple; let's talk about that one first.

The location of the object in JNDI or the JNDI server to look in; this would be the name of the JNDI server up here. This is the name of the JNDI server in the jndi-servers.mxl file. And then the isolation-level to use for that dataSource; we talked about these isolation-levels a bit in the Just Enough J2EE section of this course.

00:49:27 screen changes to entityengine.xml

There's also a comment on them here about the different isolation levels. If you need to review these, going back to the video or the transcription of the video for the Just Enough J2EE in the transaction isolation section would be a good idea. This is the one that's by far the most recommended, readComitted, if your database supports it.

00:49:52 screen changes to Framework Quick Reference Book Page 14

Okay let's look at the inline-jdbc element now. So we saw some examples of this where we specified the jdbc-driver class; this would be the fully qualified classname of the jdbc-driver class. The format for a jdbc-uri varies depending on the database. Both of these things, the driver class and the URI syntax, can be found in the documentation for the JDBC driver you're using.

If you're using one of the stock JDBC drivers, the more common ones, you'll find some examples of these inline-jdbc settings right in the entityengine.xml file. If you need a username and password to connect to the database you specify those here.

Transaction-isolation-level. This is the same as it would be for JNDI or tyrex-datasource. And again here, if your database supports it, readCommitted is typically the way to go for most applications. If you have certain specialized applications with specialized information in them sometimes even serial is justified, but if you're doing that sort of thing it's best to keep those tables in a separate database, so they don't interfere with the transactions on more highly trafficked tables.

Okay these pool- attributes are all for configuring the connection pool. Out of the box when you're using inline-jdbc we use the Minerva connection pool, which is a transaction aware connection pool, to make these JDBC operations more efficient. Minerva was a part of JBoss in the early days before they changed their licenses, and so we actually maintain a version of that in an SVN repository that Undersun Consulting runs, and the library that is a result of that code base that's in the Undersun subversion server is in a JAR file in OFBiz. And I believe it is MIT licensed and so is available to use there.

So that's the connection pool you'd be using. Typically if you're using an external application server and getting your JDBC dataSource from JNDI, all of the pool settings will be configured in the application server, just as the JDBC driver, URI, username, and password and such would be configured there.

Okay the pool-maxsize. The default is 50, so it'll keep a maximum of 50 connections in the pool. And the minimum size is 2. You may want to tune this depending on the expected traffic on your site. You'll usually want to have a good number, more connections available in the pool than you have possible connections to your web server.

So if you support 50 simultaneous connections for your web server you'll typically want to more than just 50 here. Because other things will use these as well, such as the background service execution pool, which is a thread pool that the service engine maintains. So say you have 10 threads max in the service engine pool, we'll see that in the service engine configuration, and 50 possible simultaneous connections to your web server, you'd want to have at least 60 connections available in your connection pool.

But typically much more than that. We recommend a pretty large buffer size, because sometimes connections go stale and take a while to recover, or other things can happen.

Okay the sleepTime for checking connections and the total connection lifetime can be configured here. These don't usually need to be changed, but can for performance tuning and such, the resource usage tuning.

Deadlock-maxwait and deadlock-retrywait. When there is a deadlock in the database, in other words you have 2 transactions that both have a lock on some resource and happen to be waiting for each other, this is how

long it will wait for those deadlock retries, and then the maximum deadlock wait before it rolls back the transaction. That's in milliseconds I believe, so this would be about 5 minutes before it will totally give up on a transaction, checking it every 10 seconds. So it would check it about 30 times and retry it about 30 times before it fails totally.

Okay the JDBC test statement (jdbc-test-stmt), this is something you can put in there to use just to occasionally ping the database server to make sure it's alive, which is often not necessary.

The pool transaction wrapper class (pool-xa-wrapper-class), this one is sufficient just using the default here. If you're using a transaction aware connection pool other than Minerva, which you can see uses some of the Enhydra code for wrapping a dataSource, if you wanted to use something other than that you could specify that here.

This is just a class that wraps around a non-transitional dataSource to make it a transactional dataSource. And it's needed as part of a transaction aware connection pool, because there are still JDBC drivers that exist that do not implement a transaction aware dataSource.

Okay and that's pretty much it for the inline-jdbc, and also therefore pretty much it for the dataSource element, and for the entityengine.xml file, of the entity config file as it's also known.

00:56:54 screen changes to Advanced Framework Training Outline

And that's it for entity configuration. In the next video we'll talk about entity definitions, and along with those the entity groups and delegation.

Part 4 Section 1.5

Entity Engine: Entity Definition

Starting screen Advanced Framework Training Outline

Okay now that we've covered configuration, let's go on to the next step and talk about entity definitions.

So when we were paging through the configuration stuff in the entity-engine.xml file, we saw that the delegators are configured according to entity groups, or the delegation pattern basically works through entity groups to decide on which data source to use for each entity.

00:00:39 screen changes to Framework Quick Reference Book Page 2

So each entity is assigned to an entity group. The entity group file is really a very simple one; it has an entity-group header and entity-group entries, a group name, and an entity name for each entry. So each entity does need to be assigned to a group, otherwise the entity engine will not know which data source to associate it with for a given delegator.

So here's an example of the entity group definition file. These will typically go in the entityDef directory, along with the entitymodel.xml files, and they're as simple as entity-group tags with the group name, and the entity name, the group name and the entity name. So there's an example of an entity being in a different group, and there's the name.

If you create your own entities it's a good idea to put them in the org.ofbiz group, if they're going to go in the same database as the rest of the OFBiz entities. That is the better way to do it. Otherwise you can put them in a separate group, and assign them both to the same data source.

But when it does the validation it will do them one group at a time, and so it will complain for each group about not knowing about the other entities. So you have a number of error messages on startup to ignore. The best thing to do if you create new entities, if they're going to go in the main OFBiz database, is add them to the org.ofbiz group.

And assigning entity groups is that simple. It's probably one of the more simple XML files in OFBiz.

00:02:40 screen changes to Advanced Framework Training Outline

Here's the location of the schema file, and within each component this is where you'll typically find the entity-group.xml files.

Okay entity definitions themselves. This is where the schema file is located, and the entity models entity-model.xml files will typically go here. Sometimes they'll have an extension where that * is, typically _something. In these files we define the entities fields on the entities, which of those fields are primary keys, relationships to other entities, and indices or indexes, to put an index on one or more fields of the current entity.

Also in that file are intermixed with entities you can define view-entities, and view-entities are basically made up of a number of member-entities. And then a number of aliased fields, view-links are basically how the member-entities are linked together, and then just like entity definitions they can have relationships to other entities. Then there are some special types of variations on view-entities we'll talk about.

In addition to a simple inner join you can do an outer join. We'll talk about how that looks and works.

And any time you need to do grouping and summary queries you can also do that with the view-entity; with fields to group by, and fields that represent a function rather than a simple aliased field.

00:04:42 screen changes to Framework Quick Reference Book Page 2

Okay let's go look at the details, and again we'll use the quick reference book to guide us through this. Let's start by looking at the general structure. We have an entity model as the root element. You can put some of these descriptive tags above the root element, and then you have zero to many of the entity and view-entity tags intermixed. These various descriptive tags are simply tags with a string inside of them, so a simple value for those tags.

Now for the real meat of the file. This box describes the entities, and this box describes view-entities and how they're defined. So when we were doing the framework introduction, in the Framework Introduction videos there are certain entity examples that we looked at from the example component. Let's review briefly here an entity definition and a view-entity definition. And this is a conveniently located example, so you can refer back to it whenever you need to.

This is the productKeyWord entity in the org.ofbiz.product.productpackage. The package is required; that's used for organizing the entities, although when you refer to the entity you just refer to it by the entity-name. So in other words the package is not used to separate entities into different name spaces.

Okay each field will have a name and a type. And then there are various other optional attributes that we'll talk about as we go through the reference details. But in short each one typically has a name and a type, and this type again is the type we look up in the data type dictionary, which is found in the fieldtype.xml file for each database.

Once we've specified the field we can specify which fields are part of the primary key. This entity has a two-part primary key, with productId and the keyword making up the primary key.

Here's a definition of a relationship. Type=one, which means when we follow the relationship we'll expect to find a single record. The related entity is the product entity, and the key that maps from the current entity. The productKeyword entity to that other entity is the productId.

For type one entities the entity engine will create a foreign key by default, so we do want to specify a foreign key name. The reason we want to specify a name is because these need to be eighteen characters or less. Different databases do have different constraints on that, but to be compatible to with the widest variety of databases we restrict them to eighteen characters or less as a practice in OFBiz.

A foreign key name will typically be a combination of the table that it comes from, productKeyword, and the table that it's going to, product. And to shorten it down we do this sort of a manual thing to make it look reasonable, so that you can at least partially remember what that is for if you see it later on.

Here's an example of a view-entity. Just like a normal entity, a view-entity will have an entity-name and a package. A view-entity is made up of one or more member-entities. Each member entity has an entity-alias, which will stand for a given named entity. You can include the same entity multiple times in the view-entity, as different member entity entries each with a different entity-alias. So the entity-alias must be unique for each member-entity in a view-entity.

Okay the alias-all tag allows us to automatically alias all of the fields of a given entity in the current entity. It will do that, and it will give an alias-name that is the same as the field-name unless we use the prefix attribute that's on the alias-all tag. And we'll talk about that more as we go through the reference details.

Here's the other way to alias or create a field alias, is to just do it explicitly, one field at a time. And in this case we specify the entity-alias that the field will come from, and the name of the alias to use on the view-entity. And we can also specify the field-name in the field-entity. But if it is the same as the alias-name then we can just leave it out; in other words the field-name will default to the alias-name.

Okay then we use a view-link to specify how all the different member-entities join together. So we're creating a link between the entity with the alias PPU, this one right here, to the alias OH, this one right here. So the link will go from this one to this one, and the field that links them together is the orderId. This key-map does have the same structure as the key-map in the relation element.

So when it's putting the join statement together it will know know how to link these two together. If you have three or more member-entities in a view-entity when you are linking the entities together, you can start out with any two entities in the first view-link. But the remaining view-links, the entity-alias needs to be one of those already linked in entity-aliases, or one of the already linked in member-entities. So it forms sort of a tree structure, or a list if you want to do it that way.

But whatever the case, in order for the SQL to generate properly and the structure to be valid, each view-link after the first one, the entity-alias, needs to be an already linked in member-entity.

Okay stepping back, let's go over the details of the entity definition, and then we'll go over all of the details of the view-entity definition.

So the entity element. As we've talked about each entity needs to have an entity-name, that's what it will use

to refer to it in other places. You can optionally specify a table-name, although the entity engine will automatically create a table-name based on the entity-name.

The convention for an entity-name is to use a Java style definition where we have something like this; all lowercase except the first letter of each word. So in other words we use the capital letters to separate the words. When it generates a column-name based on an entity-name, it will put an _ before each capital letter except the first one. So in other words it will use the _ to separate the words instead of the capital letter, and then it will uppercase the entire name.

So this would be in all caps. PRODUCT_KEYWORD for the table-name; that's the automatically generated table-name it will use based on the entity-name. And it does a similar thing for field-names to generate the column-names.

Okay, as we discussed each entity must have a package-name. This is used to organize the entities, though not to segregate any sort oa name space. So the entity name needs to be globally unique. If an entity definition is found with the same entity-name as one that has already been defined, it will override the previous entity-name and simply replace that.

So depending on the order of the loading of entity-model.xml files, the later ones will override the previous ones. That means if you want to change one of the entities in OFBiz you can simply copy the entity definition to the entity-model.xml file in your own component, and then define the entities as you please. And that will override the definitions in the main OFBiz components.

Okay the default-resource-name. This is used as part of internationalization, so this will be the name of a properties file typically, for example productentitylabels.properties.

00:14:45 screen changes to ProductEntityLabels.properties

I believe is one of them. Let's look at one of these real quick so you can see what these look like. This is productentiylabel.properties. The way this works is you have first the name of the entity, the name of the field, and then the primary key of the entity. This only works for entities that have a single primary-key-field, and this will be the primaryKeyId.

So the productFeatureType, the description field of the productFeatureType entity with the ID, and in this case the productFeatureTypeId of accessory will be accessory.

00:15:38 screen changes to ProductEntityLabels_it.properties

Also internationalized, in Italian that becomes accessorio.

00:15:53 screen changes to Framework Quick Reference Book Page 2

So basically to all you need to do in order to enable the internationalization in the honoring of these properties files is specify in the default-resource-name, the name of the resource to use.

00:16:03 screen changes to ProductEntityLabels.properties

Which in the example we just looked at would be the name of this file, without the .properties, simply productEntityLabels. And these go on the classpath and are found according to standard properties file naming patterns. When you call a get on a GenericValue object you can pass in a locale, and if you do then this internationalization of fields can kick in.

Now this is mainly intended to be for entities, and records in entities, that are fairly static like these: productFeatureTypes, productCategoryTypes, productCatalogCategoryTypes, productTypes, productPriceTypes, productContentTypes, goodIdentificationTypes, and so on.

They would not be a good way to internationalize things like the files on the product entity. You could use it for that but it's not recommended, as product tend to come and go, and the information about them changes over time. So we have different features in place for internationalization of things like product names and product descriptions and such.

00:17:15 screen changes to Framework Quick Reference Book Page 2

Using ties in into the content management stuff in OFBiz. But this is a simple way to internationalize fields on entities that are fairly static.

Okay dependent-on. This is not used right now, but it is part of kind of a theory of a data model, where you can specify that this entity is dependent on some sort of a primary entity. And the intent was to use this for things like data exporting or other such things, to automatically follow a tree of entities and do something with them as a group. But nothing is really implemented based on this right now.

Okay and the same, you'll note that the view entity also as a dependent-on field. Same situation there; nothing is really using it. But it was part of the initial design, as part of the theory on how a data model can fit together.

Enable-lock by default is false, but if we set it to true then it will enable optimistic locking on this entity. An optimistic lock is an alternative to a pessimistic lock, where nothing can update a record until the method

that locked it releases the lock on it. An optimistic lock basically just checks to make sure that between the time when the record was read, and when it is written back to the database, that nothing else has modified that record.

When you set this to true, as we did for the product entity, and someone was editing the product entity and they read it from the database. So Person A reads it from the database, and then Person B reads it from the database. And then Person A takes that and writes it to the database. Meanwhile Person B is still working on their copy, and they also write it to the database.

When Person B tries to write it to the database it will recognize that Person A or someone else has written something to the database so that the record that Person B is working on has changed since they read it, and so their changes would overwrite what Person A has done. And so when Person B tries to write that to the database it will throw an exception, to let them know that someone else has been modifying things. And then they can start over based on the most recent information to avoid that conflict.

Okay no-auto-stamp by default is false. The auto-timestamping is a feature of the entity engine. It will create four fields: one for transaction timestamp, create, for the create timestamp, the other for the transaction timestamp for update, and an update timstamp within that transaction. Those four fields will automatically be added to all entities and maintained as different store operations. Create and store operations are done on the database, unless you specify true here to tell it to not do an auto-stamp.

So if you're talking to an external database for read-only purposes, or to interact with data that is really part of another program, you'll typically want to set no-auto-stamp to true, so that it does not try to create those stamping fields or maintain the information in them.

So the never-cache attribute, which by default is false. If set to true it will tell the entity engine to never cache this entity. If you do a find-by-primary-key cache it will still always go to the database very time, and it will not take the resulting value and put it in the cache for future reference.

Auto-clear-cache by default is true. If you set it to false then operations on this entity will not automatically cause the cache to clear. There are only very rare circumstances where you might want to do that sort of thing, as the automatic cache clearing is very good, but such things may come up.

Then it has some descriptive attributes: title, copyright, author, and version to describe the context and owner of this entity. You can see for the most part these are not used in the OFBiz data model.

The description sub-element of the entity element can be used to describe that entity, and with text that will be maintained over time as part of the structure, a little bit more formally then putting in comments.

Okay then it has other sub-elements: one or more fields, zero to many prim-key (primary keys), zero to may relation or relationships, and zero to may index elements.

As we talked about with the primary key, each entity can have zero to many primary key fields, and to denote that a field is a primary key field you simply use the prim-key element with the name of the field. The field-name here corresponds to the name attribute on the field when it's defined.

Leading us to the field element. All fields on the entity will be defined here. A field on an entity corresponds to a column on a table in the database. Every field must have a name, the column name (col-name) if not specified will be defined from the field-name, similarly to how the table-name is derived from the entity-name. It will basically put an underscore _ before each capital letter except the first letter. So it will use an _ to separate the different words in the name, and then it will uppercase the entire thing to generate the column name.

A couple of things to note about the conventions that we use in OFBiz. We use the same conventions for naming entities and fields as we use for naming Java classes and Java fields, or Java class fields. We use a capital first letter for an entity-name, a lower case first letter for a field name.

For acronyms the first letter of the acronym is capitalized, the second letter is lower case. That is actually the Java convention, although it turns out to be very convenient here for our entity definitions. Say we had a capital I and a capital D here for example, when this was translated into a column name it would be PRODUCT_I_D. For longer acronyms you could see that becomes somewhat of a problem. So this is the conventions for naming entities and fields.

Okay moving on; each field has a type. We went over the data type dictionary, and this refers to a type in the data type dictionary, as specified for each database in the fieldtypes.xml file. So a good place to see this reference is the fieldtypes.xml file, of the database of your preference or choice.

Okay the entity engine as I mentioned before related to the credit card type fields; each field can be encrypted. If you say encrypt=true then these fields will be encrypted and decrypted automatically by the entity engine. So if someone looks at the database through some sort of a database tool they will see values in the database that are encrypted. If they look at it through OFBiz, through the entity engine, then they will see values that are automatically decrypted. So this is a

good security measure for things like credit card numbers and so on.

Each field can have a decryption for general free form English information about the field, just like the entity description. And we can have validate elements here, just like we did with the field-type XML files. The validate tag has just the name of the method to use to validate it in the utilValidate class.

These are not used very much, mainly because they are only enforced in the entity data maintenance screens and webtools, so they are not enforced in applications. Services are really where this sort of validation is meant to be implemented.

Okay the relation element is for defining relationships between this entity and other entities. So each relationship will always have to have the related entity name, the rel-entity-name, here, and can optionally have a title. If you have more than one relationship to the same entity-name then you must have a title. The name of a relationship is the combined title, if it exists, and the rel-entity-name.

So in this case down here I have a relationship from productKeyword to product. If I had two relationships to product I might use a prefix of product, and on the second one use a title of default. So the full name of the relationship then would be defaultToProduct. The titles will typically be all lowercase except the first letter, which will be uppercase. That's the convention there.

Relationships can be used for various things. Of course a type one relationship will result in a primary key being created in the database. But regardless of the type of the relationship in the entity engine API and in the simple-methods, we've seen operations like get-related, and get-related-one to follow relationships to retrieve data from the database.

Again for type one relationships. Since they do create a foreign key we'll typically want to specify the fk-name (foreign key name). It's not necessary; it can try to automatically generate a foreign key. But it's difficult to do, because usually the combination of the table name for the current entity and the table name for the related entity, and shortening those down to eighteen characters from a potential sixty characters, can result in a less than ideal foreign key name. So well typically manually specify those, as shown here.

Okay for the different types of relationships. In theory, based on looking at the key maps, the entity engine can always figure out what the type of the relationship should be. Because if the key maps all together make up the primary key of the related entity then it's a type one relationship. If they do not make up the primary key of the related entity it's a type many relationship.

So there are two reasons for the type attribute here. One of them is for sanity check, for validation, so you can specify your intentions along with the key maps, and then the entity engine can check that for you. So every time it loads the data model it validates the entire thing, and that's one of the checks that it does to validate it.

The second reason is it's also a place to specify that you want a type one but you do not want it to create a foreign key for that specific relationship, and you can use that one-nofk type to do that.

The key-map element itself is very simple: you have a field-name which is required, and then an optional rel-field-name, which is the corresponding field-name on the related entity.

It's often the case that the names of the fields on the two entities are the same. But if the name of the field on the other entity is different, then for the name on the current one you can just specify that field-name in the rel-field-name. The thing to remember here is that the field-name corresponds to the current entity, and rel-field-name corresponds to the rel entity.

Okay another thing you can do in an entity definition is declare an index. Each index must have a name. Typically the name will be in all-caps with words separated by _, because that's the name of the index that will be used in the database itself. You can specify whether there should be a unique constraint to put on the index or not, by default it's false. So it's an index for optimization, rather than for forcing a unique constraint. And index, like other things, can have description on it.

And the meat of the declaration is the list of fields. There needs to be one or more index-fields, and each one of those just has the name of the field that will be included in the index. So if you know that a query is going to be done on a certain set of fields very often then you can create an index on those fields, to make that query perform significantly faster.

And that's it for everything there is to know about finding an entity.

Now let's move on to defining a view-entity. So just like an entity, a view-entity has an entity-name by which it will be referred, though with a view-entity there is not the same constraint on the length of the entity-name. With an entity we typically restrict table-names to 30 characters. So whatever the entity-name is, when it translates into a table-name that table-name needs to be 30 characters or less. Not all databases have such a short table-name requirement, but to be compatible with the largest number of databases that is the standard that we use in OFBiz, 30 character table-names.

Same with column-names on fields, by the way, they should be a maximum of 30 characters. On that topic,

fk-names should be a maximum of 18 characters, and index-names should also be a maximum of 18 characters.

So the entity-name for a view-entity can be arbitrarily long. It's usually describing to some extent the member-entities that are part of the view-entity. And it has a number of other attributes that are exactly the same as on the entity, like the package-name, again used for organization, not for segregating the name space.

Dependent-on as we discussed. The default-resoruce-name for internationalization just as on the entity. The never-cache works the same as on the entity; by default it's false, and if you set it to true then this view-entity will never be cached. And you can cache view-entities just like other entities, and query results for them and such.

Auto-clear-cache works the same way; by default it will automatically clear the cache, but if you set it false it will not.

And the title, copyright, author, and version attributes, which are the same. As usual it has a description.

Then we get to the real meat of the view-entity with the list of member-entities that make up the view. Alias-all to quickly alias fields and alias to alias fields one at a time. View-links to link up the member-entities, and then relationship to other entities, just as with the normal entity.

Okay member-entity is very simple, just as we saw in the example. Each member-entity must have an entity-alias that is unique to the view-entity, and then each one will have an entity-name for that member-entity, and both of those attributes are required.

Next thing we'll do is fields. That will typically be done with a combination of alias-all and alias records, and I've noticed that all alias-all elements need to come before the alias elements. So alias-all allows us to specify an entity-alias for one of the member-entities, corresponds to this attribute on the member-entity. And optionally a prefix. And what it will do is go through all of the fields on that aliased entity and create an alias for them for the view-entity.

If you specify a prefix it will put that on the front. These are typically all lowercased, and the resulting aliased field-name will be the prefix, plus the field-name on the member-entity that it is coming from. With one little change, and that is the first letter on the field-name coming from the member-entity will be capitalized so that it looks like the name of a proper Java field, with a lowercased first letter and capital letters separating each word. Like other things this can have a description.

If you want to alias most of the fields on a member-entity, but not all of them, you can explicitly exclude certain fields from being aliased by simply using the exclude tag. Each exclude tag just has a field attribute on it to specify the name of the field that should be excluded from the alias-all.

For aliasing fields one at a time you use the alias element. You always need to specify the entity-alias for the member-entity that this field is coming from. The name for the alias field is also required, and this will become the field-name. When you do operations on the view-entity you can treat it just like a normal entity, except that in a view-entity the aliased fields are the fields. And this alias name is what will be used as the field-name. So you can specify it's common that the name of the field alias will be the same as the field-name on the member-entity, but if not you can specify the name of the field on the member-entity that this aliased field is coming from here in the field attribute.

Okay the col-alias is basically the column name in the jin. And it will be generated automatically if you don't specify one, but you can specify one here. This relates back to the SQL that is generated for a view-entity and is somewhat technical. But basically when an aliased field in a join statement is defined, then it will typically be given an alias to use as a reference in constraints and where-clause and order-by-clause and that sort of thing, rather than specifying the full name of the field.

That's especially important when you are using a function and that sort of thing. But there again this is automatically generated by the entity engine, so you don't have to specify one unless you want to, or need to for some reason.

Okay the primary key (prim-key) is typically true or false, to specify that the alias is either primary key or not. But this is optional, and the reason that it is optional is because typically it's simply left off. The entity engine will figure out for each aliased field whether it is part of the primary key, based on whether it was part of the primary key in the member-entity that it came from.

Function and group-by are used together. If you want to do a query that results in assembling summary data then you'll typically have a set of aliased fields that consist of grouping fields and function fields. When you do this sort of thing, because of the nature of the query that's generated, all aliases must either have a group-by or a function on them. This is so that it can take the general query results and slim them down into groups based on the fields to group by, with the functions applied to the various values within those groups to generate the aggregated elements or the aggregated values to return, the summary values basically.

These are fairly standard, some function things that you'll see in SQL: minimum, maximum, sum for addition, average for averaging them all together, counting

the fields, counting distinct values (count-distinct) so only counting unique ones, and the upper and lower.

Okay in addition to plain aliases there's also this notion of a complex-alias. Now what a complex-alias allows us to do is specify that an aliased field is made up of more than one field in a member-entity. So with the complex-alias you specify the operator, and the operator, in other words used to join the various fields together like plus or minus, these types of operations to combine the fields into a single field.

And then you specify the complex-alias. It can be a tree, so you can have other complex-alias records under it, or complex-alias-fields. Eventually you'll terminate the tree with leaves that are complex-alias-fields. But you can intermix these all you want, zero to many of either of these. So complex-alias is just obviously self referencing; complex-alias-field refers to this element.

Specifically the entity-alias corresponds to the same value that we used here, just the identifier for a member-entity, optionally a function. And you'll always specify the field name.

Okay so that's basically it for defining the aliases one at a time. The last thing to do when defining a view-entity is linking the member-entities together. That is done with the view-link element.

Here again you refer to an entity-alias, the identifier for a member-entity, and the member-entity that it's realted to (rel-entity-alias), and then key-map to specify how they're related. By default the link between the entities will be an interjoin. If you specify rel-optional = true instead of the default false, then it will consider the link between this entity and the related entity to be optional, which in SQL terms means it will be doing a left-outer-join.

So what that means in more literal terms is that if I'm linking, using this as an example, I have a productPromoUse and an orderHeader. With this definition of the view-link between these two member-entities, where rel-optional is the default of false, there must be an orderHeader for the productPromoUse record, otherwise the productPromoUse record will not be included in the view when queries and such are done.

If I want to include all productPromoUse records, regardless of whether or not they have a related orderHeader record, then I would specify the attribute rel-optional equals true here, take advantage of this attribute. And then even if the orderId was null and did not refer to an orderHeader, the productPromoUse record would still be included in the query results. And the orderId and all aliased fields that came from the orderHeader would simply be null in those records for this view-entity.

And that's what is referred to in SQL as an outer-join. And technically since it is the right that is optional, the related entity it's referred to as a left-outer-join.

Okay and that's basically it for defining view-entities. One thing to note about view-entities is that there is a way to define them on the fly; it's called the DynamicViewEntity object. This will typically be used as part of a Java method, that's a Java object that can be created.

And you do much the same thing with it, you define the view-entity, add member-entities to it, add alias-alls or aliases or complex-aliases to it, add view-links, and then you can run queries on that DynamicViewEntity. This is used for things like the product searching in e-Commerce in OFBiz, where when we are doing searching we might want to constrain by an arbitrary number of categories. An arbitrary number of product features and these sorts of things.

So we don't know when the code is being written how many categories and features we need to include in the view. Or in other words how many times we need to include the product category entity in the view, and how many times we need to include the product feature entity in the view. As well as certain other ones, too. And so a DynamicViewEntity is used to assemble the view-entity on the fly based on constraints that the user selects. We'll talk more about the DynamicViewEntity later on, when we're going over the entity engine API.

Okay and that's it for the definition of entities and view-entities. The thing that remains to be covered on this page, by the way, is the entity ECA rules, the Event-Condition-Action rules for entities. And there's an example for that down here.

00:48:23 screen changes to Advanced Framework Training Outline

And according to our outline, looks like we will be looking at that next. So see you then.

Part 4 Section 1.6

Entity Engine: Entity ECA Rules

Starting screen Advanced Framework Training Outline

Okay now that we've gone over the entity definitions, let's look at the entity ECA, or Event-Condition-Action rules. We talked about these a little bit in the Framework Introduction. But the general idea is that with the ECA rules, with both the entity engine and the service engine there are certain events that occur naturally in the processing of entity operations in the case of the entity ECAs, and calling services in the case of the service ECAs, and that's what constitutes the events.

In the ECA rules you can specify various conditions, and if all the conditions are met then the actions speci-

fied will run. So the structure of the XML file is specified here in the entity dtd/entity-eca.xsd file. And the typical place you'll find the entity ECA definitions in a component is in the entitydef directory in the eecas*.xml files.

Now generally for entity ECAs, service ECAs are excellent for tying business processes together, and for composing larger processes for other things or triggering off different things to get other processes started. Like triggering off a shipment being packed, going into the packed status in order to complete an order and trigger the creation of an invoice.

When an invoice is created then that triggers another action with another ECA rule, which checks to see if the order corresponding to the invoice, or orders, were paid by credit card; the preference is to be paid by credit card. If so then capture the credit card that had been previously authorized and things like that. Business processes like that are tied together with service ECA rules.

Entity ECA rules are a different type of thing. They are primarily good for data maintenance and not for business process. In fact they can be quite annoying and in a way dangerous if you use them for business processes. It's certainly possible to trigger off a business process from a data change at a low level like this, bet because sometimes data needs to change when you do not want to trigger those processes, then it can cause side effects that are unexpected, for example when information is being imported in the entity engine XML files, or when other code is running that does not expect that to happen.

So for data maintenance for filling in fields that are made up of other fields this is very appropriate. One good example for a entity ECA that's helpful is the inventoryItem table, or entity has a corresponding entity called InventoryItemDetail.

00:03:17 screen changes to eecas.xml

Let's look at the ECA rule for this real quick. So when an InventoryItemDetail is created, stored, or removed, before returning, before that event in the processing of these entity operations, we want to call updateInventoryItemFromDetail. And this will take the change that has just been stored in the database, and basically resum up all the inventory item details and populate inventoryItem fields with those. Namely the availableToPromiseTotal and quantityOnHandTotal fields that are on the inventoryItem, that come from the availableToPromiseChange and quantityOnHand fields that are on the inventoryItemDetail.

So there's an example of where an EECA, or entity ECA, is helpful.

00:04:22 screen changes to Framework Quick Reference Book Page 2

Okay let's go ahead and look at the definition of an EECA. Just as that one with the, inventory item, here's another one that's actually in that same file. We're triggering off operations on the product entity.

This one is just the create and store operations, so before the return if this condition is met, if autoCreateKeywords is not equal to in, in other words if it's null or yes or whatever, then run this action.

IndexProductKeywords is the name of the service we'll run. We'll run it synchronously and we'll pass in the value, the current value object that's being operated on in the productInstance ingoing attribute to this service. So basically we'll be passing that generic value object, the current product that's being changed, as the productInstance attribute going into the indexProductKeyword service. So there's an example.

Here's actually the inventoryItemDetail one that we were just looking at; it's running on create-store-remove. For return we run this service synchronously, and there are no conditions here so that one will always run. This one does have a condition, so that condition must be true in order for the action to run.

Okay let's look at the details for the different things we can specify for an entity ECA. In an entity ECA file you can specify one to many ECA elements. Each ECA element must have the name of an entity specified, an operation that the ECA rule will be triggered on, and a specific event in the processing of the operation that will trigger on it. The event is basically a combination of the operation and the event.

Let's talk about these different operations. We basically have create, store, remove, and find, and all of the combinations. This is where it becomes somewhat awkward, with all of the different combinations of create, store, and remove. So create and store, create and remove, store and remove, or create, store, and remove all together. Or the any operation; if you always want it to happen then you do the any, which basically would be the same as create, store, remove, and find. And that's the operations.

The events are basically the events of the life cycle of running an entity engine operation. There's a validate step so you can do it before the validation, before the running of the operation itself, or before the return after it's run, or for cache related things. Now these will only run if cache operations are going: the cache-check, the cache-put, and the cache-clear. So your ECA rule will run before those.

Okay the run-on-error true or false. Default is false, so if there is an error in the entity operation it will not run

the ECA rule. If you want it to run anyway even if there is an error, just say run-on-error equals true.

So we can have any combination of these condition and condition-field elements, zero to many of those, and then a list of one to many action elements. The condition is kind of like the if-compare and this (condition-field?) is kind of like the if-compare-field operation in the simple-method.

This condition will compare the named field with the given operator to the value you specify, and if you want it to it can convert it to a type. By default it uses string, but you can convert it to a type you specify here, and the format given there.

So this is very similar to how these same things are used, these same attributes exactly pretty much, that are in the simple-method and other places, Screen Widget and various other widgets too. So the type plainstring just does a two string, string conversion will do one to the format or other conversion locale where and stuff for that.

Bigdecimal converts it to that, double, float, long, integer, date, time, timestamp, and boolean convert, all those things object will just leave the object as is without doing any conversion basically.

Okay condition-field is like condition, except instead of just comparing a field to a value, we're going to compare a field to another field. Other than that it's the same. But with this you can compare two fields on the entity.

And those are the basic types of conditions you can do on the entity ECA rules. They are somewhat limited, so if you need to do more complex logic, or do a query in advance of checking something, then that will typically be done inside the action service. An action is just a service that's run through the service engine.

In order for entity ECAs to work the service engine must be deployed and running was well. The mode is either synchronous or asynchronous, and you must specify the mode to use, either synchronous or asynchronous. By the way these attributes on the action are very similar to the ones on the service ECAs, so you'll see a lot of similarity to those.

Result-to-value true or false. Basically in this case by default it's true. So what it will do when the action runs is, it'll take the result of that action, the outcoming map from the service, and it will expand it into the value object for the current entity operation. If you don't want it to do that you can just set it to false.

Abort-on-error by default is false, as is rollback-on-error. If you want it to abort the entity operation on an error then you can do true, and if you want it to roll the transaction back on an error you can set that to true, or typically it would do a set-rollback-only.

Okay persist goes along with the mode; if the mode is asynchronous then you can tell it to persist it. By default it is false, which means it will put it in the in-memory asynchronous pool to run. If you say persist equals true then when it runs it asynchronously, or when it sets it up to run asynchronously, it will save the information to the database rather than just keeping it in memory. So you're more assured that it will successfully run because the data will remain in the database until it's successful, or you can see if there's an error with it by the status in the database.

Okay run-as-user. On the entity level we do not know the user by default. There is no user passed down to the generic delegator as it is running entity operations. So when we run these services we need to run them as some sort of a user, a specific user in order to satisfy security constraints that are commonly in the services.

By default this is admin, but it has changed recently to system because the admin user is sometimes disabled. But the system user is kind of what's meant to be used for these types of things now, and actually the system user is one you cannot log into but is only used for these types of background processes.

Okay value attribute (value-attr) is basically just the same as we saw down here...I believe we had one here somewhere, yeah the value-attr. So if there is a value attribute name it will take the current value object, basically the entity value object that's being operated on for the current entity event that triggered this ECA rule, and it will put that in an attribute to pass in to the service when it's called. And the name of the attribute it puts it in will be specified here, in the value-attr, tag.

And that is it for the entity ECA rules.

Part 4 Section 1.7

Entity Engine: Entity Cache

Starting screen: Advanced Framework Training Outline

Okay the next section we'll talk about entity engine caching.

OFBiz has a generic caching utility that's used for a number of different things. It's used for configuration files, as well as user data or application data that is cached, and the entity engine cache falls into that category.

So how the entity engine itself works with the cache. The OFBiz caching facilities are basically just a number of different caches that are centrally managed, and they have an number of different features to them. And the entity engine basically just uses that cache, but of course has to have a number of facilities on top of that.

The entity engine supports caching by find-by-primary-keys, find-by-and, where you have a set of name-value

pairs that make up the constraint, as well as by-condition caching. And the condition caching is by far the most complicated.

All of these different types of caches do support automatic cache clearing, so as other operations go through the entity engine, it will run things to see which caches might be affected by that operation, and automatically clear the elements from those caches that would be affected.

And this is done by the way by a reverse association in the cache itself, as it's set up. So that when a condition or a value comes in that has been modified, it can look at what would be affected by that condition through the reverse association and automatically clear those entires without, clearing ones that would not be affected by it.

00:02:30 screen changes to Overview (Open for Business-Base API)

Okay so let's talk about the UtilCache. This is the general caching API in OFBiz. The package it's located in is the Javadoc for the base component in OFBiz.

00:02:46 screen changes to org.ofbiz.base.util.cache

In this org.ofbiz.util.cache package are the main cache classes. The main one of these is the UtilCache, the rest of these are used internally. So the cache supports a number of different features.

00:02:59 screen changes to Class UtilCache

Of course getting and setting are the basic operations, and it supports a limited or unlimited capacity. If it is limited it uses a least-recently-used algorithm to intelligently remove or limit the size of the cache. It keeps track of when elements were loaded to support an expireTime, and it counts misses and hits of various varieties.

00:03:28 screen changes to Package org.ofbiz.base.util.cache

It also supports persisting cache lines to or persistently cached to the hard disk using this JDBC jdbm open source project.

00:03:44 screen changes to WebTools Main Page

Okay to get an idea of what some of these different features do and mean, let's look at the cache maintenance page in the webtools webapp. When you log into the webtools, this framework webtools webapp which is here at webtoll/control/main, on the first page right up here at the top there's this Cache Maintenance Page.

00:04:05 screen changes to Cache Maintenance Page

So this shows all of the different caches that are currently active. You'll notice caches for different purposes here; we have properties or resource bundle caches, and we have BeanShell script caches. We have FreeMarker template caches, and Screen Widget caches.

And there are also Form Widget and other widget caches, though we haven't hit any of those yet so the caches are not yet active. Since this starting of this OFBiz instance, the only thing I've hit is the webtools webapp, and we haven't hit anything there that uses the form or other widgets yet, so none of those caches are showing up.

All of these that have the entity. prefix or enticache.entity prefixes are the entity engine cache. So these entity. ones by the way are actually the configuration for the entity engine. And the entitycache.entity ones, and there are actually some variation on this, are the actual database caches. So the serverHitType here is the name of an entity, as is userlogin, and we can see it's caching some of those records.

00:05:56 screen changes to OFBiz E-Commerce Store: Featured Products

Let's hop over here and load the main page of the e-Commerce application. E-Commerce is highly tuned for performance. So most of the data that's shown to the customer, all of the product information and such, category listings as well as product details, are kept in the cache.

00:06:04 screen changes to Cache Maintenance Page

If we go over here and reload the findUtilCache page, now we're seeing a whole bunch of other entity caches, and we're also seeing some variety. So we have entitycache.entity-list, as well as entitycache.entity. These would be the single entry ones, find-by-primary-key in essence. And these would be the multiple entry ones so they'd be in a list, and these are typically find-by-condition. Now one cool thing you can do with the cache is look at the elements in the cache. Let's look at this product cache for example, you can look at the elements in the cache.

00:06:43 screen changes to Cache Element Maintenance Page

Now if the cache is very large this isn't terribly helpful, but if it's small then the string value of the key can be very helpful for identifying what is in the cache. You can also individually remove lines from the cache here, so if I remove this line then there it goes. And then of course next time the application looks for that, when it finds it through the entity engine the entity engine will not find it in the cache, and so it will look it up again.

00:07:13 screen changes to Cache Maintenance Page

Okay let's look at some of the statistics related to the cache. This is the cache size, the number of elements

in the cache, the number of these on it. These are the successful cache hits, meaning that when a program looks for something in the cache it actually finds it already in the cache, and so retrieves it from the cache. And you can see certain things are hit very frequently, a very large number of times.

Okay for misses there are a number of different varieties. The first number is the total number of misses. The second number is the number of misses because the element was not found in the cache. On initialization all of the elements will basically be misses, because they are not yet in the cache. So that's the initial, loading, and you'll see misses for that.

The next number up here as described is exp, these are the expired elements. So a program goes looking for something in the cache, the cache finds it in the cache but sees that it's expired, and considers it a miss. And it will return null, just as if it was not in the cache.

Okay this last number described up here is SR, which stands for soft reference. Certain caches do not use a soft reference; the configuration ones will typically not use a soft reference because even when memory is low we do not want to cannibalize those caches. However for the entity engine caches, all the database caches do use a soft reference, because when memory gets low we can just remove those elements to free up some extra memory.

Now when that happens, so when the Java Memory Manager recognizes that we're getting low on heap space, it will first do a garbage collection, and then increase the heap size. Before it increases the heap size it will go and follow soft references, and remove the objects associated with the soft references.

So anywhere we're using soft references here, the Java Memory Manager can go through and wipe out these soft references to free up memory. Now when that happens you'll see misses here because of soft references. So the UtilCache can keep track of that, can see that the main object was deallocated. But it still keeps a placeholder so it knows that there was a soft reference there, but the Java Memory Manager has taken that away.

By the way, to avoid that when you're ramping up it's good to start out with a large initial or minimum size for the heap. So that's the – capital X, lower case m, lower case s parameter when starting up Java. So those are the basic things there, and this also counts removes.

Okay showing the maximum size and expire time. If they're set to zero then there is no maximum size or expire time. When there's no maximum size it doesn't maintain the most recently used list, so it does run more efficiently, and removes go faster. The expire time has a very minimal performance impact.

One thing you'll note, as I've mentioned before. Out of the box OFBiz comes with settings that cause a number of things to reload automatically, and that's done through an expire time on the cache. So for example here with the BeanShell parsed caches they expire every ten thousand milliseconds, or ten seconds. Same thing with FTL templates and so on; they're configured to expire every ten seconds.

So for production that wouldn't be a very good thing, you would want to have all of these expire times set to zero. But in development it's very handy so that you don't have to manually clear the caches or otherwise do something to reload the files, such as restart or that sort of thing.

Okay we talked about the use soft reference (useSoftRef) and what that means. UseFileStore just asks should this be persisted in the local file system or not. Persisted caches do have somewhat of a performance overhead, but in cases of certain things like stopping and restarting, as well as low memory conditions, they can be helpful. So that's another thing another part of the cache management tools here that you can experiment with when tuning the cache.

And the administration column. We already looked at the elements page where you can see the individual elements in the cache. When you're looking at the configuration caches, by the way, like this template.ftl.

00:12:52 screen changes to Cache Element Maintenance Page

If we look at these elements we'll see the locations of the different FTL files that are in the cache.

00:12:58 screen changes to OFBiz: Web Tools:

And with the entity ones you see the name of the entity and the condition, either the conditions for the list ones or the primary key name-value pairs of the single one.

Okay the clear button will just clear the cache. There's also a button up here to clear all cache, clear all expired entires from all caches, manually trigger the garbage collection to run, and finally just something to reload the cache list, a little convenience thing.

00:13:27 screen changes to Cache Maintenance Edit Page

Okay the edit button over here for each cache allows you to see statistics about the specific cache, as well as change the maxSize, expireTime, and whether or not to use a soft reference. So you can change all of those things on the fly, as well as there's another clear button and such.

00:13:46 screen changes to OFBiz: Web Tools:

Now this timeout where we have these things set to ten seconds; you can either set those on the fly with the on

the edit page, or you can set default values in the cache.properties file.

00:14:06 screen changes to cache.properties

So here's the actual cache.properties file from the current revision in the OFBiz source repository.

00:14:21 screen changes to Framework Quick Reference Book Page 16

Before we look at this, there is a little example of the cache.properties in the reference book here on page 16. This is the last reference page before we hit these diagrams.

So it has a little example of the cache.properties where the file store is, and then the name of the cache, .maxsize, .expiretime, and .usesoftreference. Just to kind of remind of you of what some of these things are.

00:14:50 screen changes to cache.properties

So that's a place where you can look at it quickly to remind you of what's there, but of course the actual cache.properties file has all of the details.

Now these guys at the bottom here, this is where we're setting all of the expire times for these different caches that basically make up the configuration file. So we have the simple-methods, BeanShells, parsed classes, BSFevents, compiled Jasper reports, and controller-config, this is the controller.xml files. And then Form Widget, Menu Widget, Screen Widget, Tree Widget, FTL templates, and even DataFile models.

There's some other things up here, like the product configuration has a cache, product content rendering has a cache, and those have 60 second expire times.

Okay so basically you can set any default, any settings for any specific caches, here in the cache.properties. Up here at the top you can also set the default settings, so for all caches you can give them a maxSize, you can give them an expireTime, and you can give them softReferences. You want to be a little bit careful with those settings, because these different cache settings do have a very real impact on performance.

So going into production, typically you'll just comment out all of these expireTime lines; these two, and all of these down here, so that in a production mode it's not reloading these resources constantly.

00:16:35 screen changes to Advanced Framework Training Outline

Okay hopping back to the browser. We've looked at the UtilCache API, and the webtools cache pages, and the cache.properties file.

Now we're to automatic and manual cache clearing. Whenever you do entity operations there are some methods in the entity engine API that allow you to specify not to clear the cache, but for the most part they will just automatically clear the cache. And we'll talk about that in more detail in just a minute. All of this caching is something that the GenericDelegator object manages for you.

Typically the different methods will have a variation that has cache appended to the name. So for example findByPrimaryKey versus findByPrimaryKeyCache which will do the caching, findByCondition versus findByConditioncache and so on.

So there is good automatic cache clearing that is generally very reliable, regardless of the type of cache you use. It even works well on view-entities and other such things to manage those caches.

But you can manually clear cache elements or clear all the caches through certain parts of the entity engine API.

00:17:58 screen changes to Class GenericDelegator

This is the GenericDelegator class. We'll be looking at this in detail in a minute.

But there are a few methods for clearing all caches (clearAllCaches), clearing just a single line from the cache (clearCacheLine), or clearing the cache line by condition (clearCacheLineByCondition). So this is either passing in the GenericPK, GenericValue object, or the condition, which are the most common ways of using these, or just clearing all caches. You can also do it for a collection of dummy PKs or values.

So these are some methods that are on the delegator for manually clearing the cache. These are not very commonly used anymore, because the automatic cache clearing is sufficient to take care of most things, or pretty much all things basically.

00:19:04 screen changes to Advanced Framework Training Outline

Okay distributed cache clearing we talked about a little bit with the entity engine configuration. Let's look at this again.

00:19:15 screen changes to entityengine.xml

For a given delegator there's this distributed-cache-clear-enable, so you'd set that to true. And then there are a couple of other elements to specify, the distributed-cache-clear-class-name and the distributed-cache-clear-user-login-id.

This has a default value that is typically sufficient, the entityext.cache.entitycacheservices class. So you can typically just leave that attribute off unless you're using you own class to implement the distributed cache clearing, which you can certainly do. Also the distributed-cache-clear-user-login-id by default is system, which is typically the best one to use. So you can leave that

one off as well. The main thing you need to do here is set distributed-cache-clear-enabled to true.

Okay now let's follow it through the rest of the chain, just undo all these changes real quick. Okay so the entityCacheServices class is the one that is used by default for the distributed-cache-clear. There is an interface in the entity engine that specifies the methods that you need to implement for distributed-cache-clear. These are the ones: setDelegator, distributedCacheClearLine, distributedCacheClearLineFlexible, byCondtion, and another line with a GenericPK and clearAllCaches.

These are different methods that you need to implement here. Basically what will happen is, as different cache clear operations go through the entity engine, either automatic or manual, these distributed methods will be called and will do their thing. The typical pattern of these is they do need to have the dispatcher set, and that'll be automatically set up by the entity engine.

It gets the userlogin to use based on the setting, so this would be the systemUserLogin, as we saw there. And then it runs a service asynchronously called distributedClearCacheLineByValue.

Now that service is defined over here. This by the way is in the entityExt component. It's not in the main entity component because it uses the service engine, and the entity component has no dependence on the service engine. So there's certain entity extensions that go in the entityExt component, as shown in the package entityExt here, that do use the service engine and so are in a separate component.

00:22:38 screen changes to services.xml

So this service right here is also defined in the services.xml, in the entityExt component. So we can find that service by default.

Okay all of the services basically will have a service named like this that has distributed at the beginning. And basically when you call this service it will result in a remote service call. In this case it's using JMS, Java Messaging Server, to send a message to a JMS server that can then be distributed to other servers in a cluster.

So the location here is the JMS definition in the service-engine.xml file. We'll talk more about JMS and other remote services when we talk about the service engine. But basically what this does is when you call a service, it's going to just call a service remotely through this JSM server, and this is the service it will invoke remotely when the service message gets to the other side. So as it describes here, clearCacheLineByValue for all servers listening on this topic.

Here's a topic associated with the location here, with this JMS server. And it will send a message to that topic, and everything listening to that topic will end up calling this service locally, with this value being passed in.

So each of these is a service that will be called on a remote machine on some other server, or pool of servers in the current cluster. So each one of those is going to have that service defined as well as here.

This is the actual definition of the service, where it's implemented. It's going to call the clearCacheLine method there and pass in these two parameters. And then that service will actually talk to the entity engine on the local machine where it's being called, and do the corresponding cache clear operation.

So in that way these cache clear messages get sent out to the other services via a remote call that goes through the service engine. When you call this service locally it will, via JMS, call this service remotely. And since it is a JMS topic it can be on potentially many different servers as intended. And on each of those servers it will call the named service here, as defined here, and actually do the cache clearing on that server. So there are different ways to use the service engine to distribute, JMS is just one of them.

You can also use a service group to call a group of services, each service in the group representing a remote machine, and then those can be called directly via HTTP or RMI. But the most convenient and efficient way typically is to use JMS because a single message will be sent to the JMS server, to the topic on the JMS server. And then that message will be distributed or picked up by all of the other servers listening to the topic, and they will get the cache clear messages coming through.

And that's basically how the distributedCacheClear works. You can change the definitions of all of these services that start with distributed, in order to change how those are called remotely. But basically these clearCacheLine services you'll always leave the same; those can just stay as is.

By the way, there's some examples here of using the HTTP engine for remote calling, and these actually just call the service locally. This is the one we were looking at before, clearCacheLineByValue, so localHostClear using HTTP.

And that's just a very rough example. If you wanted to use a service group then you could change the distributedClearCacheLineByValue, for example, to use a group, engine-group. And the invoke would be the name of the service that represents the group, or the name of the service group. And then you define a service group, and we'll talk about how to do that in the section on the service engine. Then the service engine would call each service in that group, and so each service in that group would then represent a remote server

that's in the cluster. And off you go; you get the clear-Cache messages distributed.

00:28:04 screen changes to Advanced Framework Training Outline

Okay so the reason we need something like this, a distributed cache clearing, is because each application server has its own cache that represents values from the database. So as other servers make changes to what is in the database, it sends out messages to the other servers in the pool that a change as been made and so it should clear the corresponding cache lines.

Okay and that's basically it for entity engine caching. Next we'll talk about the entity engine API, and go into some details on that.

Part 4 Section 1.7

Entity Engine: API

Starting screen Advanced Framework Training Outline

Okay we've now finished talking about the entity engine cachings. We're ready to go to the next section, the entity engine API. Now API in this case is typicall for programing APIs. It stands for Application Programming Interface, and is the basically the Java methods and such for interacting with the entity engine. So if you're not familiar with Java this is going to be fairly technical, fairly specific to the lover level entity objects and how they are used.

00:00:54 screen changes to Overview

Let's start by looking at the Javadocs. The entity engine itself is in this org.ofbiz.entity package, and we have various sub packages for different things. A lot of these main objects that you'll be working with, when using the entity engine, are things that just kind of work behind the scenes, and the main objects you're dealing with are in this org.ofbiz.entity package (briefly changes to that).

00:01:25 screen changes to Advanced Framework Training Outline

This is of the order we'll go in; we'll look at some of the factory methods, and then the basic CRUD or CReate Update and Delete for creating, storing, and removing methods. And then we'll go over some of the different finding things, including some of the more complicated ones like the entityListIterator and the DynamicViewEntity, the GenricEntity API with the GenericPK and GenericValue objects. And we'll talk a little bit about the sequenced keys and how they're stored in the database and such.

And then this last bit about the GenericHelper interface is kind of a lower level entity engine thing that could be useful in some circumstances to implement your own helper, like for testing purposes and such. But for the most part you'll just use the default helper that comes out of the box.

00:02:28 screen changes to package org.ofbiz.entity

Okay going back let's look at the GenericDelgator and its factory methods. The delegator is the main object you're typically working with when you're working with the entity engine, and it has a methods for doing all sorts of different things related to the different database operations.

00:02:48 screen changes to Class GenericDelegator

The first set that I want to look at are the make methods. As you can see it has quite a few methods on here. I'll tell you right now a lot of these methods are meant for convenience or have seeped into the API over time, and there's really a fairly small subset that you'll need to use when writing entity engine code in Java.

We've already looked at the equivalent of most of these in the simple-methods and other places in the Screen Widget and those areas with actions and such. So you've seen the basic operations you can do with the entity engine. We'll look at a few of them that are more advanced, but for the most part the ones you use on a regular basis are just those basic operations.

So different factory methods: make primary key (makePK) or makeValue, so GenericValue represents a row in the database just a value object. GenericPK represents a subset of a row in the database, just the primary key.

With either of these we have some methods here to make the value object from an XML element, or the primary key from an XML element. These are used for parsing the entity engine import XML files.

But for the most part in your code, you'll be using this format where you pass in the entity name and the map of fields, name-value pairs for the fields. You can just pass in null to leave it empty, and then set values after the fact.

The same thing is true with the makeValue object for doing a GenericValue that has all of the fields in the entity, and not just the primary key field. Which is the case with this GenericPK and makePK methods. So makevalue is meant to hold everything, and you'll typically pass in an entity name and map of the fields. Again these are just the name-value pairs, and they can be null if you don't want to do anything special there.

So once you have a GenericValue object that you've made, typically these factory methods will be something like the makeValue. One in particular is something that you'll run before calling the create method. So let's hop

up and look at that. We have a number of different creates, and typically you'll pass a GenericPK or a GenericValue object, and it just does the create.

00:05:52 screen changes to Class GenericValue

One thing to note about the create is that if we look at the GenericValue object it also each GenericValue object has a reference back to the delegator that it came from, and so it just has a create method. So when you call create on a GenericValue object it will just pass itself to the delegator and take care of everything underneath, and there are some variations on this as well, these static methods.

These ones by the way, these static methods that return a GenericValue, these are really factory methods, more in make methods than create.

00:06:32 screen changes to GenericDelegator

Okay here's a little variation on creating, where you pass in a GenericValue and createOrStore. Basically what this will do is what the storeAll method does for each entry in a collection, but this one will do it on a single value. And what it will do is, if the value you pass in already exists in the database, then it will update it or store it. And if it does not exist in the database then it will create it or insert it.

Now for other operations. To get a single GenericValue object you'll usually use a method like findByPrimaryKey, where you can pass in a generic object. Or more commonly you'll pass in the entityName, and the map with the fields of the name-value pairs that make up the primary key.

FindByPrimaryKeyCached is the cached variation on that. As I mentioned with the servifdn methods they'll typically have an alternate method with a cache suffix, that you can use to take advantage of the entity engine cache. But this is the same sort of thing; pass in the entityName, the name-value pairs in a map that make up the primary key.

Once you've found the GenericValue object, either through the findByPrimaryKey or other find methods that we'll talk about in a minute, you can do these various other operations on it, the store and remove operations. Now there are those methods on the generic on the delegator, but typically once you have a GenericValue object it's either to just call those right on the GenericValue object.

00:08:32 screen changes to Class GenericValue

So just like the .create we also have a remove method and a store method. And again these work by the GenericValue object keeping a reference to the delegator that it came from, and then it just passes itself to the corresponding method on the delegator. So that's how you can update it in the database with the store method, or remove it from the database with the remove method.

00:09:09 screen changes to GenericDelegator

Okay now let's look at some more of the intricacies of finding. Another common find method that you'll see is some of these findByOr or findByLike which is an and where instead of equal it does a like.

FindByAnd is the most common one, so we have findByAnd and findByAndCache. You can pass a list of expressions and it will add them all together, although typically the way findByAnd is used, and this is one of the methods that has remained since very early in the days of the entity engine. So here's the more complete findByAnd method where you pass in the entity name, name-value pairs where the name is the field name and the value is the value that that field needs to equal to be included in the list. And all of the fields there will be anded together, hence the name findByAnd.

You could also pass in a list of field names with an optional + or – at the front, to specify an ascending or descending sorting, but a list of field names that will be used to order the returned list. Then there is of course a cached variation on these. So findByAnd is a common one that you'll use.

Another one you'll use in some cases is findAll, either just passing in the entityName or passing in the entityName and an orderBy list, a list of fields to order by.

Okay getting down into more complicated queries, we can do things like findByCondition. Here's the most complete findByCondition.

00:11:07 screen changes to findByCondition Javadoc

These do have decent Javadocs typically on them by the way. So it'll tell you what each of these parameters means. With the findByCondition, basically instead of passing it name-value pairs or whatever, we're going to pass it an entityCondition object, and we'll talk in detail about what that is in a bit.

With this one you can pass in a wherecondition as well as a having condition. Having condition is what will be applied if you're using like a view-entity that has grouping. The having condition will be applied after the grouping is done, whereas the where condition is applied before the grouping is done. And by grouping I mean groupkng, and functions, the summary functions and so on that are done as part of that sort of query.

Okay you can also pass in the fieldsToSelect. If you pass in null it will select all the fields, or you can just select a subset of fields on the entity. Again we have a list of fields to order by, and then entity findoption, which has some options to specify what will happen on a lower level in the query.

00:12:25 screen changes to GenericDelegator

This isn't quite as important here for the findByCondition as it is when using the entityListIterator. Before we look at the details of the entityCondtion class and entityFindOptions, actually entityCondtion is an interface with a number of implementations, let's look at the findListIteratorByCondition.

Here's another useful method by the way that's kind of similar to these byCondition ones like the findByConditon and findListIteratorByCondition. FindCountByCondition will run a query and simply return a long with the number of results. That can be helpful to see if there's anything matching a certain set of conditions in the database.

So findListIteratorByCondition. This second one here is the most complete; it has pretty much all the fields you might ever need for doing a query or a find. These can be left null, so for example if you leave the whereEntityCondition null then it'll return all records, unless there's something in the havingCondition. Again the havingCondition is typically just applied when there are grouping and functions in a view-entity definition.

FieldsToSelect, the orderBy, the findOptions, just like the ones that we saw above, except the difference here is it returns an entityListIterator object. And the entityListIterator is helpful when you don't want it to pull all of the results from the database into memory, into a list all at once. And the reason sometimes you don't want that is because the potential size of the resultList is so big that it wouldn't fit into memory. And so it would tend to not work, as well as possibly cause problems for other things that are trying to run on the application server.

So if you think there might be a very large resultSet you can use the entityListIterator to page over that resultSet, rather than pulling all the results over at once. Now this does keep a connection open to the database, so it is necessary to close it when you're done with it. And we'll talk about that more in just a bit.

Let's go back to some of these objects we were looking at. The entityCondition basically represents in a single object an entire where clause. It is basically what it is. So an entityCondition can be composed of other conditions, or it can be just a single expression, or all sorts of things.

00:15:26 screen changes to Class EntityCondition

Let's look at this class. I guess it's in an abstract class, so you have to use one of the sub classes, and there are a few of them that are commonly used. The most basic entityCondition I guess we can look at the API here. Basically an entityCondition has an evaluate method (eval) to see if the condition is true, making a where string (makewherestring) to generate the actual SQL from the condition, and stuff like that.

00:16:16 screen changes to Class EntityExpr

Okay the subclasses that you'll actually use for an entityCondition are things like the entity expression (entityexpr). With an entity expression, when you construct it you'll typically have a field-name for the left hand side, let's use this one it's a little bit simpler. So a field name for the left hand side (lhs), a comparisonOperator, and an object for the right hand side (rhs). So this would be something like productId is not equal to null, or is not equal to 23, or that sort of thing.

You can also use the entityExpr to do two entityConditons and join them together.

00:16:57 screen changes to Class EntityJoinOperator

So a joinOperator is going to be something like and, or let's see if this has a list of them in here; things like and, or, and these sorts of things.

00:17:20 screen changes to Class EntityExpr

Whereas the comparisonOperator is things like equals, not-equals, greater-than, less-than, like, in-between, those sorts of operators. So that's basically what you can do with an entityExpr.

For combining entityExprs, one way to do that is with another entityExpr that wraps other conditions.

00:17:49 screen changes to Class EntityConditionList

But more commonly what you'll do is use one of the the entityConditionLists, so like the entityConditionList class itself basically allows you to take a list of conditions and specify the operator to use to join them. So the list of conditions could have a bunch of expressions in it, and you could say use the operator to and them together, or or them together, or of that sort of thing.

So with these combinations of objects you can actually create fairly complicated condition structures, because each entry in this condition list is an entityCondition. And so it could be another entityCondtionList, an entityExpr, an entityExpr that joins two other conditions together, number of these things can be done.

00:19:00 screen changes to GenericDelegator

Okay let's step back to the delegator. The other object we wanted to look at was the entityFindOptions object.

00:19:07 screen changes to Class EntityFindOptions

And this has things on it like whether the results should be distinct. In other words whether it should constrain the query to only return unique rows, so any row that's not the same as any other row.

The fetchSize, all these getter methods by the way have corresponding setter methods. So the fetchSize tells the JDBC driver, when communicating with the database, how many results to pull over to the application server each time it talks to the database. A typical number for this is about 100 results. So it'll do communication with the database, pull over 100 results, process them locally or buffer them locally, and when those are used up it'll pull over the next 100, and so on.

MaxRows is the maximum number of rows to return. If you specify that you only want it to return ten rows it'll just return the first ten results. Typically you'll want to leave this at zero, to return all rows.

Okay resultSetConcurrency. There are some constants for this that are inherited from I believe the connection. So we have concurrency ready only, and updatable (CONCUR_READ_ONLY, CONCUR_UPDATABLE). Those are the two options for concurrency. So it is possible to have a resultSet that's updatable. Typically with the entity engine the API really only supports the read only stuff, not updating the resultSet on the fly and saving it back to the database on the fly.

So this you usually just leave as CONCUR_READ_ONLY. The other one is the resultSetType, as the comment says here how the resultSet will be traversed.

So here the different types: TYPE_FORWARD_ONLY, TYPE_SCROLL_INSENSITIVE, and TYPE_SCROLL_SENSITIVE. With this insensitive and sensitive, what it's talking about being sensitive to is changes in the database.

So if it's insensitive it performs better, because it doesn't have to constantly communicate with the database to see if anything has updated. FORWARD_ONLY is the highest performance but it has some limitations. Some JDBC drivers have some little quirks in them; they won't even let you jump forward to a certain result number when you do FORWARD_ONLY. You have to do SCROLL_INSENSITIVE or SCROLL_SENSITIVE even to jump froward to a specific result number.

And by the way doing this sort of thing, jumping forward to a specific result number, getting a subset of the results, the best way to do that is with the entityListIterator. So let's take a look at that.

00:22:19 screen changes to EntityListIterator

When it returns an entityListeIterator typically what you'll do, if you want to iterate through the entire thing, is just call next over and over.

The code will typically look something like this. Where your the code to execute for each value in the result will go inside the { } braces there, or the ellipsis. So we're just saying while nextValue, which equals this.next, or entitylistiterator.next is not equal to null, then carry on and do something. And that's the best way to do it.

You can use the hasNext method along with the next method, which is more common for an iterator, but as the note mentions here that's not a good way to go because of the way the JDBC resultSet is implemented. And again it shows the recommended way of doing it here.

The reason that we recommend doing it this way, instead of using the hasNext method, is that when you're doing the hasNext it needs to check to see if there's another result. Which means it needs to scroll forward once and then scroll back to the current result in the resultSet.

So this causes a number of problems, as you can imagine. If scrolling forward happens to jump forward to the next, like outside of the current results that have been fetched from the database, it will cause it to fetch another set of results and then move back. And if the JDBC driver isn't optimized for that sort of moveback, then it may have to do another fetch of results to move back to the result that it was on before. So the best thing to do is just move forward only and go with the next. That's the most efficient.

There are also some convenience methods in here so you don't have to go next over and over. Like getCompleteList, which will make a list of GenericValue objects that has all of the results.

And then more helpfully for paged user interfaces, and this sort of thing, is the getPartialList. You pass in the result number to start at and how many elements you want it to return, and it will return at most that number of elements.

Okay some other things you can do here that can be helpful. Relative allows you to jump forward or backwards from the current position. You can also go to the last or afterLast, up here. Or first or beforeFirst. So you can set the position of the cursor in the resultSet. Absolute is kind of like relative, except instead of going forward or backward from the current position it just jumps to an absolute ro number, or result number.

And those are some handy things you can do with the entityListIterator. So you don't have to just get a list back, which is basically what getCompleteList does. Which by the way is what the entity engine uses to get a complete list, internally anyway, for the other methods.

The lowest level database interaction method always uses the entityListIterator, so this method is actually called quite a bit. But anyways instead of just having to do with a full list of results you can have this greater flexibility to move around the resultSet. Such as getting partial results back with partialLists and other things like

that, to help you deal with queries that may potentially have a very large number of results.

Again I remind you this does keep a database connection open while you're using it, so when you're done with an entityListIterator it's important to close it. If an entityListIterator is garbage collected, and hasn't been closed, then it will close itself. But at that point the connection may already be stale or that kind of stuff, so it's best to close it right away.

And when it does close through the garbage collection, when the object is destroyed, it will show an error message about the fact that this particular entityListIterator was not closed. So that's something that's important to do. The entity engine does that for you for other things, when you're just getting a full list back and stuff. But in this case, where you have more control over it, you also have more control over when to close it.

00:27:20 screen changes to GenericDelegator

Okay so those are some of the more advanced objects for doing entity engine queries.

00:27:31 screen changes to Class EntityCondition

I should note one of the EntityCondition operations here. This is a newer one, by the way, the dateFilterCondition.

00:27:38 screen changes to Class EntityDateFilterCondition

Basically it allows you to simply specify a fromDateName and thruDateName, and it can filter by the current timestamp and stuff. So that's just a convenience condition type that can be done with other things, like with entity expressions. But doing it with this is a little more efficient. Well programming wise it is; execution wise it's the same.

00:28:07 screen changes to Class EntityWhereString

Okay EntityWhereString. One of the basic concepts of the entity engine is to not have to deal directly with SQL, and one of the big reasons for that is that different databases have different variations on SQL that they use. So if you have SQL in your code when you move from one database to another it might break things. Which is why there's this big warning here.

But the EntityWhereString basically allows you to specify the SQL that will actually be sent over to the database, for the WhereString of the query. And it can be combined with other conditions like the expression, conditionList, and so on, all of which are just turned into SQL strings and strung together with their various combine operators.

And of course those combine operators are things like the ands and ors and whatnot.

00:29:22 screen changes to Advanced Framework Training Outline

Okay so that's the basics for the conditions. This is another shortcut one by the way. EntityFieldMap is like entityExpr, but you pass in a map, and it just sets the names equal to the values. You can specify the combine operation here; typically an and, but it can be an or and such.

Okay we talked about the entityListIterator, now let's talk about one that's a little bit more complicated, the DynamicViewEntity.

00:29:53 screen changes to GenericDelgator

So here in our findListIteratorByCondition methods, these are the most feature rich query methods there are, there's one where instead of passing it an entityname you pass in a DynamicViewEntity object.

Now just a little refresher when we look at an entity model, let's see if we have a view-entity in here. Here's one that's full of view entities.

00:30:31 screen changes to entityModel_view.xml

Okay when we have a view-entity, remember we specify member-entities, then we alias fields, then we link the member-entities together. Now this is a static view-entity, or a predefined view-entity that has a name.

00:30:52 screen changes to GenericDelegator

But with the DynamicViewEntity class we can also build that view on the fly programmatically. So if we have varying sets of conditions where we don't know exactly what's going to be joined into the query in advance, but instead want to limit it based on the conditions selected, then we'd use the DynamicViewEntity to support that sort of thing.

One good example of this is actually in the PartyManager. In some databases we'll have hundreds of thousands or even millions customers, and the parties in the database. And so if we used a static view-entity that always joined together all of the possible entities that we might want to query by our show information form, then we'd end up in a very wide temporary table that takes a long time and a lot of resources for the database to assemble.

But if we just take the specific conditions that the user specifies, and only include the necessary entities and necessary fields from those entities in our query, then we can minimize the size of the temporary table and the number of entities that have to be included in it. And it performs significantly faster typically. So there's one application of the view-entity.

Another application of the view-entity is in e-Commerce. Where we're doing different product searches, we might have different combinations of

categories that we're filtering by different features, different keywords, and all of these basically just become dynamic additions to a DynamicViewEntity that turns into a big SQL query that's sent over to the database.

00:32:55 screen changes to Class DynamicViewEntity

So when you're creating a DynamicViewEntity it's just like a static view-entity, except instead of using XML to create it you call methods. So first you create the DynamicViewEntity, you just create an empty one. And then first thing, you want to do things in the same order that you would in a static view-entity, with these different add methods here.

00:33:24 screen changes to entitymodel_view.xml

Again a reminder; the first thing we're doing is adding member-entities, then aliasing, then view-links.

00:33:32 screen changes to Class DynamicViewEntity

And we'll do the same thing here. We'll add member-entities and tell it the alias and the entity-name; those are the same as the attributes that we use in the static view-entity. Now for aliasing we can alias-all by specifying entity-alias and an optional prefix, or we can alias individual ones by specifying the entity-alias and name.

We have a few different combinations here that represent the defaults that are in the XML file. This is the most complete addAlias, where you specify everything. The entity-alias, the name for the current alias, the field on the empty alias that this will correspond to, a column alias (colAlias) so we can override the column alias-name that will be used in the SQL query in the join clause. Boolean whether it's a primary key or not, which the entity engine can figure out, so you can pass that null typically. GroupBy true or false will typically be null, which defaults to false. And typically you won't have a function unless you're doing some sort of a summary DynamicViewEntity, where you want to have grouping and functions involved.

And then if you want to use a complex-alias then you pass in an instance of this call, which is just like the complex-alias in the static view-entity, where you can have fields that are added together and all that kind of stuff.

Okay so once we have the aliases in place the next thing we do is add view-links (addViewLink). So entity-alias, the related entity-alias (relEntityAlias), and whether the related entity is optional or not (relOptional). If you specify a null here it will default to notOptional, and a list of the key-maps (modelKeyMaps).

So the key maps are those field-name and rel-field-name tags. And basically there's a class here related to the DynamicViewEntity class that you would use to put in this list and pass over to the addViewLink.

00:36:02 screen changes to PartyServices.java

Okay some example code. A good place to find example code for this; the partyService that I mentioned earlier is a good one, in partyservcies.java.

Here we are, the findPartyService. And if we look down here a ways we'll see where it creates a DynamicViewEntity, and then it goes through adding member-entities and aliases for them. As necessary it's adding other member-entities, and aliases, and view-links to link them in, and typically also we'll add some stuff to the expression for actually doing the query based on the DynamicViewEntity.

So here again we, depending on the different possible fields that the user selects to limit the query by or constrain the query by, we will be dynamically adding in these various entities to the view-entity: partyGroup, person, userlogin, and so on, there are various of them. PartyContactMech and postalAddress for an address search, and so on.

And then when we get down to actually doing the query we pass in the DynamicView, the condition we've put together, and so on. Here's an example by the way of the entityFindOptions, where we have SCROLL_INSENSITIVE, CONCUR_READ_ONLY, and that sort of thing. True for distinct I believe, and what was this last true.

00:38:11 screen changes to Class EntityFindOptions

And we find that right here. So the first one is whether to specify the type and concurrence (specifyTypeAndConcur) when we're doing the query, and the last one is whether to add the distinct constraint or not.

00:38:29 screen changes to PartyServices.java

There's an example of that in action. There's also some examples of using the different EntityConditon objects, like entityConditionList where we have and expressions (andExprs). This is basically a list of expressions, as you can see here. We're just adding these entityExpr elements, which are instances of entityCondition, and so we can make an entityConditionList out of them and combine them together with the and operator.

And that's really all that's doing.

And that actually becomes the main condition that's passed into the query here.

00:39:23 screen changes to Advanced Framework Training Outline

Okay and that is the basics of the GenericDelegator API, and related objects like the entityListIterator and the DynamicViewEntity.

Okay we talked a little bit about the GenericEntity API, and GenericPK and GenericValue. So GenericEntity is an object that has all of the main methods and main functionality for this.

00:39:54 screen changes to Package org.ofbiz.entity

And GenericPK and GenericValue just inherit from GenericEntity and have things more specific to representing a simple primary key versus a value object.

00:40:05 screen changes to Class GenericEntity

This still has things for clearing and cloning and adding to an XML containsKey. ContainsPrimaryKey compared to it has these different factory methods, as do the GenericPK and GenericValue ones. So all of the main get and set methods are here on the GenericEntity. Most of the time you'll use something like this, just the normal get method, and pass in a string with the name of the item you want.

This get method by the way is part of the implementation of the map interface, which takes an object coming in instead of a string. One thing to note about this one. When it's called, if you pass in an invalid field-name it will just return null and log an error message, because according to the map interface this is not allowed to throw an exception. Also on the get you can, instead of just caching the object that you get after you call the get method, there are methods like getBoolean, getBytes, getString, and so on.

On the set side there's a set method that's part of the GenericEntity. There's also a put method that again comes from the implementation of the map interface along with putAll. So typically when you're calling the set you'll just pass in a name and a value. There are certain things like setBytes, also setString which will try to convert the string to whatever type the field is, based on the field name here. And it uses the valueOf method, which is common to the various Java types like the number and date types and such.

Okay and those are the basic elements we'll be looking at in the GenericEntity.

00:43:08 screen changes to Class GenericPK

So when we get down to a GenericPK, which is one of the classes that implements this interface, there isn't actually a whole lot more in a PK that we do. We have a few create methods and a clone method. And these create methods are just the factory methods.

00:43:25 screen changes to Class GenericValue

GenericValue has a few more because this has all of the things like the create, store, and remove methods. It also has all the getRelated methods. These getRelated methods do all exist on the GenericDelegator class, but as with the create and store and other methods they're here for convenience. So you can work directly on a GenericValue object, and it will make the call over to the delegator.

So getRelated and getRelatedOne. And all these variations such as getRelatedCache, getRelatedByAnd to pass in extra constraints and so on, all basically come down to this same idea, where we specify the name of the relationship. And it gets a record from the related entity, so it takes advantage of the relation definitions and the entity definitions that we covered earlier, in order to make the code a little more concise and convenient.

00:44:46 screen changes to Class GenericValue

Okay you can see some of these operations that are the basic database interaction ones: remove, store, reset just empties it out, and refresh rereads the value from the database. Or you can refreshFromCache, which will read it from the cache if it's there, otherwise it refreshes from the database.

Okay and that is the basic stuff for the GenericValue object. And GenericEntity and GenericPK.

00:45:28 screen changes to Advanced Framework Training Outline

Okay we talked about sequenced IDs a bit in the simple-methods. Methods like the getNextSeqId, which does a top level sequenced ID. And this is really just a method on the GenericDelegator. Let me find that real quick.

00:45:52 screen changes to Class GenericDelegator

You'll recognize this basically from the simple-method instructions. All the simple-methods do is call into the delegator, call into this method, and pass in the sequence name (seqname). StaggerMax here is the same as the staggerMax attribute, so if you want it to be sequential the staggerMax would basically be 1, which it defaults to up here when you don't pass it in. Or you could do something like 20, where it randomizes between 1 and 20 to stagger the sequencing.

Long will just return a long, a number instead of a string. When you do the long it will not have a prefix; you can configure a prefix per delegator, and that will be pre-pended when you use these normal methods. But the long ones can be helpful sometimes, especially if you're using a numeric primary key. Which is not the convention for OFBiz by the way, so we most often use these justGetNextSeqId methods that return string.

Okay and so that's the sequencing itself is done by a little object the delegator points to that does all of the interactions with the database.

00:47:28 screen changes to entitymodel.xml

And there is actually a data model in the entity engine itself. It just has two pretty simple tables. The sequenceValueItem is the entity that's used to store the sequence name and the last taken sequence ID (seqid). So this is just incremented over time as new sequenced values are requested.

The only other thing in the entity engine entity model is this keyStore, which is where we store the keys for automatic encryption and decryption. Now it might seem kind of funny to put this in the database along with the encrypted data. One good idea is actually to take advantage of the delegation feature, put this in a different group in the entitygroup.xml file, and have it be in a different database than, say, your credit card table. And that ways someone has to have access to both databases in order to decrypt the information.

Now another option that's commonly brought up and asked about is putting the keys on the application server, but usually the application server is the first thing to be compromised. And database servers are often compromised through the application server because the application server is available to the outside world, whereas the database server is only available on the internal network.

Anyway this is the keystore. There are some random keynames with bogus keys in them, along with the real key that's primarily used. But the main thing I wanted to bring up here is this sequenceValueItem. So sequence name, sequence ID (seqName, seqId). If you import a bunch of your own information that has its own sequences, you'll want to go through and set sequence IDs above the highest numbers that you used in your own data, so that as the sequence generator carries on it does not run into any conflicts with your imported data and the sequences it's trying to generate.

Of course another way to do that is to prefix, either your imported data or your ongoing sequences.

00:49:44 screen changes to Advanced Framework Training Outline

Okay the last thing we wanted to talk about here in the Entity Engine API is the GenericHelper interface.

00:49:56 screen changes to entityengine.xml

Now when we were looking at the entity-engine.xml file, for each dataSource it specifies the helper class.

00:50:14 screen changes to GenericHelper.java

Now this helper implements an interface called GenericHelper, and this is basically what the GenericDelgator uses to pass all of the things you ask the delegator to do back to the database.

So we have things like create, store, remove, then we have the findByPrimaryKey. The cache by the way is a level above this in the delegator, so the helper doesn't do anything with the cache. These are just the actual low level database operations.

So it would be possible to implement this interface and do other things, like have a test in memory data store, which is actually one of the things that's been attempted, and how well that's supported is kind of anyone's guess. But it's certainly something that's interesting that you can test.

00:51:48 screen changes to Interface GenericHelper

So if we look at the GenericHelper interface there's the GenericHelperDAO, which is the one that we had specified in the entity-engine.xml file. And that's the default one that has this generic data access object, that just generates SQL and sends it over to the database. And then there's this in memoryHelper, which also implements the interface and kind of simulates an in-memory database. It's kind of basic and not real fancy, but it can be very good for testing when you just have some basic operations to look at.

For the most part, though, databases and their interactions are not so heavy that using an actual database along with a real GenericHelperDAO is a very good, reasonable way of doing things.

00:52:42 screen changes to Advanced Framework Training Outline

Okay there's your whirlwind tour of the Entity Engine API. So we've talked about some of the main objects that you typically interact with to do basic operations. And some more advanced things like finding with conditions in the EntityListIterator. And even more advanced in the DynamicViewEntity, which is where things can really get fun. Also the GenericEntity subclasses, sequenced keys, and some of the ideas behind the GenericHelper interface.

Okay in the next one we'll talk about the Entity Engine XML files, and importing and exporting those files.

Part 4 Section 1.9

Entity Engine: XML Import/Export Files

Starting screen: Advanced Framework Training Outline

Okay now that we're done with the Entity Engine API, moving on to talking about the entity XML import and export files. So first we'll look at file formats for basic and pre-transformed files, then the command line tools and webtools webapp import and export pages. We'll start here with the file formats.

00:00:38 screen changes to DemoProducts.xml

So let's look at a couple of example files. There is no XSD schema definition or DTD or anything like this for

these XML files. They just have one main thing that's required, the entity-engine-xml opening and closing tag for the file, and that's the root element to the file.

And then each element underneath it. The element name is just an entity-name, and each attribute is one of the fields on that entity, so you just specify the value of the field and the attribute value. Now one variation on this is for larger fields, or fields that have longer text data in them and such. You can also take any field and instead of putting in an attribute put it in a sub element, just like the LongDescription here.

Notice this is an entity-name so it has a capital first letter. These are all field names, so they have a lower case first letter, and this is also a field-name so it has a lower case first letter. The main reason you typically want to put it in a sub element is so that you can use the CDATA wrapper in XML. Which is very convenient because then you can put HTML tags and such right in the text, without having to escape all of the special characters like the <, >, $, and all of these sorts of things.

So they'll go in just as literal text because they're in a CDATA block. Which of course opens like this; <![cdata[and then ends with]] and a > sign to close off the tag. And then of course outside that you close the field tag, the LongDescription tag in this case.

So this is t he basic format you can used for XML files. You can use XSL or other tools to transform other data to these files, or transform these files to other data. Or you can use a variation on this, where instead of using entity engine XML as the wrapping tag you use entity-engine-transform-XML, and then specify the name of a template, or the location of a template in an FTL file. I believe this does also support XSL files for doing that sort of transformation, to transform this file into the entity engine form.

So you'll see what's under here is just an arbitrary XML file. It's not in the entity engine format, but it will be transformed by, in this case this FTL file right here. So that by the time the transformation is done it will be in the entity engine XML format and can be imported through the standard entity engine XML tools.

So what's under here is just an arbitrary XML file, and it can be formatted in any way. This format is actually a content format that's used by certain sites, or certain content repository type places, and was used as part of an implementation based on the content manager to migrate content from that site to the OFBiz-based managed content site.

00:04:51 screen changes to content.ftl

Okay so let's look at the FTL file here. Basically the output of this is going to be entity-engine-xml, and the elements underneath that are just the standard entity elements, entity with field attributes or field sub elements.

Here DataSource is an entity, dataSourceId is a field on that entity, and so on. This is actually somewhat similar to XSL in the way the FTL transformation stuff works, although it is just a template that's rendered, and it has some of these recurse type things that are similar to what XML has. It also supports the macro feature in FTL and many other features in FTL. You can even combine this with Java code and stuff, although typically when you're just transforming an XML file you don't have a Java environment or Java context to build from.

00:06:15 screen changes to
www.freemarker.org/docs/index.html

So for reference on how to use this style of FTL. If we look at the freemarker.org website, here in the documentation there is a section on these sorts of different styles of XML processing; imperative XML processing and declarative XML processing.

00:06:35 screen changes to
www.freemarker.org/docs/xgui.html

In general, here in the XML processing guide this is basically for taking an input XML document, and transforming it to an output template, or output document of any sort.

00:06:52 screen changes to content.ftl

In the case here that we're looking at in the entity engine XML files, like here we're transforming from XML (changes to DemoContent.xml then back) from this XML snippet basically everything under the entity-engine-transform-xml document. We're transforming all of that to another XML file in the standard entity engine XML format.

00:07:15 screen changes to
www.freemarker.org/docs/xgui_declarative_basics.html

So there's some good information in here about these things. You'll notice files using these declarative XML tags like recurse and macro for convenience. So anyway this is a good thing to read through to see how those work.

00:07:37 screen changes to content.ftl

But the basic idea of it is just kind of like XSL, where we recurse and process it based on the other things in the XML file, (changes to DemoContent.xml). So the contents element, the content element here, and all of the sub elements under content (changes to content.ftl).

We'll find corresponding things here like contents, content, and then the sub elements under content. The element, that's ignored so other elements are just ignored. The content here, basically the different things

under it. So we have node, we haven a ID field that's inserted here, and that's assigned to the contentId with a prefix, and then that's used in the variable expansion here. And we also have node.content_text.

So this is how we can refer to sub elements or attributes (changes to Democonetn.xml). As we can see we have things like the ID here, and the content_text here (changes to content.ftl). And so these various aspects of content element are just put into the different attributes and such in the data model for the content management thing, that's basically all this does.

So it's a fairly concise file, and not terribly complex. It's just basically mapping from an arbitrary XML format here (changes to democontent.xml) to the entity engine import format that we would normally expect (changes to content.ftl). So data source, data resource, electronic text and content in the entities that's its dealing with.

00:09:31 screen changes to DemoProduct.xml

Okay so before we move on to looking at how these are actually imported there's one other thing in wanted to mention. There is a feature in here to automatically generate Ids; in the entity there are two conditions that have to be met for this to activate. The entity has to have a single field primary key, so that would qualify for the facility, and then we can just leave off the primary key field for it, in this case it's a facilityId. So if there was no facilityId, or if we left the facilityId empty, then it would automatically generate a facilityId.

Now that can work well in some circumstances, but in many cases that's not very helpful because this facilityId is not only referred to here, but it's also referred to in a number of other entities that are related to that entity. So this is not an example of where that would be possible, but that is one of the features of the import, is that it will automatically sequence keys. And the sequence name it will use is just the convention in OFBiz; it'll use the entity-name for the sequence name and then automatically sequence an ID.

00:10:54 screen changes to Advanced Framework Training Outline

Okay let's hop back to our outline here. So we've looked at the file formats for the basic standard entity engine XML file, and the entity engine XML transform file. Let's look at the command line import.

00:11:30 screen changes to install-containers.xml

A good place to look at this, actually there's a pretty good comment in the code. Let's talk about how the command line style works.

When you run the ofbiz.jar file with the -install argument, then it will use the install-containers.xml file instead of the ofbiz-containers or test-containers. So test-containers would be used with the -test, ofbiz-containers is used for the normal startup. And we talked about how those are chosen earlier on in the Advanced Framework course in the container section, and the load a container unloading stuff.

So basically in here it has some of these standard things: loads a component, loads a classloader, and then we use the dataload-container, which basically is implemented by this class, entityDataLoadContainer. The delegator name we use for data loading is default, and the entity-group-name is org.ofbiz. So these are the two thing when you're loading data that you need to specify; the delegator and the entity-group-name.

00:12:35 screen changes to entityengine.xml

Looking at our entity-engine.xml file, let's go up to where the delegator is defined. So the default delegator will have a group name associated with it, in this case it's LocalDerby. When we look at the different data source definitions, each data source can have a number of different data readers associated with it.

So these ones all do the default set. And the point is so that with different data sources you can have different sets of files that are meant to be loaded in them. For example, this is actually a mistake here in this LocalDerbyODBC definition because we would not want to load the same seed data into those tables; they would have different sets of seed data of which there are none right now. There's no seed data that goes into them; that's an integration database so it's populated by other programs. But anyway so having these data readers associated with the data source allows us to have data more specific to different circumstances.

00:13:55 screen changes to install-containers.xml

When we're running the install routines, the data loading routines, these are the things we specify, the delegator-name and entity-group-name. Let's take a look at the EntityDataLoadContainer class.

00:14:11 screen changes to EntityDataLoadContainer.java

This will of course implement the container interface. In the init method is where it does all of the setting up of the parameters and such, and so that's the main thing we'll look at. The actual work is really done down here in the start method, which does all of the reading itself. So but the comments we'll want to look at are right here.

So here's an example of how to run the reader with an number of these different parameters, java-jar ofbiz.jar -install. So this -install argument after the ofbiz.jar is what tells it to run the install routine, the data load routine isntead of the normal startup. These are some of the parameters specific to it; you can specify a file, with a full or relative path, or if you're using the readers that

are configured through the entity engine you simply specify the reader names.

They can be separated, or you could have seed for example to just load the seed data, or these comma separated loads, all three of he standard data readers in ofbiz: the seed, demo, and external data. You can override the timeout. The default is 7200 which is in seconds. So this is the transaction timeout in seconds, and that's about two hours right here. You can specify the name of the delegator; it will default to the default delegator.

00:16:05 screen changes to install-containers.xml

This will actually override the delegator specified right here, if you specify one on the command line. So the default one is the one by default because of that.

00:16:15 screen changes to EntityDataLoadContainer.java

And then the same thing with the group; you can specify the group to use, or it will default to the one that's configured here in the install-containers file (flashes to install-containers.xml then back). So the command line will override that.

So you'll either typically specify readers, or you'll specify a directory, or the file. When you specify readers, you'll specify the delegator and group name optionally associated with those. If those are not specified then it will default to the ones inside the install-containers.xml file. And with any of them you can specify the timeout, which typically defaults to two hours since these are longer than the normal transaction, which has a default timeout of sixty seconds. Okay and that's the basics for loading data from the command line. Another way to load data once you already have OFBiz running is to use the pages in webtools.

00:17:32 screen changes to advanced Framework Trianing Outline

So we've finished the command line import, now moving on to the webtools webapp.

00:17:40 screen changes to Framework Web Tools, WebTools Main Page

So we've looked at some of the other tools in here, and we'll be looking at some more of them in other parts of the framework videos. But right here in this entity XML tools, in this section we have things to do in Export, Export All, and we'll talk about what each of these is, and Import and Import Directory.

So the Import does a single file, Export does a single file. Export All will export multiple files which can then be imported with this Import directory. Which will just import a full directory of files and optionally delete them as it goes, so it can keep track of which ones have succeeded and failed.

Okay so let's look at these. So first data exporting.

00:18:35 screen changes to XML Export from Data-Source(s)

You can either specify the Output Directory and the Maximum Records Per File, or a Single Filename. So you'll either use these two together, or the Single Filename. You can filter the date range for the records, so the Updated Since and Updated Before. So this is the from-date and thru-date basically for the updates.

By the way, these two you can specify at any time regardless of the output. Those are related to the input, but for output you have two options; either specifying an Output Directory and Maximum Records Per File, to split it into multiple files, or you can specify a Single Filename to put it all in. Or you can tell it to go out to the browser. We'll do that real quick so you can see the results.

Okay then there are various options here for choosing the entities that are actually exported. You can select entities individually here; there's a Check All and Un-Check All. These do a round trip to the server. When you Check All it will check only the non-view-entities, but you can also export data from view-entities if that's more convenient for what you want to import, like if you're going to transform it into something else or something.

So the view-entities like this one are in there, they're just not checked by default. That's one option, is to check each one that you want. Another option is to specify an Entity Sync ID here. In the next section we'll be talking about the Entity Sync tool. And basically when you have an Entity Sync Id, each entity synchronization setup has a set of entities associated with it, and so that is what it will use to output the entities here.

There are also these pre-configured sets, so this is a third option. Exporting the entity catalog, all of the catalog data, or the product data which is split into four parts. The reason for these four parts is so that you can import them in the same order, in Part 1, then Parts 2, 3, and 4. And that will satisfy all of the foreign key dependencies. When we talk about importing we'll talk about some other ways you can deal with foreign key issues on importing and such.

So those are the different ways you can specify how to select the data to return it. Now for each of the entities you select it will export all of the records in that entity unless you use this from date and thru date, or the Records Updated Since and the Records Updated Before, and then it will filter them by that.

Okay so if we look at a fairly common one like, let's go down to where we have some sample credit card data.

This is a good one to look at, because it will remind us of the sensitive nature of the security related to this. Fields that are encrypted by the entity engine will be automatically decrypted. If you're doing a one way encryption like with a password...let's go ahead and select userlogin as well so you can see the encrypted passwords.

But for the credit card number, which is encrypted in the database, when it's accessed through the entity engine it will be decrypted. But the userlogin has a password on it that's one way encrypted, so for that it would not work.

00:23:00 screen changes to view-source, source of webtools/control/xmldsrawdump

Okay so when we do this we'll end up with kind of an empty looking screen like this, unless there's an error. But it should be all right. But basically because it's an XML file, and it's trying to view it as HTML, it's not going to really work there. But if we do a view-source then we can see the actual XML that is exported here. So this is what I was talking about here with the card number; it does show the actual card number because it's two-way encrypted, and so it decrypts it coming out of the database. So access to these tools by the way for exporting is just like having access, even worse actually, than having access to all of the data in the database.

For passwords, since these are one-way encrypted on the application layer and not in the data-layer, you can see the hash value of the passwords here. You'll notice a lot of these are the same because that's OFBiz in lowercase letters, that's basically what the hash result is. I believe the default is using an SHA hash, but that's a detail related to the userLogin management.

Okay and that can be customized in the security.properties.

00:24:16 screen changes to Framework Quick Reference Book Page 16

Which by the way is included in here, I believe. So you can see here for example the has that's used,a dn the default is sha, but you can change it to other things. Okay that's a little side note.

00:24:36 screen changes to view-source source of: webtools/control/xmldsrawdump

So there's an example of doing an export with that, what the XML file will acutally look like.

00:24:53 screen changes to OFBiz: Web Tools: XML Data Export All

Okay going back to webtools, let's go to the main page and look at the export all page. So this one we just specify the output directory. It kind of has a short description of what it does up here, but basically it will export one file for each entity. And it will go through and export all of the data from the database, so it exports everything.

Now the way these are written, it basically goes through a record at a time. This does use the entityListIterator that we talked about before, and an output stream so it does not have to have everything in memory at once. So the only things in memory are the records that the JDBC driver fetches, the results that the JDBC driver fetches on each communication with the database, and a little bit of the buffer of the output stream. So this can handle very large tables and such, as long as you have a sufficient timeout of course. And this timeout by the way is for each entity, so each entity is done in the transaction.

When you're running this sort of an export it's best to not have anything else hitting the database or running against it, or you can end up with some inconsistent results.

00:26:19 screen changes to XML Export from Data-Source

If you need to have incremental change data going out you can use the other export page, with the from date and thru date on it. Or even better use the EntitySync tool that we'll talk about in just a bit, for moving data around between OFBiz servers.

Okay, let's look at the import pages

00:26:41 screen changes to OFBiz: Web Tools: XML Data Import

This first one is the name of a FreeMarker template to filter the data by or transform it by, along with a URL.

00:27:05 screen changes to DemoContent.xml

So when you're doing this, this is kind of a substitute with a separate field for the embedded one, the entity-engine-transform-xml here. So in this case we're specifying the template right in the field, but you can also do the same thing with any arbitrary XML file. It doesn't have to be wrapped in this tag, in the entity-engine-transform-xml tag, it can be any XML file.

00:27:28 screen changes to XML Data Import

And you just specify the name of that XML file here, and the name of the FTL file to transform it by here. So they can be independent, or as we saw in that example they can be combined together. And then you can treat it like a normal entity engine XML file and the import will take care of the transform for you.

Okay some of these options; Is URL. If you specify a filename here that is a URL you could access through HTTP or FTP or whatever, then you would just check

this box. If it's a local file location then you would just leave this unchecked.

Okay Mostly Inserts. This is an optimization parameter; it should not affect any functionality. If you know that this is mostly new data that's going into the database, then what it does is instead of following the normal pattern, where for each record it looks to see if it is in the database already and if it is not in the database it will insert it, if it is in the database it will update it in the database.

If you're going to have mostly inserts you check this box and what it will do is, instead of doing a read and then a write to the database, it will just write all of the records to the database. And then for any records that already exist it will get an error back on the database, and based on that error it will do an update. So this is much faster when you have mostly inserts; it will still work fine if you have a lot of updates in there. But if it's mostly updates, or has enough updates in it, then it will be slower to do it this way because each exception takes longer to handle then the checking to see if a record already exists in the database. So that's what that does.

This next one is maintain timestamps. If you check this the entity engine automatically maintains certain timestamps, the various ones that we talked about, and will talk about in more detail as part of the EntitySync.

But the create and last last-updated timestamps, and the associated transaction timestamps with those. If you check this box when you're importing a file, if you specify values for those fields on an entity then it will honor them, and it will insert those values as is is into the database. If you don't check this box then the entity engine will reset those values to the actual time that it was created or updated in the database.

Okay Create “Dummy” FKs. This is a workaround for foreign key problems that sometimes come up with data, and can become a big problem when you're trying to import data or move it around. This is something that the EntitySync tool can do as well, and it was actually written for the EntitySync tool, and then supported over here because it can be very convenient. Basically if you have dependencies between different records in the database, and the order of those records is incorrect, and you don't have initially deferred turned on for the foreign key definition in the database, then you will get errors because of foreign key violations.

This is especially problematic where you have circular references where one record references another, and then that other references back to the first one. And there is no insert order that would satisfy those. You can do a combination of the inserts and updates that will work fine, but if you want to just do an insert on those it will not work; there's no way you can order those two records to have it successfully go in. In the data model in OFBiz we try to avoid situations like that, but there are certain situations where those circular references are necessary for the functionality requirements.

And so for any of those circumstances this Create Dummy FKs will get around the problem. What it will do is, when it sees a field with a value going into a foreign key, or if it's inserting a record before it does the insert, it will look at all of the relationship definitions to see if there are foreign keys going to other tables.

And it will check each foreign key to make sure that it is okay. If it's not okay, if there's no record there, then it will know. So this is basically doing the same sort of check that the database will do on the foreign keys, and it will recognize the fact that the foreign key is not there. And it will create a dummy record, hence the name Dummy FK, that just has the primary key with the rest of the record empty. And then it will do the insert on the current record and move on.

Later on if there is data that goes into that other record, it will see that the record already exists in the database and just do an update, so it won't cause nay problems there. The only real danger with this is you can get those dummy records in there that don't have any other data, where you intended really to have other information in there but you forgot export it or put it in your seed or demo data files. And if you know all of the data is going to be there eventually then you use this Dummy FK option. If you don't know then it's better to see the foreign key errors, and deal with them as they should be dealt with.

Okay so that's the Dummy FKs option. Then we also have the transaction timeout as usual, and it defaults to two hours. You can specify other things there. In addition to being able to specify a file here with this form up here on top, you can also just paste in an XML document, the complete XML document with the entity engine XML root tag, so you can just paste it in right here. This is especially helpful if you have a few lines that you need to drop into the database from a seed or demo data file or whatever. And then you can just copy them, paste them in here, and add a couple of entity engine XML opening and closing tags.

00:34:21 screen changes to Framework Web Tools XML Import To DataSource

Okay let's look next at the import directory one. This one is very much like the other import page that imports a single file, except instead of specifying a file name or a file URL, you specify a directory that contains XML files that have the root tag of entityengine.xml, or they can also have the root tag of the entity engine XML transform.

So this has the same options here, mostly inserts, maintain timestamps, and Create Dummy FKs. Those

are exactly the same as the ones on the other import page, with an additional option Delete Files Afterwards. By default this is set to true, or it's checked.

What that will do is, if a file successfully imports, then we delegate it from the directory so that we know which files remain. This can be helpful when we have a whole bunch of files, where each file represents a different entity, and there may be various dependencies between them.

So basically we can just run the import on the directory over and over, and the same directory just over and over run the import. And then on each pass it will delete the files that are successful, leaving only the unsuccessful files remaining. So if you get into some sort of a circular dependency or something, then even after as many passes as you care to try to do there may still be some files left over. So what you'd do in those cases is look at the specific error messages to see what is going on with the file, and if it is a circular dependency problem and not a missing data problem. If you can verify that then just check the Dummy FKs and it'll go in.

Okay transaction timeout same thing; pause between files. With this you can specify the number of seconds to pause between files, to give it a little breather I guess. Okay and then you click on the Import File button and off you go.

00:36:35 screen changes to Framework Web Tools, WebTools Main Page

So those are the screens for exporting and importing data through the webools web applicaiton. So with these you can do it with an instance of OFBiz already running. And generally depending on how it's configured you should also be able to use the command line tools when another instance of OFBiz is running.

00:36:55 screen changes to Advanced Framework Training Outline

Okay and that is it for the entity engine XML file import and export.

Related to this, just something that hit me, there is another service that does importing from a directory. There is no user interface for this since it's not as commonly used, but it will import flat files instead of importing these XML files. The name of the file should be the entity-name, and within the file it should have rows with comma separated fields. The first row should have the field-names that are comma separated, so it knows which position on each row is that way.

00:37:56 screen changes to services.xml

If we look at this services.xml file in the entityExt component, this is where the service definition is. So the importEntityFileDireoctyr will import an entire directory of such files, and importDelimitedFile is called repeatedly by this other one, by the directory one, for importing each individual file.

Okay you can pass in a delimiter to these to specify what the field delimiter is, typically a comma but you can have a tab separated or whatever, and it will pass out the number of records that were successfully imported. So these are some services that you can use if this is closer to what you need.

Another option for flat files coming in is to use the DataFile tool to transform them into the entity engine XML file format, or write special code that uses the DataFile tool for parsing, but then just uses the entity engine API for inserting those records in there.

00:39:10 screen changes to Advanced Framework Training Outline

We'll talk about the data file tool further down in this outline, and that's actually the last section we'll cover.

Okay so now we're done with the entity XML files. Let's talk next about the EntitySync, or Entity Synchronization.

Part 4 Section 1.10

Entity Engine: EntitySync

Starting screen Advanced Framework Training Outline

Okay we've finished the entity XML file import and export section. Now we're moving on to the entity sync or entity synchronization section. The general idea, or theory of operation of the entity sync is that it does a time based synchronization from one server to another. So these are independent instances of OFBiz running. Each has their own database, and what we want to do is send data from one to the other. And it's possible to set up multiple synchronizations so we can send information in two directions.

It also supports things such as pushing data to the other server and pulling data from the other server, so that only one server needs to know the address of the others. So you don't have to have two way addressing over the internet, which is helpful for remote sites like for example a store that is synchronizing with a central server, that lives in a data center somewhere.

The central sever does not need to have an IP address or contact the store server, the store server just pushes data to the central server and pulls from it. Of course a point of sale scenario like that we can do multi-tiered, so there's a per store server that synchronizes with a dozen different cash registers, all running with independent databases. So if network or power on other machines or anything goes down they can continue processing sales and such.

And then the per store server can synchronize with the central server, like a company wide server that will keep track of all sales and inventory and all of these sorts of things. So the point of sale application in OF-Biz actually supports all of that kind of stuff.

Okay one of the main things about this is that it is a time based synchronization, and not a data comparison based synchronization. This is a fair bit different from say a PDA synchronization where you're synchronizing with a desktop computer or a cell phone or whatever, and it compares the various entries; address book, calendar, to-do list. So it compares all of these sorts of entries to see which devices have seen updates since the last synchronization, to see what needs to be done to get the data onto the different devices to get the same set of date, adding and removing and merging records, and that sort of thing.

So that kind of synchronization is fairly different from what we're doing here. This is a time based synchronization, so more of a synchronous synchronization, as sync does refer to time. And this is much more suitable for very large date sets, so we can have updates on hundreds of products being synchronized out on potentially many hundreds of point of sale terminals. Or for integrating purposes we could have, for example, data synchronized from a production e-Commerce server down to an integration database, that is then used for sending data back to another system and these sorts of things.

Because it's time based and not data comparison based, all we're doing is keeping track of when different records are created and changed, and then we're sending sets of records over from one system to another.

It is suitable for large data sets, although best for ongoing incremental changes as opposed to large infrequent changes. And what that means is that, even though we can handle very large data sets, if we have a whole bunch of changes that happen in a very short time then the way the synchronization code works is it's going to try to send all that data over in a single pass.

And the servers may run out of memory, because it has to load all the data into memory for a certain time cyhncjk, and then do various sorting to merge records from multiple entities into a single stream of records based on their timestamps. And then that gets sent over to the other server. So both servers need to be able to have the entire set of records in memory at once.

If you have a very large import that's done then this doesn't always work so well. There is something for off buying synchronization where we export everything to a file instead of running it over the network in chunks, and we'll see the services and stuff for that.

So that can work better for initializing setups and such. Or another alternative is to use the entity XML export and import that we talked about before, and use that to move data over for initial setup of a system.

And then from there, as long as changes are incremental even if there is a large backlog of incremental changes that need to be sent over, as long as there are not any single extremely large transactions that need to be pushed over the wire then there shouldn't be any problems

Okay let's look at the entitysync entities and related entities, which are used to configure the entity synchronization. We'll also look at some sample settings and some demo data, and a demo scheduled job that's driven from seed data or demo data. And we'll look at the status page in webtools for this.

00:06:12 screen changes to org.ofbiz.entity.group

Okay for the data model this is the entity reference chart, or the entity reference pages that are in webtools. We'll be using these to get an idea of what the entities look like, and we'll go over these in more detail, what everything is, here in a bit in the next section.

00:06:32 screen changes to Advanced Framework Training Outline

Where we're covering other of the entity engine pages in webtools.

00:06:40 screen changes to org.ofbiz.entity.group

Here we have the org.ofbiz.entity.group package of entities. And this is used optionally by the entity sync entities for determining a group of entities that will be involved in a synchronization. This can also be used for other purposes though.

So unlike a package that each entity fits in one and only one of the groups are a data structure, rather than a configuration file we can use to group an arbitrary set of entities. Each group will have an ID and a name. In the entity reference pages we do see the timestamps that the entity engine automatically maintains for all entities, unless you use the attribute we talked about in the entity definitions to turn it off. But by default all entities will have these four fields; lastUpdatedStamp, lastUpdatedTxStamp (transaction), createdStamp, and createdTxStamp..

The transaction stamps are when the transaction was begun that the change is part of, and the stamps are within that transaction, although they are an absolute value. But to order the actual records that changed within the transaction.

Okay and the entity engine does do things for these stamp fields when it maintains them, by the way, that

ensure that within a transaction each updated or created stamp, depending on whether it's a create or updated, is unique for that transaction.

So it'll keep track of the last one and always increment it by one, even if it's been less than a millisecond since the other record went through. As long as you don't have more than a thousand records going into the database per second in a single transaction then you shouldn't have problems with overflowing it too much.

But it is important that those are unique, so we can order all the records by these stamps and basically reproduce the transaction to send it over to the other server. Even reproducing the transaction we can't always eliminate problems with foreign keys, but we'll talk about that in a bit. Anyway that's the intention here, to reduce problems with foreign keys as much as possible and recreate the data changes on the remote server when the synchronization goes through.

Okay so each gourp will have an entity or package field. So this can be either the name of an entity or a ful or partial package. So it could be something like entitygroup, or something like org.ofbiz.entity.group. Or it could even be just org.ofbiz.entity to include all the sub packages under entity, which we can see over here are the crypto, group, sequence, synchronization, and test packages.

So you can take advantage of the package organizations to define your groups, or you can specify the entity names individually. And we'll see there's also a similar pattern used when associating entities directly with the synchronization, rather than through a group. But it's nice to use a group because these are reusable for multiple synchronization definitions, as well as for other purposes.

Okay so here's our sequence value item entity that we looked at before.

Let's look at the org.ofbiz.entity.synchronization package. The main entity here is entitySync. Any time we run an entitySync one of the main things we pass in is the entitySyncId, so the entitySync services are all based on this. The status of the synchronization is kept here in the entitySync. There's a runStatusId.

00:10:58 screen changes to EntityExtTypeData.xml

The different statuses, let's go ahead and look at those real quick. The entityExtTypeData has the different status items for an entity sync status, and these are the entitySync run statuses. We have nuot started; running, offline pending, complete, and then different error statuses. These others we'll see in just a minute, the include application type, we'll see how those are applied to the entitysync.

So these are the different statuses that happen in the life cycle of an entitySync running. If there is an error it will flag the sync as an error and save in the entitySync history table, which we'll look at in a bit, The details of the error and then that can be corrected. And then the status can be changed so that the next time the entitySync service is scheduled to run for that entitySync it'll pick up and continue from there.

00:12:08 screen changes to org.ofbiz.entity.synchronization

Okay let's look back at the data model. This in an important field, the lastSuccessfulSynchTime. And what it means is that for the date and time stored here, everything up to and including that date has already been pushed over to the other sever.

Now if you're synchronizing two multiple servers, what you'll want to do is create a different entitySync record for each server. They can have the same source of information, but they each maintain their own lastSuccessfulSynchTime so that the data can be sent over to the different severs independently.

So what happens in the source database is happening all the time and the timestamps are saved. And then when a synchronization needs to be done to a different server it just keeps the status of the lastSuccessfulSynchTime, and that way it knows which records need to be updated between that and the current time.

And it actually doesn't go up to the current time. This is configurable, but it will typically run just up to five minutes before the current time. The reason we don't want to run the data within the last five minutes is because you have these timestamps are set by the application servers. And so as long as all of the application servers hitting a database are within five minutes of each other, which is typically plenty of time for regular server time synchronizations that are typically set up, then you won't have any problems with lost records, where one server is running syncronixation and a different server has put in records that are too far in the past or something to be picked up by that synchronization.

Okay lastHistoryStartDate is just to keep track of the history records. The details of the history of what happens in each synchronization pass are stored down here in the entitySyncHistory table. So there's a lot of information that's kept here that's automatically maintained.

The entitySyncId and the startDate make up the primary key. The runStatusId of that particular run. It'll eventually be complete, or error when it's done. If it is an error then the next one that starts up, this one will stay independent, the next one that starts up will have a new entitySyncHistory record.

The lastSuccessfulSyncTime, like up until what time did we synchronize. LastCandidateEndTime, lastSplitStartTime, these various fields are all just describing

what was done in the synchronization. So the splits are basically within a synchronization, we split it into different groups. This is also configurable in the entitySync record, so the syncSplit you can configure a different one for offline synchronization splits (offlineSyncSplit) and for a buffer on the end.

This syncEndBuffer is that five minutes by default that I was talking about. Actually this may be an optimization thing; I'll need to review that real quick. We'll see it as we're looking through the process.

Some of these things are for optimization purposes. So it will see periods of inactivity for example, and skip over them and try and look into the future. So these are just optimizing things that are not necessary, they'll change the speed of running but won't necessarily change the functionality or the results.

The syncSplitMillis is about the same; it can change the results for running because if this is too big then it will try to get too large of a chunk or data. Like for very busy databases you might have this down to four or five seconds of wa window that it looks at for each split that it sends over. So within each split it basically needs to be able to fit all of the data, all of the changes from that split from the entire database into memory.

And then those are sorted and sent over to the receiving server. So if the database is not very busy this can go up to thirty seconds or even a minute without running into a problems, and it will perform better if you do that. But for very busy databases you're basically getting down to a few seconds or even sub-seconds that you want to consider for a given split.

Okay this is keeping track of when things are sent over. We keep track of whether it was supposed to be created, stored, or removed. So when we for the list of items that need to be created, and we'll keep track of how many were inserted, how many were updated, and now many were not updated. Now the reason that it would not update is if on the target server it was updated more recently then on the source server, and so to avoid overriding changes that are more recent it will check that and not update if it is more recent.

Okay toStore, a similar sort of thing. These are meant to be stored, but if it wasn't in the target database then we keep track of the fact that it was inserted; toStoreUpdated is what was expected to happen.

ToStoreNotUpdated. Again if there is a record on the target server that has a newer timestamp then the one that we're sending over then it will not be updated. So you can watch for exceptions and stuff here.

Okay toRemoveDeleted and toRemoveAlreadyDeleted. We have totalRows, some statistics on those, and the total number of splits. So if you had two hours of data to send over and they were split into five second splits then you'd have quite an number of splits being sent over. And finally totalStoreCalls.

Okay it also keeps track of the runningTime, the perSplitMinimum and maximum times, the perSplitMinimum items and maximum items. So you see, you can use this historical data to see if any of these parameters might need to be tuned for ongoing synchronization in that specific circumstance. There's nothing in there right now to do automatic tuning or that sort of thing, although I suppose such a thing could be created that would help optimize some of the parameters on the entity sync record here. Based on these, here, based on the history information of what actually happened in different synchronizations.

Okay some other things that are important here: the targetServiceName, and targetDelegatorName, which is optional. You can tell it which delegator on the remote server to use to store the data. But the targetServiceName is basically the service to call, to send the information over. This is usually a remote service, and when we look at the example we'll see how that's set up. And so the remote service that's called locally that will result in a remote service call to the other server, is what we're looking at.

So we're taking advantage of the remoting feature in the service engine for this. You'll typically set up a single service per target server that you're going to be sending information to, and then the service engine takes care of the remote communication. Either via HTTP if necessary, or more efficiently via RMI. Those are the typical remoting protocols to use anyway; others are certainly available, but those are the main ones. RMI is good because it encrypts and compresses the data, and sends it over in a binary format so it can be as efficient as possible.

Okay keepRemoveInfoHours tells the synchronization services how long to keep/remove data. What it will do is there's a default of twenty-four hours, but you can configure it to keep/remove information longer. There's a service that runs periodically, and we'll see the definition setup of this service as we're looking at the others, that will automatically remove data over time, will remove this data.

So what happens is, every time a record is removed the primary key is saved off in this removeInformation entity, and that's used to send information over to the target server to tell it which records need to be removed. And then that information is cleared out periodically, by default once every 24 hours. If you think there are longer periods of time where you might want to keep that information, then all you have to do is define an entity sync and set this value, the keepRemoveInfoHours higher, and that will kick up the time. So it looks through entity sync records to find the highest value for this, and that's what it uses to keep the information.

Okay forPullOnly and forPushOnly. As I mentioned before there are different services that will do pushing or pulling data, and so you can configure these to run just in those modes, to kind of use as a sanity check so that if a different mode is used then it will complain.

Okay the entitySyncHistory table that we talked about. The entitySyncInclude we talked about, and the entitySyncIncludeGroup. So these two entities, entitySyncInclude and entitySyncIncludeGroup, are the ones we use to specify which entities will be included in this synchronization. So we looked at the group definition, basically identify it by entityGroupId, so we can just associate those. We end up with a list for a given entitySyncId of the different groups that are associated.

When we are associating individual entities we specify the entitySyncId, entityOrPackage, which is the same as on the group member entity up there that we looked at, and then the application enum ID (applenumid). And I think the other one, this group entry, also has this applEnumId. I forgot to mention it though when we were talking about that.

So here's the entityGroupEntry, the same pattern we talked about: entityOrPackage, and applEnumId. Down here on the entitySyncInclude we have that pattern entityOrPackage, and the applEnumId. So again the entityOrPackage can be an entity-name; it should be either a full entity-name, or a full or partial entity package.

00:24:30 screen changes to EntityExtTypeData.xml

The applEnumId has three options as we saw over here: include, exclude, and always include. These are especially helpful for dealing with packages and such.

We can include an entire package, except maybe one sub-package or certain entities. But if we want to include an entire package, like org.ofbiz.entity, and then exclude one of the packages underneath that, like .synchronization, but always include a certain entity that's in that package regardless of the exclude, like entitySyncHistory, then we could use three different records to accomplish that.

00:25:19 screen changes to Entity Reference Chart

So here's that example. We can see this a little more visually if we wanted to include org.ofbiz.entuity. If we just did that as an include it would then include all of these, but if we wanted to then exclude one of the packages we could exclude the synchronization package. If we wanted to exclude everything in the synchronization package except this entity, the entitySyncInclude, then we'd have one record that does the exclude on this package and then one record that does an alwaysInclude on this entitySyncInclude here. So that's the order of operations or priorities for those different application types.

Okay here's the entitySyncRemove entity that we talked about. This keeps track of the primary key, in text form, of all removed entities. This is another thing, in addition to these lastUpdated and createdStamps, that the entity engine keeps track of when removes are done. So we have an entitySyncRemoveId that's sequenced, and then the primaryKeyRemoved, and then of course we have these stamps that we use to determine when we need to remove these. So these will typically stick around for 24 hours and then be removed automatically by a scheduled service that we'll see in just a bit.

Okay so that's it for the data model for the entity synchronization. Let's look at what it actually looks like when we set up an EntitySync record, in this entityscheduledservices.xml file.

00:27:00 screen changes to EntityScheduledServices.xml

Here's an example EntitySync record; we just have an arbitrary ID sitting in here that all of the entitySyncIncludes refer to. The initial status is not stated, so it knows that it's ready to go. NotStarted or completed are the ones that it looks at when determining if a synchronization is ready to run, so we'll talk about the importance of that in just a bit.

Okay syncSplitMillis. So this actually has a very large split time of six hundred thousand milliseconds, 600 seconds or ten minutes. Here's the service it will call, remoteStorageEntitySyncDataRmi, so it's doing a remote call over RMI. This is just a generic example one. Again if you're synchronizing to multiple servers then each EntitySync record will refer to a different service, and each of those services will talk to a different server. So the server that it talks to is configured in the target service here, keep information (keepRemoveInfoHours) forPushOnly, in other words it won't allow that client that we're sending information to to pull information from this server.

Okay in the entitySyncInclude, this is going to exclude the ServerHit entity, and it's going to exclude the ServerHitBin entity. It's going to exclude all of the .service and .entity records, but it's going to include all of the org.ofbiz.* (whatever) packages. So any package name that starts with this will be included. So this is basically all of the entities except the entity specific entities, service specific entities, and the ServerHitBin and ServerHit entities.

So that's what those mean, and we'll look at some more examples of this in just a minute. Okay so here's a sample EntitySync setup, but as it is it doesn't do anything until we specify a JobSandBox record that will do something special. Here's how we can set up in imported XML data something that will run every two minutes with no limit on the count, so it's -1, and no untilDateTime, so it will run infinitely every two minutes.

So that's the recurrenceRule, the recurrenceInfo, which refers to the rule, and then the JobSandBox refers to the recurrenceInfo record over here. See, the recurrenceInfoId points to the recurrenceInfo record, the recurrenceInfo record points to the recurrenceRule, and that's how those data structures work. So the recurrenceInfo is basically saying start at this time, and the recurrenceCount that the number of times to do it is limited to zero, so in other words is unlimited.

Okay the RunTimeData here that is associated with the JobSandBox, so the JobSandBox record will also refer to a RunTimeData record. And RunTimeData basically just keeps track of the context that will be passed to the service. So when we do a scheduled service, or a persisted asynchronous service, what it's going to do is serialize the context that's passed into the service in an XML snippet, and that will be saved in this RunTimedata Entity.

So here we can see the format of that. It has a primary key, runTimeDataId, and then we have the runTimeInfo field that will be populated with this, everything starting at the CDATA down to here.

This ofbiz-ser is what the XML serializer class does; it creates these as well as parsing them. So all we're doing in this case is we have a map, a HashMap, the map structure that it uses has entries, so basically we have a list of entries underneath the map-HashMap element, and underneath the entry element we have a key element and a value element. So the key, in this case there are different data types that can be represented with XML elements here, so we have a standard string (std-String) of entitySyncId and a standard string of the value 1500, so that's going to be pointing to this EntitySync that we set up right up here.

Now the different values that are used, the different elements in here and stuff, you can look at the XML serializer class if you want more information about those.

00:32:27 screen changes to XmlSerializer.java

But typically those are all automatically generated, and automatically read and parsed back into the data structures. So we have these serialization returns, and it's checking to see the types of the objects coming in. If it's just a string we get a standard string, if it's something like an arrayList or a linkedList, whatever, we get these, we create elements for those so we know the data type to pull back out. And then we serialize each entry in that list, which calls back into this method to create the structure in an XML file or an XML snippet.

00:33:24 screen changes to EntityScheduledServices.xml

So the intent of this is to be human readable, just in case you did need to hop into the database with a tool and adjust whatever has been serialized there. And also so you can read it, and of course it's every machine readable so that it can be used by the service engine job manager. So there's some details about that, and we'll be talking about more of the job stuff in the service engine section.

Here's an example in this XML file of how to not only define an EntitySync, but how to define in seed data the setup for this. When we look at the scheduled services we'll see that there's also a webtools page, where you can schedule a service through the application. And it will put together all of this information for you and drop it in the database.

But if you want to do it through seed data like this, which is often a good idea when you're setting up a number of services to synchronize with and such so that it's more controlled, this is an example of how you would set all that up.

Okay there's another job up here's that's run. I mentioned the fact that the remove information is kept for a limited time period. This is the job that's run every day, once a day, and it calls the cleanSyncRemoveInfo service. Which goes through and checks to see how long in the past it should keep the remove information, like 24 hours for example, and everything older than that is removed.

Okay, now that we've seen how to define the EntitySync and such, let's look at how these services are actually defined. Let's look at the remoteStoreEntitySyncDataRmi. We'll trace that through the call, so when this service is scheduled to be called the runEntitySync service, this is the one we're actually scheduling, and we're telling it the entitySyncId that we want to run is 1500.

So when it runs that, it's going to find this EntitySync record, and for each of the splits it will call this service to send that information over to the other server. And then when the information is successfully sent it will update the lastSuccessfulSyncTime and then move onto the next split, until it gets up to five minutes before the current time. And then it will quit and wait for the next time to run.

Now when this runEntitySync automatically runs, it may be that the previous one, because we're running it, we have it set up to run every two minutes, and so we can see that in the recurrence rule here it runs every two minutes. And so if it takes longer than two minutes for an EntitySync to run for all of the data that's in the database, basically when this runs again it will see, based on the runStatusId, that there's already an EntitySync running against that. And so it will just quit immediately and wait. So another runEntitySync will not start for a given entitySyncId until the previous one has finished and the status is set to completed, or an error status.

If it's an error status it will not start again until you go and clear that error status, look at the details of it, fix the problem, and then clear the errorStatus, set it to not-started or to completed. And then the next runEntitySync will pick it up and go with it frorm there.

Okay so let's look at the definitions of these services. We saw the runEntitySync service here, and that will be called. And basically we're just passing in the entitySyncId, and then for each split of data, for each time section of data, it will call this service to send the data over to the remote server.

00:37:48 screen changes to services.xml

Let's look at that one first actually. The example remote services are defined down here. So we have two examples of pushing, and two examples of pulling services. Each has one HTTP example and one RMI example, so there's an HTTP and an RMI example.

Here's the one that we're looking at, remoteStoreEntitySyncDataRmi, which is going to push data over to another server using RMI as the remoting mechanism. So regardless of the rest of the service definition, the service that it's going to call in the remote server is storEntitySyncData, so that's the service that the push ones will call. The pull ones will call this pullAndReportEntitySyncData.

So that's the name of the service on the remote server that will be called when this service is called on the local server. We can see here for example the storeEntitySyncData and pullAndReportEntitySyncData.

Now in the schedule, the service that we schedule to run for each EntitySync, if we're going to do a pull we schedule runEntitySync and just pass in the entitytSyncId. And this will go through, look at the current data in the database, split it up into splits, and for each split call the local service like remoteStoreEntitySyncDataRmi. Which will end up in the remote service being called on the target server, storeEntitySyncData as defined here.

For the pulling it's the same sort of thing, except for instead of scheduling runEntitySync you schedule runPullEntitySync. And then that will result in whatever service you want being called, for example remotePullAndReportEntitySyncDataRmi here, which we'll call locally through the service engine. Which will result in a call on the remote server of the pullAndReportEntitySyncData, which is defined here. And that's the service that will end up being called in the remote server.

Now to actually get the data, bring it back to the current server and then this server will persist all of that data in the local database. When we're doing a pull we keep track of the status on an EntitySync record on the local server, but we're pulling data from the remote server. We just tell the other server what our last successful synchronization time was, and then it returns the records accordingly.

Okay one thing to note about these remote service definitions. Typically what will happen is in a services.xml file, the same services.xml file will often be deployed to both machines, although they can have different ones. But it's usually easier to just deploy the same code base to all of the machines. They'll just have different things in their databases as far as which services to schedule and call automatically, and also which EntitySync records are in each of the different databases.

But anyway, when we're defining one of these remote services that runs over HTTP or RMI it's very common that we'll use the implements and say which service is going to be called on the remote server, so that this service has the same set of input and output attributes. So for storeEntitySyncData, instead of redefining all of these different attributes here we just say that it implements that service, and then all of those attributes will be inherited basically. And that's another thing that we cover in more detail in the service engine section of this advanced training.

Okay so that's an introduction to the basic services. There are a few other services that take care of some housecleaning and other things. Here's a cleanSyncRemoveInfo; this is where the service is defined that cleans up the old remove information records.

ResetEntitySyncStatusToNotStarted. This is what you'd use in the case of an error. This is something that you can call through the syncStatus user interface in webtools that we'll see in a minute. So this will just reset the status to notStarted once and error has been corrected, so that the next time the runEntitySync or runPullEntitySync service is called it will pick up that EntitySync record and go with it as expected.

Okay now here is some for the offlineEntitySync. Instead of having a scheduled service to run an EntitySync this is another alternative; you call runOfflineEntitySync, specify the entitySyncId and the filename. It will do everything that the normal runEntitySync does, getting data ready to push over to the other server, putting it all together, and ordering it properly and all of these things. But instead of sending it over to another server it will save all of that into an XML file.

If you don't specify this name, by the way, as this comment shows it defaults to offline_entitysync-, then the entitySyncId, then another -, then the current datetime in that format, and then .xml. So you can schedule this to run if you want, the runOfflineEntitySync, and over time just save files to a directory. And then those can be copied over via FTP or a floppy disk or whatever you want, to the other server. And then on the other server each of those files is loaded using this loadOfflineEnti-

tySyncData, and you're just passing the filename to load.

You'll note one thing that is missing here. When this offlineEntitySync runs, once the file is successfully written it will assume that the EntitySync is successful. So if anything happens when loading it, that needs to be corrected and then loaded, in order to insure that all data gets moved over successfully. So once it's in the file it's considered successful, and next time the runOfflineEntitySync runs for that entitySyncId it will take the next timeset past the previously successful one and put it in the next file.

So that's the basic pattern for doing onlineEntitySyncs. Okay and that's it for the entitySyncServices.

00:46:01 screen changes to Advanced Framework Training Outline

So the last thing we have to look at now is the EntitySync status page in webtools.

00:46:18 screen changes to OFBiz.org Framework Web Tools, WebTools Main Page

So let's hop over here in our browser and make sure I'm still logged in; I may have timed out. Okay so the EntitySync status is right here, under the Entity Engine Tools section.

00:46:28 screen changes to Framework Web Tools, Entity Sync Status

And basically what it shows us is all of the entity syncs. For each ID it shows us the EntitySync, its Status, the Last Successful Synch Time field that we started that shows up to what time data has been moved over.

You can see that none of these are active, and so none of them have a value there. By the way if there is no Last Successful Synch Time, instead of starting from that time when looking in the database it will go through all of the entities that are in the group for the synchronization and see what the oldest time is, and use that for its starting time.

Okay all of the other information here is not yet populated because these haven't run yet. But it will show the target service name as applicable, the target delegator names you can see are typically empty, and different things. There's also a link here to run the offline sync. If any of these is in an error state there will also show up a little button over here to reset the status, I think in this column, to reset the status back to not-started so that it can try again the next time the scheduled service runs.

But there's also a link here to run the offline sync, which will result in the file being saved out that represents the next set of splits for the synchronization, and then if you have a file you can specify the file name here and load it. So basically these are just little forms in the user interface, that call into the services that we looked at for running an offline entity sync and loading the data from an offline entity sync.

So it's a fairly basic user interface. Right now it doesn't have anything to show the history; you can look at that directly in the database and all of that sort of thing. But this is a basic thing to manage, the entity syncs, so you don't have to manually do low level service calls and stuff so much.

00:49:04 screen changes to PosSyncSettings.xml

Okay you'll notice in the database we have a number of other EntitySyncs that are set up in the demo data for OFBiz; these are the point of sale ones. PosSyncSettings.xml is the name of the file that defines a number of entity groups, and then corresponding entitySyncs that are associated with those groups. Then here we have commented out the different scheduled services to run, push and pull.

These are examples of doing two different phases of synchronization, so you'll see one for each of these. This is a point of sale example meant for pulling data from the Main Central Server (MCS) to the Per Store Server (PSS). This is one for pushing data from the Per Store Server to the Main Central Server. So this is the case where we know the IP address of the Main Central Server, but the Main Central Server may not know the IP address of the Per Store Server. So we're pulling data from the MCS to the Per Store Server, and the Per Store Server is pulling data from the Main Central Server and pushing data to the Main Central Server.

The data that it pulls, you can see how these are grouped over here; Main Central Server to Per Store Server. We have things like the entity group definitions, just to make sure you can actually change the sync settings on the point of sale server, or how the groups are defined, by just changing them on the Main Central Server, that's why that is there. But the main information that's pushed down from the central server is product information, customer information, an the POS terminal settings.

The stuff that goes from the Per Store Server, let's look at the other example, the Per Store Server to the Main Central Server, is things like customer information and orders, along with their corresponding invoices and payments. Also shipments, inventory changes, and point of sale terminal log and status. So the Per Store Server in this setup will have all customer information, all inventory information, and so on for that store.

The terminal, on the other hand, when we look at the terminal settings we're sending much more limited information. We send over party records but not any contact information. It doesn't keep track of any inven-

tory, on the POS terminal itself it doesn't keep track of order history except for its own orders, and these sorts of things.

So the POS terminal itself has a smaller database. The inventory tracking is not done there, it's done on the Per Store Server, which is coordinated with the Main Central Server. So basically this is an example of setup of the Point of Sale, an application of the EntitySync stuff for pushing and pulling data between three different tiers: the Main Central Server, a single one of those; a Per Store Server, so one server for each store, you could have dozens or whatever number of stores are involved; and then inside the store we have a number of POS terminals that are synchronized with the Per Store Server.

So when you make a product change you would send it from the Main Central Server to the Per Store Server, and then from the Per Store Server to the POS terminals. That way the Main Central Server does not have to send details down to the POS terminals individually, saving espically on the wide area network traffic and such.

And similarly with information that's trickling back to the Main Central Server; it doesn't have to communicate with so many terminals because it's not talking to each point of sale terminal, it just has to talk to the Per Store Servers to get the information back. So that can be sent in larger chunks, just a more consolidated setup.

But you could set this up to skip the Per Store Server and go directly from the point of sale to the Main Central Server. For smaller stores, or smaller groups of stores, that would typically be more appropriate. But as you can tell with how a lot of this is set up, it has to be set up fairly manually with a good awareness of the EntitySync tool and other things, including the customization of the POS screens and such.

Usually the point of sale application as it is right now is not very easy for smaller stores to use, so it's mostly used by larger stores. And this is the default setup that we have in place in OFBiz, or the example setup, and a good example of how to use the entity synchronization to move data around in a real environment like this.

00:54:25 screen changes to Advanced Framework Training Outline

Okay that is it now for the entity synchronization. In the next video we'll look at the entity engine pages in webtools.

Part 4 Section 1.11

Entity Engine: WebTools

Starting Screen Advanced Framework Training Outline

Okay in the last section here in the entity engine we're talking about the entity engine pages and webtools, and we just finished the one about the entity sync.

00:00:22 screen changes to Framework Web Tools

Let's just hop over to the webtools web application and look at some of these. We already looked at the entity XML tools, and let's look at the entity engine tools now. We've already looked at one, the EntitySync status right here in the last section on the entity sync.

00:00:43 screen changes to Entity Data Maintenance

Okay entity data maintenance. The first screen here shows a list of all entities and for the view entities we just have all-links. For the rest of them we have create, find, and all. The difference between find and all is find just shows a search form without doing any sort of a search. All will immediately do a search on the entity for all elements, in other words with no constraints, and then show the first page of results.

So you can use this once you've found an element through either of these. For example let's look at one of the type records like accounting transaction type here (acctgTransType).

00:01:34 screen changes to Find AcctgTransTypes

If we click on all we get all of the elements. This will show a table with a summary of the records in it, just showing the value of each field, and we can view or delete a record from here.

00:01:52 screen changes to View Entity: AcctgTransType with PK

When we view it, it shows some interesting things. First of all there's just a view tab that's open. This does use Javascript to switch back and forth between these, and the formatting isn't quite perfect which is why it runs down the foot of the page, but it's very handy.

So for here for example the acctgTransType we can edit all of its editable fields, which means all of them except the primary keys. And the editing here is fairly simple; we do have data popups for the data fields and certain things like that, but for the most part you just have a free form text entry box.

You can also view related entries. If it's type one and it finds one it will actually show the details here, or it will just tell you it wasn't found. If it's type many like this one then it shows links to create a new one with the field populated to point back to this current record, or find based on a current record.

00:03:01 screen changes to Find AcctgTranss

So if we click on this we can see it's doing a find by the acctgTransTypeId. In this there are no results, but there's an example of it. Let's hop back to the main

page here and look at an entity that will have some better examples. Let's look at the product entity, or productType actually, so you can see the relationship to the product entity. Which is way down here.

00:03:41 screen changes to Find ProductTypes

So these are all the productTypes.

00:03:47 screen changes to View ProductType Finished_Good

If we look at one of these like the finished_good then it has a parent type. So here's an example of a type one relationship that is populated. And then type many relationship we can do a query to find all of the products that are related to this.

00:04:04 screen changes to Find Products

So it's showing one to ten of thirty. You'll notice for these larger entities this table ends up being extremely wide. This is rather generic code so it doesn't have anything special to handle this sort of thing, or to select which fields to pull out of the display. And because of that certain things, text fields and such, will sometimes become long, so we end up with tall things like this.

Let's find out which one that does, like the longDescription here which has a lot of spaces so it can wrap. So the browser optimizes the space by doing this. But anyway you can see the summary of them, and for each one you can view the record.

00:04:52 screen changes to View Product

And product. You can see it has a lot of different relationships. So we can see the related things, and we can edit the product entity itself, all of these different fields on it and such.

So that is basically it for those pages. Let's see, this does use the entity list iterator. If you're doing a search on a field with large tables, like this one is showing one to ten of thirty, you can click on next and get the next set. And that's it for the entity data maintenance pages.

00:05:46 screen changes to WebTools Main Page

Okay the next one is this Entity Reference, and you can generate a static version. The difference is this one is meant to be saved to disk, and then you can just refer to it locally in a browser.

00:06:09 screen changes to org.ofbiz.accounting.budget

If we just click on this one. These are fairly large web pages because of the size of the data model in OFBiz. But on the left here we have a frame that has first of all a list of all of the packages, and then after the package an alphabetical list of all of the entities themselves.

When we click on one of these it will jump to that, the anchor in this other frame for that package or entity. So if we wanted to look at the product entity for example we could hop to the product package.

00:06:45 screen changes to org.ofbiz.product.product

In the product package the first entity we have here is this. These are in this site, they are alphabetically organized by package, and then within each package they are ordered alphabetically by the entity-name. So here we have the communicationEventProduct, goodIdentification, and so on down through here. Entities and view-entities will be shown here.

Here's the product entity. You can see the details for all of the fields. The Java name; this is the field name, the DB name, this is the column name in the database. The field-type, and then for the given database the Java-type it translates to, and the SQL-type it translates to. So this is for the current database that's associated with the default delegator, the SQL-type and such.

Okay then it shows all the relationships, both type one and type many relationships. Where a relationship has a title it puts the title in black with the name of the entity as a link, so you can see the full relationship name as well as pick out which part is the related entity.

It also shows you the key mappings here, so we know it's productTypeId on this entity and on the related. Here we know that the key mapping has one name on this entity, primaryProductCategoryId, and another name on the related entity, just productCategoryId.

So this shows all the relationships. This can be handy for seeing how different entities inter-relate, and it's also handy for seeing, since in the entity definitions it will explicitly define type one relationships. But in many cases the type many relationships go in the other direction and are defined implicitly, or the entity engine automatically figures them out. So if we looked a the entitymodel.xml file, for the product entity we would not see all of these different ones, because these are automatically created.

With this you can browse around the data model. You can hop up to an agreement, and in the agreement you could hop over to the party. Party is like product; it has a large number of relationships. From here we have communicationEvent, so basically you can hop around the data model and explore things doing it this way.

I guess another thing I forgot to mention with relationships is when there is a foreign key name specified explicitly then that's shown here as well.

Okay over on this other side we can go back to the webtools main page. It has a little link for that, although typically this will open in another window, or as I have Firefox configured to do it'll just open up in another tab in the same window. So we'll typically still have the

webtools page open. You can pop over to the entity reference main page, the check/update database, and the induce model for XML

00:10:16 screen changes to WebTools Main Page

Those are just some convenience links which are also linked to from the main page, so we'll hop back here and look at them from here.

00:10:22 screen changes to Entity SQL Processor

The entitySQL processor is a little bit newer in OFBiz, just within the last few months, but it is convenient. Specify the group you want and it will look up the database that's associated with that group for the screen delegator. And basically what you do is you type in an SQL statement and then click submit.

So you could do something like select * from product, where product_id=”WG-1111”.

Okay so with a statement like this where we select all from product we can specify where-conditions, all sorts of things. It's basically just a full SQL statement, so we can do that sort of thing, or we could just do it without a condition and it'll return multiple results.

This is basically just taking the resultSet and displaying it in a table. You can do things other than selects here, such as updates and inserts and so on; just basically type in the SQL and execute it. Obviously the SQL would be database specific. And it's mostly meant for convenience tool, so you don't have to switch to a database client when you're working with this sort of thing. Or when you need to execute some SQL directly in the database.

00:11:43 screen changes to WebTools Main Page

That was the entity SQL processor. EntitySync status we looked at before, and it shows the status of different synchronizations in the database.

Okay induce XML model from database. What this will do is connect to specify a helper-name. This is the same as the datasource name. It doesn't have to be associated with the delegator or anything, it can be any datasource in the file. So we can do localDerby, and then induce, then basically what this does is it looks at the current database and tries to automatically figure out entity definitions.

00:12:32 screen changes to LocalDerby XML Model

Notice this isn't an XML document coming backs, So what we're seeing is just stuff. The best way to see the details is to do a view-source. This has gone off the edge of the page.

Anyways so you can see here in example, this is a fairly large document. Okay anyway you can see it is figuring out the entity-names and field-names. It's looking at the table-names and figuring out what the entity-name is, and looking at the column name and figuring out what the field-name would be. It does a best guess based on the size of the string for the type to use for each field, and it also does a very simple best guess on the primary key. This is not really very intelligent right now, it's just fairly basic.

So typically when you finish with this you'll have to go through and edit the entities to use a more fitting entry from the data-type dictionary, like these ones should really be something like id-ne instead of short-varchar. But it's just guessing the field-type to use based on the size of the string and that sort of thing. So it's fairly simple, but it can be helpful to get a start on an entity model.

Note that this does the entity model for the entire database, not just a certain group of records or anything, and so you'll end up with a single very large file.

00:13:58 screen chagnes to Check/Update Database

Okay the next page, Check/Update Database. This has some administrative tools related to the database. CheckUpdate is the same thing that's done on the start, and you can specify some of the same options that are in the entityengine.xml file for each data source. Like whether you want it to check the primary keys, check the foreign keys, check the foreign key indexes, add missing, and repair column sizes.

Repair-column-sizes is something that's unique to this. What that will do basically is, if the column size is larger, it will only do larger, make the column size bigger. Then it will create a new column with the proper size, copy all of the data, remove the old. and rename the new column to the old column name. So that's the repair column sizes; just a helpful little thing.

Okay remove all tables. There are certain circumstances where you may want to do this.

Specify the group form that it'll look up the datasource to use. You can see all of these default to org.ofbiz, which is the most common one. So that's for convenience. Notice the remove button is disabled, so you click the enable button and then remove.

Okay create/remove all primary keys. So create all the primary keys and remove all the primary keys. You can do the same with the declared indices, the foreign key indices, and the foreign keys; you can create and remove them all.

The character-set and collate. This will update these settings base on the current settings in the entityengine.xml file, the datasource element. This is something that is somewhat my-SQL specific. It does some funny things with the character sets and collations, and so sometimes you'll end up with problems

because of that and need to update them later on, which is what this is for.

Note that when you're removing all foreign keys, or actually for instance if you're removing all tables, there are certain cases where foreign keys will still be there and cause a table to not be removable. And so it's helpful sometimes to just remove all the foreign keys first and then remove all the tables.

So certain funny things like that can come up. These are basically just database administration tools that, like the rest of the entity engine, generate SQL based on the current entity model and send it over to the database.

One thing to note about that is when I'm saying, like remove all foreign keys here for example, it's only going to remove the ones that it knows about, the ones that are in the entity model. If you've added other foreign keys in the database that are not in the entity model then those will not be removed here. The same thing with indices and primary keys. This doesn't really come up, but same thing would be true if that was appropriate there.

00:17:11 screen changes to WebTools Main Page

Okay so that's it for the entity engine tools, and we've looked at all of those different pages and what they're used for.

00:17:25 screen changes to Advanced Framework Training Outline

And that is also the end of this section, so next thing we'll be moving on to is the service engine.

Part 4 Section 2.1-2.2

Service Engine: Introduction

Starting screen Advanced Framework Training Outline

Okay now we begin the section on the service engine. Let's review real quick where this fits into the bigger picture here in the reference book.

00:00:17 screen changes to Framework Quick Reference Book Page 18

This green box over here has these couple of boxes in it that represent the service engine, how they fit into the artifact reference diagram. And there are of course other parts of it; this is not meant to be a comprehensive diagram of everything in the OFBiz framework, but just the main pieces and how they fit together.

Service definitions; when a service is referred to through an event, or a service action in the Screen Widget, or a call service in the simple-method. There is also various very complete API for calling services in Java methods, or even when a service is triggered through an ECA rule based on the calling of another service that triggers an event.

So when a service is run it looks up the service definition. Depending on how that service is defined it may be implemented in Java, in a simple-method, or in a variety of other languages or remote communication mechanisms or these sorts of things. And we'll look at all of those as we get into the details of the service engine.

00:01:38 screen changes to Advanced Framework Training Outline

First of all the architecture and design goals of the service egnine. One of the most important things is something we call agnostic service calls, or in other words it is implementation agnostic and location agnostic. So when you call a service by name you just tell the service the name and the parameters, or the attributes that you want to pass to it.

You don't have to know where it's located, or where the file is located that it's implemented in, or on which server it's located, or any of these sorts of things, nor do you have to know how it's implemented. You don't care whether it's implemented in Java, or is a simple-method, or using some sort of a scripting language, or a remote call to a SOAP service, or whatever. You don't really care when you're calling it.

When you're designing the application in general of course you need to know those things, and those things are configured in the service definition. And of course those things have an impact on performance and such, but the code that actually makes the service call doesn't need to know those things. So this provides a nice loose coupling interface between the code that's calling the service and the services themselves, and provides a lot of flexibilities, increases re-usability, and decreases brittleness, so as different things change there's typically less of an impact on other parts of the code.

Another advantage of the service engine is that it supports different types of service calls, including synchronous, asynchronous, and scheduled calls. So with the same service definition, same service implementation, the code can call it in various different ways depending on the needs of the application.

Okay the service engine provides attribute or parameter validation for both incoming and outgoing attributes. The validation is somewhat simple that it provides, so typically you'll have more complete validation inside the service implementation, but it will at least validate the types of the parameters coming in: whether they're required or not, optional or not, and other such things. There are a few other validation options that we'll talk about when we talk about service definitions.

Okay there are a number of different ways of implementing services, and the objects that handle these different ways of implementing services are called service engines. So first of all there's a generic engine interface that each of the objects that handle these different types of services will implement. So if you want to implement your own, you basically just implement this interface. There are also some convenience objects, and we'll look at these down here in the designing a new engine section.

So there's some convenience objects in here for doing things such as handling asynchronous calls for languages that are naturally synchronous, and handling synchronous calls for languages that are naturally asynchronous, like a workflow engine for example.

Okay so you can see the definition of all of the existing engines in the serviceegine.xml file. Let's take a look at the current serviceengine.xml file in OFBiz.

00:05:20 screen changes to serviceengine.xml

And we see all of these engine definitions.

00:05:30 screen changes to Framework Quick Reference Book Page 15

For convenience these are also located in your quick reference book on the service configuration page, which is page 15. There is a service configuration example which has all of the general settings, and here in this section we have all of the different engines that are available. Let's zoom in a little bit so this is more readable.

Okay so these are the main ones: BeanShell (BSH) to implement a service with a BeanShell script, and a service group, so you can implement a service with a group of services that basically results in calls to other services. And we'll be talking about, as we go over the details of the service engine here, exactly how you define a group, and how you define a service that will call the group, and so on.

HTTP means call a service remotely on another server that is running a service engine via the HTTP communication interface. So it runs over HTTP or HTTPS, and you specify the URL to contact in the other server. And then it will encode the service call, send it over, call it on the remote server, the other server encodes the response and sends it back to this server, and that can decode the response and return it to the calling program.

So again the calling program does not need to know how it's implemented or where it's located, that's taken care of by the serviced definition. Okay interface is a service that's not actually implemented, it just acts as an interface. So it has attribute definitions and stuff that other services can inherit by extending or implementing this service.

Jacl is another scripting language, that's basically a tickle interpreter written in Java. By the way I noticed there are a few of these like Jacl that call through the BeanScripting Framework, the BSF engine. So calls to Jacl, Javascript here, uses the BSF engine, JPython uses the BSF engine, and so on.

So there are various scripting languages even in addition to those that you can access through the BeanScripting Framework or BSF, and call the scripts that way. So you could add other definitions here, or you can even call scripting languages more directly like we do with BeanShell up here; we use the BeanShell engine. BeanShell is more commonly used in OFBiz, so we have something that's a special case to handle that.

Okay so the Java engine. These would be static methods implemented in Java, and we'll look at these in a bit and see exactly how they're defined and such.

Javascript, the scripting JMS, is another facility for doing a remote service call that goes through a JMS server. JMS is inherently asynchronous, so this is great for asynchronous service calls that are distributed.

Whereas for synchronous service calls that are distributed we'll typically use something like RMI or HTTP. RMI is the preferred one because it's more efficient, being a binary format. The RMI service engine also by default does both encryption and compression, although you can change the configuration so that it just does compression or that sort of thing.

So this is typically the best one to use for synchronous calls, unless you have trouble running over the internet with firewalls and such, where you can't get RMI connections open, and then HTTPp is a good option for doing the remote calls.

And Jpython, another one of the scripting ones we talked about.

A route is like just like a place holder. I'm not actually sure if this is even used anywhere; let me see if I can find an example real quick. This is related to the workflow effort that was implemented on top of the service engine, and just isn't used very much. But it's kind of similar to the route concept in an XPDL workflow.

Okay the RMI engine is the one that we talked about for doing a remote call via RMI. Simple is for calling a simple-method, in the MiniLang that we talk about in that section of these training videos. SOAP is another remote invocation interface. It will call a SOAP service that is on another server using an automated mapping based on the service definition. So we'll talk a little bit about SOAP services, calling them on other severs as well as accepting or receiving SOAP calls for service engine services, and different methods of doing that.

So the service engine does have some stuff to automatically map between a service engine service and a

SOAP service. But the definitions are really quite different, so unless you have control over the definitions chances are you will have to write some mapping code, either using something like a service bus that can do transformation of a SOAP envelope on the fly, or writing SOAP client and server code in Java or some other language of your choice, and in that SOAP specific code making calls OFBiz services to handle. So the code basically is doing the mapping between the SOAP service and the OFBiz service.

Okay workflow will call into the OFBiz workflow engine and invoke a workflow to find an XPDL. So we can look at some examples of that.

So these are the different engines. We'll go over the rest of what is in the service config file of the servicengine.xml, and what everything means, in he configuration section. Which is coming right up by the way.

00:12:27 screen changes to Advanced Framework Training Outline

So those are some of the main types of engines that we'll typically work with.

And by the way the workflow reference to point to he old OFBiz workflow engine, but is now pointing to the Shark workflow engine. This is something that is still in development but it is usable. There are certain tools that don't work in the integration, but for the most part it is functional. There is a web based UI for the Shark worfklow, and once it is more stable will eventually replace the old OFBiz workflow engine.

00:13:19 screen changes to serviceengine.xml

If we look at the current workflow type then you can see there's the Shark service engine. This is part of the Shark integration, so that you can call a Shark workflow through the service engine.

00:13:38 screen changes to Advanced Framework Training Outline

Okay so that's basically it for the architecture and design goals. Next we'll talk about the service engine configuration.

The main element of the service engine configuration, or the main file, is this serviceconfig XSD that defines the structure of the file. And the location of the default file itself in OFBiz right now is this serviceengine.xml in the config directory in the service component, which is one of the framework components. So there are various things that we define in here, references to JNDI servers and resource loaders. The resource loaders are very similar to those in the entityengine.xml file, in the ofbiz-component.xml files, and so on.

On to global service definition files. Just like with the entity you can refer to service definition files from the service engine file, whereas with the entity engine you can refer to them from the entityengine.xml file. But typically these are not referred to there because each component has its own list of services, as we looked at in the component configuration section.

So that's typically not done, and service definitions, service ECA rules, and service groups are all typically configured just through the ofbiz-compoennt.xml file for each component. Okay we'll talk about the service group and SECA or Service ECA rule files. It can refer to those, again they are not referred to so much typically through the serviceengine.xml file because the ofbiz-component.xml file is what really governs those.

You also configure JMS servers through this. JMS servers in a service definition are referred to by name, and then that name is used to look up the server definition here. And for these server definitions we'll see the details, but they basically include the location of the JMS server, name of the topic or queue, whether it is a topic or a cue, and those sorts of things.

00:16:13 screen changes to serviceengine.xml

Okay so let's dig right in and look at the service engine. The current serviceengine.xml file is here.

00:16:16 screen changes to Framework Quick Reference Book Page 15

This one here in the quick reference book is based on that one, although it's slimmed down a little bit to use just as a quick reference with a reminder and such. And also in the quick reference book we have a complete list of all of the elements and attributes in the serviceengine.xml file.

00:16:47 screen changes to serviceengine.xml

So let's look at it from a high view real quick, based on the current one. Authorization is the name of the service to use for authorization, I guess exactly as it says there. You just specify the service name, which by default is userLogin. So if you want to replace this with your own service you can do that for authentication or authorization.

Thread-pools. The thread-pool is what we use for running the all of the asynchronous jobs and scheduled jobs, so this is the pool of threads that is used for running these separate from other things that are happening on an application server. And these are basically various parameters to configure that, and we'll talk about these in detail as we go over the reference.

Okay we already talked about the engines. These are basically just the different engines you can use in service definitions, and represent different ways of implementing services. Service locations can be defined

here. Each as a name and a location. This is so that you could have a central location that is shared among multiple services, especially for remote service calls, rather than specifying the same location in the service definitions over and over.

Okay startup-services. These basically just specify the name of the service. Runtime-data-id is for a runTime-Data entity reference. Delay, so how long after startup it should run, and which pool it should run in. So run-TimeData, by the way, I guess I should be more specific. As it mentions here it's for the in-parameters, for the context that will go into the service. So this is a service that will run every time the server starts, basically is what this accomplishes.

Okay and then we have the jms-service servers. We have just a general name here. Each server has a name, a jndi-name, where you can find the JMS interface. The topic or queue (topic-queue) name such as this; this would be a topic name. The type is topic, and it can also be queue here; we'll see that in the reference. Username and password for it. And whether to listen to this server or not for other service calls, that are coming from other servers that are sending messages to that server. So that's basically what that's for.

00:19:48 screen changes to Framework Quick Reference Book Page 15

Okay let's go over some more of the details of these things from the quick reference book. So in the Service Configuration file there are a number of these different elements, and let's go over them and their details one at a time.

We talked about the authorization, there will just be one of these, and we're just specifying the name of the service that will be used for user authorization or authentication, basically checking their username and password.

Okay the thread-pool. You'll basically have just a single thread-pool defined. When services are run through this service engine, or the service engine for the current server, this is the name of the pool that they will be sent to, and each thread-pool has a name. So specified here is the name of the pool that these would be sent to. There's also a run-from-pool element where we specify the name, and notice that there can be zero to many of these, where we specify the names of the pools that this service engine will run from.

So this thread-pool that's running that takes care of asynchronous persisted, as well as scheduled persisted jobs. These are the different pool names, the ones specified here in the run-from-pool element. These are the different pool names that it will be looking at when it reads from the database.

So you can have separate sets of service engines that are doing different things. For instance you could have a server that just reads service calls from the database, scheduled or asynchronous persisted service calls, and it only reads a certain pool. And then you could have all of the application servers not read from that pool.

Now this send-to-pool is the one that services are sent to by default. But when you're calling a service scheduled or asynchronous persisted, there are options in the API to specify the pool name that it will be sent to. And in the data model for these service engine pools it also keeps track of the pool that the persisted job is in. Okay so you can use this to manage basically which servers will be looking at which sets of services to run.

The next attribute is purge-job-days and the default is 30 days, so if you don't specify one then the default is 30. What this means is the number of days to keep old jobs around before they are purged. So there's a service that runs periodically, I believe once a day is how it's scheduled to run, and it will remove the job information data about old jobs that have run, anything older than this many days before the current date and time.

So you'll notice in the configuration file, the value out of the box is actually set to 4, a little bit smaller. So these defaults are typically just a safe high.

Failed-retry-min is the number of minutes to wait after a service fails, after there's an error in it, before retrying the service.

TTL, this is the time to live for a thread. You can see here it's set to a very big number, looks like eighteen million. This is in milliseconds, so that's about five hours. And the so the time to live for the thread means how long after the thread is created it will remain until it is recycled, basically until it is removed from the pool.

Actually related to that is this jobs attribute, and the jobs attribute tells it how many jobs to run before the thread is recycled. And you can see the typical value here; the one out of the box is ten jobs that will run before the thread is recycled. So the intent of these is to keep the threads fresh, and to make sure there's nothing that's kept around long term in these threads.

Okay wait-millis, or wait milliseconds. Once a thread basically finishes all of the jobs that it's currently running, this is the amount of time that it will wait before it looks for new jobs to run in the job queue that the service engine maintains. So again once it's finished a thread will basically be running jobs. Once it's finished with its batch of jobs it will check to see if there are any jobs left to run. If there aren't any it will wait for this amount of time, for this many milliseconds, before checking the queue again to see if there are any jobs to run. So this just reduces busy waiting on the server, reduces the CPU load.

Okay we talked about the jobs, the minimum number of threads (min-threads). So this is the minimum number of threads to have in the pool, this is the number it will have when it initializes, and then as the load on these threads increases more threads will be allocated up, to the maximum number of threads (max-threads). As their time to live expires, or they run the number of jobs, they'll be removed, and as needed new threads will be created, up to the maximum number. So when it's quiet threads will be removed until it gets down to the minimum number of threads.

So you can see the out of the box values are 5 and 15, some good example values, though for a busy server this can be significantly higher. One thing to keep in mind, though, is that typically these thread-pools will be running on the same server as the web application server, so you don't want to have too many threads in the pool. Unless you see that it's just not handling the jobs, the persisted and scheduled jobs, and then you may want to consider adding a dedicated server to start running those jobs.

Okay poll-enabled. This is whether this service engine thread-pool will poll the database for jobs. This'll almost always be set to true. Very rarely it will be set to false, although you can set it to false if you don't want it to run jobs from the database, but rather you only want it to run asynchronous in memory jobs. So it would not run any asynchronous persisted or scheduled jobs or services.

The poll-db-millis is how many milliseconds. So if the poll is enabled by the way there's a separate thread that does the polling, that will basically look for the upcoming jobs to run and put them in the in-memory job queue to be run by the thread-pool. This poll-db-millis specifies how many milliseconds to wait between polling the database for new jobs to run, and defaults to 20,000, or 20 seconds.

Okay and we talked about the run-from-pool. You can have multiple instances of this to specify all of the service pools that this thread-pool should read from, and again each service engine instance just has a single thread-pool that it maintains on each server. And generally while there can be multiple dispatchers they'll all use the same configuration, basically share these sorts of things, share the thread-pool and such. So there'll be typically one per instance of OFBiz that's runnign with the service engine in it.

Okay engine. We looked at all of the out of the box engines that are available for running services. We talked about startup-service; you specify the name of a service that will be run when OFBiz starts up. Runtime-data refers to a runtime-data record in the database that has the context information, or the parameters or attributes to pass into the service. And runtime-delay, how long to wait from the startup of the service engine to actually run this startup service. And which pool to run it in (run-in-pool). If you don't specify it it will default to the send-to-pool here, just like all services do.

Okay service-location. We talked a little bit about these where you can specify a location name and then the location value. This is to avoid having URLs in service definition files over and over, giving you a central place where you can change things to point to different servers and such.

So these for example: the main-rmi, main-http, entity-syn-rmi, and entity-sync-http. These are used instead of having these locations in the service definition for example. We have the entity-sync-rmi used instead.

00:30:22 screen changes to services.xml

Now to refer to those, to use those in a service definition, let's look at the example of the services in the entityext component that we were looking at in the entity engine training, under the entity synchronization section. Instead of specifying the full location you'll simply specify the location name here, entity-sync-http, and entity-sync-rmi.

00:30:45 screen changes to serviceengine.xml

So that's what we can use for the location instead of specifying the full URL, and then it can look up the full URL in here. So for the HTTP one and the RMI one we can look up the full location for those. Now you'll note here that when we are using the HTTP service engine, we actually use a request map that is defined in the controller.xml file, so his is actually a control servlet request, and by default it's in the webtools webapp.

That should be located somewhere, and if you're not deploying webtools you need to deploy something that has these if you're going to be using the HTTP services.

The RMI ones are a little bit different. When OFBiz starts up, you can specify an rmi-dispatcher. This is done in the ofbiz-containers.xml, so we saw it there as we were reviewing it. Oops not this ofbiz-containers, we're looking at the wrong one, that's why, that's the template that's in this other file.

00:32:16 screen changes to ofbiz-containers.xml

Okay we want the ofbiz-containers in the base directory, not in the app server config.

In the ofbiz-containers file we have the rmi-dispatcher container and the name of the dispatcher here, that's the name it binds to, and the port and host, or IP address that it binds on. So there's the name of the dispatcher.

00:32:41 screen changes to serviceenigne.xml

And that corresponds to this URL here. The port is there, and the name of the dispatcher here on the end. So those two need to match up, or it needs to refer to one that exists.

00:32:57 screen changes to Framework Quick Reference Book Page 15

Okay going back to the quick reference. That's basically how the service locations are used.

Resource-loader is not used too commonly in here. There aren't very many things that refer to it, and the ones that do are these things like the global-services, the service-groups, and the service-ecas. And those as we've talked about are these different service resources that are loaded through the ofbiz-component.xml files typically, instead of the serviceengine.xml file.

It is still possible to load them centrally here, but it's better organized if they're configured for each component, rather than in a central location. So when you create your own component you can configure your service-resources, and just drop it into OFBiz, and when the component is deployed the service engine will pick those up. And there's no need to change any central file to get your services and other service resources recognized.

But the resource-loader here is very similar to the one in the entity engine; it's basically the same thing. Give each resource-loader a name, and that name is the same as what will be used in the loader attribute for these. The class is the same as the class in the entity engine one, and the one in the current entityengine.xml file shows the one to use for a file. Prepend-env will typically be ofbiz.home, which is an environment variable that's always populated, and you can put a prefix which will go after the env and before the location that is specified in whatever uses the loader.

00:34:59 screen changes to ofbiz-componenet.xml

So just to clarify where entity-resources are actually referred to, let's look at the ofbiz-component.xml file. We'll just do whichever of these, we'll start with the accounting one, it's the first one on the list, and it has quite a few of them actually.

We have all of these different service-resource elements. Each of them has a type, and these are all service model or service definition XML files. Here's a service group XML file, and a service ECA XML file. So this component actually has all three.

00:35:29 screen changes to serviceengine.xml

And this is how we refer to the service resources, rather than putting all references to them in the serviceengine.xml file. It's far better organized and more flexible.

00:35:38 screen changes to Framework Quick Reference Book Page 15

Okay moving on. Jms-service. Rhe jms-service name here is what is used for the location in the service definition.

00:36:12 screen changes to services.xml

Let's look again real quick, I believe there are some JMS ones in here. Here we go. So when the engine is JMS, this is one of the distributed cache clear services (distributedClearAllEntityCaches) that we look at in the entity engine section.

So when engine is JMS the location will be the name of one of these JMS service elements in the serviceengine.xml file.

00:36:38 screen changes to serviceengine.xml

So the serviceMessenger here and the location of this service definition corresponds to the serviceMessenger here. Notice this one is commented out, as there is no JMS configured by default in OFBiz.

00:36:50 screen changes to Framework Quick Reference Book Page 15

But when you do configure such a thing, that's the name to use. Now if you have multiple servers underneath it, you need to have at least one, but you can have many, if you have multiple servers underneath it then you can specify a send-mode. This is very handy actually, and it can be used if you have multiple JMS servers and say one is down, and you want to send it to the next one if the first one is down, you could use the first-available option here. It will use the first server that's defined, and if that one's down it will use the second server, and if that one's down and there are more servers defined it will keep on going down the list until it finds a server that is available, that will successfully take the JMS service call.

You can also send it to all of them. You can distribute the load to random, round-robin, or least-load. I don't know if the least-load is implemented actually. But anyway that's the general idea with the send-mode, is how to handle the multiple servers.

Okay each server will have a jndi-name and a jndi-server-name. Again the jndi-server-name comes from the jndiservers.xml file. So for example we have the default JNDI server, which is typically just when you get an JNDI context it's the default one it will use when you don't specify any parameters. And that's typically the one that's configured in the jndi.properties file, although it depends on the application server. Some application servers do things a little bit differently for the default JNDI server for the application server.

Okay then the jndi-name is the name of the resource in JNDI to look up, so here's an example of that. For the JMS connection factory either a topic or a queue connection factory. Topic-queue is the name of the topic or queue to use on that JMS server, and then in the type you need to specify whether it is a topic or a queue, and that is required, so you specify one or the other.

If you're not so familiar with JMS the main difference between a topic and a queue a topic is something that multiple servers will listen to or can subscribe too, so this is a publish subscribe model. So when you send a message or publish a message to the topic, the JMS server will send that message to all of the listeners on the topic. With a queue there will be just a single listener, and it will queue up messages that come in as needed to guarantee that they are sent out to the server. Now depending on your JMS server it may or may not support queuing for a topic. So in other words if it can't send the message out to a certain server it may just throw the message away once it's done sending to the rest of the servers, rather than holding on to it remember that it it still needs to send to that server.

So depending on your JMS implementation just be aware that there is sometimes that sort of an issue with the topic versus a queue, which is typically guaranteed to get the message through, or it will keep, it depending on how the service is configured. Sometimes keep it indefinitely, though of course sending email alerts and whatnot telling people that it needs to be resolved.

Okay the username and password to connect to the JMS server. And the client-id. Some JMS servers require a client-id. This is optional, as not all JMS servers do require it, but that's something that you can use to identify yourself as a client.

Then you can specify whether to listen to this particular JMS server or not; the default is false. If listen equals true then it will basically be either listening to the queue, it'll be the receiver of the queue the one and only receiver registered as that, or it will be one of the subscribers to the topic.

Okay listener-class is optional. You can specify a class that's used for the listening. You can see in the current configuration (screen changes to serviceengine.xml, then back) this is not used. But so the listener-class will allow you to specify a class to use for listening to the topic or queue, rather than the default implementation.

Okay and that is it for the service config or servcieengine.xml file

00:41:54 screen changes to Advanced Framework Training Outline

And here in our outline we have gone through now the service engine configuration. For most of this information by the way you can always of course reference it in the transcription of this training video, or come back and listen to my voice again. But a lot of it is also available in this document that has been around for quite a while, this serviceconfig.html. This is the service engine configuration guide.

That's the current location of it, and as noted with some other things this will eventually be moving to the docs.ofbiz.org server.

Okay that's it for the service engine configuration. Next we will be looking at service definitions.

Part 4 Section 2.3

Service Engine: Service Definition

Starting screen Advanced Framework Training Outline

Okay in the next section we'll talk about service definitions, or defining services, now that we've covered the service engine configuration. We saw a taste of defining services in the Framework Introduction videos with the various example services, mostly for the purpose of create update or delete, or CRUD services. And now we'll look at quite a variety, and in more detail, about what all of the attributes and elements within a service definition mean.

First off the schema file for service definitions is here, in the framework/service/dtd/services.xsd file. And in a component you'll typically find services here, in the servicedef directory named services, optionally something.xml. Often if there is some sort of an extension here, the *, beyond the services it will be separated with an _, that's just the convention that has developed over time.

So that's most typically how you'll find them. Before we get into all the details, which we'll cover in the framework quick reference book, let's look a the webtools webapp where there is a service reference page.

00:01:38 screen changes to OFBiz: Web Tools:

Let's go look at that. Down here in this section on service engine tools the first item is the service reference.

00:01:53 screen changes to OFBiz: Web Tools:

When we got to the firs page it basically does a listall services for the current dispatcher associated with this current web application. One thing I should note about dispatchers is that, as you can see here, there are multiple dispatchers. Each dispatcher can share global services as well as local services, so it's possible to have a dispatcher for a web application that uses the classloader context of that web application, which by the servlet standard is separate from the classlaoder of other webapps.

So if you have service implementations that you want to use in a webapp, that are on the classpath of that webapp but are not avaiable to other webapps, then you can use a per-webapp dispatcher. That's not used very much anymore, in fact I don't know that we use it at all, but it is still possible to use it.

Now when you do that you specify the location of the file in a web.xml initParam. That's not really used typically, so all the different dispatchers basically just share a global the global set of services that are available.

So regardless of which dispatcher you setlet you'll basically get the same set of results. But there is typically one dispatcher for each webapp, is what these come down to. And the reason is because of that feature that is available, although typically not used, to have a webapp or local dispatcher specific services. Typically we define services, put them in the service, and notify the service engine about them through the ofbiz-component.xml file, which adds them to the global services list.

Okay so going to the first page here we get an alphabetically ordered list, by service name, of all services. You can see up here it tells you how many services there are, 1,552. And you can scroll down through this as desired, or limit the list by clicking on these letters up on top here.

Notice most services do start with a lower case letter. There are just a few that start with an uppercase A here, but for the most part they start with a lowercase letter, which is the convention and recommendation for OFBiz.

These by the way, for example aim is an acronym which, if it started with a capital letter, would be Aim. This should be lowercase aim, and to separate cc as a separate acronym you can do an uppercase C then a lowercase c, then an uppercase R and so on, as is here. The same would be true with these others. So the naming convention that we use for service names follows basically a method name in Java, that is the convention that we are going with.

Okay if we select a letter we can see just the services that start with that letter, just a few here. Notice this is only moderately useful, because many of the services are Create Update and Delete services, so when we get down here to the CRs we have a whole bunch of create services.

Now on this page, for each service it just tells the service name, engine name, default entity name, the method or target service or whatever to invoke, and the location of the service implementation. When you click on any of these things it will filter the list by that.

So if we want to see all the services that are in this file we can click on that, and if we want to see all the services in Java we can click on that. All the interface services, which wouldn't be too terribly many, we can click on that. You can see there are a number but not terribly many, 43 it looks like.

00:06:50 screen changes to https://localhost:8443/webtools/control/avaialbeservices?sel_service_name=calcbilling

Okay let's look at a particular service, one of these that's not an interface so we have a more complete set of information. And it's quite a bit of information about the service: the service name, the description, whether it's exportable, or whether the export attribute is set to true. Also some other miscellaneous information that comes from the service element in the servicex.xml file, so we'll go over these in detail there. Any permission groups associated with the service and service it implements, if there are any. And we'll go over what these mean as well, as we go over the details of the service definitions.

So then it shows a list of all the in parameters and the out parameters. Notice that inout parameters are in both lists, like userlogin here. The inout parameter is in both lists, otherwise this list just has the out parameters, and this list just has in parameters. Now this is set internally, every single service has a whole bunch of attributes that are automatically added by the service engine for general purposes. So all services have a userlogin coming in and a locale coming in. We see these are set internally is set to true. So these are parameters or attributes that are automatically set up by the service engine.

So coming in we always have a userlogin and a locale, same going out, and going out we also have these standard sets of response information. So the response message is the success or error response code might be a better code, but response message is the parameter; ErrorMessage, errorMessageList, successMessage and successMessageList.

00:08:57 screen changes to https://lcoalhost:843/webtools/control/availableservices?sel_servce_name=calcship

So all of the services that we look at, if we hop over to another one here we'll see it has the same set userlogin and locale, and a bunch of these response ones. All of these ones that have the is set internally set to therue are in that category. So the ones that have is set internally set to false are the ones that are actually defined specifically for that service, so all of these sorts of things.

Okay here's an example of one that does implement an interface. It shows a link for that interface, and if we click on it we get the definition for the interface. another thing you can click on here, there is a schedule button that pops us over to the job scheduling page

(moves to that page). Which you can also get to from here (jumps to Webtools Main Page and back to Job Scheduling Page), schedule job.

And when you click on that schedule button it bypasses this screen. So you just use the default options, which are basically meant to run once but with no restrictions on the dates or intervals or anything.

Max retries will just run it once by default. So normally when you come in here and schedule a job you'll do it through with a service name here, but that button bypasses it. We'll talk more about this scheduling job screen and following screens in the job scheduling process when we get to the Scheduled Services section.

00:10:33 screen changes to WebTools Main Page

So one last thing I wanted to talk about here is the service log.

00:10:42 screen changes to Service Log

This shows the last two hundred service that were executed. That number two hundred can be changed, it's just a constant in one of the Java files. So basically what you'll get when you come to this page is the last two hundred services that were called. It shows the name of the service, the dispatcher name that it came through, which typically corresponds to the webapp that it is called through, the mode, which is synch or asynch, the start time, and the end time for the service.

So you can see some information about what's actually happened. This is just a list that's kept in memory and cycled through as more services are called. So just a little convenience tool.

00:11:29 screen changes to WebTools Main Page

Okay these other three pages we'll talk about when we get to the job scheduling section.

00:11:37 screen changes to Advanced Framework Training Outline

Let's move on to service definitions and the details about the services.xml file, all of these things. And a few more actually. We'll get into all of the details here.

00;11:54 screen changes to Framework Quick Reference Book Page 3

This is page 3 of the quick reference book, covering service definitions, service groups, and service ECAs. As you can see the service groups are fairly simple. The service ECAs are fairly similar to the entity ECAs, but we'll cover those later on when we get to those topics. So now we're going on to the service definitions, and this panel is also covered, which covers the details about an attribute in a service definition.

So we've seen a number of examples. Down here we have an example of an interface and then a create/update. Some more functionality specific or process specific services like the index product keywords, and some other examples. We've already seen some examples in the Framework Introduction videos, so this is just to review quickly. Every service will have a name, can have a default entity-name, and will always have an engine, which specifies how the service is implemented.

So here's an example of using auto-attributes. Where we say include the non primary keys we can also say include primary keys or include all. The mode in, out or inout, and optional is true or false. So all of these automatically created attributes will have option equals true, mode – in, and it will go through all of the non primary key attributes.

You can specify an entity-name when you do auto-attributes, or if you don't it will default to the default entity-name. Then we're excluding these fields, and then that interface is used with the implements tag here in other service definitions. This has to complement that, its own auto-attributes, to pull in the primary key fields from the default entity, since no entity is specified the product entity. Their inout and optional is true. So then it's overriding, it's saying the productTypeId is not optional, and the neitnernalname is not optional.

Okay so there's some quick overview of what some of these things mean in the service definition. By the way we looked at the engine example here which does not have or need a location in invoke. But with the simple or Java or whatever other type service we typically have a location, the location of the service implementation, either in a file or a server somewhere, depending on what type of engine it is, and the method to invoke at that location.

Okay so let's go look at all the details now. First of all in a services.xml file you can have a number of these different things: a description, vender version created service, and then it's this service tag that gets all of the details. So you'll have one to many service tags, and for each service tag you're defining a service.

So every service must have a name, and the name should be globally unique. If the name is the same as the name of a service that's been defined before it will override that service definition; it will basically write over top of it.

Okay then each service has an engine that we specify. We've talked about all of the different engines.

00:15:42 screen changes to Framework Quick Reference Book Page 15

You can see a list of them in the reference book here on page 15, the Service Configuration page, and it has

in the service config example all of the different engines that are available. So these are all of the different ways of implementing a service.

00:15:57 screen changes to Framework Quick Reference Book Page 3

And back on page 3.

Okay the location and invoke; we talked about those. The location of the service. Depending on the engine it could be a local file or a classpath resource typically, and although you can use things like the component:// we may be moving to that in the future rather than using classpath resources. Which is what we typically use even for simple-methods right now, which don't have to be on the classpath. For Java files and such it'll always be the fully qualified class name of the class that the static method is implemented in. When we talk about service implementation we'll see more details of how those look.

Okay so we have the location, and at that locaiton the method to invoke. So either a method in a Java class, a simple-method name within a simple-methods file. If it's a remote call like vitjms or RMI, this is the name of the service on the remote server to call. As we've seen with the distributed cache clearing services for the entity engine, and also the entity sync remote calling services.

Okay auth=true or false. By default authorization is not required, so auth=false is the default, so if you want the login to be validated and require valid credentials coming in, either in a userlogin object or in separate username and password fields, then you should set auth=true.

Export by default is false. And that means that it is not available to call externally or to call remotely, so if you want to be able to call it remotely then you need to set export=true. Now this export flag is basically used for certain remote calling facilities that are a little bit less secure, like calling a service through HTTP. There's a request that calls into an event that calls the service in the webtools webapp through HTTP, where you can just pass in which HTTP parameters you want, the service name and different incoming attributes, and it will call the service for you. But you can't use that unless export is set to true.

So when export is set to true you need to make sure that it's set so that the service can stand alone with regard to security related things, so you'll pretty much want auth to be true for an exported service, unless it's a publicly available service that doesn't involve sensitive data. You'll also want it to check permissions or other things inside the service.

Okay validate equals true or false, by default is set to true. This tells the service engine whether to validate the incoming and outgoing contexts or parameters that are passed according to the attributes that are defined on the service. So by default it's set to true, which is typically the behavior we want. But if you want to have a service where you can pass anything to it and it can return anything back without the service engine validating those, then you can just set this to false.

Okay the default-entity-name. Each attribute can have an entity-name specified. If there is no entity-name specified but there is a default entity-name specified then each attribute will default to being associated with that entity. If there's no default-entity-name specified and no entity-name specified, then there's just no entity associated with the attribute. To go along with that, each attribute can have a file-name within that entity that it is associated with.

So this is especially helpful for things like the Form Widget for example. When it knows the service attribute definition as well as the entity field definition, it can much more intelligently automatically create a user interface element for that service attribute. And then also for general information, and in the entity reference and stuff, we can look at all of the the services that have a certain default entity and so on.

Okay use-transaction, require-new-transaction, and transaction-timeout are three related attributes that govern the transaction management options for a service. By default use-transaction is true, and what that means is when the service starts or when the service is called, the service engine will check to see if there's a transaction already in place. If there is it will just use that transaction, if there is not it will begin its own transaction.

If you do not want the service engine to begin a transaction for your service then just set use-transaction to false. But if there's already a transaction in place that transaction will still be in place when your service is called. Setting use-transaction to false just means that if there is no transaction in place the service engine will not create a new transaction for you.

Require-new-transaction by default is set to false. If you set this to true the service engine will always create a new transaction for the service call. If there is not a transaction already in place when the service is called, it will simply begin a transaction.

Okay so when require-new-transaction is set to true and there is a transaction in place, as opposed to not being a transaction in place and it can just start one, for a transaction in place it has to pause the existing transaction, create a new transaction, and then when the service is done running it will commit or rollback the new transaction that it created and then resume the original transaction that was there. So it's a little bit more complicated, but not significantly so.

In a case where you want your service to run in its own transaction, independent of whatever might be calling it, then just set require-new-transaction to true and off of you go.

Okay for the transaction-timeout you can specify here the number of seconds to use for th timeout of the transaction. The default is typically sixty seconds, although that is configurable usually in the transaction-manager. And for the embedded Geronimo one it is sixty seconds by default.

So you can set a different transaction timeout. The trick with the transaction-timeout is that it only works when a new transaction is started, so if you set require-new-transaction to true then you know that it will always use the transaction-timeout. If require-new-transaction is set to false, the default, and use-transaction is set to true, the default, then the transaction-timeout will only be used if there is not a transaction already in place. If there's already a transaction running, and use-transaction is just going with the existing transaction, then it will not be able to change the timeout of that transaction.

Okay max-retry by default is set to -1, which means no limit on the max retries. If this is a service that you know will be scheduled asynchronously, and you do not want it to run indefinitely when it fails, then you should set the max-retry to a certain value such as 3, if you want it to retry 3 times before giving up on that service call. It will store in the database the results for the 3 service calls so you can look there and see that there was an error. But other than that after those three calls are done it will give up, and that's what the max-retry does.

Okay debug by default is set to false. If you set it to true what it will do is turn on verbose debugging for the duration of the service call, so this is useful for a development environment. Though I would not recommend it for a production environment, because it doesn't just turn on verbose for that service only but rather turns it on for the entire server, and then turns it off for the entire server when it's done.

Okay so those are the attributes on the service element. Next let's talk about the sub elements underneath the service elements. Description is just like a description anywhere, you can put text to describe the service.

Name-space will define the name-space for the service. This is typically just used along with SOAP services for the name-spaces there. So the name-space here does not segregate the name-space for service names; those do need to be globally unique.

Required-permissions. Now we're starting to get into the interesting ones. For required-permissions you can have zero to many of these. And with each one of these you'll specify a set of permissions that is required and they can be combined with the join-type here of and or or, and there's no default so you must specify which join-type to use. So you can either specify zero to many permissions, and/or zero to many role-members.

So let's look at what each of these means. Check-permission you just do the permission and action. This is just like the check-permission in the simple-method, where it we'll have something like a permission might be product, or catalogMgr for the catalog manger base permission, the general catalog manage permissions. And then the action will typically be something like _create, _view, _admin, that sort of thing.

So the full permission name is the action plus the permission. One of the reasons we separate them is because it clarifies semantics, but also because if you have an action in here like _view or _create, if the user has the _admin extension on the permission then that will qualify for any of the other ones.

Okay check-role-member is a fairly simple one; just check to see if a party is in a given role. So these required-permissions are not used all that often. These two elements that exist right now, the check-permission and check-role-member, are the beginning of what was planned to be a larger set of functionality, but this was sufficient for the needs at the time. This is an alternative to doing permission and other checking security checks inside the service implementation. Which, especially before this was created, was on the only way to do it. And that still remains for the most part the traditional way to do it.

You don't see this required-permissions being used very often, but it is certainly one way to do a certain variety of permission checks. And it is nice to have those in the service definition so that they are closer to the surface, rather than down inside the service implementation. It's also nice to have this sort of validation in a single place, because there is other validation going on in the service based on the attribute definitions and such, of the types and whether they're optional or not. And so it's nice to combine all the error messages for those with the permission error messages, if there are nay.

We may be moving more towards this more in the future, so it's good to know about. But you won't see this used a whole lot in the current practices for the most part, to do permission and other checks inside the service implementation rather than the service definition.

Okay the implements tag; we looked at how this works. Basically with implements you just specify the name of a service to implement, and it inherits all of the attributes from that service. Auto-attributes allows you to define a set of attributes all at once, based on an entity definition. Attribute explicitly declares a single attribute,

and by the way these must be in this order in the service definition; the implements, then the auto-attributes tags, and then the override tags.

The override tags for any attributes that already exist. This is especially helpful for when an attribute is inherited from another service, or is automatically created with auto-attributes. You can override things like whether it's an in or an out, whether it's optional or not, and these sorts of things.

So let's look at these definition things in more detail. For auto-attributes we looked at the example down here in the interface product service. So you can optionally specify an entity name, and if you don't specify the entity name it will look at the default entity-name. If there is no default entity-name and you don't specify an entity-name here then this auto-attributes tag will be invalid, and you'll get an error message.

Okay you can specify whether to include the primary key fields, the non primary key fields, and/or all fields from the entity. It defaults to all, of course, and for each of the fields that are included it will create a new attribute with these other attributes set as the options on it. So for the mode it is required that you specify that. It will be in for incoming, out for outgoing, or inout for both incoming and outgoing. And then optional would be either true or false, and defaults to false, defaults to required basically.

Form-display also true or false, this form-display here. So all 3 of these, mode, optional, and form-display, correspond with attributes over here on the attribute tag: the mode, optional, and form-display down here. Form-display refers to whether these attributes in this case should be displayed by default on a form. So when a form does an auto-fields from a service, using that tag that we talked about under the Form Widget section, then it will look at this form-dispaly attribute for each of the service attributes to see if it should be included by default there.

Okay then the exclude tag, if we have a number of different fields in an entity that would normally be included in the auto-attributes. If there are certain of those that we do not want it to define an attribute for on the service, then we can just use the exclude tag and specify the field-name that we don't want to be included. And there can be zero to many of those so you can specify multiple exclude tags to exclude multiple fields. Just as we do here.

So under this auto-attributes we exclude the createdDate, and we also exclude the lastModifiedDate and so on. So in other words these are fields that are not meant to be maintained through the client, the user of the service, but rather typically it implies that the service itself will be maintaining those fields.

Okay we've looked at auto-attributes, now let's look at the details for the attribute element. Each attribute must have a name, as we've seen, and a type. The type can be a fully qualified Java classname. Or we have some simplified types, like if you specify it without a package it will look in java.util and java.lang for that object. So let's see if we have an example. Oh also there's a special case for the entity engine objects like GenericValue.

So if we just specify string instead of specifying java.lang.string it'll find it just fine. And if we just specify GenericValue instead of org.ofbiz.entity.genericvalue then that will also work just fine. So there are some special cases to simplify the service definitions, and these are some of the more common ones you'll see. But you can use any fully qualified classname.

And mode we've talked about. Optional we've talked about; defaults to false again here. So if you do not specify an optional = true then the attribute will be required.

Okay the form-label is the label to use if this attribute in a form field that is automatically defined from the service attribute. And that goes along with form-display, which tells the Form Widget when doing an auto field service whether to include this attribute in it or not.

Okay entity-name and field-name are just for information purposes as we talked about before, related to the default-entity-name over here.

String-map-prefix and string-list-suffix. These are for informational purposes for a specific variety of service calls. If you're calling this service through the control servlet using the service event handler, or the service multi event handler, then you can specify the map-prefix on an attribute on the service. And instead of looking for the attribute-name incoming, it will look for the map-prefix plus the attribute-name. List-suffix does a similar thing for lists. So basically what these will do is, with the information coming in, it will find all.

Okay so with the map it will find all of the request parameters coming in that have this prefix. And it will put them in a map, with the name-value pairs of the remainder of the parameter name after the prefix and the value of the parameter. Those will be the name-value pairs that go into the map, and then that map will be put into the incoming service context with the name of the attribute. So in that case the type of the attribute should be something like java.util.map or .hashmap or that sort of thing, typically java.util.map is the name of the map interface for Java.

Okay the same sort of thing with a list except, it's a suffix instead of a prefix. It will look for this suffix on all incoming parameters. And if it finds it then it will put those parameters into a list, and that list will be passed

into the service after this attribute. So there the type should be java.util.list.

Okay we've already talked about the form-display. Type-validate is another of the validation extension, beyond just specifying whether it's optional or not, and the object type that it should be coming in. So type-validate allows you to specify a method to use for validation, and you can do this in the attributes or in the override. This is just like some of the other validation methods that we've seen, where you specify the class-name and the method name. Class-name defaults to org.ofbiz.base.util.utilvalidate, so the UtilValidate class has a number of validation methods on it. And you can use the ones there or add your own.

And then a method to call within that. These methods will always accept a string coming in, so whatever the attribute object type is will be converted to a string, passed into the method, and the method should return a boolean whether it validates or not. If it does not validate then we get a fail-message or a fail-property, and one and only one of these two should be selected. And that will be used to return an error message along with the failure of the service validation. So the service engine will add that to the error messages that it returns to the calling, to whatever called the server.

So we have the fail-message and fail-property. These are exactly the same as in the simple-methods. Here you just specify a message. And here you specify a resource, a properties file, and the name of the property within that file.

Okay the override element here. All of the attributes here are basically just the same, have the same meaning as the attributes on the attribute element up here. They are just used a little bit differently. So the name is always required, and the rest of them are optional. They all have the zero to one, and so for any of these that you specify, what it will do is override that value on the existing attribute definition.

So again this is mostly useful for attributes that are created through auto-attributes, or that are inherited by the service definition by implementing another service. By the way, with this implements tag I should mention that the service that it implements, we show in this example how this service, the createProduct service, implements the interfaceProduct service. And the interfaceProduct service is a type of interface, so this engine type of interface is used specifically for these things.

But you can implement any service of any type; it does not have to be engine=interface, it can be anything.

Okay and that's it for the definition of attributes, which also brings us to the end of the definition of services.

00:41:50 screen changes to Advanced Framework Training Outline

So we're done with the service definition section. Next we'll talk about the different ways of implementing a service.

Part 4 Section 2.4

Service Engine: Service Implementation

Starting screen Advanced Framework Training Outline

Okay now we've moving on to the section on implementing services, now that we've finished looking at how to define services. We've seen a number of examples of implementing services in the Framework Introduction, as well as certain other parts of this Advanced Framework series. In the Framework Introduction the main ones we saw were simple-method services. Now we'll look at Java services, BeanShell services, as well as remote services. We've also seen some remote services, so we'll just review those examples. Java services are probably the most complicated, we'll talk about those in the most detail because of the increased flexibility in options and such there.

However you implement service, you have some basic things coming in and going out. Basically a server will always have a context or parameters coming in. And that's just a map, just a set of name-value pairs, and the return values are put in a map as well, which is passed back from the service call.

00:01:36 screen changes to Open For Business-Service Component, DispatchContext

Along with that there is also available, when a service is called, this DispatchContext object. And the DispatchContext has a number of things that are helpful for finding out about information available about other services. So you can get service definitions, these modelService objects that represent a service definition, by just passing in the service name. Related to that you can also get an XML snippet, an XML document that represents the WSDL for this that corresponds with the automatic mapping for the SOAP service, if the named service here was called as a SOAP service.

There's some other things that you'll use more commonly as well, such as the delegator to get the delegator, getDispatcher, and getSecurity. These are common things that you'll call to get objects to of course do entity engine operations with the delegator, call other services through the dispatcher, and the security object to check permissions. There are also some helpful methods here for making a valid context. You can either pass in the name of the service, or the model service if you have looked that up already or have it sitting here somehow.

You pass in the mode which is either in, out, or inout, those three different options, and you pass in the context source. So it'll take variables from this context that

you pass in, create a new map, and return it with just the valid incoming variables. So as it notes here it may not be 100% valid, as it does not fill in missing fields or anything. It just copies over all of the fields from this context to the new one, from this map in the context variable passed into the new one. And it will also do type conversions and such as those go in.

00:04:27 screen changes to Advanced Framework Training Outline

Okay so that is the DispatchContext object. And that's basically what's passed into a service when it's called. We'll start by looking at the Java services, the most basic and low level type of services to implement.

00:04:43 screen changes to CommonServices.java

When you're implementing a service in Java it follows this sort of pattern. This is the CommonServices.java file, in the common component.

00:04:57 screen changes to services_test.xml

Here's the services_test.xml file that corresponds with it, and these are the ones that I wanted to demonstrate. We have a number of examples of test services in here, and they're fairly simple cases to look at when you're just getting started. And here are various ones implemented in Java, as well as Javascript, and we'll look in a little bit more detail at the one implemented in BeanShell here. But first let's go ahead and look at some of the testJava services, things like testEntityFailure, entitySortTest, and whatnot.

As you can see in the service definition these are defined by the fully qualified classname, so that org.ofbiz.common.commonservices is what we have right here.

00:05:50 screen changes to CommonServices.java

And we have some things like the testServcie and so on down through here. This is the typical pattern that a service definition will follow. The name can be whatever you want, but the rest of it will always be the same. It should be public static, so in other words we don't have to have an instance of the object in order to call the method. It always returns a map and it takes two parameters; the DispatchContext that we just talked about, and the map of the context coming in. So that's basically what makes up a service definition in Java.

Now there are all sorts of things you'll typically do in a service. These are very simple test services so they don't do a whole lot, but one object you'll note here that is used quite a bit, and we'll go over in a little more detail, is the serviceUtil object. And this has a number of different, mostly static methods on it, that are little utility methods that simplify things when you are writing a service in Java. So these are some of the things that are a little more automated when you're using a service in like a simple-method or that sort of thing.

The serviceUtil is also very handy, of course, for services written in BeanShell, and a service written in BeanShell is very similar to this. It's almost just like taking the body out of the service and putting it into a .BSH file. But we'll talk about that in more detail in a minute, when we get to the BeanShell services.

So other things that happen. These are some transition testing services, entitySortTest for example. Here's a very common thing to do, getting the delegator from the dispatch context so we have a delegator object, and then doing a number of operations on the delegator. In this case they're just metadata operations, used for just doing a sort test, so it's really fairly simple.

And all it does is, when this service is called as log, is sorted model entities. But when you do more complex things the delegator is there, the dispatcher, the LocalDispatcher, you can also get that from the dispatch context, and all of these sorts of things.

It's very common inside a service that's implemented in Java to do entity and service engine operations, and one issue with those is that they can throw exceptions. So there are various ways of handling that. This shows one way, where we have the exception as narrow as possible. And another way which is sometimes better, but not always. is to have a larger try catch block. So for example if we took the try and moved it up here outside of the for loop, and the catch outside as well, and then to reformat this you just do this little trick.

So this would be another style where we have a whole bunch of these delegator operations going on, and if any of them fail then they all fail. And typically with this style you would then return something like serviceutil.returnerror, and then return some sort of an error message, maybe even something as simple as this. Although it's a good idea to put a little more descriptive context around it, so they know not just what happened on a low level, but what was happening on a higher level. Something like “error making a lot of visits”, or something like that, so there is some sort of context.

When you do a larger thing like this it's a good idea also to separate this out into another variable, and then use that in the error log as well as in the return error statement. We'd end up with something like this. And there's an example of a higher level one.

00:11:15 screen changes to SubscriptionServices.java

Let's move on to some better examples of this. There are a couple things I want to show. These are the subscription services, and the reason I wanted to point to these is that this is one that I've worked with fairly recently. And so we process or extend a subscription

record by product (processExtendSubscriptionByProduct). There's a byOrder one, and that one calls the byProduct one. Down here you can see we have a dispatcher.runSync, processExtendSubscription, which does individual subscriptions, and then that one is defined up here.

So we're going to use these as an example of some patterns. So one of the things that you'll want to do that's common in a service implementation is database interactions. So here for example we have some degelator operations and such, and they're all wrapped in a try and catch that closes the block. And here note that we're just catching the genericEntityException. We log the details of the error and then we return immediately with the error that was there. So this is another place that would be good to have some sort of a message on top of it to show the context, and we'll see that elsewhere here.

Another thing that's common to do is call another service from a service implementation. So this Java method that we're in right now is a service implementation, and we're running dispatcher.runsync. We'll talk a little bit more in detail about how to call a service in a bit.

So this is a natural part of implementing a service. When you do call the service and get the result back, you'll want to use something like this serviceutil.iserror method, and pass into it the result object, and then return the map created by servicutil.returnerror. Now this is the version of returnError that has the most parameters to pass into it. So we start with a general message that kind of describes the context of what we were trying to do. It's very helpful to put a productId in this case, or some sort of an Id in the message, so a user or another developer or tester that sees this message can know kind of more what they're dealing with.

Some of these other things: there's an error list, and an error map. We don't have any of those to return with this error, but we do want to pass in the result from the service that we just called. So this map right here is created here as a result of the runSync call, and it's what we use to see if it was an error, if the service returned an error. And if that service returned an error we're going to return an error here.

When we use this version of the returnError method and we pass in as the fourth parameter here the request, the parent request basically from the service call that resulted in the error that we are now returning. And what it will do is it will take all of the error messages out of this; if there was an error message map, or an error message list, or an individual error message, it will take all of those out and put them in the error message map, or for the list or individual messages in the error message list, and pass those on to whoever called this service.

So that's another common pattern that you'll see used when handling results of service calls. Note here that dispatch.runsync can also throw an exception, the genericServiceException, but we're not worried too much about that, and we're not catching that exception here, in fact we're just letting the method throw it so that the service engine can handle it.

So the service engine that called this service will just pass it up into the top level service engine call, and it will handle it just fine. So if there was some error like this “service didn't exist”, then we just let it throw something like the “processExtendSubscription didn't exist”. Then we let it throw the exception and off we go, let the service engine take care of that.

Okay another thing to note about calling services is this context right here that's passed into the service. We're using the model service, so from the dispatchContext we get the modelService object for the service name, and then on that modelService object we're calling the makeValid.

We talked about this when we were looking at the dispatchContext object a minute ago. And this we pass in the context, and actually this is a variation on the one that we saw in the dispatchContext a minute ago, because this is one that is on the modelService object. So another option is to get the modelServcie object and pass it into another dispatchContext call.

But once you have the modelService object itself you can use the makeValid method right on it, and you pass in the context and whether you want in parameters, out parameters, or the inout. So there are constants on here, in_param, out_param, and there should be an inout somewhere. It's going off the screen. Anyway you can also pass in just a string instead of using the constant; you could pass in something like inout or just in, or whatever. But using the constant is a nice thing to do I guess.

Okay so those are some of the patterns that you'll typically see when you're implementing a service in Java. Now the thing to keep in mind when you're implementing a service in Java is, this is of course the lowest level way to implement a service. If you're not experienced with Java programming this is probably going to seem fairly difficult and extreme, what all of these things mean with exceptions and all of this syntax, and so forth. So these are really meant for more complicated services that do certain things.

This service, actually it's kind of debatable whether it should be written in Java anyway, whether it's complex enough to write in Java. Because the same service written in a simple-method would be significantly shorter and less complicated, because you don't have to do this sort of explicit error processing. The simple-method stuff does that for you with the call-service operation and stuff.

So what you end up with here is having to handle everything explicitly. Which is a bit more work, it tends to be more verbose, and also, especially when you're just getting started, more error prone. But when you have complicated logic, or if you have a Java API that you need to interact with, writing a service in Java is a great way to go, and it's a valuable tool.

00:19:20 screen changes to Advanced Framework Training Outline

Okay so those are the basics of Java services. Let's now look at the BeanShell service that I mentioned before. Actually before we go onto that this reminded me of one more thing.

00:19:29 screen changes to org.ofbiz, Class ServiceUtil

Let's look at the rest of the ServiceUtil class. There are number of other messages here that are helpful.

CancelJob and cancelJobRetries. These are part of the job management, and we'll talk about that stuff later. Okay with a result object, with a server call we can get the error message (getErrorMessage) from that, get a combined error message. This getMessages is used kind of at the higher level, where we get messages from the result of a service call and put them in like the request attribute. For errorMessage we have errorMessageList, and errorMessageMap, those sorts of things. Actually just errorMessagelist, at that level there is no errorMessageMap.

Okay some other things that are helpful. The isError, we saw the use of that method, and there's also an isFailure method that can be helpful. If the service returns a failure it means that the service call failed, but it's not an error. And the difference is that with an error it will rollback, the service engine will rollback the transaction that the service is in if there is a transaction, or call setRollbackOnly on it. If it is just a failure it will just return that failure.

Okay making an error message from a result (makeErrorMessage). You can pass in a message prefix, a message suffix, and an error prefix and suffix, to put things around your messages. This is convenient in some cases but really isn't used a whole lot, I guess it's not very common to use it.

Okay the returnError method. We saw the use of a simple returnError method, this one right here where we just pass in a string with the error message, down to the most complicated one where we have this returnError. Usually when you do this you'll have a string for the error message and a map with the nested result. But there are two other parameters, the nulls that we saw before; the errorMessageList and the errorMessageMap that you can also put in.

Okay similar things for returnFailure, and then there's also returnSuccess. ReturnSuccess you can do it with no arguments, or you can do it passing back a success message.

So those are some of the common things to use on the serviceUtil object. Some of the other things are a little bit more obscure and you can explore those, but the need for them doesn't come up as often.

00:22:30 screen changes to Advanced Framework Training Outline

Okay we looked at the dispatchContext, and I think that really is all we want to cover in Java services. Now we're moving on to the BeanShell, how to implement a service using BeanShell.

00:22:49 screen changes to services_test.xml

So I guess one thing to note. You won't see services in BeanShell implemented very often, and that's why we're using this test BeanShell service as our example. And the main reason for that is, usually if a service is complicated enough to write in a Java-style syntax we might as well just write it in Java, although using a BeanShell script like this to implement a service can be a good way to do a mock-up. Some quick testing and playing around before you copy that code into a real Java method and use it.

Or of course you can write and deploy these in production and so on. But usually that's not done, because if you're going to do something that you'd write a script for you'll typically do it in a simple-method. It's just a more convenient tool, especially for implementing services.

But let's look at what this does. Basically in this test one we have a message screen going in, and a result screen going back out, so we'll see how to take a parameter coming in and return a result.

00:23:59 screen changes to BshServiceTest.bsh

So here is the BshServiceTest BeanShell file and system.out.println (print line). You can use all of these sort of standard Java objects; this is just interpreted Java basically, the same BeanShell that we talked about in the user interface layer.

You'll have a context coming in, and you can get from that context, like getting the message. Note that this is BeanShell, so a lot of this type casting and stuff is not necessary, but there you have it. Then for results you just put them back in the context, and then when the service engine calls this will figure out which were the outgoing parameters, and create the real outgoing map and use that and return that. Basically for the incoming as well as the outgoing you just use the context map to get input.

So this is a very simple service written in BeanShell.

00:25:09 screen changes to Advanced Framework Training Outline

Okay the next style of service that I want to talk about is simple-methods. We've already covered these a fair amount, especially in the Framework Introduction, so we'll just review these real quick. And we'll use the same example that we used in the Framework Introduction.

00:25:28 screen changes to services.xml

Okay the createExample service, for example; typical service definition with the default-entity-name. So the engine will be simple when we're implementing it as a simple-method. The location is the name of the location of the XML file that has the simple-method implemented in it that we want to call. That is a full resource location on the classpath typically, although as I mentioned before we are considering, especially for the simple-methods, moving over to the component:// type location that we see so often with the screen and other widgets, so that they're not so dependent on the classpath. But then the name also of the simple-method within that file to invoke. And we've seen these a bit.

00:26:36 screen changes to ExampleServices.xml

Now in the exampleServices file itself. When we were looking at the simple-methods we talked in a fair bit of detail about the different operations, and also about some of the difference between implementing a simple-method that's meant to be called as a service versus one that's meant to be called as a control servlet request event.

This service for example does not really have anything that is specific to service call except this instruction right here, the field-to-result. So we want to take the sequenced exampleId and put it in the outgoing result map, and this is how we'd do that.

Incoming parameters just come in in the parameters map, so here's the set-nonpk-fields operation, where we're taking all the non primary key fields from the parameters map and putting them on the new entity.

So that's how you handle inputs and outputs in the simple-method, just as we discussed before. Contrasting this to the subscription services that we just saw that were written in Java, calling a service is far simpler, even the setting service fields (set-service-fields) is far simpler. It's fairly verbose, but the actual complexity of this is fairly simple; all we're doing is passing in the service-name, the incoming map name, and the outgoing map-name. And that's the same map that is commonly passed into the service call.

And all we have to do here is the service-name and the in-map-name, the context going into the service. And then if this service returns an error the simple-method stuff will handle it. It will return an error if that's what you've configured it to do, which is what it does by default. But you can also tell it to not break on error, to just ignore the error and carry on.

Okay we've already covered most of the details of simple-methods on the user interface section in part 3 of these training videos, so there's not a whole lot more to cover. That's a good thing to review when you're getting into these.

00:29:10 screen changes to Advanced Framework Training Outline

Okay the last type of service implementation I want to talk about is the remote services. Now technically for these services we're not really implementing them, all we're doing is telling the service engine to call a service that exists on another server.

00:29:31 screen changes to services.xml

So when we call this service for example, which uses the RMI engine to use the Java RMI library to remotely call a service, and there's some service engine tools that are built on top of the RMI transport layer to take care of this. But basically when we call this service it's not really implemented at all, we just specify the location of where the real implementation is, and the name of that implementation at the location.

This storeEntitySyncData actually is a service that will be invoked on the remote server, and it's the implementation of that service that really matters. So this is where the actual service is defined, and the implementation we can see is written in Java. And here's the real implementation of it, the name of the method within that file.

And this is an example, as we saw before, of an implement, where all of these things that call storeEntitySyncData remotely will typically just implement that service so that the same parameters going in and coming out are used. Which is usually the pattern when you are remotely calling a service.

00:30:57 screen changes to Advanced Framework Training Outline

Okay and that is it for the section on implementing services. As I mentioned before the most complex part, the one we spent the most time on, was implementing service in Java. And that is by far the most flexible and powerful, although also the most work and requires the most knowledge and experience to use.

And that's it. The next section we'll be talking about calling services.

Part 4 Section 2.5

Service Engine: Service Calling

Starting screen Advanced Framework Training Outline

Okay we're now moving on to the section on calling services. We just finished the implementing services.

So the first thing about calling services, as we talked about in the general overview of the service engine, is that it supports synchronous, asynchronous, and scheduled service calls. So synchronous means it calls and returns, and the current thread will block and wait for the result, or in fact if it's a local method it will just run in the current thread.

Asynchronous means it will be set up to run in a separate thread, in the thread pool that we looked at in the configuration of the service engine. And asynchronous can be either in memory, so it's put on the eny memory cue, or persisted to the disk. And then other servers if there are more servers pointing to the same database, or other servers may in fact pick that up. So the server that set up the persisted asynchronous call and dropped it into the database may not be the one that eventually runs it.

The same is true for scheduled services, though scheduled services are a little bit different. Persisted asynchronous service basically means run as soon as possible. Scheduled services are run at approximately a certain time, or every so often; a certain frequency like every hour or every day or every month or that sort of thing.

00:01:36 screen changes to Framework Web Tools, WebTools Main Page

Okay we looked at the Web Tools Service Call Log. Let's review that again real quick; it's a pretty simple page. We'll be looking more at the scheduled services here. We're talking about calling services and specifically calling services in a scheduled way, so actually scheduling a job list and a thread list. Let's look at the service log.

00:01:59 screen changes to Framework Web Tools, Service Log

As you can see, since I've started up very little has been run. This will show the last 200 services that have been run. There's start and stop time, the service name, dispatcher name, and the mode for them. So just kind of some general information.

00:02:14 screen changes to WebTools Main Page

Another thing that can be handy, general information looking at this, is the thread list.

00:02:22 screen changes to Framework Web Tools, Thread Pool List

What the thread list does is shows the current threads in the invoker pool. So you can see, according to the configuration we have a minimum of five threads. Which is what we have here; threads 2-6, so five threads. We have the status of each one, the job number if there is a job number, that's the ID in the database of the job. And finally the name of the service that's currently running and how many milliseconds it's been running for, I believe is what that is for.

With this thread list you can see kind of what's going on in the application server. That could be useful for looking at asynchronous persisted jobs as well as scheduled jobs or services rather.

00:03:20 screen changes to WebTools Main Page

Okay then with the schedule job and the job list, we'll look at those in a minute to see what they do.

00:03:28 screen changes to Advanced Framework Training Outline

Let's look at calling services from a different source, different languages and stuff. We've looked at some examples of implementing these services in these different ways, like Java and the simple-method. Let's go back and review those again from this other perspective, of looking at them in terms of how to call a sevice. So let's look first at calling a service from Java.

00:03:58 screen changes to SubscriptionServices.java

This is the same SubscriptionServices.java file that we were looking at before. And basically down here is where we are setting up, calling the makeValid method to set up the context, based on this other context that's being then populated. Then we do dispatcher.runsync, pass it the name of the service and the context to go into it, and then get the result coming back.

So we're going to look at the dispatcher. The type of object this is is the LocalDispatcher, which is actually an interface. And we'll look at the definition of that to see how you use this to call services from a Java method, or this could be in a BeanShell script or other such things. Even in FreeMarker files you can use the dispatcher to call services and such.

Okay just a reminder of this pattern real quick. When we get a result back we usually want to see if there was an error in the service call. We can do that with the serviceutil.iserror method here. With the isError method we just pass in the result, and then if it returns true we know it was an error. Then depending on how we want the service that we're writing to flow, we'll typically call serviceutil.returneror to put together the error map based on a message and the nexted result.

So that's the basics of calling a service in Java. Let's look at the API and see what other options are around, since we just looked at the very simple runSync.

00:05:52 screen changes to Interface LocalDispatcher

Here's the LocalDispatcher interface. There are a few different dispatchers that implement this. We don't really have to worry about those, the actual objects that implement the interface.

So we looked at the very simplest one in that example, which is basically the runSync method where we pass in the name of the service and the context. There's another variation on runSync where we can pass the name of the service, context, transactionTimeout, and requireNewTransaction.

If we specify requireNewTransaction = true basically what it will do is override the setting that is on the service definition. So remember in the service definition we had a requireNewTransaction attribute, but we could override it on the API level by using this method and passing that in.

There's also a variation on runSync where we ignore the result, we just tell it to run. It will will still run synchronously but the result is ignored, it doesn't return the map. So the utility of that is maybe somewhat limited, but there it is.

Okay now if we want to run a service asynchronously, in other words this will set up the service run and then return immediately before the service is run. Then we call the runAsync method. And it's basically just the same as the runSync method; we pass in the service name and a context, and by default this will just go into the in-memory service processing queue.

This variation on it is the same except we also have a boolean parameter, a true or false “should we persist this asynchronous service call?” If we say true, persist, then instead of putting it in the in-memory queue it'll put it in the run queue in the database. And as I mentioned before, running it asynchronously basically means that the service engine will set up the service call and then return immediately. When it's persisted, what it will do is persist the context in the database and the service running information, and then it will return.

So when this returns you know that, and if you passed persist = true then you know that it's safely in the database, unless the transaction that it's currently running in is rolled back, and then it can be pulled from the database.

Okay another variation on runAsync. We're passing in the service name and context, and there's one with the persist parameter and without it. But the point of these ones is to pass in this GenericRequester object, and we'll talk about that in a second.

This last variation on the runAsync method has the complete list of parameters to pass in: serviceName, the ingoing context, the GenericRequester, the persist boolean persisted or not, the transactionTimeout, and the requireNewTransaction. Now when we pass in a GenericRequester we can use this object to...let's open this up in another tab.

00:09:32 screen changes to Interface GenericRequester

Because something is running asynchronously, we can use this object to get the results back. Now you can implement your own GenericRequester, this is just an interface, but typically what you'll do is use the GenericResultWaiter.

00:09:57 screen changes to Class GenericResultWaiter

So when you create an instance of the GenericResultWaiter you can just create a new object; you don't need to pass anything to the constructor.

00:10:09 screen changes to LocalDispatcher

And then you keep a handle on that object and you pass it into the runAsync method, either using this most simple form with the serviceName, the context, and the requester, or the GenericResultWaiter, all the way up through this most complicated form right here.

00:10:25 screen changes to GenericResultWaiter

And then you have an object that you can use to check and see, you can call isCompleted over and over. So for example you could have a while loop like while-not isCompleted, and inside the loop just wait for half a second or something. You might want to wait longer if you know it's going to be a very long lived service.

So that will be executing in a separate thread, but the current thread can sit there and wait for it. Which is basically running it synchronously, right? Except you can do some interesting things with this. So say you have four services to call, and they can all run in parallel, or they don't depend on each other. Maybe they're all running on different servers in fact. So what you'd do is set up four GenericResultWaiter objects, one for each service you're going to call.

00:11:19 screen changes to LocalDispatcher

And then call the runAsync method, and pass in the serviceName and context and GenericResultWaiter object for each one, so you basically have four calls to runAsync. They will all run pretty quickly, and get down to the bottom.

00:11:40 screen changes to GenericResultWaiter

And then you can call the waitForResult method on each of the GenericResultWaiters, and basically what you'll do here, what this will do is that busy waiting that I talked about where it waits a certain amount of time. If you want to specify the amount of time it waits you can use this version of the method and tell it how many milliseconds to wait to check the status of the asynchronous service call.

Anyway with the four GenericResultWaiter objects we just call waitForResult, which returns a map that has the result of the service call, and we just call that for all four of the GenericResultWaiters, just one after the other, bam bam bam bam. So we set up all four map variables.

Now by the time the last one is finished then we know of course all four are finished, and we have the results from all four of them. But instead of the time of A+B+C+D for the four different service calls we did, each with its own GenericResultWaiter, the time it takes to run all four of them is only the time of the longest one. If B was the longest running one it takes the time of B, so we basically get the time of A, C, and D for free. And that's one of the things you can do with these asynchronous calls, and the waitForResult and the GenericResultWaiter.

So that can be a nice pattern to use when you have a fairly large set of things to do that can be split out to different servers, or different threads of the current server or that sort of thing. And things can typically run significantly faster using that sort of a technique.

00:13:38 screen changes to LocalDispatcher

Okay another variation on this. Instead of generating the GenericResultWaiter and passing it in to this version of the runAsync method, you can just call runAsyncWait with the serviceName and the context, and get the GenericResultWaiter back. So with that four service call thing I was talking about before, you call this four times, get the four GenericResultWaiter objects back, and then we call the waitForResult method on each of those and get the result map back.

00:14:04 screen changes to GenericResultWaiter

And there we have it. With the simple eight lines of code we have something that will give us the efficiency of taking only the amount of time of the longest service to run.

00:14:16 screen changes to LocalDipatcher

Okay we talked about runSync and runSyncIgnore. Now let's talk about scheduled services. There are various different ways of setting up scheduled services, so we're just going to talk about this from the perspective of running it from Java with a LocalDispatcher. This is kind of like an asynchronous persisted call; in the simplest form we just pass in the serviceName, the context, and the startTime.

00:15:00 screen changes to LocalDispatcher, schedule

The startTime is a long here. This doesn't show a lot more information, but basically this is a long.

00:15:11 screen changes to LocalDispatcher

But the best way to do it to get that long value is to use the timeStamp or the date-time objects, these sorts of things like the javaUtilDate or the javaSqlTimeStamp, and set up the date and time you want it to start. And then each of these objects has a get-long method on them, to get a long number that represents that start time, and then you can pass that into this schedule call.

Okay you can also do, again serviceName, context, startTime, pass in the frequency, the interval, and the count. So let's go ahead and jump down to the more complicated ones.

The endTime. Here's the one that has all of the parameters. These other ones are just variations, or just different combinations of the parameters for convenience.

Okay so in the most complicated form we can pass in the poolName. When we're looking at the serviceengine.xml file we were talking about the different thread pools, and how each applicaton server can configure its thread-pool in the servicengine.xml file: which pool to send to by default, and which pools to listen to.

So with all of these other methods they will use the default pool, but if we call this one and pass in the poolName, that's the pool that the scheduled service will go into. And then of course that can be read by the current server or a different server or whatever, however you want to set it up. We have the serviceName, and context, startTime same as we talk about.

Okay frequency and interval. These two together are used to specify things like once every day, once every two days, once every four hours, these sorts of things.

00:17:22 screen changes to LocalDisaptcher, schedule Parameters:

I believe somewhere we have some constants for this. Okay here we go, this thankfully refers to it. The frequency variable, if we look at the recurrenceRule object, let's see if that is in here.

00:17:43 screen changes to RecurrenceRule

On the recurrenceRule object we have these different constants for minutely, monthly secondly weekly and yearly. So we have all of these different frequencies.

00:17:58 screen changes to LocalDispatcher

And then along with the frequency we pass in an interval. Interval just specifies how many of the frequencies we want. So this would be two months, or four for intervals, and then hour we'd use the hourly constant for the frequency here that is on the recurrenceRule object.

Okay account as it shows here is the number of times to repeat the call. If it's -1 then there will be no count limit.

EndTime, the time in milliseconds when the service should expire. This is like the startTime; it has the time in milliseconds using the Java standard time represented as a long, that can come from a java.utli.date object or a java.sql.timsetamp object. So both of these can come from those objects, that's the most convenient way to put them together really.

Okay then the maximum number of times to retry (maxretry) should the service call fail. This value just like some of the other ones we've talked about. There is an attribute on the service definition for the maxretry, and this will simply override that value on the service definition, if there is one on the service definition.

00:19:25 screen changes to LocalDispatcher

Okay and that's the basics for calling a scheduled service from Java. The API is fairly simple, basically all these parameters will result in something being dropped into the database. You can also drop something into the database directly through seed data or your own application or whatever, although the easiest way to do it is just to call these scheduled services if they meet your needs.

00:19:54 screen change to Advanced Framework Training Outline

Okay so these are all the different ways of calling the services from Java. We'll talk in more detail about the scheduled service in a bit, review quickly the calling from Java as well as setting them up in other ways.

Okay but moving on to the calling from section here. Let's look at calling from a simple-method.

00:20:20 screen changes to ExampleServices.xml

So in the ExampleServices here, the one we were looking at, there's this example as part of the create example service, of calling the createExampleStatus service through the set-service-fields and call-service.

Another variation, another way to call a service, you can call a service asynchronously here using the call-service-asynch operation. It's very similar to this one; you specify the service-name, the in-map-name, and include-user-login, just like you would with the call-service method here. So it's fairly simple, it just results in an asynchronous service call instead of a synchronous.

So that's really just a review of what we covered in the section on the simple-methods. And so really calling a service from a simple-method is very simple. It's just a fairly natural thing for that, and that's pretty much all there is to talk about there.

00:21:37 screen changes to Advanced Framework Training Outline

Okay calling it from the control servlet. We have seen some examples of this in the Framework Introduction when we were looking at the example application. That's how they were all invoked.

00:21:50 screen changes to controller.xml

So for example, in the createExample request we have a request event here defined, type-service, and invoke is just the name of the service. And the path by the way we don't have to include here; either way it's left empty.

So basically all you do is specify the name of the service to call. It'll automatically map the input parameters, and will do certain things like the map prefix and the list suffix. If you want multiple fields from the form that's being submitted to be passed to a single incoming attribute to the service, then you can use those as we talked about in the service definition area. But in general it's that easy.

The other variation on this is the service multi. And we talked about the service-multi in a fair amount of detail when we were talking about the Form Widget and such, so that would be a good place to refer back to. But the basic idea of the service-multi, as opposed to the service, basically the service-multi will have the forms set up in a special way.

So instead of calling the service once, you can have a table of forms that are all combined into one form so it has a single submit button. And then basically each row in the table will have a special suffix on it to tell it which row it's part of, and each row will then be split out from the rest and passed into the service for the input context. So it'll call the service once for each row, that's basically the service-multi thing.

And again for more details on that the best place to look is probably in this Advanced Framework course, at the Form Widget section on the multi-type forms, because with the multi-type form it does all the special preparation and such for the service-multi type here.

00:24:06 screen changes to Advanced Framework Training Outline

Okay so that's basically all there is to calling a service through the control servlet; you just map it to be called and it takes care of the rest.

Okay HTTP, SOAP, and other remote clients. We've looked at one variety of this where you have one service engine talking to another service engine. But there's also a way for remote clients to call services by just doing an HTTP request or a SOAP call to the server, that is not going from a service engine to a service engine, but just an arbitrary client to the service engine.

00:24:51 screen changes to controller.xml

And it goes to the service engine for both of these; we can see how these are set up. Here this is the controller.xml file for the webtools webapp, and we have an HTTP service request and a SOAP service request. Now for both of these one thing to keep in mind is that in the service definition the export-true flag must be set. So the nice thing about that is any services that you don't want to be available through this sort of fairly arbitrary external calling, all you have to do is leave the default on export, which is false, and then if you try to call the service through these it will treat it as if the service doesn't exist.

00:25:55 screen changes to HttpEngine.java

Okay this is the the HttpEngine.java file. There are various parameters you can pass in. What is the one that it calls? (changes to controller.xml then back) On that class it calls the httpEngine event, so the httpEngine method right here is the one we're looking at.

So there are a few parameters to note here. The serviceName, just the standard serviceName, so you'd have a parameter something like serviceName equals something like createProduct or createExample or something. Servciemode, and then xmlContext, or the service context. Now the xmlContext is meant to be a little XML snippet which typically you would not be able to pass on a URL parameter; this would have to be part of a form submission in essence, is the only way you'd get that context in there.

The serviceMode is either sync or async. As you can see here, if you don't specify a serviceMode it will default to sync. But you can also specify async, like that, to result in an async service call.

Okay so basically when you do it, you could do it through a browser, you could do it through any sort of HTTP client in another programming language or whatever.

00:27:37 screen changes to controller.xml

But basically you just have the location of this request as the URL, and you pass in those parameters: the serviceName, the serviceMode, and serviceContext optionally. And then you can pass in any other arbitrary parameter, and those will automatically be mapped as well I believe.

00:28:03 screen changes to HttpEngine.java

And then it will return the result, actually I don't know that it does, yeah really the only way to get anything into the context here is to put it in this XML block, the serviceContext block.

00:28:41 screen changes to EntityScheduledServices.xml

And this is serialized. So for an example of what this looks like, when we have a scheduled service it has the same sort of serialized XML block, so this is what it would look at. This is in the entityScheduledServices.xml file if you want to look at this in more detail.

00:28:52 screen changes to HttpEngine.java

And the class that does this is xmlserializer.deserialize. There's aslo an xmlserializer.serialize. So after it does the service call it gets the result back, and it serializes the result here and it returns that in the HTTP response in the resultString here. You can see where it's setting the content type and writing this response back out.

Okay and that's pretty much it for using the HTTP engine to call a server. So for any programming language including Pearl, Python, whatever, even another Java application or something, you could use this HTTP method of calling a service, and that would be an alternative to using a SOAP client. So instead of the SOAP envelope basically the only thing you'd really have to do, if it's not in Java, is to write something to serialize and deserialize in the way that the xmlserializer does. But it's fairly simple, so you can hack something together without too much trouble typically.

00:30:21 screen changes to EntityScheduledServices.xml

Okay with the SOAP service it's a little bit different. This is the target URL for calling a service from SOAP, and the SOAP client will just basically package everything into the envelope, including all of the parameters to pass in, the name of the service to call, and all of this sort of stuff. And that gets passed in.

And then this, the SOAPService. Basically that's just the URL to send it to if you're using HTTP for the transport mechanism for SOAP. You can use other mechanisms; I don't know if we have a JMS listener set up for receiving SOAP calls that way, but that's certainly something you could set up as well. But anyway this is basically how you would call a SOAP service.

00:31:10 screen changes to Advanced Framework Training Outline

Now I'm not going to go into detail with that here, as in the last section here we'll talk more about calling or service engine services from a SOAP client, and calling SOAP services through the service engine from an OFBiz application.

00:31:40 screen changes to LocalDispatcher

Okay so moving on to the next section, scheduled services.

So we looked at calling them from Java through the LocalDispatcher interface. The schedule method here

where we can specify the pool name. Then the serviceName, the context, the startTime, the frequency constant; daily monthly hourly, or whatever the interval or number of intervals of the given frequency, and these by the way can be -1 to ignore them. Then the count, the number of times to run, and endTime, max number of retries (maxretry), and so on.

And so that's basciclaly calling it through the Java API, and all this really does is take all this information, serializes the context, and drops it all into some records in the database.

00:32:31 screen changes to Advanced Framework Training Outline

Now the next style of calling a service that I want to look at is from an imported entity XML file. And we actually looked at one of these in the section about the EntitySync tool that's part of the entity engine extensions.

00:32:50 screen changes to EntityScheduledServices.xml

So let's look at that file again, to see in more detail what this is all about. Here's an example of a scheduledService that's fairly simple. We have the name of the service, when to run, and if it's in the past it will run it as soon as it picks it up, as is the case there. Then the pool to put it in, and the user to run as.

And finally a recurrenceInfoId, that points to a recurrenceInfo record with that given ID. So this has the startDateTime, the recurrenceRuleId and the recurrenceCount. This recurrenceCount just keeps track of how many times it has recurred, so you can set it to zero or let it initialize by itself. And then of course this refers to a recurenceRuleId.

00:34:08 screen changes to LocalDispatcher

For that ruleId we have an untilDateTime. So we have the startDateTime and an untilDateTime that correspond with, here in the LocalDispatcher, the startTime and endTime.

00:34:19 screen changes to EntityScheduledServices.xml

Frequency, interval, and count are all configured here. Frequency equals “daily” for example. Then intervalNumber and countNumber. CountNumber in this case is -1, so there will be no limit on the count. And intervalNumber and frequency, take these together to mean that it will execute once per day.

In this example it's every two minutes, so not twice per minute but every two minutes is what that means. And again this has no restriction on the count. So basically that scheduledService just drops this kind of stuff into the database; each job is given a jobId and then off it goes, and the service engine picks them up and runs them.

Here's another example. This is the EntitySync example, where not only do we have recurrenceRule and recurrenceInfo, but we also have runTimeData that's referred to in the JobSandBox record. So this has a JobSandBox record, it just doesn't refer to any runTimeData so the runTimeDataId is null. But this does refer to a runTimeData, and here's the record here with the ID and the runTimeInfo field that has all the details. So when we're in an XML file this is basically what we see. We put it into the CDATA so we don't have to escape everything.

And then from here, all the way down to the close of the CDATA, basically all of that that's selected now is the XML snippet that represents the context for the service call. So if you were using that HTTP service, that event that we looked at in webools, this is the same format you would use. This is the format of the OFBiz XML serializer object.

So if you want to drop something in the database that the service engine will pick up and have a context to run with, you can just drop this kind of stuff right into the database. Here we have a map with a single entry, the key is this, the value is this, and so it's a very simple context that we're passing in.

And that's basically how these work on a low level. So you can take advantage of that to get things going by putting things in the database, and without going through the service engine to get the scheduled service set up.

00:37:08 screen changes to Advanced Framework Training Outline

Okay the last way we want to look at, and we'll actually look at some examples now, is the webtools service job screens.

00:37:19 screen changes to OFBiz.org, WebTools Main Page

So in the webtools webapp we have all of these service engine tools. From the schedule job screen (changes to Framework Web Tools, Schedule A Job, and then back) we can schedule a job to run, and we'll walk through that in just a second.

00:37:34 screen changes to Framework Web Tools, Scheduled Jobs

And the job list shows jobs that have already run. So if they have a finish time and a start time then you can know that they have already run. If you look in the database you'll also see a different status on all of these jobs.

This is the job name and the corresponding service name. When we schedule a job through CDATA, through an XML file that we just looked at, we can put a job name on it like this. When we schedule through the user interface it will just be given a number basically. For jobs that are scheduled to run but have not run yet, those all show up at the top here, and you can click on this link to cancel the job before it runs the next time.

There aren't any jobs running right now, and hopefully we'll catch one when we run it manually. But when a job is running you won't be able to cancel it. It's only when there is a job that's scheduled to run sometime in the future. And we can see the scheduled run time here in the run time column, the date and time for it, and if it's very far in the past it usually means that it's just barely been put in the database, and the service engine hasn't picked it up to run it yet.

And this also shows the pool that the job is in, and that's convenient as well. So that's basically what this page does.

Let's go ahead and take a look at scheduling a job. Let's look at one of the services that we've been playing with...I don't think I still have the test. Let's open that up and look at services_test.xml.

00:39:31 screen changes to services_test.xml

Okay so here's the test service. All it does is, we take a string in, a message in, and it returns a response string out. And I believe this log a few things to the console, so we'll watch the output here (flashes to the console and back) as we run it. Let's go ahead and throw a few enters in here to leave some white space, so we know about where we are.

00:39:55 screen changes to Framework Web Tools, Schedule A Job

Okay so the service name, testScv. If we leave all of these things in the default values then it will default to just run once. Or we could have it run once every minute, make it fun, or why don't we make it run every ten seconds, and we'll have it run ten times. Actually let's do it once every five seconds, or three seconds, so we don't have to wait so long.

00:40:26 screen changes to Schedule A Job, Step 2: Service Parameters

Okay so we submit that, and then this screen will automatically look at the incoming parameters for the service, and show a little text entry box for each one. This is not necessarily the easiest way to get data in, but it is one way. For complicated services that require a list or a map coming in or something, you're out of luck, you can't really use this user interface as it is right now. But if it's just a string or a number or a date or something then you can use it.

00:41:00 screen changes to Framework Web Tools, Scheduled Jobs

Let's just throw something in here, submit it, and now our job is scheduled. So we can see an example. When this screen was rendering this service had started, and it has not yet finished. And so we can see that's one of the cases where there is no cancel job link.

Now if we refresh this (refreshes), so now we see that it has successfully run once and completed, and it's waiting to run again. If we hit cancel now, actually we're probably too late to hit cancel because the interval is too small. Every three seconds this should be running again, so we see another one there.

00:41:39 screen changes to Console

So if we look at the console here. Okay there goes another one that just ran. Let's leave some space, and we should see in just a minute the stuff that's logged. So this does show kind of some debug stuff for the scheduled services, actually, and then in the service implementation itself this is the log message that's coming out of it: “This is the test message.” Service Test: “This is the test message.”

Now it doesn't have the normal debug header like these things: the date and time, and the name of the thread and stuff, because this used the system.out.printline method instead of the debug.log method.

00:42:31 screen changes to Framework Web Tools, Scheduled Jobs

So eventually this'll run through all ten times that we restricted it to, and we'll see all ten of those calls coming through here. And here's the job name, just kind of a randomly generated number that is given for the name when you don't specify one. So that's how you use the schedule job user interface and webtools.

00:42:54 screen changes to Framework Web Tools, WebTools Main Page

And that's another way to get jobs scheduled to start up, to get running. So if you just want to run a service real quick, this can be handy for that.

00:43:05 screen changes to Framework Web Tools, Thread Pool List

It's also handy for other things like if you want to set up a service to run monthly you can just do that right there. Although usually for things like that you'll want to do it as part of something like CDATA so you'll have a little bit more control over it. And if it's in an XML file then you'll have revision control for it, so any new systems that are set up will have that going into it and stuff.

Okay I'm trying to hit refresh and see if we can catch one of these in action and see one of the threads with

the job actually running, but they may have already all finished. Oh we did catch one, here we go.

So here's the running status, and this thread is the one that picked up the job. There's the running status, there's the job number, the name of the service, and the time. So that's what that looks like when th re's actually a service running in the thread pool.

00:44:09 screen changes to Advanced Framework Training Outline

Okay and that's it for calling scheduled services. Now the last bit here in the calling services section is about the service engine RemoteDispatcher. Now what this is used for is if you have another Java application that wants to communicate with an OFBiz server, but is running in a different JBM, possibly on a different server itself on a different box, then what you can do is get a RemoteDispatcher from the service engine in OFBiz. And when you call services on that RemoteDispatcher it will execute them on the OFBiz server, but when you're calling them through the RemoteDispatcher you can treat them as if it was just a local call. So you can pass parameters to it; it's basically very similar to the LocalDispatcher interface that you're calling services on.

So to do this basically we look up the RemoteDispatcher in JNDI, and then the actual underlying communication is done through RMI. So if you have an issue with RMI and firewalls on your network, for either the other server or the OFBiz server, then this may not work. So RMI calls do have to be able to get through.

Okay the rmi-dispatcher is the one that's loaded here in the ofbiz-containers.xml file, and we've seen some examples of that. To see how to use this, there's some good comments and sample code right here in this exampleremoteclient.java file. So let's go take a look at that real quick and walk through a bit of it.

00:46:16 screen changes to ExampleRemoteClient.java

Okay so first of all in order to use it in your application, if you have a separate Java application, you'll need the following things from OFBiz on the classpath, all of these different files. Notice this last one, ofbiz-service-rmi.jar. When the jar file is built it's called .raj so that it's not deployed, so you'll just copy it and rename it to .jar instead of .raj.

Okay and then there are also a few third party libraries that need to be on the classpath. If these are already on the classpath, if you're using them in your application, then great. But if not then they need to be added.

Okay once you get all those things set up, basically this is what you can do. You'll have a RemoteDispatcher object, this is in the service engine API, it's very similar to the LocalDispatcher, it just has methods on it you can call.

00:47:37 screen changes to Interface RemoteDispatcher

We can look at the Javadoc for it real quick here, RemoteDispatcher, here we go. So the RemoteDispatcher has these same sort of runAsync methods, the runSync methods, and the schedule methods. It has all of these. The ones that it does not have, that the LocalDispatcher has, are the lower level methods for managing the service engine.

00:47:54 screen changes to LocalDispatcher

So the LocalDispatcher does have some of these methods like up here: addCommit, addRollback, deRegister, getDelegator all this kind of stuff. That's only on the LocalDispatcher.

00:48:04 screen changes to RemoteDispatcher

On the RemoteDispatcher, though, you have all the methods you need for all of the varieties of service calls, even scheduling services to run. So synchronous and asynchronous and everything. You call it as if the service engine was right there running in your application, even though when you do these calls all it does is result in a remote call done over the network on the OFBiz server. And then the results are passed back to you.

00:48:33 screen changes to ExampleRemoveClient.java

Okay so back to the code snippet. Here's a very simple example of doing a JNDI lookup, just use the naming object, .lookup, and then the URL. Here's the default URL. This is actually referring to LocalHost so you usually want to change this. This is just example code by the way; it's not meant to be run as is technically, although it does work run as is.

But it's meant to be basically copied over into your application, and fit into however you want to do things working with the RemoteDispatcher. So it's basically just this simple. You do a naming.lookup of the RMI URL, and this is the location of the RemoteDispatcher that's set up in the ofbiz-containers.xml file.

00:49:22 screen changes to ofbiz-containers.xml

So it corresponds with this value right here, that's the binding name; the rmi-dispatcher. That's basically where that's set up.

00:49:35 screen changes to ExampleRemoteClient.java

These are just standard exceptions by the way: the NotBound, MalformedURL, and RemoteException, these are standard exceptions for the JNDI naming object.

Okay once we have our RD object, or RemoteDispatcher object, we can just do things like we would

normally do on the thing. Here's an example of rd.runsync, passing the name of the service and the context of the service, and the context here is very simply that message parameter. So this is the same service that we were using in our example of the webtools service scheduling. And this is something you can put in your own application, this service call, and then the service will run on the OFBiz server and the results will come back to you.

So this little runTestService method, I guess if we want to do the full loop here, sets up the exampleRemoteClient, creating an instance of the current object, of the exampleRemoteClient object. When that's initialized, in the constructor is where it's doing the JNDI lookup with the RemoteDispatcher. So once this is created we can just run test service (testscv), get the result back, and this is just using systemout.println (system out print line) to print out to the console the result that we got back. So it's really that simple.

Okay some general ideas about using this. A lot of the resources in OFBiz, pretty much everything, uses services for making changes to the database and such, but only in a few cases are services used for searching and data retrieval. If you have a separate application where you want to get data from OFBiz, the best way to do it is even better than getting it from the database that OFBiz runs on or any of these sorts of things.

The better way for a few reasons, especially if it's fitting into a larger business process, is to write a service that prepares the data and deploy that service on the OFBiz server, and then call that remotely using the RemoteDispatcher. And then in the context that comes back from the runSync, you'll have all of your data and you can do what you need to with it. And then in that way you can also offload some of the processing, data preparation stuff, to the OFBiz server, rather than running it in the client application.

Okay so that could be used for other server applications, it could be used for a Java desktop application, it could be used in an applet that's running the in the browser, all sorts of different things to do these remote service calls.

00:52:41 screen changes to Advanced Framework Training Outline

Okay and that is basically it for this section on calling services. In the next section we'll talk about service groups.

Part 4 Section 2.6

Service Engine: Service Groups

Starting screen Advanced Framework Training Outline

Okay now we're moving on to service groups. The last section we covered was calling services. Service groups are really fairly simple. Service group just defines a group of services or kind of a list of services that can be called in different ways according to the group definition. So we'll see how a service group is defined and also how a service is defined, so that when you call that service it calls the group.

Okay the group XML file. The structure of it is definid in this XSD file, service/dtd/service-group.xsd. In a component, this is where you'll typically find the group definition files, in servicedef/groups*.xml. Okay now let's look at a service definition for a group and then how to define a group.

00:01:05 screen changes to services_test.xml

So a service definition for a group. We're looking at the services_test file; these are the test services in the common component. Here's just a simple example to give you an idea of what the groups are like. This is how we define a service, and of course all services have a service name. The service engine will be group, and the location is the name of the group. And that's really all there is to it.

You can specify other additional attributes like the authorization and all this kind of stuff, but for the most part basically we just have the end of the group. You can also define attributes coming in to it and that sort of thing, but it's not really required.

00:01:54 screen changes to groups_test.xml

When we look at the definition of this group, our testgroup, it has two services in it: the testScv (test service) and the testBsh. So test service is the one that we looked at, that we called through the web interface and such. And testBsh is the BeanShell example service that we looked at. When this service group is called it's going to call this service synchronously. And when that finishes then it will call this service synchronously. It can do other modes like the asynchronous, and it also has options for certain other things.

We'll look at the details of all of the attributes in a bit when we look at the quick reference book. Okay one of the important things here is the send mode for the service. There are a few different options. This is kind of like the operating mode for the JMS servers, where we have some of these different options for either handling fail-over or for handling load balancing types of things. So randomly distributing among the service, or round robining, between them, or calling the first available going through the list, or finally calling all of them. If you call all of them, which is the case in this one, then it will just go down through the list and call them.

00:03:17 screen changes to OFBiz Quick Reference Book Page 3

Okay let's look at the quick reference book. On this same page, page 3, we have the service groups here as well, and the definition for them is in this box. It's a pretty simple file; we have a service-group root element and then one to many groups underneath it. Each group has a name and a send mode.

Let's look at the send mode. The default is all. If you select none, by the way, then it won't call any of the services. It's a handy little way to turn off the group. The all mode will call them in order, so each service element it will call those, and then when that one's finished it will move on to the next one.

First-available will go through the lists of services under it. If the first one succeeds then it stops there. If the first one fails it goes to the next one and keeps on going through them until it gets to the end or finds one that succeeds.

Random. Basically every time you call the group it will randomly call one of the services. And round-robin every time you call the group it will cycle through the services, remembering the one it called last time. So it will cycle through the services and call all of them an equal number of times, just looping through them over and over.

The first all is basically if you're implementing something that pulls a bunch of services together and calls them all. That would be an all group. The first-available group is something you'd use for fail-over types of situations, such as maybe you have services that maybe are doing the same thing on a number of different servers, and you want it to just run on the first available one.

This is basically a fail-over list, the first server being the primary one and the other ones being backup servers, and so on. Random and round-robin are for cases where you could have the service in the group representing a number of different servers, so they each do a remote call to a different server basically. So the random and round-robin modes are just ways of distributing the load somewhat evenly among those servers.

Okay each service basically just has a name, a mode, whether it will be called synchronous or asynchronous, and result-to-context true or false, by default false. If you have services in the list, especially in the all mode where all of the services will be called, and one of the services depends on something in a previous service in a list, then in that previous service you typically want to set the result-to-context to true. You do that so that after that service runs and it gets down to the service that depends on it, the output from the first service will be available for the second service.

00:06:34 screen changes to Advanced Framework Training Outline

Okay and that's pretty much it for service groups. Like I said they are fairly simple. So utilyt of them they can be used to combine different services together. Although usually the best way to do that, to compose services together, is to just create a new service, a simple-method that calls the other services.

And the nice thing about that is you can interleave other logic or validation or whatever. It's a little bit more flexible then using a service group. Service groups are also useful though, for distributed things where you're doing service calls to various things.

They're also useful for the distributed cache clearing, if you're not using JMS but instead just using the service engine transport protocols like RMI or HTTP. Then with a service group you can have one RMI service, for example, representing each server in the pool.

And in the service group you just have it so when you call the service group you set it up to call all those services. When you add a server you just add a new service definition and then add that service to the group. So that's another way they can be useful.

Okay so that's it for service groups. In the next section we'll talk about the service ECA rules.

Part 4 Section 2.7

Service Engine: Service ECA Rules

Starting screen Advanced Framework Training Outline

Okay now we're onto the service Event-Condition-Action, or ECA, rules. These are fairly similar to the entity ECA rules, covered in the entity engine section. The service ECA rules were actually implemented first, and the entity ones based on them, based on the concept of them and such.

And the service ECA rules are really the more important ones. these could be used for the a number of different things, but the main purpose is to tie high level business processes together. So as one thing happens in one area of the applications these rules can sit and watch for it, and when that happens trigger something else that happens elsewhere.

They're also very good for customization and building your own logic that runs on top of and along with the out of the box OFBiz logic. So if you want something special to happen every time a payment is received or a product is changed or something, then you can set that up, write the custom code, custom logic, and have it triggered through an ECA rule.

So ECA of course stands for Event-Condition-Action. The events are basically in this context, business events. Or they can be fairly technical; it depends on the nature of the service that they stem from. But basically as a service executes there are different events

that occur in the life cycle of executing the service, and on each of those events you can attach an ECA rule so it'll be triggered on an event. It can have conditions, and if all the conditions are true then all the actions will be run. So it's fairly simple.

The schema file for the structure definition for the XML file is here in the service component, in the dtd/service-eca.xsd file. Inside a typical component you'll find the service definitions in the servicedef directory, in the secas*.xml files. Okay so when you define ECA rules, for each rule you define the event it's associated with, the conditions, and the actions.

00:02:40 screen changes to secas.xml

Let's look at a couple of examples and then we'll look at the quick reference book to see the details of what all of the different things mean for them.

Okay here is where were looking at some of the subscription stuff earlier, mainly the processExtendSubscriptionByOrder service, and this is where that service is called. So there's a service called changeOrderStatus that's called whenever an order status is changed when that service is called before the transaction commit is done. But it will not run if there is an error in the service.

These conditions will be evaluated, so it checks to see if the statusId = order_approved, the orderTypeId is equal to sales_order, and the statusId is not equal to the oldStatusId. In other words we're not only making sure the status is equal to order_approved, but that it has changed to order_approved. If all those three conditons are true then it will run these two actions: it will call this service updateContentSubscriptionByOrder. and processExtendSubscriptionByOrder. These are both run synchronously, as selected here.

So that's basically that simple. As you can see these are very high level compositions, triggering other processes on things that happen in more some core processor or something. Okay so that's the general idea there.

00:04:22 screen changes to secas_shipment.xml

Here's another one that is an interesting example. This is also on a status change, so we look and make sure the statusId is not equal to the oldStatusId, and that the new statusId is a shipment that's packed, and that the shipmentTypeId is a sales_shipment. And then this is where, once a shipment is packed, we want to create the invoices for that shipment. So there may be one or many invoices actually.

And this comes down to a business rule that is actually a very common requirement for accounting, as well as different fulfillment practices and such. The rule is that when you're selling things the invoice is created once the shipment is packed. So that's literally what we're representing with this rule.

The updateShipment is the source service, or the one that we're triggering from, before the commit is done on the transaction for that service. And if the conditions are met these are the services we're calling to trigger off the other processes.

00:05:41 screen changes to Framework Quick Reference Book Page 3

So having looked at some examples, let's look at the details of all of the fields here. There's some more examples in the file right here, and you can look at them for other things. Here's actually the updateShipment example. It's maybe a little bit more brief. So that can give you an idea of how those might be used.

I guess this other example right here is interesting; maybe I'll mention this real quick. Because unlike these other two that we looked at it's not something that triggers on a service change. Any time we create product content, before the input validation, we check to see if the contentId is empty, and if it is then we run the create-content-synchronously.

So basically what this is doing with productContent, if we're creating it and not passing in a contentId, then we want to use the information that's passed into the service call to create productContent, to create the content in the database. And this service will return the contentId, which will then be available for passing to the createProductContent service.

Now the interesting thing about this is that this is done before the in-validate, so before input validation I guess is maybe a better way to phrase that. Which means basically we're doing what's necessary to create the missing contentId and slip it in there before the inputs for this service are validated.

Okay as we saw in the service ECA file, the root element is a service ECA, and then there are one to many ECA elements. Each ECA element represents an ECA rule. We'll always have the name of the service that we're triggering on, and we'll also always have the event.

Let's talk about these different events. In the life cycle of a service we talked about the transaction related things. Let's skip these global ones for a second and go down to authorization.

So the first thing that happens is we check the authorization. If authorization is required then we check to see if the user is logged in and such. So if we set the event to auth, before the authorization is done, whether it's required or not, before that life cycle of that part of the service execution cycle is done then the event will be triggered. Next we have the input validation, then we have the actual invoke of the service, then we have the

output validation, then we have the transaction commit. And finally the return.

So with each of these stages in the execution of a service, note that when you say that event=return it means it will go before the return. If you say that event=invoke, then it's triggered just before the service implementation is actually invoked by the service engine.

With the commit, whether you have use-transaction set to true or false on the service, so whether it actually has a transaction in place or not. Also if the transaction was already in place so it didn't start a new transaction, it won't actually commit. But this phase will still be crossed in the life cycle of a service execution and so the rules will be run; that's how all of these work.

Now global-commit and global-rollback are a little bit different. These are some special case things that are a little bit fancy and very handy. Basically with the global-commit and global-rollback these will only run if the service is part of a transaction, and when the transaction it is part of successfully commits, or in the case of an error rolls back, then the global-commit or global-rollback will be done.

So it may be that this service call finishes and there are a hundred other service calls that are done, and they're all part of the same transaction, and eventually the transaction succeeds and commits. As soon as that happens then this global-commit will be done.

If it rolls back then the global-rollback will be done. Now the way these work in the process of actually doing a commit or a rollback, the transaction manager will look for these sorts of things to do that we have registered, before the rollback is actually finalized. I take that back; in the case of a rollback it's after the rollback is actually finalized, so the transaction will be done, it will be rolled back, and we go on.

In the case of a commit, with the transaction manager that we use, the JTA implementation, it is a distributed transaction manager, so it does a two phase commit. So these global-commits will be run between the two phases. When the first phase completes successfully then these will rerun before the second phase of the commit is finally done.

So those are the different events that you can trigger on. And then we have some run-on-failure and run-on-error attributes. By default they're both false, so if you want the ECA to run anyway in spite of a failure or error then you can set them to true. Now as I mentioned before in this service section, a failure basically just means that it's the same thing as an error, except it doesn't represent a case where a rollback is required. There's was just some failure that happened.

For each event we have a number of any combination of these; condition and condition-field, zero to many of them. And then we have the actions. The conditions will be run in order, and in order for the actions to run all the conditions must evaluate to true. And then if they all do, all of the actions will be run, and there can be one to many actions. So note that you don't have to have a condition, you can just trigger actions immediately from the event.

Okay condition is very much like the if-compare in the simple-method, so we have a map-name and a field-name. The map-name is not required so much anymore, as you can use the . syntax to reference a map right in the field-name. An operator just like the operators in the simple-methods and widgets and so on. And we're comparing this field to a value, as desired you can specify the type that both should be converted as needed too, and for certain types there may be a format string you need to specify along with it.

Condition-field is the same as condition, except instead of comparing field to a value you compare field to another field. The other things, operator, type, and format, do the same thing.

Okay for each action that's part of the ECA rule, and action in this case is just a service, we specify the name of a service and the mode to run it in. These are both required, they're the only two required attributes. So you specify sync or async, for synchronous or asynchronous. If you do specify asynchronous you can set persist = true; it is false by default. If you set persist = true then it will be a persisted asynchronous service, in other words saved to the database and then run from there. And persisted asynchronous services as we've discussed before are basically more reliable.

Once it's saved in the database then it moves on, and then if it fails it can be re-run. And if necessary things could be changed to make sure; to fix a failure so that it doesn't fail again and so on.

Okay the result-map-name. When the service is complete, you can take the result-map and put it in the context for other actions to use. Or if it's one of them that runs before the invoke, like invoke or before, which includes auth, in-validate, and invoke, then that map will also be available to the main service call. So that's what the result mapping does.

Result-to-context true or false. By default this is true, so if you don't want it to do this then you set it false. And what it does is it takes all of the results of the service and puts them in the context. So they can be used optionally for the future service calls, other actions whether within the same ECA rule or in other ECA rules. And they'll also be available again if it's before the invoke, including auth, in-validate, and invoke, then the results of this action will be available to the service as well.

Okay by default failures and errors will be ignored. But if you want it to stop the execution of not just this action and the ECA rule, but of the service that's being called as well, then you can set the ignore-failure or ignore-error to false, and an error in the ECA action will be considered an error just like an error in the main service action, and the same for failure. So for an error it will cause the transaction to rollback and everything else, like an error message to be returned and so on.

But by default if something happens in an action of an ECA then you'll have log messages, but it won't affect the flow of the normal service call.

00:17:08 screen changes to Advanced Framework Training Outline

Okay and that is it for service ECA definitions, service ECA rules. They're a fairly simple tool that provides a lot of power and a lot of flexibility. Again, like the service engine, they provide very good loose coupling that allows for flexibility and also to make things less brittle, so that if changes are needed in certain places it's less likely those changes will have a cascading effect on other areas.

And you can also keep your own code that extends the OFBiz code fully separate, because you can define an ECA rule that triggers on one of the core OFBiz services. But it's defined in your own component and the services and actions it calls can be in your own components or can be in the core application components in OFBiz. And basically you have a separate way of attaching to processes without modifying any of the code.

Okay so that's it for service ECA rules. Next we'll talk about a variation on them, the Mail-Condition-Action, or MCA, rules. These are for processing incoming email.

Okay this is a quick addendum to the service ECA rules.

00:18:37 screen changes to Framework Quick Reference Book Page 3

We covered the details and went over the conditions, but there's one that was missed in that, one that I just added to the Framework Reference book. And that is the condition-service. So with this one you basically use the condition-service tag and specify the name of a service, and it will call that service and pass in certain things, and then the service returns a boolean.

00:19:03 screen changes to services.xml

When you implement the service and define the service, it should implement this interface, the serviceEcaConditionInterface. And as you can see here it takes a serviceContext going in, and a serviceName, and then returns the conditionsReply as just a boolean, either a true or false. So you can implement the service, do whatever you want inside, check whatever in the context, and you also get the serviceName. You can look up the model service from the dispatch context when this is called and so on.

So that's pretty much all the information that is available in the service call to do things with.

00:19:58 screen changes to Framework Quick Reference Book Page 3

So this is another type of condition that you can fit into your ECA rules along with condition and condition-field, and another one that must be true if specified before the actions will be run.

00:20:11 screen changes to Advanced Framework Training Outline

Okay and that's it for that addendum. Next we'll move on to the MCA rules.

Part 4 Section 2.8

Service Engine: Service MCA (email) Rules

Starting screen Advanced Framework Training Outline

Okay this is the section on the service MCA or Mail-Condition-Action rules. These are somewhat similar to the Event-Condition-Action rules, except instead of being triggered by service event they're triggered by the receipt of an email. So the main purpose of these is the processing of incoming emails. The rules themselves are defined in an XML file, and the structure for that XML file is defined in this schema file, the service/dtd/servie-mca.xsd file. And in a component you'll typically find them here in the servicedef directory, and the file name for them is xmcas*.xml.

Okay in addition to this XML file, which has these various rules in it that we'll talk about, there's also the Mail Client Setup in the ofbiz-containers.xml.

00:01:19 screen changes to Framework Quick Reference Book Page 15

In the quick reference book you can see the information about the MCAs, on the service configuration page on page 15, and for convenience even though the ofbiz-containers.xml file. The main thing for it is over here on page 16, right here (screen flashes to Page 16 and back to 15). Over here on page 15 there's an excerpt from it for the Java Mail Container, that focuses on that specifically. Let's look at that along with the MCA rules here.

So when you set up the Java Mail Container, and you can set up multiple of these for different accounts and such, but you'll typically just have one and then that will point to a mail account depending with the rules in the MCA, the conditions in the MCA rules. You can get multiple email addresses or different other things that

you filter on, for incoming emails to send them to different services for processing.

Okay so this Java Mail Container: the delegator-name, this is the name of the entity engine delegator to use, and the name of the service engine dispatcher to use, and it will create a new service engine dispatcher for this. Then there's a run-as-user. The default value is system, so that's the user that will be used to run the services that are the actions in the MCA rules.

There's a poll-delay that specifies how long to wait between checking the email server for new messages. This is in milliseconds, so three hundred thousand would be three hundred seconds, or five minutes.

Okay delete-mail true or false, this is mainly for iMap but could also be used with Pop3. With iMap it's a little bit more flexible; there's this comment down here that says Pop3 can't mark messages seen, or you need to delete them basically. You typically set the delete-mail to true, so they'll be deleted as they come in from the Pop3 mailbox. With the iMap mailbox you can choose to delete them. Or if the delete-mail is false, as is shown here, then it will just mark them as seen, so it will not re-download them over and over.

The default-listener; you can have various different listeners here to listen to different servers. Okay the first property is the protocol to use, and this is showing iMap. The other possible value is Pop3 as there, so if you can use an iMap server it is more flexible and tend to work better.

You'll also specify the mail host, so this is the domain name or IP address of the mail server. There's a user name and password for that server, if required, and then this debug value which is set to true. You can set to false if desired, but this will basically log additional information to the console or the runtime logs, so you can see what's going on with incoming mails and such.

Okay so that's basically it for the configuration of the Java Mail Container in the ofbiz-containers.xml file. Now let's go ahead and go through the details of the MCA rule file format, since it's a fairly simple file, and then we'll look at some examples so we can see how these are actually implemented.

Okay the root element is service-mca, and each MCA element under that represents a Mail-Condition-Action rule. Each rule has a name, mostly used for messages and such related to it, and then there are three kinds of conditions: field, header, and service. And then you can have one to many actions associated with the rule if all the conditions are met.

So as with the other types of rules the action just has the name of a service, and the mode synchronous or asynchronous. If it's asynchronous you can specify persist=true, if we want it to go to the database queue instead of the in-memory queue, and you can specify a runAsUser like the system user or that sort of thing.

Okay let's look at the conditions now. The conditions are fairly simple; they're a little bit more simple in the MCA rules than in the ECA and other rules you'll see in other places. They're not so similar to the compare and compare-field things in the MiniLang and such.

So we have a field-name, an operator and a value. For the operator for both fields and headers the operator is either empty, not-empty, matches and not-matches. Matches and not-matches are for doing a regular expression match. These are ORO; ORO is the regular expression tool that we use typically in OFBiz, and that's used here.

And then we have equals and not-equals. So for empty and not-empty the value would not be used, for matches and not-matches the value is the regular expression, and for equals and not-equals the value is the value to compare it to. The condition-header is almost the same, except instead of comparing it to the field we're comparing it to one of the email headers.

So in the email message all the headers that you have sitting up at the top, those are available for rules here. You can also use the condition-service, where you just specify the name of that service to call, and there's a special interface just like with the ECA condition service. There's a special service to implement that passes in the mail information, and then the service returns a boolean true or false where it passes.

00:08:46 screen changes to services.xml

Let's go look at those. This is the services.xml file in the service engine component. Here we have the serviceMcaConditionInterface which is very similar to the ecaConditionInterface, but for the Mail-Condition-Action rules. It takes the messageWrapper object in, which is the same object that the mail processing interface takes in, and then this returns a boolean true or false in the condition reply.

So the mail process interface is the other one that we'll see used. Any MCA action should typically implement this interface, because this is the object that will be passed into the service when it is called.

00:09:38 screen changes to services_test.xml

Let's see an example of one of those that's implemented, and the corresponding MCA rule. Here's the testMca service. This is in the same file as the other test services we've been looking at; so the testBsh service and such are in here. Also this is in the common component.

So in the common services Java file we have an MCA test. This is the common pattern you'll see for these MCA action services, that they implement the mail

process interface. Of course you can have other services that do other things regardless of the mail process interface, so that's optional.

00:10:28 screen changes to smcas_test.xml

Okay before we look at the implementation of this service let's continue following the chain back and look at the testMca file. Each MCA has a rule-name, conditions, this has just a couple condition-fields, and then we have a single action that calls that testMca service that we were just looking at.

These fields are good examples. We have the to field and the subject field, and they're both using the match so this will be a regular expression. We have .*; any number of any character at ofbiz.org is what this first one represents, and the subject basically just has to have the word “test” in it. So it can have any number of any character before, and any number of any character after.

Okay so if it matches those two conditions then that message will be sent to the testMca service.

00:11:42 screen changes to CommonServices.java

Let's look at the implementation of that service in the CommonServices file here. And hop down to the MCA test.

Okay there's the mimeMessageWrapper that's coming in, and you can get a mimeMessage object. That mimeMessage object, if we go and look at the imports we'll see that that's part of the javax.mail.internet package. And the mimeMessageWrapper by the way is specific to the service engine, and mainly just meant for these MCA rules for the mail processing stuff in it.

Okay so once we get the message there are a whole bunch of different things you can do with that. This test one basically just logs information about the message that's coming in. It shows the to if there are recipients, the from if they're not null. It always shows the subject, sent date, received date, and such.

So it's just logging these things and then returning. This is an example basically of how you can get the message information in. And the main point is you get a service that's called, and you can get the message object from the mimeMessageWrapper. And then there are all sorts of different things; basically you'll be using the get ones you can get from that about the mail message, including all of the headers, all of the different standard mail fields, all of the message information attachments, and so on.

00:13:34 screen changes to Advanced Framework Training Outline

So that is the basics for the service that is called from an MCA rule. And there's been some work to take incoming emails and store them in communication events in the Party Manager. So that's one application, one of the more natural applications actually, of the MCA rules.

And then those communication events can be used for all sorts of other things, but you can basically use this for whatever you want. The thing to keep in mind with the mail processing here is there's no mail server embedded in OFBiz; it just acts as a mail client and checks an email account as configured in the ofbiz-containers.xml file

Okay next we're going to look at SOAP services, and then that will be it for this service engine section.

Part 4 Section 2.8

Service Engine: Service MCA (email) Rules

Starting screen Advanced Framework Training Outline

Okay this is the section on the service MCA or Mail-Condition-Action rules. These are somewhat similar to the Event-Condition-Action rules, except instead of being triggered by service event they're triggered by the receipt of an email. So the main purpose of these is the processing of incoming emails. The rules themselves are defined in an XML file, and the structure for that XML file is defined in this schema file, the service/dtd/servie-mca.xsd file. And in a component you'll typically find them here in the servicedef directory, and the file name for them is xmcas*.xml.

Okay in addition to this XML file, which has these various rules in it that we'll talk about, there's also the Mail Client Setup in the ofbiz-containers.xml.

00:01:19 screen changes to Framework Quick Reference Book Page 15

In the quick reference book you can see the information about the MCAs, on the service configuration page on page 15, and for convenience even though the ofbiz-containers.xml file. The main thing for it is over here on page 16, right here (screen flashes to Page 16 and back to 15). Over here on page 15 there's an excerpt from it for the Java Mail Container, that focuses on that specifically. Let's look at that along with the MCA rules here.

So when you set up the Java Mail Container, and you can set up multiple of these for different accounts and such, but you'll typically just have one and then that will point to a mail account depending with the rules in the MCA, the conditions in the MCA rules. You can get multiple email addresses or different other things that you filter on, for incoming emails to send them to different services for processing.

Okay so this Java Mail Container: the delegator-name, this is the name of the entity engine delegator to use, and the name of the service engine dispatcher to use,

and it will create a new service engine dispatcher for this. Then there's a run-as-user. The default value is system, so that's the user that will be used to run the services that are the actions in the MCA rules.

There's a poll-delay that specifies how long to wait between checking the email server for new messages. This is in milliseconds, so three hundred thousand would be three hundred seconds, or five minutes.

Okay delete-mail true or false, this is mainly for iMap but could also be used with Pop3. With iMap it's a little bit more flexible; there's this comment down here that says Pop3 can't mark messages seen, or you need to delete them basically. You typically set the delete-mail to true, so they'll be deleted as they come in from the Pop3 mailbox. With the iMap mailbox you can choose to delete them. Or if the delete-mail is false, as is shown here, then it will just mark them as seen, so it will not re-download them over and over.

The default-listener; you can have various different listeners here to listen to different servers. Okay the first property is the protocol to use, and this is showing iMap. The other possible value is Pop3 as there, so if you can use an iMap server it is more flexible and tend to work better.

You'll also specify the mail host, so this is the domain name or IP address of the mail server. There's a user name and password for that server, if required, and then this debug value which is set to true. You can set to false if desired, but this will basically log additional information to the console or the runtime logs, so you can see what's going on with incoming mails and such.

Okay so that's basically it for the configuration of the Java Mail Container in the ofbiz-containers.xml file. Now let's go ahead and go through the details of the MCA rule file format, since it's a fairly simple file, and then we'll look at some examples so we can see how these are actually implemented.

Okay the root element is service-mca, and each MCA element under that represents a Mail-Condition-Action rule. Each rule has a name, mostly used for messages and such related to it, and then there are three kinds of conditions: field, header, and service. And then you can have one to many actions associated with the rule if all the conditions are met.

So as with the other types of rules the action just has the name of a service, and the mode synchronous or asynchronous. If it's asynchronous you can specify persist=true, if we want it to go to the database queue instead of the in-memory queue, and you can specify a runAsUser like the system user or that sort of thing.

Okay let's look at the conditions now. The conditions are fairly simple; they're a little bit more simple in the MCA rules than in the ECA and other rules you'll see in other places. They're not so similar to the compare and compare-field things in the MiniLang and such.

So we have a field-name, an operator and a value. For the operator for both fields and headers the operator is either empty, not-empty, matches and not-matches. Matches and not-matches are for doing a regular expression match. These are ORO; ORO is the regular expression tool that we use typically in OFBiz, and that's used here.

And then we have equals and not-equals. So for empty and not-empty the value would not be used, for matches and not-matches the value is the regular expression, and for equals and not-equals the value is the value to compare it to. The condition-header is almost the same, except instead of comparing it to the field we're comparing it to one of the email headers.

So in the email message all the headers that you have sitting up at the top, those are available for rules here. You can also use the condition-service, where you just specify the name of that service to call, and there's a special interface just like with the ECA condition service. There's a special service to implement that passes in the mail information, and then the service returns a boolean true or false where it passes.

00:08:46 screen changes to services.xml

Let's go look at those. This is the services.xml file in the service engine component. Here we have the serviceMcaConditionInterface which is very similar to the ecaConditionInterface, but for the Mail-Condition-Action rules. It takes the messageWrapper object in, which is the same object that the mail processing interface takes in, and then this returns a boolean true or false in the condition reply.

So the mail process interface is the other one that we'll see used. Any MCA action should typically implement this interface, because this is the object that will be passed into the service when it is called.

00:09:38 screen changes to services_test.xml

Let's see an example of one of those that's implemented, and the corresponding MCA rule. Here's the testMca service. This is in the same file as the other test services we've been looking at; so the testBsh service and such are in here. Also this is in the common component.

So in the common services Java file we have an MCA test. This is the common pattern you'll see for these MCA action services, that they implement the mail process interface. Of course you can have other services that do other things regardless of the mail process interface, so that's optional.

00:10:28 screen changes to smcas_test.xml

Okay before we look at the implementation of this service let's continue following the chain back and look at the testMca file. Each MCA has a rule-name, conditions, this has just a couple condition-fields, and then we have a single action that calls that testMca service that we were just looking at.

These fields are good examples. We have the to field and the subject field, and they're both using the match so this will be a regular expression. We have .*; any number of any character at ofbiz.org is what this first one represents, and the subject basically just has to have the word “test” in it. So it can have any number of any character before, and any number of any character after.

Okay so if it matches those two conditions then that message will be sent to the testMca service.

00:11:42 screen changes to CommonServices.java

Let's look at the implementation of that service in the CommonServices file here. And hop down to the MCA test.

Okay there's the mimeMessageWrapper that's coming in, and you can get a mimeMessage object. That mimeMessage object, if we go and look at the imports we'll see that that's part of the javax.mail.internet package. And the mimeMessageWrapper by the way is specific to the service engine, and mainly just meant for these MCA rules for the mail processing stuff in it.

Okay so once we get the message there are a whole bunch of different things you can do with that. This test one basically just logs information about the message that's coming in. It shows the to if there are recipients, the from if they're not null. It always shows the subject, sent date, received date, and such.

So it's just logging these things and then returning. This is an example basically of how you can get the message information in. And the main point is you get a service that's called, and you can get the message object from the mimeMessageWrapper. And then there are all sorts of different things; basically you'll be using the get ones you can get from that about the mail message, including all of the headers, all of the different standard mail fields, all of the message information attachments, and so on.

00:13:34 screen changes to Advanced Framework Training Outline

So that is the basics for the service that is called from an MCA rule. And there's been some work to take incoming emails and store them in communication events in the Party Manager. So that's one application, one of the more natural applications actually, of the MCA rules.

And then those communication events can be used for all sorts of other things, but you can basically use this for whatever you want. The thing to keep in mind with the mail processing here is there's no mail server embedded in OFBiz; it just acts as a mail client and checks an email account as configured in the ofbiz-containers.xml file

Okay next we're going to look at SOAP services, and then that will be it for this service engine section.

Part 4 Section 2.9

Service Engine: SOAP Services

Starting screen Advanced Framework Training Outline

In this last section on the service engine we'll be talking about SOAP services and service engine services, and how they relate and how they can call each other and whatnot.

So first thing to note. You've probably noticed already if you have any familiarity with SOAP that service engine services are a fair bit different from SOAP services. They both fall under the category of that service oriented architecture, but the service engine is not a web services standard. It does have various facilities in it for distributed services, but there's not a single standard for the distributed services, and that's not the intent.

The service engine in OFBiz is meant to be the framework or infrastructure that the logic layer of the applications are built on. So it facilitates the various things that we talked about in the introduction to the service engine.

So the service definitions, the way services are called, the optimization for local services in calling local services synchronously in a thread, these sorts of things are a fair bit different from what you'll typically deal with when looking at a service oriented tool that is SOAP based or web service based.

The service engine, though, can interact with those. And there are various ways of making OFBiz services or service engine services available as SOAP services themselves, as well as calling soap services from OFBiz services or other OFBiz logic.

So let's talk about some of this. The first one we want to talk about, the default context service engine context to SOAP envelope mapping.

00:02:12 screen changes to Framework Web Tools

In the webtools webapp when we're looking at this is in the service reference that we've looked at before, and this is the reference page for the test service. This does have export set to true, and when export is set to true you'll find this little link here, show WSDL. Now the main things that we'll expect to see in here are messages going in, and the resp for response going out.

So what this is going to do, this service engine has a built in feature to generate WSDL based on a service definition. Now there are of course some limitations to this, and for more complex services it may not work out very well, but this is the default mapping. So in some cases it can save you a lot of time and effort and make OFBiz services available in other tools, even other languages and other operating systems and all sorts of things.

So the alternative to this for more complex services we'll talk about as well a bit later.

00:03:28 screen changes to Framework Web Tools Service: testScv

So we do show WSDL. It just drops it into a text box here for convenience and then you can copy it. And if you're using a SOAP client generator tool you can use this as a basis for generating the client code, or if you have some sort of a service bus tool where you need to do mapping between SOAP envelopes then you can use this for feeding into the mapping tool and such.

So this is basically just standard WSDL file, and all the details about the different standards used and such, namespace for WSDL, SOAP standard used and so on, are all in here. So I don't know if there's much more to say about that. Basically you can use this text in your tools, or to get an idea of what the default mapping looks like between the OFBiz services and SOAP services according to the WSDL definition.

00:04:50 screen changes to Advanced Framework Training Outline

Okay now let's look at some of the other things here; how to call a SOAP service from the service engine, and also how to call a service engine service from a SOAP client. With both of these we have this context envelope mapping going on, or in other words they both need to do that. Let's start first with this calling SOAP services through the service engine.

00:05:21 screen changes to services_test.xml

There is an example test service here in our good old services_test.xml file in the common component, called testSoap. When you want to do a service that when you call it locally will result in a call to a SOAP service on another server, then you just specify the engine = soap and the name of the service to call, or the location of the SOAP server and then the name of the service to call there. So this is a test server that Apache runs for Axis.

Here's the namespace for the SOAP call. This'll feed into the WSDL file, the envelope and such, as it's put together. And we have the message going in and result going out. So those are the names of the parameters that are used there. And there's an example of how to implement a service that, when called locally, will result in a call to a SOAP service that's elsewhere.

By the way, we talked a bit earlier on in this service section of the Advanced Framework training about using HTTP to make OFBiz services available to other clients. Those HTTP services can also be called through a service definition, and here's a client that calls the test service through that HTTP service request that we were looking at, the one that is in the webtools webapp.

So that's one way to call it. This uses the XML serializer in OFBiz for the context that's going in and out. That's how it's text encoded and submitted, as it should be a form field type submission to pass it to the server. And then the body of the result that comes back from the server is also an XML serializer serialized XML file or XML snippet that can then be parsed in the client to get the result back. But the HTTP client that's in the service engine can call that type of service that's specific to OFBiz just like the SOAP client can all a SOAP service using the more standardized envelope and such.

00:08:16 screen changes to controller.xml

In the controller.xml file is where we saw the http service defined so that an OFBiz service can be called externally through this. And similarly we have a request here for the SOAP service. So this is a request so that OFBiz services can be called as SOAP services using the same mapping.

00:08:47 screen changes to Framework Web Tools Service:testScv

Now this is where that WSDL file that we saw generated in webtools over here really directly applies, because that SOAP client should expect to work according to this WSDL definition. When it calls a SOAP service through this request.

So notice the event here has type = soap, but then no method or anything. There's an event handler, the SOAP event handler, that just handles generic SOAP requests coming in and maps them to service calls, and then takes care of them.

00:09:25 screen changes to SOAPEventHandler.java

So let's look at a couple of these things. Let's look at the SOAPEventHandler. Basically when it's invoked this can do a couple of things, like through this request it can handle requests for WSDL documents as well as actual service calls, this uses Axis to do the mapping, and then you can look at the details of how it does the mapping of the envelope in here to result in the OFBiz service call.

So it'll call the OFBiz service using the default mapping, and then take the result from the OFBiz service and

again map it back to the SOAP envelope for the response, and send it back down to the client.

So for more details, or if you want to see exactly how it's done, this is a good place to look.

00:10:28 screen changes to services_test.xml

Going back to the other side for a second, where we have like this testSoap service that when called locally results in calling this SOAP service on another server.

00:10:38 screen changes to SOAPClientEngine.java

That's implemented with the SOAPClientEngine. And so you can see in here this is also using Axis to make the SOAP call and do the automated mapping. So for more details you can look through these files. Right now the state of SOAP support in OFBiz is okay and it's usable, but mostly just for simple services.

00:11:15 screen changes to Advanced Framework Training Outline

So I guess we kind of talked about both sides now. If you have more complex services like SOAP services that you need to call from OFBiz, then rather than, like the test_soap service that we saw, using the engine-soap in a service definition, the better thing to do is use another web service client tool. Whatever your preference is, to write web service client code, especially if you're doing a lot of them that will go along with some sort of a visual tool that can take a WSDL definition for whatever service you need to be calling, and generate stub code in Java for making that remote SOAP call. And then you can just use that code inside an OFBiz service.

So basically you have a translation so that when the OFBiz service is called locally, the service implementation maps the input parameters to the SOAP envelope that will then be sent to the SOAP server. And then response from the SOAP server that comes back, that code would also map that back to the response that is defined in the service definition for the OFBiz service.

So it's basically creating a custom wrapper around the SOAP service, and you can do the same thing in the other direction. If there's a certain SOAP standard that you need to comply with and create SOAP services that will comply with, then chances are you're not going to be able to use the automated mapping in the service engine. So you'll need to use a tool to generate SOAP services that a client can connect to and use, and then those SOAP services, when called, can in turn use a dispatcher or delegator to call other services, do entity engine operations, and so on.

So sometimes that sort of custom code is required to do mapping and to support the complexity of envelopes that are available, which can be very different structures in fact than how the internal services work. Or you may need to call multiple services, and all of these sorts of things.

So that's generally what ends up happening with more complicated SOAP services. But for the simple ones you can actually get pretty far just using this automated mapping and the existing tools that allow you to interact between a SOAP client or server and the OFBiz service engine.

Okay and that's it for SOAP services, and that's also it for the service engine in part 4 of the Advanced Framework series.

Part 5 Section 1

Business Process Management

Starting screen Advanced Framework Training Outline

Okay in this part of the Advanced Framework Training we'll be talking about some general concepts of business process management or workflow types of things; rule engines, data files, etc. For the most part through this part five we'll be staying at a higher level and be talking about more concepts, and how they relate to OFBiz.

For the most part there are no real out of the box examples for certain things like the OFBiz workflow engine. There is an older example that is not active by default, and that's complicated by the fact that the OFBiz workflow engine is planned to be replaced by the Enydra Shark workflow engine. And we have a partial integration already in place for that.

So we'll talk about some of these things, starting though with more general concepts and how they fit into other tools, like the service engine and such in OFBiz.

Okay first of all the workflow engine and business process management concepts and such. In general I prefer to use the term at a high level of business process management rather than workflow, and there are other standards related to business process management and splitting them up. There are two main camps in them; one of them is the workflow style, and the other one is the message flow style of business process management.

The workflow engine in OFBiz is more of the work is definitely fully in the camp of the workflow style. With the workflow it's kind of like a state machine, so you'll have states and transitions between those states. The workflow engine in OFBiz and the Shark workflow engine are both based on the WfMC and OMG standards.

These are complimentary standards that work together for defining WfMC, and they have a whole bunch of things, XPDL for an XML file that defines a workflow through various interfaces and such that can be used to

interact with to manage and to get information from the workflow engine. And then the OMG, their primary concern in the world is with interfaces. So WfMC worked with OMG on certain interfaces that are related to workflow control and interaction from outside programs.

The other main style is the message flow style,. With this style, rather than having a single workflow that's very centralized and managed by a workflow engine, we have a whole bunch of different nodes, and these nodes pass messages around. The terms are typically consume and produce messages, so nodes become the consumers and producers of messages.

Some standards that are related to this, the ebXML BPSS (Business Process Specification) is basically an interface definition for a message flow style workflow, whereas the BPML standard from BPMI is kind of a way of implementing that sort of an interface. So this is an XML file that can tie in with other applications and lower level locigin and such, but basically BPML can be used to implement a message flow style workflow.

There are various SOAP related standards that build this sort of a business process management system on top of web services, which is what they use to pass the messages around.

Now one of the reasons the workflow engine in OFBiz has not been used as much. Early on we did plan on having something for higher level business process management, and these sorts of things, and we found some applications for a workflow style process management system. That's why the OFBiz workflow engine was implemented originally. However, finding other applications that were better for workflow than other tools became a little bit more difficult over time.

So it's still very good for certain types of workflow in particular, very good for certain types of work management, especially where you have combinations of automated processes and manual processor things done by humans that need to be managed by the system, and very centrally managed and controlled. That's what the workflow is good for.

Message flow is much better for integration between systems, especially integration on a process level. That's where it really shines, in integration on a process level, and through these message passing getting things going on.

In OFBiz basically what has happened over the years is, since these service ECA rules were introduced, and an ECA rule is usually considered more in the camp of a rule engine, it's kind of a variety of rule engine. But in a way I kind of like to categorize it more as a business process management tool, even though it consists of rules based on events that are happening in the system.

But those events are kind of like messages being sent to nodes that consume the incoming messages or events and produce the outgoing messages or actions. So it's very similar in terms of application, although of course the ECA rules in OFBiz are not nearly as formal as something like the BPSS or even BPML definitions, where you have a set of XML files that define more globally what the processes are and how they fit together.

Okay let's talk more about some of the workflow concepts. This is specifically for WfMC style workflows. At a high level you have processes that are composed of activities. Activities can have either a manual or automatic start and end. If it's automatic it means there's typically some sort of logic that's associated with the activity that's used to implement and execute the activity. Whereas for manual starting or ending is used where a person needs to start or end the activity. So in other words you can have combinations of these where an activity is automatically started and manually ended, and vice versa.

This comes back to where the real strengths of the workflow style thing come into play, and that is highly centralized management of combinations of manual and automatic activities basically. So activities are basically like states in the state machine.

And then we have transitions between the activities. When an activity does end there can be an arbitrary number of transitions going out of the activity. They can optionally have a condition associated with them. And if the condition or conditions are met, or if there is no condition, then the transaction will lead to other activities that will be set up for starting, either manually or automatically started and executed. And then the workflow moves on.

So there are different concepts of splitting and merging. If you have multiple transitions going out of an activity then that is a split. And you can tell it how you want it to split, like choosing one of the transitions or going through all of the transitions where the conditions are met, and that sort of thing.

Merging is the other side of things, where you have multiple transitions coming into an activity. When that happens you can specify that all of the transitions must come in before the activity can start, or maybe just one transition needs to come in before the activity can start.

So those are the concepts of splitting and merging. As a workflow executes it has a data context that's a lot like the context of a service engine. And speaking of which a workflow by nature is asynchronous; it runs in the background. You can sometimes inject data into the context of a workflow, and there are things you pretty much always have to have in any workflow engine: some way of interacting with the workflow, seeing

activities, especially manual activities that are waiting for a start or end, and other things.

Sometimes you'll have specialized user interfaces related to the workflow. So for a manual start or end it might be associated with a special screen that a user goes to to enter information, or that sort of thing. And then once that's completed it can end the activity and allow the workflow to move on. These sorts of things.

But a workflow by nature is asynchronous; it runs the background. So when called through the service engine they're typically asynchronous. You can call a workflow synchronously, and there is a wrapper around things that are naturally asynchronous to call them synchronously in the service engine.

But if you do that you could be waiting for a very long time, especially if there are manual steps that someone needs to go in and do, or other processes that it's dependent on, real world processes liking printing or packaging or assembly or these sorts of things.

Okay so those are some of the general concepts for working with a WfMC style workflow, and how they kind of fit in with the service engine and other things. So workflows can be triggered by a service call, typically asynchronous, and passing in a context that becomes an initial context for the workflow. But it's also possible, especially for automatic activities that they be4 implemented by, or the logic associated with the automatic activity.

In OFBiz the best tool to use for that is basically a service call, so you'll usually pass in certain parts of the workflow context. The full workflow context will map into the input attributes for a service, and you can execute the service.

So both the OFBiz workflow engine and the Shark workflow engine with the OFBiz integration can do both of those things; you can call the workflow as through a service. So in a service definition you can say that it is implemented with a workflow. And inside the workflow you can also call out to services, in other words implementing automated activities through service calls.

Now specifically for the OFBiz workflow engine, I'm not going to cover the details here because we are planning on moving away from the workflow engine. There are some examples if you want to look at them; in the orderManager there is a custom user interface for order approval, and an order approval workflow.

00:13:08 screen changes to orderProcessXPDL.xml

If you look in the servicedef directory in the order component you'll see here we have this orderProcessXPDL file. And this is a service definition file written in XPDL, which is one of the WfMC standards. And in addition to the standards, the standard has facilities for extensions in the extended attributes, and that sort of thing, that specific workflow engines can add.

So the first thing to learn about using the OFBiz workflow engine, or the Shark and Enhydra Shark workflow engine, is XPDL and some of general concepts of how these fit together.

One of the reasons we've moving towards Shark is that it has a visual workflow editor which is very convenient. So you don't have to get into these details, and what is really honestly a fairly messy XML file.

00:14:17 screen changes to http://incubator.apache.org/ofbiz/docs/workflow.html

But the extended attributes for the OFBiz workflow engine, if you do happen to be using the OFBiz workflow engine, are available in the workflow engine guide, on the ofbiz.org site. This has some information about the workflow engine in general and related standards, and then all these extended attributes related to different parts of the file: related to activities, implementation or tool extended attributes, and so on. There's also some commentary about the APIs, general activity types, and so on.

00:14:49 screen changes to Advanced Framework Training Outline

So this is a good resource, the workflow engine guide, for more information about the OFBiz workflow engine.

The OFBiz workflow engine by the way uses a lot of stuff in the service engine. In fact a lot of the features in the service engine were driven by the fact that something like them was needed in the workflow engine. But we found them to be very useful independent of a workflow engine, just through the semantics of the service engine.

Okay so moving on to the Enhydra Shark Workflow Engine. There is an integration with Shark in place in OFBiz, so that Shark can use the entity engine for persisting its workflows and statuses and such. In the integration it uses its own data model and such in the database.

The OFBiz workflow engine, instead of using its own data model, used other things. There's a specific data model for the OFBiz workflow engine for the process definitions and such, bu the runtime information for the OFBiz workflow engine was put in the workEffort entity.

But the Enhydra Shark integration basically just has a number of entities that are specific to Shark. So the main source of information, and everything about Shark, is at this URL.

00:16:21 screen changes to http://www.enhydra.org/workflow/shark/index.html

And here's what the page looks like. This has a number of nice features; it has a good API for interacting with the workflows, and the visual editor is fairly open.

One thing to note about it, however, is that it is LGPL licensed. OFBiz has been in the Apache incubator for a while, and we have cleaned out all of the LGPL licensed libraries because we cannot distribute them with the project. However, we can distribute code with the project that is dependent on LGPL licensed libraries. Not on LGPL licensed libraries, but we can on LGPL licensed libraries.

So what you'd do if you want to use the Shark workflow engine in OFBiz is copy the libraries into place, and then you can run it from there.

00:17:29 screen changes to OPTIONAL_LIBRARIES

With all such libraries the resource to look at is this OPTIONAL_LIBRARIES file. And the very first entry here just happens to be in the workflow section, the shark libraries. So these are all the shark libraries that you need, or quite a few of them. Most of these libraries we have maintained available in the old ofbiz.org subversion server that is hosted by Undersun, and we are continuing to host that. So if you check out the old OFBiz repository, in the specialized directory, there's a libraries component that has all of these libraries, for Shark as well as for various other things.

00:18:16 screen changes to Advanced Framework Training Outline

Okay so there is a pretty good integration in place. Although there are a few issues with it, as the shark API has developed over time.

We actually started using Shark pretty early on in its life cycle, and it has matured significantly since then. And the current version of the libraries, that are in the specialized directory in the old OFBiz server, is the easiest place to get them, because I believe they were just a build from source fairly early on.

One of the things that really needs to be done with this integration is updating to the most recent version of Shark, and then building out from there. There is a web application that has some good tools for interacting with Shark workflows: for managing the process repository, looking at running processes, and so on.

00:19:21 screen changes to OFBiz.org Framework Shark, Shark Manager Main Page

And that's the Shark webapp; you can see the tab for it up here. It has the XPDL process repository, process list for running processes and such, and then the work list for manual activities that are waiting for a user response. So that's a generic work list.

You can also build customized applications, as is often what's done with these, although a generic work list is usually helpful and often used as well. Anyway this is some of the basic stuff that you'll need.

But when Shark is deployed in OFBiz you can also connect to it remotely with the Shark client, which is a desktop application. And it has a pretty extensive set of functionality for interacting with the running workflows, starting workflows from that tool, and even defining workflows and such.

00:20:08 screen changes to Advanced Framework Training Outline

Okay that's shark. And that is it for the workflow engine in general, in general business process management stuff.

Part 5 Section 2

Rule Engines

The next topic that I wanted to cover from a high level is rule engines. We'll go into a little bit more detail about the data file tools when we get there, and look at some examples and stuff.

But for rule engines; first by way of general concepts there are two main categories of rule engines that I wanted to distinguish.

There's actually a third one, and that is the ECA rules, or business event rules is another term for them. But as I mentioned, I kind of like that better in the business process management category, even though it is rules driven business process management. So it kind of bridges the gap between the two.

On that note on bridging gaps, I should mention in this workflow area there have been some efforts to combine the WfMC and BPML standards to create a new standard that features both workflow style and message flow style business process management. So that might be an interesting thing to look at, and you'll probably see open source projects like (screen flashes to enhydra.org/wrokflow/shark/index.html and back) the Enhydra Shark workflow engines adopting some of those things in the future.

Okay rule engines generally fit into two categories, one of which is inferencing engines. This is where you'll see terms like forward and backward chaining, the ready Rete algorithms and this sort of thing. This is kind of a prolog style; logical language is another term for these, and basically prolog is used for inferencing.

The general idea with inferencing is the output of one rule can affect the inputs of another rule. So the rules are chained together; you can either go forward, starting with some input data and following the rules forward to find the output results. Or you can start with output

data and go backward through the inferencing chain, and verify or find rules that contradict the proposed results. Basically that's the general idea for backward chaining.

Another type of rule engine, or this is more of a general tool based on a rule engine I guess, is a constraint-based optimization tool. These are very helpful tools, where you can set up a series of constraints and structures.

Basically at a low level what it's doing is going through all combinations of the different options for the structure, and eliminating the ones that violate the constraints, and then finding all of the combinations. So this falls back into combinatorics as well, but it finds all of the combinations that fit all of the constraints, and then it will go through those combinations of options in the structure. And with an optimization function, or a fitness function, determine the best option of all of the combinations.

And typically you'll let it run for a while. If there are too many constraints, if it's over constrained, you might have a very small number of options and not really get a very good result by the fitness function. If you have too few constraints, you'll have so many options that it takes a long time to go through them.

But typically what will happen is that a good system that implements these sort of things can do things like more intelligently choosing the combinations of options, to get what will most likely be a good fitness result. And in any case you'll typically let it run for a certain amount of time, or check a certain number of combinations of the options. And when that's done you'll find the best result so far and just use that result.

So constraint-based optimization is very helpful for problems that don't have a single solution, or are very complicated. And sometimes even for problems where you're not seeking an optimal solution; you're just checking to see if a solution is even possible, doing evaluating combinations of the different factors in a structure and checking those against the constraints. And that alone can be valuable.

So there are a number of commercial tools that do constraint-based optimization. You could build such a tool based on some of the open source ones, but for the most part you're stuck with commercial ones in this area, that are really intended for constraint-based optimization.

But this is one of the things that really is highly complementary to business software like that which is found in OFBiz. And it can do things like, if you have shipments that you are running with your own trucks. Or even manufacturing processes and you're trying to find the ideal order to manufacture things, or you're trying to optimize based on profit. So if you have agreements for manufacturing that will pay more, if they're done more quickly and such, then you can see how those will fit in with the rest of them, and how you can optimize that for maximum profit.

For shipment routing you'll usually minimize based on distance or time or these sorts of things. These are also used for applications like assigning flights to gates at airports, and all sorts of things.

For UPS I saw recently that they are deploying a very large system that basically looks at all of the packages that will be delivered in a route for a truck, and it determines the optimal path to go through. Basically the roads for a driver to take and such; to go through this route and drop off all of the packages, or pick up packages. So basically optimizing the stops that need to be fit into a route.

So there are a lot of really neat applications for this. I think in a lot of ways this is more exciting than just the plain inferencing engines. Although the inferencing engines are basically an important part of constraint-based optimization, and have a number of applications, a number of helpful things like problems they can solve effectively, that are very difficult using other techniques.

One interesting thing about inferencing engines that I'd like to note, and this came out with the OFBiz rule engine, which is now very much gone, is that early on in the days of OFBiz we had this rule engine in here which was based on a book called Building Parsers in Java, by Steven Metsger I think was the author's name. And that's actually a very helpful book for doing parsers, especially dynamic parsers.

But the rule engine in there was a good starting point. We started to expand beyond that, but the main impetus behind that was the fact that open source engines were extremely weak at the time. That has since changed, and there are some pretty good open source ones now, like JBoss Rules and Mandarax.

00:28:55 screen changes to
labs.jboss.com/portal/jbossrules

Jboss Rules especially is looking very promising. This is based on the old Drools rule engine, which was part of the code house projects. And they appear to have done a lot of good stuff with it. So if you need some sort of a rules engine that provides for this declarative programming, specifically declarative inferencing, then this is a good place to look. It's is a very interesting project. They do have an Eclipse plugin and such, with auto-completion and all sorts of handy things for these.

But so far we have not found a really good home for this sort of thing in OFBiz. We may introduce it more in the future at some point, but it's still kind of a solution searching for a problem in a lot of ways.

00:29:50 screen changes to Advanced Framework Training Outline

Okay another pretty good one is Mandarax, and you can Google that and find some information on it.

And there are other open source rule engines, and over time these are actually progressing impressively well, which is why we removed the OFBiz rule engine from the project. So if you need such things then the best thing to do is use one of these other open source projects.

But one interesting thing that came out of the research, the time that I was putting into the rule engines, is the similarities between a rule engine and a relational database. SQL in it's own way is a declarative programming language that has, it's when you're dealing with a rule engine you typically have facts and rules.

And facts are basically just rules with no preconditions; they're just assertions, only without conditions. So a database is basically just an assertion management system, and when you run SQL against the database it's actually fairly similar to running rules against a set of facts to determine an output. And more complex SQL queries or stored procedures and such actually start looking a lot like sets of inferenced rules, except that they don't do the chaining. It's basically just a single step inferencing engine, which is not really technically an inferencing engine.

But the structures are very similar; the concepts on a low level are very similar. Even the term twople? is often used for data structures in a rule engine, as well as for a relation in a relational database or a table in a relational database.

Okay there are also a number of good commercial rule engines that you might want to check out, especially if you are working for a big company that has a lot of money. These are typically very expensive; they can be like ten to fifty thousand for a single license. Sometimes significantly more, depending on how you're deploying it and how much server power is going to be launching these.

The two mains ones are Blaze, which was purchased by fair isacc, and iLog rules is a pretty good rule engine. And these are both still being actively developed and used and are pretty good things. iLog in particular also has a pretty good constraint-based optimization product that might be of interest.

Okay and that's it for the workflow or business process management section, and the rule engine section.

Next we'll talk about the data file tool and go into more detail on that, and look at some examples and such.

Part 5 Section 3

DataFile Tool

Starting screen Advanced Framework Training Outline

Okay this section is about the data file tool. This is one of the tools that is not used in the applications a whole lot because the main purpose of the tool is to read and write usually proprietary files, as part of communication with another system or JDA migration or whatever.

So the efforts that are based in the data file tool are usually pretty custom. But it does tend to get used quite a bit actually, so I wanted to mention some things about it here in these Framework Training videos.

The main point of the data file tool is to work with flat files. There's some tricky things that come into these, lots of different styles of text files and stuff. They could be character separated fields and character separated records, usually with one record per line type of thing in either comma or tab separated, so it supports that sort of file.

It also supports fixed width fields and fixed width records, or fixed width fields on one record per line style file. It even supports files where they have hierarchical structures, and you can specify records that are children of records, and different record types in the same field.

Usually in that case the record type will be identified by some sort of a few characters in the record, and so with that type of data file you can specify for each record where the type identifier is located, and then for each type identifier what you should expect in the record.

So the main point of the data file tool is to provide generic data structures that are similar to the entity engines. The record object in the the data file component is very similar to the generic value object in the entity engine. Once the records are read in then you have this simple object to work with them, so your code for dealing with the data files or with the flat files or other text files is significantly simpler. And if there's ever a change to the specification of the flat file then your code doesn't have to change so much. The main change will go into the data file definition.

So data file definitions are done in XML files, and the structure of those XML files is specified in this XSD file in the datafile component, the dtd directory; it's called datafiles.xsd. And the locations of the files will vary, but they're usually treated kind of like a script; they're resources that are loaded from the classpath, so they're usually under the script directory.

Let's look at some data file examples and the data file API.

00:03:45 screen changes to TaxwareFiles.xml

There are a few places where we've found a use for data files in integrations with other tools, or loading data from fairly standardized sources. So this one is for Taxware files. Fixed-record is the separator-style for records in the file, and for fields in those records.

The three styles, and I kind of mentioned these before, are delimited. So the fields in a record are character delimited typically, and will have also one record per line. So the records are also basically character delimited.

Then there's the option of fixed-length, where you'll have fixed-length fields, but rather than a fixed-length record you'll have characters delimiting the records. Like an enter character turn, a line feed, or whatever, to separate the record so you end up with one record per line essentially. Fixed-record is kind of the ultimate in fixed-width formats, where both the fields and the records are fixed-width.

This Taxware file is that way. This record is 60 characters, this record is 2,592 characters. And with this sort of thing you can see a type-code, and each field in a record will have...actually the type-code is irrelevant for these because it looks like there's only one type of record per file. Let's verify that real quick...yeah none of these have more than one type of record per file. So the type code is basically ignored.

But each record has a number of fields on it, and for a fixed-width field we basically specify the position and the length of each field. So zero and it's six characters; six, one character; seven, one character, and so on up to ten, which is fifty characters.

And then the same is true here. Basically the positions are just counting up through the full length of the record. So for each field you can specify the type of the field. And if it's just a string, basically that's what you'll get in Java, just a string.

If it is a number then it has different options for parsing numbers. So you can specify a default value, description to go along with the field name is optional, expression, the format, whether to ignore the field or not, and whether it's effectively a primary key field.

Let's look at the format one real quick. The format can be used to specify a number format or a date format, or that sort of thing. Okay, whether it's expression or not for this.

Let's see, with one of these we should go look through the schema file. But basically we're specifying. For different types there's the format string to specify how to parse it, and for some things, especially for fixed-width fields, you'll often end up with cases where the numbers are zero padded or space padded. So the parsing is not necessarily a whole lot different for those, but for writing the file back out it is important to know what sort of padding you're dealing with, spaces or zeros or whatever.

One of the goals of the data file tool from the beginning was to be able to produce output that is the same character for character through the file as the input, which is especially important for fixed record files. So that was what a lot of the early testing was based on, to be able to read it into the data file structures, write the data file back out, and have them be the same.

So when you have data in a data file structure, and you can use the data file definition to go along with that, it can be used for both reading and writing of these text files.

00:09:00 screen changes to ZipSalesTaxTables.xml

Okay another example is the ZipSalesTaxTables. This is fixed-length as opposed to fixed-width, so we have one record per line. They still specify the the record name here but that's not necessary. So basically you specify here just the same for fixed-widget fields: where each one starts, and how long each one is in terms of characters.

There are a number of them here that are doubles, so numbers and double precision floating point numbers. So as this is read in it will parse these values and convert them into a double object. And for any failures, any problems there, as it's going through the file it keeps a list of error messages so you can see all the problems in the file.

00:10:05 screen changes to Framework Web Tools, Work With Data Files

Okay there is a webtools user interface for this. It's a fairly simple page in the DataFile tools; we just have one page in here. This can be used to read in a flat file, so you can specify the definition file name or URL, the DataFile definition within the definition file.

00:10:28 screen changes to TaxwareFiles.xml

If the definition file for example was pointing to this TaxwareFiles, with the full resource location on the classpath of course, then the data file within it would be taxwareOutHead, or taxwareOutItem. These sorts of things.

00:10:39 screen changes to Work With Data Files

So that's the name of the DataFile definitions within this file. Okay and then the DataFile name or URL; this is the file to read in and then optionally the file to save it out to.

Now there's this other option to save it out to an entity XML file. If you have a DataFile definition that matches an entity definition, or the records match an entity and the fields match an entity field, then it can do automated output. This actually can be somewhat useful.

The save-to file name will basically use the same definition and just write it out for testing purposes, to make sure it's reading and writing that data file properly. But if you write it out to an entity engine XML file then it actually transforms it to that XML file, and that can be imported into the database through the entity engine XML import tools.

00:11:44 screen changes to Package org.ofbiz.datafile

Okay so looking at the API for this. These are the Javadocs for it, and this is for the package org.ofbiz.datafile. The main objects you'll typically be dealing with are datafile and record. And if you have a very large set of records in a data file then rather than iterating, like getting a list of records from the datafile object, you can use the recordIterator. This is kind of like the entityListIterator, to read the records in through a stream and then you can process them one at a time. So if you have very large data files then that sort of thing is very often necessary.

00:12:39 screen changes to Work With Data Files

This datafile2entityxml class is what is used for that field we saw over here, for saving the data file data out to an entity engine XML file.

00:12:48 screen changes to Package org.ofbiz.datafile

Again the main things we're working with are the datafile and record objects.

00:12:53 screen changes to Class Record

And the record object has on it things for getting and setting the fields on the record. As I mentioned before, this also supports. One big difference between this type of the record object and the GenericValue object is that a record can have child records. So this will happen in flat file where there's like a header record, and then a number of detail records. And those detail records will be considered children of the header record.

00:13:27 screen changes to TaxwareFiles.xml

And in the datafile definition, for a record definition you can specify the name of the parent to use. So that's what that would be used for. And then you end up with basically an hierarchical data file.

00:13:49 screen changes to Class Record

So a lot of flat files, especially fixed-width files and stuff, are that way. Not so much for character delimited files; it's a little bit tricky to try to fit a character delimited file into a multiple record-type-style format.

Okay this has an object kind of like the GenericValue object. GetDate, getDouble, getFixedString, and so on.

When you get a record, if it has child records you can get a list of the child records to iterate over and process. Often when you have child records you'll need the parent record at the same time in order to get all of the data related to that record, for populating in the database or whatever you're doing with the data that's incoming or outgoing.

00:14:48 screen changes to Class DataFile

Okay the datafile object itself. This will represent a datafile that's been read in. It has a pointer to a modelDataFile. So there's a getModelDataFile, and typically when you construct it the default constructor here, the no parameter constructor, is protected, so you can't construct it that way.

00:15:14 screen changes to TaxwareFiles.xml

The only way you can construct the datafile is with the modelDataFile, and the modelDataFile basically represents this XML that's read in for a specific datafile element like this guy right here.

00:15:27 screen changes to DataFile

So once you have your modelDataFile you can create a datafile record for it. And then you can do a few things with this object. You can use it for reading or for writing, so you can either read in the data file from an input string, plus locationInfo for messages, or just directly from content. So you have the datafile content sitting in memory somewhere, you just pass it in. Or from a URL; this could be a local file URL, or it could be an HTTP URL or whatever. And here's a static one that will create a datafile object based on the fileUrl, definitionUrl, and the dataFileName.

00:16:12 screen changes to Framework Web Tools, Work With Data Files

Basically the same three fields that we saw here, the dataFileUrl, DataFile definition name within the definition file, and the DataFile name or URL itself. So those corresponds to the same things.

00:16:30 screen changes to DataFile

Okay and then once you do that you can either get all of the records. If you've read the file in you can get all of the records. Or, as an alternative to reading the entire file in, you can make a record iterator and pass in the dataFileStream and locationInfo, which is for messages. All of the data is just in the stream coming in. So if you've already opened the file or whatever you can just pass in the stream, or you can pass a URL.

Then it will make the recordIterator object, and you can use this recordIterator to read in all of the records in the file. That's more memory efficient for larger datafiles, and is an alternative to calling the readDataFile methods.

This makeRecord method is just a factory method to make a record object, an empty record object based on a record name.

00:17:40 screen changes to TaxwareFiles.xml

And that record name here will correspond to the record name here in the datafile definition.

00:17:46 screen changes to DataFile

Okay alternatively you can make these records and add records to the data file, and when you have a record you can also make records to be children of that record, and add the parent to the datafile. So when a parent is added all of its children are inherited or implicitly added. And then if you populate all these records that way, then you can just use the write methods.

This one will write it out to a string that sits in memory, this will write it to an outputStream, if you already have a file outputStream open, or whatever, however you're outputting it. These streams could be used for an HTTP connection, like the input stream coming from an HTTP request, an output stream going to an HTTP response, that sort of thing. Or you could write it to a filename.

Notice there isn't anything to write it to a generic URL, because typically URLs are very easy to open for reading, but opening for writing if it's an HTTP URL is a little bit more tricky.

00:19:13 screen changes to Class Record

Okay so that's basically the datafile API.

00:19:20 screen changes to Package org.ofbiz.datafile

For related objects and for information on modelDataFile and modelField and modelRecord you can look at these things. But these objects are just populated from the XML files for the datafile definitions.

00:19:43 screen changes to Documentation for DataFiles

Okay this is a little bit of generated documentation for the datafile itself, so we can look at some of these elements.

00:19:56 screen changes to TaxwareFiles.xml

So for datafile we saw the name, the type code is also required, as we saw in the example over here. It's kind of silly that it's required in a way. Specify a value but it's ignored if there's only one kind of a record and stuff. Actually it's (mumble(, so for the record type code it's ignored if there's only one type of record.

Type-code for the file is just for kind of management purposes and stuff.

00:20:26 screen changes to Documentation for datafiles- datafiles.xsd

Okay sender and receiver are for documentation purposes. The delimiter to use between records, or alternatively the record-length. And then separator-style, we talked about these different separator-styles: fixed-length, fixed-record, and delimited, and description is just a free form description.

And they can have one to many records designated for each data file. The record itself has a name, optionally a type-code, so if we do have a type-code to identify records there are a number of these fields that are used for identifying it. The most important ones are the tc-position (type code position) and the tc-length, so where in the record to look for the type-code to find it. And it will look at that position and length and compare it to the type-code specified here, and if it matches then we know it's this type of record.

So we have the parent-name for the parent-record-type, general description, limit is one or many, so do we expect to find just one of this type of record or many of them. And then each field will always have a name, if it's a fixed-width

record or a fixed-width field then you'll have a position and length for each one. Each field will always have a type.

00:22:07 screen changes to TaxwareFiles.xml

So for the example we can look over here, the type is the object type that is associated with the field.

00:22:14 screen changes to datafiles.xsd

Format is for converting a string to dates or numbers or whatever.

Valid-exp (validation expression). So you can tell it whether to validate with the expression or not with the true or false here. With the valid-exp it'll basically be used to validate the incoming or outgoing data, to make sure it follows the proper patterns. If not you get lists of error messages, as with other things.

General description, default-value-if which will be inserted if there nothing in the datafile for reading and such, whether it's a primary key field (prim-key), and then this is a simple shortcut for ignoring it (ignored).

Okay and that's basically the structure of the DataFile XML files.

00:23:18 screen changes to Advanced Framework Training Outline

And that brings us to the end of the section on the Data File Tool.