
The Best Kept Secret of The Best Kept Secret: Part 2

DirectToWeb. Any new words come to mind? Maybe, really-flexible -framework-for-creating-dynamic-WOComponents? No, probably not yet. So for the observant reader, what is the really cool part of DirectToWeb that we looked at in Part 1? Easy. DirectToWeb allows for the reuse of the most important components of a project .... the ones the graphic design artists spent hours and hours designing. In a normal WebObjects application every time you want to display the contents of an EOEnterpriseObject or a list of objects, you normally have to develop a new WOComponent , get the look and feel right, get all of the properties showing up correctly, set date and number formatters, etc. Time consuming. If now you want a limited view, no problem just hack in a WOConditional here and there, and presto limited view. Now what about the customer service agent who needs to also look at this EnterpriseObject? No problem, just add a few more WOConditonals ... you get the idea. Now you have this Frankenstein component that so heavily depends on user state and user type that the only way you can possibly reuse this component is by cutting and pasting bits and pieces into other components. Sound familiar?

So, how does DirectToWeb help you solve this problem? Simple, for the style of pages that you find yourself using over and over again (inspect pages, tab inspect pages, wizard creation pages, list pages, edit pages, ect) DirectToWeb provides an extremely robust framework for reusing the part that requires the most work, the look of these templates. In the first article we looked at how the DirectToWeb List interface and embedded D2WList component could be used with a custom template. A custom template being a generic WOComponent that can be customized to a certain look and feel while at the same time retaining its extremely dynamic nature. In the list example we just displayed a list of property keys that just spit out what was in the database. I'll be the first to admit that really without being able to interact with the list of objects, a list for the sake of a list is not very exciting, nor very useful. In this article we will explore how to get our own components into the list and then into a DirectToWeb inspect page, as well as using a custom template for inspecting. If you haven't read the first article on DirectToWeb located here, I would highly suggest you take five minutes and at least skim over it as most of what will be presented in this article directly builds upon concepts introduced in the first article.

Common DirectToWeb Questions

After the first article went up on Stepwise I received a number of questions and comments. Thank you very much for the wonderful feedback and please keep it coming! Before moving into the heart of this article I thought I would spend a bit of time going over a few of the more common questions I received.

How does DirectToWeb render a page? The way DirectToWeb renders a page is actually fairly straight forward. The heart of DirectToWeb is the rule engine. It is the rule engine's responsibility to answer questions. So, let's imagine that we have a page configuration named 'foo.' This page configuration has a task (list), an Entity (Bar) and an array of displayPropertyKeys((barName)). The object that embodies the rule engine in DirectToWeb is the D2WContext. The way the D2WContext answers questions is through key-value coding. So imagine we want to know what the displayPropertyKeys are in the page configuration 'foo.' This bit of code could find that out for you:

```
D2WContext myContext = new D2WContext();
myContext.takeValueForKey("foo", "pageConfiguration");
NSArray displayPropertyKeys = (NSArray)myContext.valueForKey("displayPropertyKeys");
```

So how did the D2WContext know to return the correct array given just a pageConfiguration? Easy, the D2WContext resolved a few rules. Rule resovling is fairly straight forward, in the above example a key 'pageConfiguration' was set to a value 'foo' on the D2WContext, next a question was asked, essentially "What are your displayPropertyKeys?." To answer this question the D2WContext first asks itself "What are all the possible rules I have that will answer the above question?" Given that rule set the D2WContext then sorts the rules by priority (that's the number on the right hand side in Rule Editor), once the rules have been sorted by priority the D2WContext returns the rule with the highest priority (it also caches this result so that

if asked again it won't have to perform the sort all over again). Given a tie, meaning two rules hit the criteria and both have the same priority the D2WContext will choose the most specific rule, ie the rule with the most cluases. These are the basics, don't worry if it doesn't make perfect sense right now, with a little playing around I the above should become more clear. If you would like a more detailed exaplanation of how DirectToWeb resolves rules I would refer you to the DirectToWeb article "Developing WebObjects Applications With DirectToWeb" in the WOInfoCenter. Returning to the above code snippet, if you look in the DemoListTemplate WOComponent from the first article (also used in this article), you will find that the WORepetion that displays the array of displayPropertyKeys for a given list page configuration has the following bindings:

```
list = d2wContext.displayPropertyKeys
item = d2wContext.propertyKey
```

In this way the repetition will be enumerating through the displayKeys and setting the propertyKey on the context to the current displayPropertyKey. We will cover more of the D2WContext object in another article. Also note that you should never have to programmatically create a D2WContext, DirectToWeb handles all that for you.

Performance. What about performance of DirectToWeb? In WebObjects 4.5 DirectToWeb has been tuned and optimized so that dynamic pages only take about 25% longer to render than a regular WebObjects component of the same complexity. So the real question you should be thinking is gee, how did they make DirectToWeb so fast? Simple, the D2WContext caches rules as they fire. So the first time DirectToWeb renders a page, that page will render slower than any other time that page is rendered. The rule caching mechanism and performance tuning will be covered in a later article. For now just know that once the rule cache has "warmed up", to use the EOF term, page rendering is not noticeably different compared to a static WebObjects page.

How are DirectToWeb and DirectToJavaClient related? Good question. DirectToJavaClient is built upon the same underlying rule engine as DirectToWeb, so if you are interested in learning more about how DirectToJavaClient's rule engine works, here is a pretty good place. Where DirectToJavaClient differs significantly from DirectToWeb is that instead of spitting out html it spits out xml, which is used to dynamically build the interface to an applet. DirectToJavaClient is an incredible piece of technology which really pushes the limits , but that's a story for another time ;)

Custom Components. How can I get custom logic into a dynamic page? Good question. First we need to differentiate bewteen custom property level components and custom page level components, becuase in DirectToWeb we hvae both. I custom page level component is referred to as a template and is used to add a custom look to an entire page. This was demonstrated in the first article and is extended in this article. The second type of custom component is a property level component. These are the components that are used to display a particular propertyKey or attribute. Property level custom components are the topic of this article, so keep reading to learn about all these wacky and wild components. If you have any other questions on DirectToWeb, please send them on, and I will try to answer them in a future article. Before reading further, it would be a really good idea to download the example project. It contains all of the code, rules and links to pages referenced below.

Custom Property Level Components in DirectToWeb

In the context of DirectToWeb there are two different kinds of property level custom components. A custom property level component being a component that is used instead of the normal property level display component, i.e. instead of a D2WDisplayString or a D2WDisplayDate. Recall that in the list example from the first article all of the attributes being displayed were either a string or a date. In this case DirectToWeb was using the default components to display this information, i.e. D2WDisplayString and D2WDisplayDate. Now imagine that we don't want the name attribute of the Movie to show up as just a String, but instead we want the name to be a hyperlink that takes us to a page that displays all of the information about that movie. How would we do that? With a custom component of course.

The first kind of custom component is a direct subclass of D2WComponent. These components are only

available within a DirectToWeb page, for example the D2WDisplayString, a subclass of D2WComponent, cannot be used outside the context of a DirectToWeb page. These types of custom components are generally used to create new ways of editing attributes. These will be covered in a future article about EnterpriseObject editing and creation. The second kind of custom component is a plain vanilla subclass of WOComponent. When used with DirectToWeb this component will be handed an object and a key through its bindings. With these two bits of information the component can do whatever it likes. I usually like to make these components non-synchronizing and stateless, seeing as they can be used many times on a given page. If you don't make your custom component non-synchronizing then you will need to have a variable of the type EOEnterpriseObject named 'object' and a String variable named 'key' in your component. Let's actually look at the code from a custom component named LinkToViewMovie1.java taken from the D2WInspectExample project:

```
public class LinkToViewMovie1 extends WOComponent {

    public boolean isStateless() { return true; }

    public Object object() { return valueForBinding("object"); }
    public String key() { return (String)valueForBinding("key"); }

    public Movie movie() {
        return (Movie)(object() instanceof Movie ? object() :
object().valueForKeyPath(key()));
    }

    public WOComponent view() {
        MyD2WInspect1 view = (MyD2WInspect1)pageWithName("MyD2WInspect1");
        view.setMovie(movie());
        view.setBackPage(context().page());
        return view;
    }
}
```

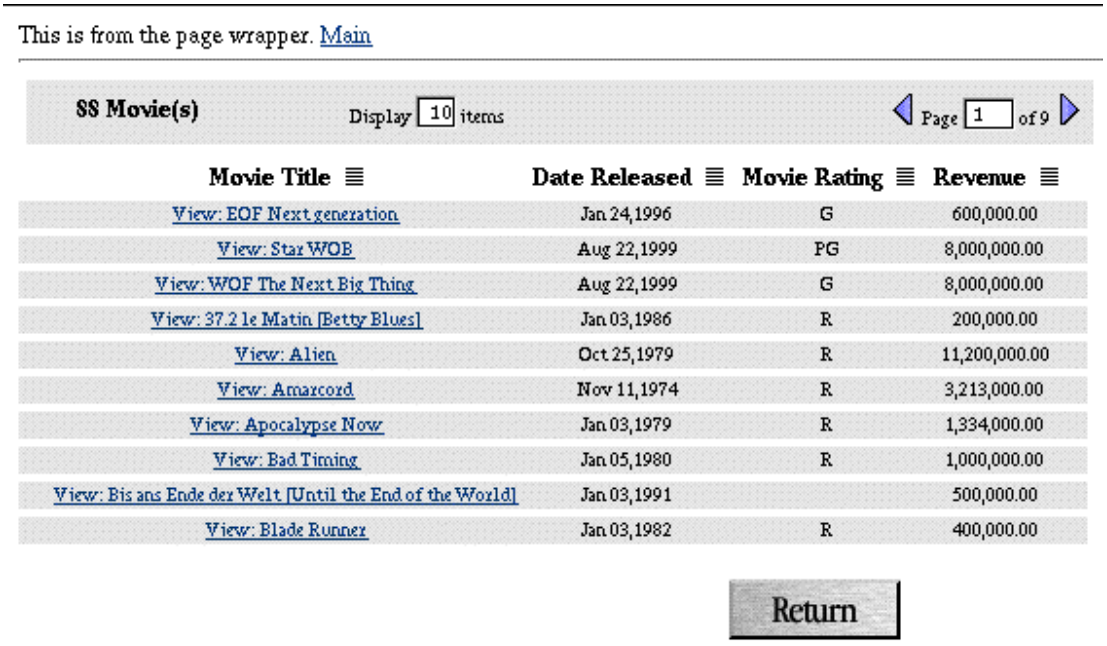
Looking at the .wod of this component we can see that the only function of this component is to display the movie's name in a hyperlink with the action method bound to the view() method . So how do we get this component into the DirectToWeb list? With a rule of course. Well actually with two rules, they are:

```
(pageConfiguration = 'ListMovies1') and (propertyKey = 'title') => componentName =
"D2WCustomComponent"

(pageConfiguration = 'ListMovies1') and (propertyKey = 'title') => customComponentName =
"LinkToViewMovie1"
```

Tiny bit of review on the DirectToWeb rule system: All of the rules for a project are found in the DirectToWeb model file, named user.d2wmodel in the example project. To edit/create and find rules we use the RuleEditor application. The 'Find' command is your friend. Want to see all the rules associated with the propertyKey title? Type 'title' in the find box and presto, every rule that has 'title' in it appears in the list. Note that this will find all rules that have title in either the lhs (left hand side, the part of the rule before the => above) or the rhs (right hand side, the part after the =>).

The two rules above tell DirectToWeb two things, first when displaying the propertyKey 'title' I want to use a custom component as oppossed to the built-in D2WDisplayString component and second, the custom component's name is 'LinkToViewMovie1.' The generated list page will now look like this:



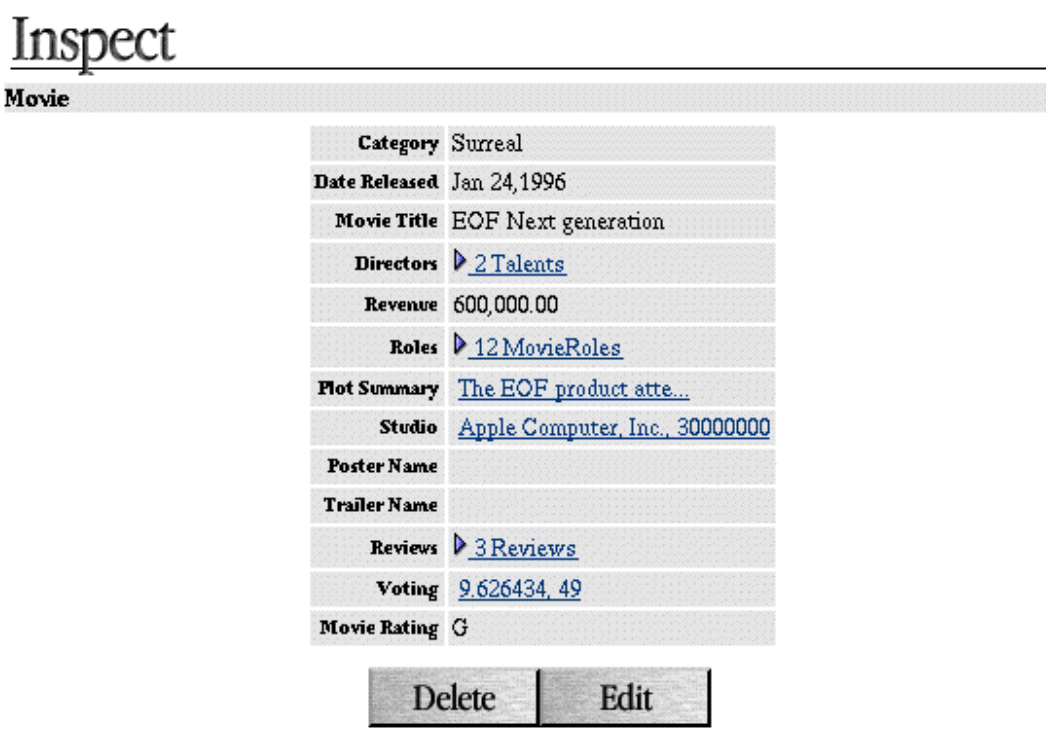
As you can see in the above list the movie's title is encapsulated with a hyperlink. Also note that the name at the top of the column displays 'Movie Title,' instead of Title which is the name of the attribute in the EOModel By default DirectToWeb beautifies the attribute's name, so 'title' would become 'Title' and 'movieRating' would become 'Movie Rating.' Often however (as anyone who has ever had to work with a content team can attest to) the name in the EOModel will not be the name that you would like the users of the application to see. In this case the display name can be specified with a rule, like so:

```
(entity.name = 'Movies') and (propertyKey = 'title') => displayNameForProperty = "Movie Title"
```

Notice that this rule does not include a pageConfiguration in the lhs of the rule. The scope of this rule is not tied to a single pageConfiguration, i.e. a single dynamic page. Instead this rule will apply anytime you are inspecting, listing, editing or searching on an object whose entity name is 'Movie.' This is a very, very powerful concept and the main reason why you want to have a rule system instead of a plain lookup mechanism or code. For example imagine you wanted to build two build a suite of applications and have the Movie entity's title appear as 'Movie Title' everywhere it is used, except when listing Movie entities or when you are in the customer service application. Building up this logic with a few rules is pretty simple, but trying to implement the above case in code would be non-trival and probabely require constant fiddling everytime you wanted a different behaviour. The DirectToWeb way is the exact opposite, provide a default general rule and then override that rule in specific pageConfigurations or applications. Gernally when writing rules for a large project you will have two rule files, the rule file at the business logic layer and a rule file at the application layer. A good example of a rule that belongs in the business logic layer or framework is the above rule. Once that rule is in place everytime the title attribute of a Movie object is displayed it will be displayed as 'Movie Title' instead of 'Title'. A good example of a rule that belongs in the application layer is any rule that deals with a particular pageConfiguration. Remember a pageConfiguration is the equivalent to a static WOComponent's name, so in most cases you wouldn't place specific WOComponents in the business logic layer, likewise rules that correspond to a particular pageConfiguration mostly belong in the application rule file. To see the above rule in action click on one of the movie title's hyperlinks from the above list. The component returned from the view method above should look like this:

This is from the page wrapper. Main

Plain vanilla D2WInspect



Back

Notice that the attribute 'title' is still getting displayed as 'Movie Title' from the display rule above. This can be extremely handy, especially if you have a fickle content team ("We realize you just changed the entire site to be Movie this and Movie that, but well we decided that we really like the word Film better ..."). From the above screen shot you can get a feel for the plain vanilla D2WInspect component which is embedded in the component '`MyD2WInspect1`'. The D2WInspect component is very similar to the D2WList component, i.e. you give it an object, an entity and optionally a pageConfiguration and it produces a D2WInspect page inside your WOComponent. Notice how easy it is to move between DirectToWeb dynamic components (the page with the list in it) and regular WOComponents ('MyD2WInspect1' is a regular WOComponent) by using custom components.

Let's jazz things up a bit, actually let's jazz things up a lot. In the list page let's add another column that lists the number of reviews of that movie with a hyperlink to view the list of reviews. To show that the display keys of the list don't actually have to correspond to attribute keys, the key added to the display keys will be 'dummy,' likewise let's add a corresponding displayNameForProperty so that 'Dummy' won't show up at the top of the column. Let's also format the movie revenue number as '$ #,##0.00.' In the movie inspect page let's display the relationship to the plot summary just like a normal attribute, and let's display a table of the directors (a flattened relationship) showing each director's first and last name. Plus let's add a link to write a movie review in the inspect page. To make things even more interesting let's use a custom inspect template instead of the built-in template (just like we are currently using for the list pages). Plus let's do all this with dynamic pages (no D2WInspect or D2WList components embedded in normal WOComponents), three custom components and 31 rules in less than nine minutes. Ya think it's possible? I do. The final product will look like this:

Movie List Page:

This is from the page wrapper. Main

88 Movie(s) Display 10 items Page 1 of 9

Movie Title	Date Released	Movie Rating	Revenue	Number of Reviews
View: EOF Next generation	Jan 24,1996	G	$ 600,000.00	3
View: Star WOB	Aug 22,1999	PG	$ 8,000,000.00	1
View: WOF The Next Big Thing	Aug 22,1999	G	$ 8,000,000.00	No Reviews
View: 37.2 le Matin [Betty Blues]	Jan 03,1986	R	$ 200,000.00	No Reviews
View: Alien	Oct 25,1979	R	$ 11,200,000.00	No Reviews
View: Amarcord	Nov 11,1974	R	$ 3,213,000.00	No Reviews
View: Apocalypse Now	Jan 03,1979	R	$ 1,334,000.00	No Reviews
View: Bad Timing	Jan 05,1980	R	$ 1,000,000.00	No Reviews
View: Bis ans Ende der Welt [Until the End of the World]	Jan 03,1991		$ 500,000.00	No Reviews
View: Blade Runner	Jan 03,1982	R	$ 400,000.00	No Reviews

Return

Review List Page:

This is from the page wrapper. Main

2 Review(s)

Reviewer	Review
Max	It moves me.
Me	It is really neat

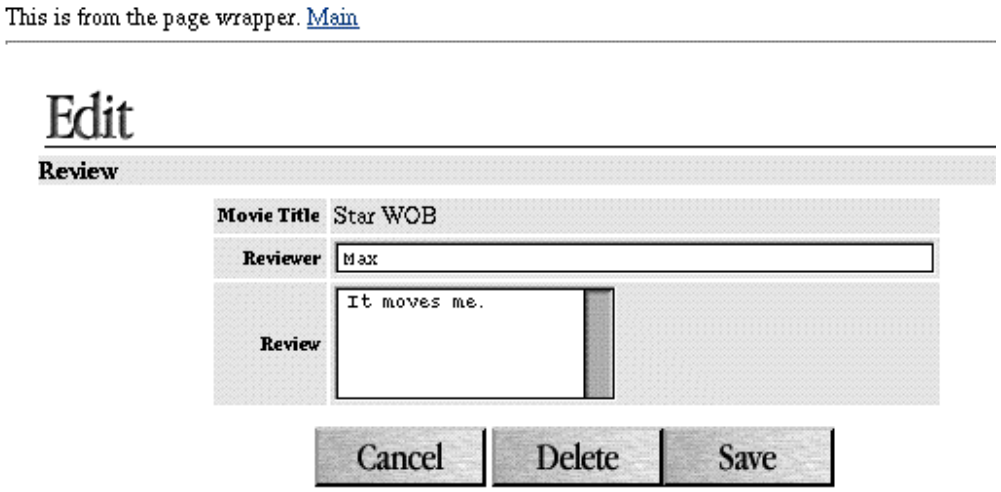
Movie Inspect Page:

This is from the page wrapper. Main

Movie Title	Star WOB
Directors	Patrice Gautier Toni Trujillo-Vian
Date Released	Aug 22,1999
Movie Rating	PG
Revenue	$ 8,000,000.00
Number of Reviews	1
Write A Review	Write a Review

Return

Write A Review Page:

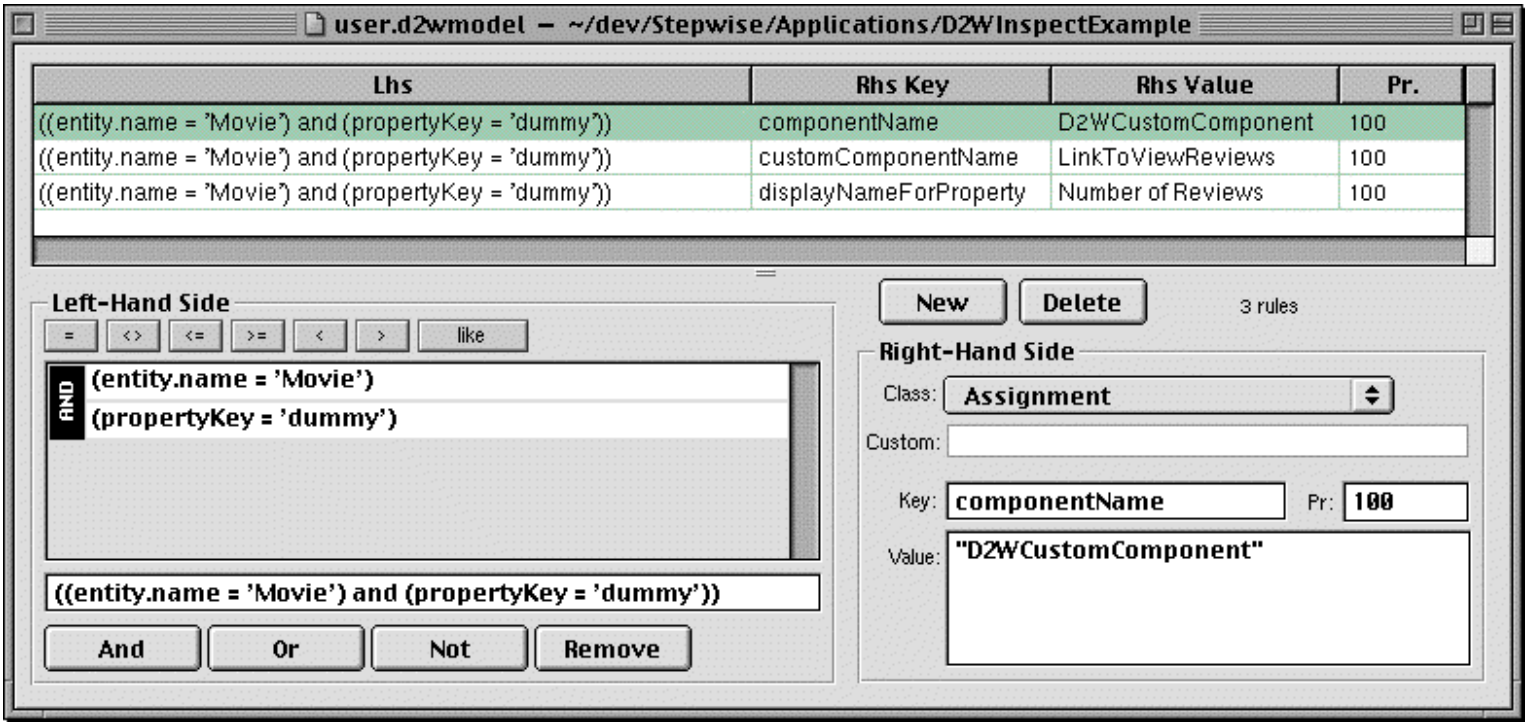


Note: when you try out the example project for the first time on your machine you will notice that the number of reviews shows up as "No Reviews" for all the movies. Little did I realize when designing the example that the OpenBase Lite Movies database did not include any reviews, thus the addition of the "Write A Review Page".

So where to begin? Let's start with the list page. It looks very similar to the first list page, except it has the added column 'Number of Reviews.' This is another custom component called 'LinkToViewReviews.' All it does is test if the number of reviews is greater than 0, if so it displays a hyperlink to another dynamic list page displaying the movie's reviews, if not it displays 'No Reviews.' As indicated above, the key in the displayPropertyKeys is 'dummy' and the corresponding componentName, customComponentName, and displayNameForProperty have all been set for the entity. They look like this:

```
(entity.name = 'Movie') and (propertyKey = 'dummy') => componentName =
"D2WCustomComponent"
(entity.name = 'Movie') and (propertyKey = 'dummy') => customComponentName =
"LinkToViewReviews"
(entity.name = 'Movie') and (propertyKey = 'dummy') => displayNameForProperty = "Number
of Reviews"
```

Note that these three rules correspond to the following screen shot in Rule Editor:



Notice also that in the inspect page we have included the key dummy in the displayPropertyKeys of the inspect page and sure enough all three of the above rules fire. Adding the formatter to the revenue attribute is really straight forward. Simply add the rule:

```
(propertyKey = 'revenue') and (pageConfiguration = 'ListMovies2') => formatter = "$
#,##0.00"
```

This formatter rule functions just like the WebObjects binding numberFormat or dateFormat on a WOString. So if you wanted all numbers being displayed to by default be formatted with the format "$ #,##0.00" you could add the rule:

```
(attribute.valueClassName = 'NSNumber') or (attribute.valueClassName =
'NSDecimalNumber') => formatter = "$ #,##0.00"
```

Note: This and many other high level hooks can be very handy for localizing applications, but that's a story for another time ;)

Notice here that we are using the key 'attribute' in the above rules. In the rule system, just as entity refers to the EOEntity of the object, attribute refers to the EOAttribute of the propertyKey if it is an attribute of the entity*. As you could probably infer, the rule system also has a key 'relationship', which corresponds to the EORelationship if the current propertyKey is a relationship of the entity. Notice also that the valueClassName in the above rules refers to the Obj-C class names, not the corresponding Java equivalent. Even though DirectToWeb is written entirely in Java, the rule system still uses the Obj-C class names and most importantly nil instead of null.

In the inspect page, everything is pretty straight forward. If you open the rules and perform a find for 'InspectMovie1'. The following set of rules should come up:

Lhs	Rhs Key	Rhs Value	Pr.
true	look	NeutralLook	100
(pageConfiguration = 'InspectMovie1')	task	inspect	100
(pageConfiguration = 'InspectMovie1')	entity	Movie	100
(pageConfiguration = 'InspectMovie1')	displayPropertyKeys	(title, directors, dateReleased	100
((propertyKey = 'revenue') and ((pageConfiguration = 'ListMovies2	formatter	$ #,##0.00	30
((pageConfiguration = 'InspectMovie1') and (propertyKey = 'direct	allowCollapsing	false	100
((pageConfiguration = 'InspectMovie1') and (propertyKey = 'direct	disabled	true	0
(pageConfiguration = 'InspectMovie1')	showBanner	false	100
true	isEntityEditable	true	0
(pageConfiguration = 'InspectMovie1')	pageName	DemoInspectTemplate	100
(pageConfiguration = 'InspectMovie1')	isEntityEditable	false	100
((pageConfiguration = 'InspectMovie1') and (propertyKey = 'writeA	componentName	D2WCustomComponent	100
((pageConfiguration = 'InspectMovie1') and (propertyKey = 'writeA	customComponentName	LinkToWriteMovieReview	100
((pageConfiguration = 'InspectMovie1') and (propertyKey = 'writeA	displayNameForProperty	Write A Review	100

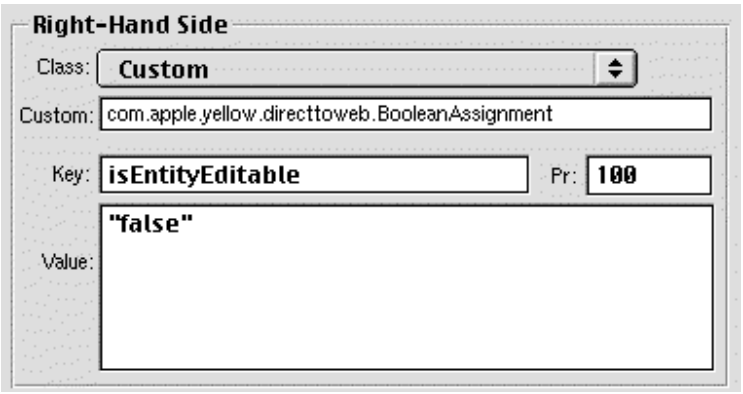
Note that we have specified the pageName in the rules as 'DemoInspectTemplate.' This is the hook we use to specify our own custom look and feel. The 'DemoInspectTemplate' is pretty much a plain vanilla inspect page, but note that you have complete control over the look and the feel. Want style sheet elements, easy as pie. Want graphic images instead of displayNames, sure no problem you have the template. Note that we have added one ability to the DemoInspectTemplate which is to disable the 'edit' button. To do this we add the default rule:

```
*true* => isEntityEditable = "true" (BooleanAssignment)
```

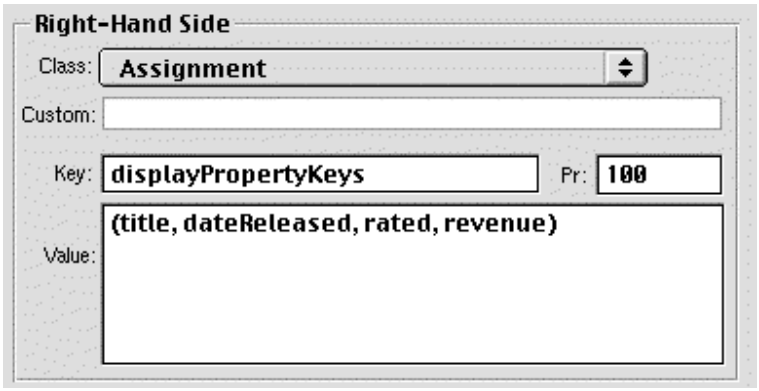
To turn it off in the inspect configuration used, 'InspectMovie1' we simply add the rule:

```
(pageConfiguration = 'InspectMovie1') => isEntityEditable = "false" (BooleanAssignment)
```

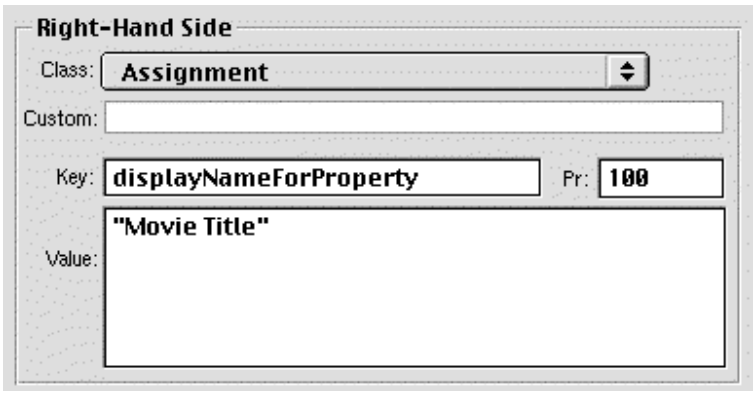
Notice that the above two rules use BooleanAssignment instead of the regular rule assignment. A screenshot of what this type of assignment looks like in Rule Editor:



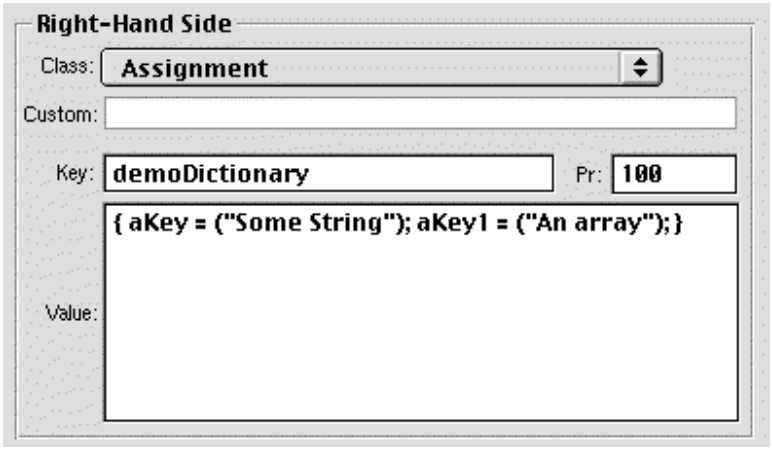
So what is the difference between the default assignment and using a custom assignment class? By default the value part of the key-value pair of the rule is just a plist that gets turned into a String, NSArray or NSDictionary. For example, the displayPropertyKeys rule in Rule Editor looks like this:



Whereas a rule for a displayNameForProperty will look like this:



Finally, if you wanted to have a rule that returns a dictionary it would look something like this:



So if the rule engine is asked for any of the rhs keys from above (displayPropertyKeys, displayNameForProperty and dictionaryDemo), either a String , NSArray or NSDictionary object will be returned. So how do the BooleanAssignment and EntityAssignment differ? Simple, by providing a custom Assignment class DirectToWeb is able to return other objects, like for example Boolean and EOEntity objects. Note however that due to the Java Bridge Boolean objects get turned into Java Integer objects. In the next article we will look at adding our own custom assignment classes which can be a very powerful extension of DirectToWeb. (This is actually really simple, if you can't wait until the next article check the WOInfoCenter Assignment class for further information).

Back to the custom inspect page and its table of directors. How was this done? With four rules of course! First note that in the displayPropertyKeys "directors" appears. This is the name of the flattened relationship to the Talent entity. The component that DirectToWeb uses by default to display to-many relationships is called D2WDisplayToManyFault. This component appears the same way it did in the plain vanilla inspect page (a collapsible component with the directors first name displayed wrapped in a hyperlink that takes the

user to another inspect page about the selected director). To create the look used on the above inspect page we simply tell DirectToWeb to use another component. We could have written our own custom component to display the list of directors first and last name in a table, however this would not be a very reusable approach. Instead what we want is a generic way to display a to-many relationship in a table. Well it turns out that DirectToWeb has a component named D2WDisplayToManyTable that does exactly what we need. To use this component to display the directors we need to specify four rules. They are:

```
(pageConfiguration = 'InspectMovie1') and (propertyKey = 'directors') => componentName =
"D2WDisplayToManyTable"

(pageConfiguration = 'InspectMovie1') and (propertyKey = 'directors') => disabled =
"true" (BooleanAssignment)

(pageConfiguration = 'InspectMovie1') and (propertyKey = 'directors') => allowCollapsing
= "false" (BooleanAssignment)

(pageConfiguration = 'InspectMovie1') and (propertyKey = 'directors') =>
keyWhenRelationship = "fullName"
```

The first rule tells DirectToWeb to use the component D2WDisplayToManyTable to display the relationship directors. The second rule 'disabled' tells DirectToWeb to not place a hyperlink to the director inspect page around each director. By setting the 'allowCollapsing' to false this tells DirectToWeb to not use a collapsible component around the D2WDisplayToManyTable component. The last rule, keyWhenRelationship, tells DirectToWeb what to display for each director. Note that fullName is not in the model, instead it is a method off of Talent that simply returns firstName() + " " + lastName().

As a preview of things to come in the movie inspect page the key writeAReview was added to the displayPropertyKeys. This key is used to get the custom component 'LinkToWriteAReview' into the inspect page. This component displays the text "Write A Review" in a hyperlink that when clicked creates a Review object and then takes the user to a regular DirectToWeb edit page. Let's take a quick peak at the code that creates the review object and returns the DirectToWeb edit page. Here it is:

```
   public WOComponent writeReview() {
        // This will be a peer of the session's default editingContext.
        EOEditingContext peer = new EOEditingContext();
        // The below methods would be better suited in the business logic.  But for
simplicity it it here.
        EOClassDescription cd =
EOClassDescription.classDescriptionForEntityName("Review");
        EOEnterpriseObject review = cd.createInstanceWithEditingContext(peer,null);
        peer.insertObject(review);
        Movie localMovie = (Movie)EOUtilities.localInstanceOfObject(peer, movie());
        if (localMovie == null)
            throw new RuntimeException("Attempting to create a review for a null
movie.");
        localMovie.addObjectToBothSidesOfRelationshipWithKey(review, "reviews");
        EditPageInterface epi =
(EditPageInterface)D2W.factory().pageForConfigurationNamed("CreateReview", session());
        epi.setObject(review);
        epi.setNextPage(context().page());
        // Need to make sure that the ec survives, ie after the
EOUtilities.localInstanceOfObject method the ec could be released.
        peer.hasChanges();
        return (WOComponent)epi;
    }
```

As you can see, it's all pretty straight forward. A peer context is created so that if the user hits the back button then s/he is not left with a dirty editing context. The page configuration "CreateReview" has the following four rules:

```
(pageConfiguration = 'CreateReview') => task = "edit"
```

```
(pageConfiguration = 'CreateReview') => entity = "Review" (EntityAssignment)

(pageConfiguration = 'CreateReview') => displayPropertyKeys = "(movie.title, reviewer,
review)

(pageConfiguration = 'CreateReview') and (propertyKey = 'movie.title') => componentName
= "D2WDisplayString"
```

Again, all pretty straight forward stuff. Note that by using a D2WDisplayString for the propertyKey 'movie.title' we have effectively made this attribute read-only.

Recap

So what the heck did we just cover? In a nutshell we looked at how to use custom components in either a DirectToWeb list or inspect page. We also explored more of how the rule system works, in particular we looked at formatters, display names, EOAttributes and Boolean assignments. We also looked at how to use a custom inspect template with DirectToWeb. In a preview of things to come we showed how to create objects and edit them with the standard DirectToWeb edit page. In the next article we will be looking more at object creation, editing and validation as well as custom assignment classes. Slightly more impressed with DirectToWeb yet? If not at least give me the benefit of the doubt and read the next few articles because believe me the best is yet to come!