Code Reuse: Taking Advantage of Includes and Overwrites in Transfer

Posted At : November 7, 2008 1:41PM | Posted By : Matt Quackenbush
Related Categories: ColdFusion, Transfer

Code reuse is something that I am very passionate about, and I go out of my way to think through the various intricacies of my applications looking for ways to make my code more reusable. Some might even say that I'm a bit obsessive-compulsive about it. :D

Today I want to brag on Transfer , and show a couple of awesome features that Mark Mandel has built into Transfer that allow OC guys like me to have our cake and eat it too.

The <include /> Tag

I would assume that most people that use Transfer are aware of the <include /> tag. For those of you who aren't, let me give a quick summary of what it does for you. When working with a larger scale application, your Transfer config file might end up being rather lengthy. The larger it becomes, the more cumbersome it is to edit. To make things easier, you can break your packages out into separate config files, and then include each one into the main application config.

For the sake of demonstration, let's say that we've taken our User Management package, and placed it into its own config file. It might look something like this:

/config/transfer.userpackage.xml.cfm

```
<?xml version= "1.0" encoding= "UTF-8" ?>
<transfer xmlns:xsi= "http: //www.w3.org/2001/XMLSchema-instance" xsi:noNamespaceSchemaLocation= "transfer.
xsd" >
<objectDefinitions>
<package name= "user" >
<object name= "User" table= "tblUser" decorator= "com.User" >
<id name= "ID" column= "userID" type= "numeric" />
<property name= "UserName" column= "user_name" type= "string" />
<property name= "Password" column= "user_password" type= "string" />
<property name= "Email" column= "user_email" type= "string" />
<manytoone name= "Role" lazy= " false " proxied= " false " >
<link to= "user.Role" column= "user_id" />
</manytoone>
</object>
<object name= "Role" table= "tblUser_Role" decorator= "com.Role" >
<id name= "ID" column= "user_roleID" type= "numeric" />
<property name= "RoleName" column= "role_name" type= "string" />
<property name= "Description" column= "role_desc" type= "string" />
</object>
</package> <!-- END: user -->
</objectDefinitions>
</transfer>
```

Nothing fancy there, it's just a standard Transfer config file, defining our 'user' package, which contains two object definitions: 'User', and 'Role'. We want to use this in our application, so we open up the application's main Transfer config, and add it in by using the <include /> tag.

/config/transfer.application.xml.cfm

```
<?xml version= "1.0" encoding= "UTF-8" ?>
<transfer xsi:noNamespaceSchemaLocation= "http: //www.web-relevant.com/transfer/transfer.xsd" xmlns:xsi= "
http://www.w3.org/2001/XMLSchema-instance " >
<includes>
<include path= "/config/transfer.userpackage.xml.cfm" />
</includes>
<objectDefinitions>
<!--- object declarations here --->
</objectDefinitions>
</transfer>
```

Another Code Reuse Tip: Mappings

Mappings in ColdFusion have been around for a long, long time now, but CF8 has added per-application mappings , which is, in my opinion, one of the greatest enhancements to CF8. I'm guessing that most of you are already well aware of mappings, but maybe you haven't thought of using them in this manner.

We already have our User Package broken out into its own Transfer config file, and we're already using <include /> to utilize it within our application, but hey, we worked long and hard on our User Package, and we've made it so generic that it can be reused across multiple applications. Awesome! Let's then take advantage of mappings, and store that in a central location, so that we can use it in multiple locations.

Application.cfc

```
this.mappings = structNew();
this.mappings[ "/globalconfigs" ] = "{DriveLetter}:Globalconfig" ;
```

Now we can update our <import /> with the new, central location.

/config/transfer.application.xml.cfm

```
<?xml version= "1.0" encoding= "UTF-8" ?>
<transfer xsi:noNamespaceSchemaLocation= "http: //www.web-relevant.com/transfer/transfer.xsd" xmlns:xsi= "
http://www.w3.org/2001/XMLSchema-instance " >
<includes>
<include path= "/globalconfigs/transfer.userpackage.xml.cfm" />
</includes>
<objectDefinitions>
<!--- object declarations here --->
</objectDefinitions>
</transfer>
```

"Great, Quack," you murmur, "but what happens if I need to tweak my User Package for a particular application?"

No worries, you can do that, too!

Code Reuse on Steroids: Overwriting Package and/or Object Definitions

Now we're *really* getting into the fun stuff! Overwriting is, in my opinion, one of the most overlooked - yet extremely powerful - features of Transfer. There are two ways to utilize overwriting in Transfer:

1. Create a package/object definition in the local file named the same as in the included file, and the local definition will take precedence
2. Set the overwrite attribute on the <include /> to 'true', and the included definition will take precedence

Option #1

So, let's say that the application we're working on needs to add a 'JoinDate' property to the User object. No problem, we just overwrite that object in our local config, like so:

/config/transfer.application.xml.cfm

```
<?xml version= "1.0" encoding= "UTF-8" ?>
<transfer xsi:noNamespaceSchemaLocation= "http: //www.web-relevant.com/transfer/transfer.xsd" xmlns:xsi= "
http://www.w3.org/2001/XMLSchema-instance " >
<includes>
<include path= "/globalconfigs/transfer.userpackage.xml.cfm" />
</includes>
<objectDefinitions>
<!--- object declarations here --->
<package name= "user" >
<object name= "User" table= "tblUser" decorator= "com.User" >
<id name= "ID" column= "userID" type= "numeric" />
<property name= "UserName" column= "user_name" type= "string" />
<property name= "Password" column= "user_password" type= "string" />
<property name= "Email" column= "user_email" type= "string" />
<property name= "JoinDate" column= "join_date" type= "date" />
<manytoone name= "Role" lazy= " false " proxied= " false " >
<link to= "user.Role" column= "user_id" />
</manytoone>
</object>
</package> <!-- END: user -->
</objectDefinitions>
</transfer>
```

Note that we used <include /> to bring in our global User Package, and then we defined a new User object in our local file, but we did *not* overwrite the Role object definition. Transfer automagically melds the two <package name="user"> definitions together, giving precedence to the definitions contained in the local config file.

Option #2

This time, let's use another <include /> to do our overwrite. First we'll create a local UserPackageOverwrite file:

/config/transfer.application.userpackageoverwrite.xml.cfm

```
<?xml version= "1.0" encoding= "UTF-8" ?>
<transfer xsi:noNamespaceSchemaLocation= "http: //www.web-relevant.com/transfer/transfer.xsd" xmlns:xsi= "
http://www.w3.org/2001/XMLSchema-instance " >
<objectDefinitions>
<!--- object declarations here --->
<package name= "user" >
<object name= "User" table= "tblUser" decorator= "com.User" >
<id name= "ID" column= "userID" type= "numeric" />
<property name= "UserName" column= "user_name" type= "string" />
```

```
<property name= "Password" column= "user_password" type= "string" />
<property name= "Email" column= "user_email" type= "string" />
<property name= "JoinDate" column= "join_date" type= "date" />
<manytoone name= "Role" lazy= " false " proxied= " false " >
<link to= "user.Role" column= "user_id" />
</manytoone>
</object>
</package> <!-- END: user -->
</objectDefinitions>
</transfer>
```

Now we'll include both the global User Package config and the local UserPackageOverwrite config into our application's config file:

/config/transfer.application.xml.cfm

```
<?xml version= "1.0" encoding= "UTF-8" ?>
<transfer xsi:noNamespaceSchemaLocation= "http: //www.web-relevant.com/transfer/transfer.xsd" xmlns:xsi= "
http://www.w3.org/2001/XMLSchema-instance " >
<includes>
<include path= "/globalconfigs/transfer.userpackage.xml.cfm" />
<include path= "/config/transfer.application.userpackageoverwrite.xml.cfm" overwrite= " true " />
</includes>
<objectDefinitions>
<!--- object declarations here --->
</objectDefinitions>
</transfer>
```

By adding the overwrite flag and setting it to true, we've accomplished the exact same thing as we did in Option #1.

IMPORTANT NOTES!

1. This is a ridiculously over-simplified example, and it would make no damn sense to use these techniques if your object packages are as small as the ones in this example. This example is purely meant to be educational.
2. One must give due diligence when planning to utilize these techniques. If not, you could end up overwriting things you don't want to.
3. I *highly* encourage the use of <package> in your object definitions. Not only does it help with the logical separation of application concerns, but it will also greatly assist in taking advantage of the overwrite features that Transfer provides.

Closing Kudos

I want to say a HUGE "Thank You!" to Mark for authoring such an invaluable tool. I also have to give another HUGE "Thank You!" to Mark for his incredible support and response to bug reports and feature requests. I am constantly amazed at his tireless efforts. Thanks, Mark, as always. :-)