What is Direct to Web?

What is Direct to Web?

by Ramsey Gurley

When I first started learning about Direct to Web (D2W) I had a few misconceptions about what it really was. Since then, I've encountered other WebObjects programmers with some of the same misconceptions that I had. Given the relative obscurity of D2W, that seems understandable. To address this, the first question we should cover is: What is D2W?

D2W is to WebObjects apps as the assembly line is to automobiles. Without D2W, WebObjects components are, for the most part, custom built. They are hand tooled to fit a specific need. If at any time you make a change to your data model, you have to go back into your pages and update them to make use of those changes. At a minimum, you will likely need a page to inspect, edit, list, and query each entity in your model. If you add one relationship between two entities, you will need to retool eight pages! This is not an efficient way of doing things.

By contrast, a D2W app produces new pages for you whenever you update the model. There is no need to hand tool pages anymore. In this respect, understand that D2W is not just a framework or a finished product. It is not an end unto itself, it is a means. It is a programming methodology. With it, you provide customization directions (D2W rules) to the assembly line (D2W) to build your end products (WOComponent pages). Carrying the analogy further, D2W allows you to be the Production Manager of an automated assembly line rather than the Production Technician doing the hand-build of the product all by yourself.

Consequently, the D2W framework provided by Apple or available in frameworks like Project Wonder are not so much a solution as they are a starting point. No matter how complete a D2W framework happens to be, you'll eventually run into a case where you need a custom component. When that time comes, you will need to do more than use D2W. You will need to understand how D2W works so you can extend it.

So now that we've covered what D2W is, we should know how a D2W app is built. Apple has documentation on D2W available at this page. While I still recommend you read that document, parts of it are a bit out of date or not terribly relevant. What follows here is a condensed version.

What are the major parts of a D2W app?

D2W applications have several significant parts that I will discuss in detail. Those common to every D2W app are the D2W factory, the D2WContext, page components, property level components, assignments, and rules.

The D2WContext and the D2W factory

The D2WContext is the centerpiece of D2W programming. It is the hub. All roads lead to the D2WContext. It's pretty important 😃 But it's also pretty simple to use. It is basically a NSMutableDictionary containing a list of key/value pairs. It is typically created by the D2W factory and given a few initial values for keys. If you request a value for a key that is not available in the D2WContext, it will infer the value of the key from the rule system.

That's quite a mouthful, so let's look at an example. Consider the following method from the D2W factory:

```
D2W.factory().listPageForEntityNamed(String entityName, WOSession session)
```

When you call this method, the returned object is a page component that conforms to the ListPageInterface. To do this, the factory initializes a D2WContext. The factory then sets the value for the D2WContext's 'task' key to "list" and the 'entity' for the page to the entity named by 'entityName'. With that information in place, the factory can then ask the D2WContext for the name of its page component! The D2WContext now has enough information to consult the rule system, and return the page component name needed to finish the method. Once the factory gets a name, it can call pageWithName, assign the D2WContext to the page, and return the result. Wow, that's pretty cool! But you're probably wondering, how exactly did the D2WContext know what page name to return? What is the rule system?

The rule system

For more in depth treatment of this topic see The D2W Rule System

The rule system is made up primarily of two parts: Rules and Assignments. A rule file is basically a configuration file. You will find your rules stored in the .d2wmodel files of your D2W application. Rules are commonly represented textually as:

```
Priority: LHS => RHS Key = RHS Value [Assignment]
```

You can see, there are five parts to every rule for us to discuss.

RHS Key

Shorthand for "right hand side" key, this is the actual keypath the D2WContext is attempting to infer. If for example, we were to call

```
d2wContext.valueForKey("pageName");
```

Then the rule system would take all the rules in your rule files with a RHS Key equal to "pageName" and begin to evaluate them in the order of their priority.

Priority

The priority of a rule determines the order in which it is evaluated. Higher priority rules are consulted before lower priority rules. A rule with priority 0 is the lowest priority rule. Convention is that rules of priority of 0-10 are reserved for framework fallbacks, 11-99 are used by the framework for overrides, rules from 100-105 were used by the web assistant in the user.d2wmodel. Higher values obviously continue to override lower values.

LHS

Shorthand for the "left hand side." The LHS is basically a boolean evaluation. It's like an if statement.

```
if(LHS) { fire this rule }
```

Unlike java though, the evaluations are typically keyPaths and can look a little strange when you first see them. For example,

```
smartRelationship.isToMany = '1'
```

fires the rule if the valueForKey("isToMany") for the object assigned to the key 'smartRelationship' is true. LHS can also contain wildcards. For instance,

```
pageConfiguration like 'List*'
```

fires the rule if the pageConfiguration is 'ListPuppies' or 'ListKittens'. You can also have compound rules using and, or, & not:

```
((task = 'list' or task = 'select' or task = 'inspect') and propertyType = 'r' and not (smartRelationship.
isToMany = 1))
```

If a rule does not fire, then the next lower priority rule is tested. Once a rule fires, evaluations stop.

RHS Value and Assignment

I discuss these two together because they are basically inseparable. Once a rule fires, a value is assigned to the RHS key in the D2WContext. Assignment classes take the value and coerce it into the actual value assigned to the key. For instance, if we have a RHS Value that equals "false", how do we know what will be assigned to the RHS Key? Is the string "false" assigned literally? Or is the key given a boolean value of false? That depends entirely on the assignment you choose. The default assignment class (com.webobjects.directtoweb.Assignment) will assign the string value, where the boolean assignment class (com.webobjects.directtoweb.BooleanAssignment) will coerce the string and assign a boolean value instead. Of course, there are many different useful assignment classes, and you can write and use your own assignment classes as well.

Now that you understand the rule system, you can imagine how in the above example, the D2WContext finds the name for the page with nothing but values for task and entity in its dictionary. The rule that returns the page name might look something like:

```
100: (task = 'list' and entity.name = 'Kitten') => pageName = "MyAwesomeD2WCustomListPage" [com.webobjects.
directtoweb.Assignment]
```

The factory would then call pageWithName("MyAwesomeD2WCustomListPage"), binds the D2WContext to result, and return the resulting WOComponent found in MyAwesomeD2WCustomListPage.wo. You can now begin to see how D2W allows you to construct apps in an extremely dynamic way. Rather than hardcoding bindings in the WOComponents, your D2W pages are actually being assembled at runtime. To keep the app peppy, RHS Keys are cached by the D2WContext so that it does not need to continually re-evaluate rules. How it does this is largely dependent on the D2W framework you are using.

Pages and property level components

Now that the D2W factory has initialized the page, you have a page component that contains your d2wContext. Since the d2wContext is ultimately just a fancy NSDictionary, it is used to pass values around during the construction of the page. In a typical WebObjects page there are quite a lot of component bindings. In a D2W WebObjects page, the D2WContext is where the binding values are stored. Continuing with our list page example... Somewhere in the course of your list page, you are going to end up at a WORepetition that iterates through the enterprise objects in your page's displayGroup. For each enterprise object, you'll have another WORepetition that iterates through a list of the object's attributes and relationships (the property keys) that you would like displayed. For that second Repetition, your WOD bindings might look something like:

```
PropertyKeyRepetition: WORepetition {
        list = d2wContext.displayPropertyKeys;
        item = d2wContext.propertyKey;
}
```

You can see the WORepetition is iterating through the d2wContext's displayPropertyKeys key and then assigning the value of each iteration into the d2wContext's propertyKey key. Once a value is assigned to the propertyKey key, you would then begin the process of creating the component to represent the values contained by that object's property... the value of which **is also** assigned dynamically by the D2WContext:

```
PropertyKeyComponent: WOSwitchComponent {
        WOComponentName = d2wContext.componentName;
         localContext = d2wContext;
         object = object;
}
```

You can see the component that displays the property value for the object isn't even in the page! It is assigned dynamically at runtime by the d2wContext. The d2wContext can infer this value just like it inferred the value for the page name. Because the values of these components are not hard coded into the pages, you can now see how a D2W application dynamically updates itself to reflect changes you make in your data model and in the rule system. With D2W, you can change hundreds of bindings for many different WOComponents with a single rule. You can also change out entire components based on what your application logic demands in a much more flexible and reusable way. As a result, you can assemble applications much more rapidly with only a limited amount of coding for custom components and flow control.

Related Links

Apple's D2W documentation
rule modeler