

[<< Back to Dashboard](#) | [<< Extras Viewer](#)

ColdBox ORM Event Handler

([Base ORM Services](#) |
[Virtual Entity Services](#))

Covers up to version 3.1.0

By using the ColdBox ORM event handler as your base class for your application's event handler you will inherently get the ability to talk to your ColdBox application instance and even inject objects into your ColdFusion ORM entities using [WireBox](#). Not only that but we have also enabled the event handler to re-transmit the Hibernate Interceptions and announce them as ColdBox Interceptions. This way, you can intercept very easily the ColdBox way, create several chains, etc. The first thing you need to do is tell the CF Engine that you want to enable event handling and also which CFC will handle your global ORM entity events.

Application.cfc settings

```
this.ormSettings = {
    cfcllocation= "model" ,
    dbcreate = "update" ,
    dialect = "MySQLwithInnoDB" ,
    logSQL = true ,
    // Enable event handling
    eventhandling = true ,
    // Set the event handler to use, which will be inside our application.
    eventhandler = "model.ORMEventHandler"
};
```

In this basic ORM configuration structure I have two lines that deal with ORM events:

```
// Enable event handling
eventhandling = true ,
// Set the event handler to use, which will be inside our application.
eventhandler = "model.ORMEventHandler"
```

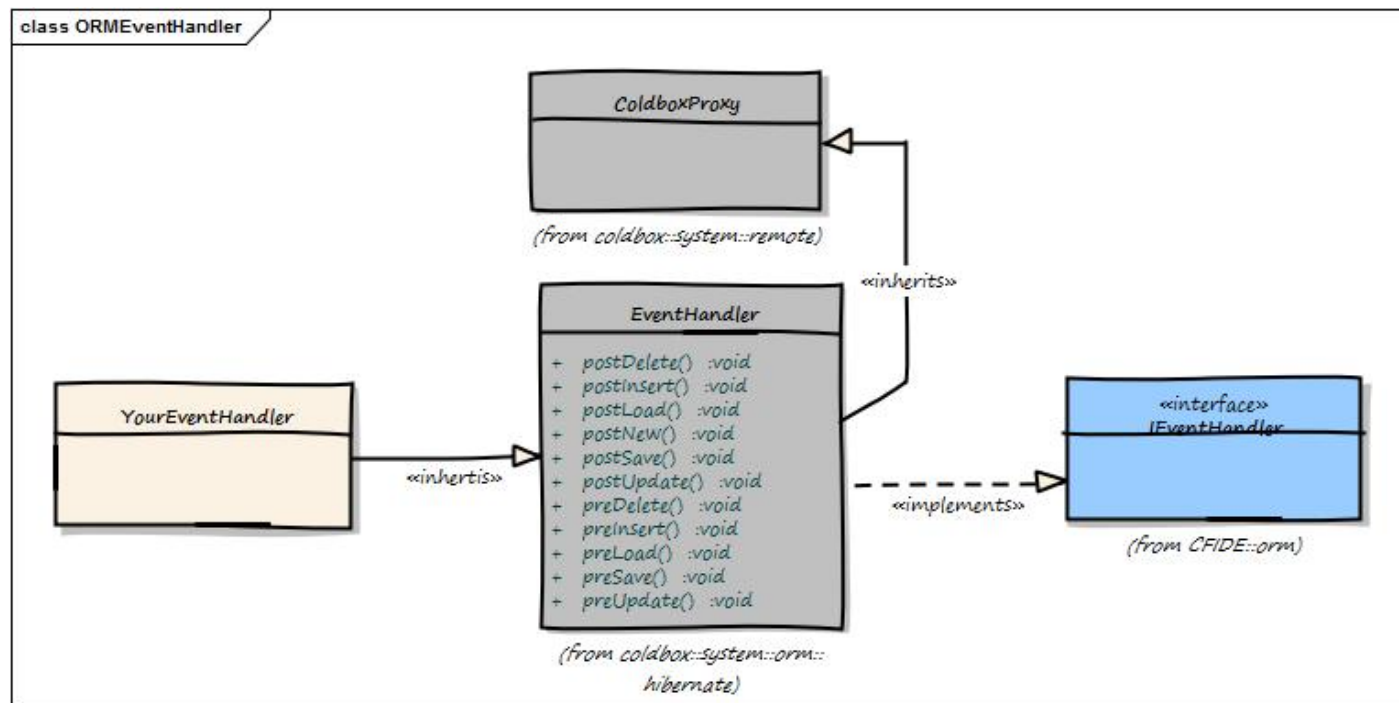
This enables the internal Hibernate interceptors and then you map a global CFC that will handle them. Mine will be in my local application's model folder named *ORMEventHandler*. It has to be inside of my application's folder structure in order to be able to talk to the ColdBox application. This is extremely important, as this creates the bridge between the ORM and ColdBox.

Important: The ORM Event Handler CFC MUST exist within the application in order for the bridge to work.

Event Handler CFC

Contents

- [ColdBox ORM Event Handler](#)
 - [Application.cfc settings](#)
 - [Event Handler CFC](#)
 - [Entity Injection Capabilities](#)
 - [Autowire Interceptor Entity Injection Properties](#)
 - [ORM To ColdBox Interceptions](#)
 - [Related ORM Services](#)



The ColdFusion documentation says that in order to create a global event handler that it must implement the `CFIDE.orm.IEventHandler` interface. So all you need to do is create the CFC and make it extend the core class that will provide you with all these capabilities: `coldbox.system.orm.hibernate.EventHandler`

```

component extends= "coldbox.system.orm.hibernate.EventHandler" {
}
  
```

That's it! Just by doing this the CF ORM will call your CFC's event methods in this CFC and satisfy the interface. Of course you can override the methods, but always remember to fire off the parent class methods in order for the ColdBox interceptions to still work. You can then override each event method as needed. Below is a chart of methods you can override:

Method	Description
postNew(entity)	This method is called by the ColdBox Base ORM Service Layers after a new entity has been created via its <code>new()</code> method.
preLoad(entity)	This method is called before the load operation or before the data is loaded from the database.
postLoad(entity)	This method is called after the load operation is complete.
preInsert(entity)	This method is called just before the object is inserted.
postInsert(entity)	This method is called after the insert operation is complete.
preUpdate(Struct oldData, entity)	This method is called just before the object is updated. A struct of old data is passed to this method to know the original state of the entity being updated.
postUpdate(entity)	This method is called after the update operation is complete.
preDelete(entity)	This method is called before the object is deleted.
postDelete(entity)	This method is called after the delete operation is complete.

The base event handler CFC you inherit from has all the ColdBox interaction capabilities you will ever need. You can find out all its methods by referring to the [API](#) and looking at that class or by inspecting the ColdBox Proxy class, which is what is used for enabling ColdBox interactions: `coldbox.system.remote.ColdboxProxy`

Entity Injection Capabilities

By enabling the global event handler, [WireBox](#) can also be enabled to listen to these events and use them to wire up dependencies in these ORM entities and thus provide **dependency injection** for ORM entities. The injection takes place during the `postLoad()` and `postNew()` event cycles. All you need to do is enable the entity injector via the properties of the [Autowire](#) interceptor in your configuration file.

Important: If you have to also override the event handler's `postLoad()`,`postNew()` method, you will then need to call the super class method so the injection procedures can take place:
`super.postLoad(entity) super.postNew(entity)`

The video below describes the entire process:

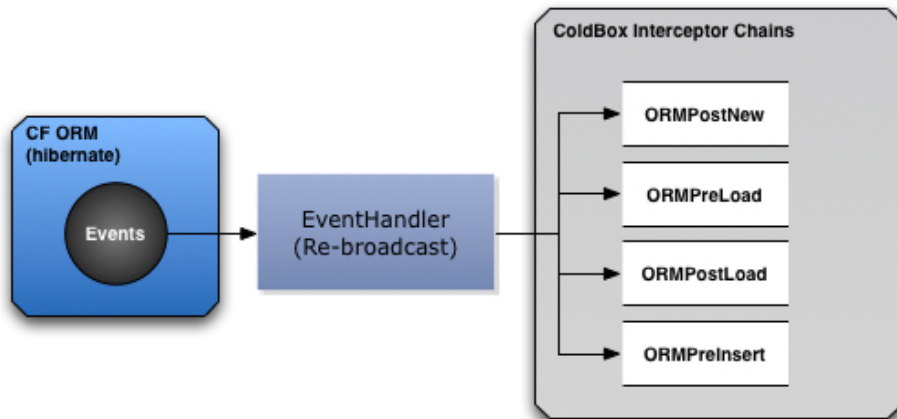
Autowire Interceptor Entity Injection Properties

Property	Type	Default	Description
<code>entityInjection</code>	Boolean	false	Enable the entity injection events
<code>entityInclude</code>	List	[empty]	A list of entity names to include in the injections ONLY!
<code>entityExclude</code>	List	[empty]	A list of entity names to exclude from dependency injection!

Example:

```
interceptors = [
  {class= "coldbox.system.interceptors.Autowire" ,
    properties={
      entityInjection = true, entityInclude = '', entityExclude = 'BadEntity'
    }
  }
]
```

ORM To ColdBox Interceptions



We have also expanded the hibernate ORM events to bridge the gap to the ColdBox interceptors. So now all the hibernate interceptors will relay their events to ColdBox via interceptors. This means that each hibernate event, like *preLoad()* for example, will announce its ColdBox counterpart: *ORMPreLoad()*. Below are the new interception points the ORM Event Handler exposes with the appropriate interception data it announces:

Interception Point	Intercept Structure	Description
ORMPostNew	{entity}	Called via the postNew() event
ORMPreLoad	{entity}	Called via the preLoad() event
ORMPostLoad	{entity}	Called via the postLoad() event
ORMPostDelete	{entity}	Called via the postDelete() event
ORMPreDelete	{entity}	Called via the preDelete() event
ORMPreUpdate	{entity,oldData}	Called via the preUpdate() event
ORMPostUpdate	{entity}	Called via the postUpdate() event
ORMPreInsert	{entity}	Called via the preInsert() event
ORMPostInsert	{entity}	Called via the postInsert() event

With the exposure of these interception points to your ColdBox application, you can easily create decoupled executable chains of events that respond to ORM events. This really expands the ORM interceptor capabilities to a more decoupled way of listening to ORM events. You can even create different interceptors for different ORM entity classes that respond to the same events, extend the entities with AOP, change entities at runtime, and more; how cool is that.

Related ORM Services

- [Base ORM Services](#)
- [Virtual Entity Services](#)