

Contents

- [Writing Decorators](#)
 - [Authoring a Decorator](#)
 - [Overwriting methods](#)
 - [Access to Transfer](#)
 - [Access to the Datasource](#)
 - [Access to the Transaction](#)
 - [Setting up values in the Decorator](#)
-

Writing Decorators

A 'decorator' is used when you wish to write your own CFC to be used to represent data, in place of the Transfer generated Object.

An example of this being to write a User.cfc to represent a User record within a database.

The 'decorator' that has been written wraps around the TransferObject that is generated by Transfer and automagically extends any public method generated on the decorated TransferObject.

Authoring a Decorator

Decorators are set using the 'decorator' attribute on the 'object' element in the [Transfer Configuration File](#)

Decorators must extend [transfer.com.TransferDecorator](#)

Overwriting methods

If a method exists in the Decorator, it will overwrite the method of the generated [TransferObject](#)'s method.

If you wish to have access to the TransferObject stored within the decorator, and any of its original generated methods, you can use [getTransferObject\(\)](#) to return the original TransferObject

An example of an overwritten method that utilises these methods, is if a User TransferObject has a 'setHomePageURL()' method generated, that you wanted to ensure had '[http://](#)' at the beginning, you could put in your User.cfc decorator

```
<cffunction name="setHomePageURL" access="public" returntype="void" output="false">
  <cfargument name="url" type="string" required="Yes">
  <cfif findNoCase("http://", arguments.url)>
    <cfset arguments.url = "http://" & arguments.url>
  </cfif>
  <cfset getTransferObject().setHomePageURL(arguments.url)>
</cffunction>
```

This overwrites the generated setHomePageURL() method, and extends its functionality.

Access to Transfer

The method of [getTransfer\(\)](#) is available to the Decorator CFC, and gives access to the [transfer.com.Transfer](#) CFC.

Access to the Datasource

The method [getDatasource\(\)](#) is available to the Decorator CFC, and gives access to the [transfer.com.sql.Datasource](#) CFC.

Access to the Transaction

The method [getTransaction\(\)](#) is available to the Decorator CFC, and gives access to the [transfer.com.sql.transa](#)
[transfer.com.sql.transaction.Transaction](#) CFC.

It is advised that you use the [execute\(\)](#) method in the Transaction CFC to execute Transfer based Transactions, as the AOP [advise\(\)](#) methods are tailored more towards singletons.

Setting up values in the Decorator

[configure\(\)](#) on the TransferDecorator will be run when the object is first created. You can overwrite this when you implement your own Decorator CFC, and is a perfect place to set default values for your Transfer Object.

For example, this will set the 'name' property in this TransferObject to 'foo' when it is first initialised:

```
<cffunction name="configure" access="private" returntype="void" output="false">
  <cfset setName("foo") />
</cffunction>
```

Categories: • [Decorators](#)