

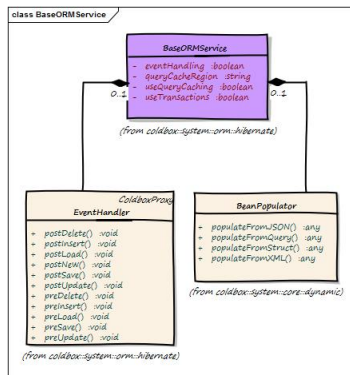
[<< Back to Dashboard](#) | [<< Extras Viewer](#)

Base ORM Service

[\(Virtual Entity Services | ORM Event Handling | Hibernate Criteria Builder \)](#)

Covers up to version 3.1.0

Overview



The *BaseORMService* is an amazing service layer tool that can be used in any project. The idea behind this support class is to provide a very good base service layer that can interact with ColdFusion ORM via hibernate and entities inspired by [Spring's Hibernate Template](#) support. It provides tons of methods for query executions, paging, transactions, session metadata, caching and much more. You can either use the class on its own or create more concrete service layers by inheriting from this class. We also have [virtual service layers](#) that can be mapped to specific entities and create entity driven service layers virtually. ColdBox also offers several integrations to this class via plugins and the wirebox injection DSL.

Note: The best place to see all of the functionality of the base ORM services is to check out the latest [API Docs](#).

Let's explore these possibilities before we get to digest the class.

ColdBox Integration

There are a few ways to interact with base, virtual or concrete entity services:

1. ORMService Plugin
2. Injected Services
3. Injected Concrete Services

ORMService Plugin

A core plugin exists called *ORMService* that extends this base service class and offers a nice integration to this service layer as a plugin that can be retrieved from any handler, interceptor or plugin *viagetPlugin()* calls.

```

// A handler with an injected plugin
component {
    property name="ORMService" inject="coldbox:plugin:ORMService";

    function saveUser(event){
        // retrieve and populate a new user object
        var user = populateModel( ORMService.new("User") );

        // save the entity using hibernate transactions
        ORMService.save( user );

        setNextEvent("user.list");
    }

    function list(event){
        var rc = event.getCollection();

        //get a listing of all users with paging
        rc.users = ORMService.list(entityName="User", sortOrder="fname", offset=event.getValue("startrow",1), max=20);

        event.setView("user/list");
    }

    // Calling the plugin directly
    function index(event,rc,prc){
        prc.data = getPlugin("ORMService").list("Permission");
    }
}
  
```

As you can see from the code above I can either use the WireBox injection DSL to inject the plugin or use it directly via *getPlugin()* calls. Once you have a reference to the base ORM service then you can just any of its methods.

Injected Services

The WireBox injection DSL has an injection namespace called *entityService* that can be used to wire in a *BaseORMService* or any [Virtual Entity Services](#). The question is, which base or virtual entity service? Well, they are completely virtual! You don't have to write these service layers, we provide them for you and it pretty much takes you about 80-85% off the way.

Injection Content	Description
<i>entityService</i>	Inject a <i>BaseORMService</i> object for usage as a generic service layer
<i>entityService:{entity}</i>	Inject a <i>VirtualEntityService</i> object for usage as a service layer based off the name of the entity passed in.

```

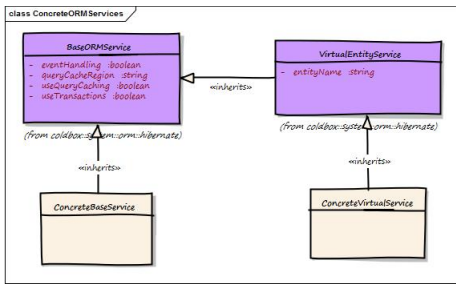
component {
    // Generic ORM service layer
    property name="genericService" inject="entityService";
    // Virtual service layer based on the User entity
    property name="userService" inject="entityService:User";
}
  
```

The example above showcases property or field injection, but you can easily also use it for constructor or setter injection. You can also encapsulate them in your WireBox binder and inject by Identifiers.

Concrete Injected Services

Contents

- [Base ORM Service](#)
 - [Overview](#)
 - [ColdBox Integration](#)
 - [ORMService Plugin](#)
 - [Injected Services](#)
 - [Concrete Injected Services](#)
 - [Base Properties](#)
 - [criteriaQuery](#)
 - [Returns](#)
 - [Arguments](#)
 - [Examples](#)
 - [criteriaCount](#)
 - [Returns](#)
 - [Arguments](#)
 - [Examples](#)
 - [evictEntity](#)
 - [Returns](#)
 - [Arguments](#)
 - [Examples](#)
 - [getAll](#)
 - [Returns](#)
 - [Arguments](#)
 - [Examples](#)
 - [save](#)
 - [Returns](#)
 - [Arguments](#)
 - [Examples](#)
 - [saveAll](#)
 - [Returns](#)
 - [Arguments](#)
 - [Examples](#)
 - [getQueryCacheRegion](#)
 - [Returns](#)
 - [Examples](#)
 - [count](#)
 - [Returns](#)
 - [Arguments](#)
 - [Examples](#)
 - [sessionContains](#)
 - [Returns](#)
 - [Arguments](#)
 - [Examples](#)
 - [delete](#)
 - [Returns](#)
 - [Arguments](#)
 - [Examples](#)
 - [deleteAll](#)
 - [Returns](#)
 - [Arguments](#)
 - [Examples](#)
 - [getTableNames](#)
 - [Returns](#)
 - [Arguments](#)
 - [Examples](#)
 - [setQueryCacheRegion](#)
 - [Returns](#)
 - [Arguments](#)
 - [Examples](#)
 - [findWhere](#)
 - [Returns](#)
 - [Arguments](#)
 - [Examples](#)
 - [getKey](#)
 - [Returns](#)
 - [Arguments](#)
 - [Examples](#)
 - [merge](#)
 - [Returns](#)
 - [Arguments](#)
 - [Examples](#)
 - [evictQueries](#)
 - [Returns](#)
 - [Arguments](#)
 - [Examples](#)
 - [findAll](#)
 - [Returns](#)
 - [Arguments](#)
 - [Examples](#)
 - [deleteByQuery](#)
 - [Returns](#)
 - [Arguments](#)
 - [Examples](#)
 - [isSessionDirty](#)
 - [Returns](#)
 - [Examples](#)
 - [deleteById](#)
 - [Returns](#)
 - [Arguments](#)
 - [Examples](#)
 - [setUseQueryCaching](#)
 - [Returns](#)
 - [Arguments](#)
 - [Examples](#)
 - [new](#)
 - [Returns](#)
 - [Arguments](#)
 - [Examples](#)
 - [getSessionStatistics](#)
 - [Returns](#)
 - [Examples](#)
 - [list](#)



Let's say you are using the virtual services and base ORM service but you find that they do not complete your requirements, or you need some custom methods or change functionality. Then you will be building concrete services that inherit from the base or virtual entity services. This is the very purpose of these support classes as most of the time you will have custom requirements and your own style of coding. Then you can just inject them like normal model objects as you will be building them in your `model` folder. `model/security/AuthorService.cfc`:

```

/**
 * Service to handle auctor operations.
 */
component extends="coldbox.system.orm.hibernate.VirtualEntityService" accessors="true" singleton{

    // User hashing type
    property name="hashType";

    /**
     * Constructor
     */
    AuthorService function init(){
        // init it via virtual service layer
        super.init(entityName="bbAuthor");
        setHashType( "SHA-256" );

        return this;
    }

    /**
     * Author search by name, email or username
     */
    function search(criteria){
        var params = (criteria=="Arguments.criteria#1");
        var r = executeQuery(query="from bbAuthor where firstName like :criteria OR lastName like :criteria OR email like :criteria",params=params,asQuery=false);
        return r;
    }
}
  
```

Then you can just inject your concrete service in your handlers, plugins, etc.

```

component{
    // Concrete ORM service layer
    property name="authorService" inject="security.AuthorService";
    // Aliased
    property name="authorService" inject="id:AuthorService";
}
  
```

Base Properties

There are a few properties you can instantiate the base service with or set them afterward. Below you can see a nice chart for them:

Property	Type	Required	Default	Description
<code>queryCacheRegion</code>	string	false	<code>ORMService.defaultCache</code>	The name of the secondary cache region to use when doing queries via this base service
<code>useQueryCaching</code>	boolean	false	false	To enable the caching of queries used by this base service
<code>useTransactions</code>	boolean	false	true	Enables hibernate safe transactions around all operations that either save, delete or update ORM entities

So if I was to base off my services on top of this gem, I can do this:

```

import coldbox.system.orm.hibernate.*
component extends="BaseORMService"{

    public UserService function init(){
        super.init(useQueryCaching=true);
        return this;
    }
}
  
```

Let's start digesting all the beautiful methods this support class can offer. However, please note that we only show partial functionality here, for full functionality, arguments and methods, please refer to the [API Docs](#)

criteriaQuery

Do a hibernate criteria based query with projections. You must pass an array of criterion objects by using the Hibernate Restrictions object that can be retrieved from this service using `getRestrictions()`. The Criteria interface allows to create and execute object-oriented queries. It is powerful alternative to the HQL but has own limitations. Criteria Query is used mostly in case of multi criteria search screens, where HQL is not very effective.

Returns

- This function returns *any* which can be an array or query depending on passed arguments

Arguments

Key	Type	Required	Default	Description
<code>entityName</code>	any	Yes	---	The entity name to run the criteria on
<code>criteria</code>	array	No	[]	Array of criterion
<code>sortOrder</code>	string	No		
<code>offset</code>	numeric	No	0	
<code>max</code>	numeric	No	0	
<code>timeout</code>	numeric	No	0	
<code>ignoreCase</code>	boolean	No	false	
<code>asQuery</code>	boolean	No	true	

- [Returns](#)
- [Arguments](#)
- [Examples](#)
- [createService](#)
 - [Returns](#)
 - [Arguments](#)
 - [Examples](#)
- [findAllWhere](#)
 - [Returns](#)
 - [Arguments](#)
 - [Examples](#)
- [countWhere](#)
 - [Returns](#)
 - [Arguments](#)
 - [Examples](#)
- [exists](#)
 - [Returns](#)
 - [Arguments](#)
 - [Examples](#)
- [exist](#)
 - [Returns](#)
 - [Arguments](#)
 - [Examples](#)
- [findIn](#)
 - [Returns](#)
 - [Arguments](#)
 - [Examples](#)
- [deleteWhere](#)
 - [Returns](#)
 - [Arguments](#)
 - [Examples](#)
- [executeQuery](#)
 - [Returns](#)
 - [Arguments](#)
 - [Examples](#)
- [refresh](#)
 - [Returns](#)
 - [Arguments](#)
 - [Examples](#)
- [getPropertyNames](#)
 - [Returns](#)
 - [Arguments](#)
 - [Examples](#)
- [get](#)
 - [Returns](#)
 - [Arguments](#)
 - [Examples](#)
- [clear](#)
 - [Returns](#)
 - [Examples](#)
- [getUseQueryCaching](#)
 - [Returns](#)
 - [Examples](#)
- [populate](#)
 - [Returns](#)
 - [Arguments](#)
 - [Examples](#)
- [populateFromJSON](#)
 - [Returns](#)
 - [Arguments](#)
 - [Examples](#)
- [populateFromXML](#)
 - [Returns](#)
 - [Arguments](#)
 - [Examples](#)
- [populateFromQuery](#)
 - [Returns](#)
 - [Arguments](#)
 - [Examples](#)

Examples

```
// Assuming the following code has a dependency on a virtual entity service:
property name="authorService" inject="entityService:Author";

var restrictions = authorService.getRestrictions();
var criteria = [];
ArrayAppend(criteria, Restrictions.eq("firstName", "Emily"));
ArrayAppend(criteria, Restrictions.eq("firstName", "Paul"));
ArrayAppend(criteria, Restrictions.eq("firstName", "Amy"));
example1 = authorService.criteriaQuery([Restrictions.disjunction(criteria)]);

var disjunction = ArrayNew(1);
ArrayAppend(disjunction, Restrictions.eq("firstName", "Emily"));
ArrayAppend(disjunction, Restrictions.eq("firstName", "Paul"));
ArrayAppend(disjunction, Restrictions.eq("firstName", "Amy"));

var criteria = ArrayNew(1);
ArrayAppend(criteria, Restrictions.gt("id", JavaCast("int", 7)));
ArrayAppend(criteria, Restrictions.disjunction(disjunction));
example2 = authorService.criteriaQuery(criteria);

var criteria = ArrayNew(1);
ArrayAppend(criteria, Restrictions.eq("firstName", "Michael"));
prc.example1 = authorService.criteriaQuery(criteria);

var criteria = ArrayNew(1);
ArrayAppend(criteria, Restrictions.ne("firstName", "Michael"));
prc.example2 = authorService.criteriaQuery(criteria);

var criteria = ArrayNew(1);
ArrayAppend(criteria, Restrictions.in("firstName", ["Ian", "Emily", "Paul"]));
prc.example3a = authorService.criteriaQuery(criteria);

var criteria = ArrayNew(1);
ArrayAppend(criteria, Restrictions.in("id", JavaCast("java.lang.Integer[]", [2, 5, 9])));
prc.example3b = authorService.criteriaQuery(criteria);

var criteria = ArrayNew(1);
ArrayAppend(criteria, Restrictions.like("lastName", "M%"));
prc.example4 = authorService.criteriaQuery(criteria);

var criteria = ArrayNew(1);
ArrayAppend(criteria, Restrictions.ilike("lastName", "s%"));
prc.example5 = authorService.criteriaQuery(criteria);

var criteria = ArrayNew(1);
ArrayAppend(criteria, Restrictions.isEmpty("books"));
prc.example6 = authorService.criteriaQuery(criteria);

var criteria = ArrayNew(1);
ArrayAppend(criteria, Restrictions.isNotEmpty("books"));
prc.example7 = authorService.criteriaQuery(criteria);

var criteria = ArrayNew(1);
ArrayAppend(criteria, Restrictions.isNull("bio"));
prc.example8 = authorService.criteriaQuery(criteria);

var criteria = ArrayNew(1);
ArrayAppend(criteria, Restrictions.isNotNull("bio"));
prc.example9 = authorService.criteriaQuery(criteria);

var criteria = ArrayNew(1);
ArrayAppend(criteria, Restrictions.between("lastName", "A", "M"));
prc.example10a = authorService.criteriaQuery(criteria);

var criteria = ArrayNew(1);
ArrayAppend(criteria, Restrictions.between("id", JavaCast("int", 3), JavaCast("int", 7)));
prc.example10b = authorService.criteriaQuery(criteria);

var criteria = ArrayNew(1);
ArrayAppend(criteria, Restrictions.gt("id", JavaCast("int", 8)));
prc.example11 = authorService.criteriaQuery(criteria);

var criteria = ArrayNew(1);
ArrayAppend(criteria, Restrictions.ge("id", JavaCast("int", 13)));
prc.example12 = authorService.criteriaQuery(criteria);
```

criteriaCount

Get the record count using hibernate projections and criterion for specific queries

Returns

- This function returns *numeric*

Arguments

Key	Type	Required	Default	Description
entityName	any	Yes	---	The entity to run count projections on
criteria	array	No	[]	The array of hibernate criterion to use for the projections

Examples

```
// The following is handler code which has been wired with a virtual entity service
property name="authorService" inject="entityService:Author";

function showAuthors(event){
var rc = event.getCollection();
// Get the hibernate restrictions proxy object
var restrictions = authorService.getRestrictions();
// build criteria
var criteria = [];
// Apply a "greater than" constraint to the named property
ArrayAppend(criteria, restrictions.ge("firstName", "M"));

// Get the criteria query first
prc.example1 = authorService.criteriaQuery(criteria=criteria, offset=(prc.boundaries.STARTROW-1), max=getSetting("PagingMaxRows"), sortOrder="firstName ASC");

// get the total records found via projections
prc.foundcount = authorService.criteriaCount(criteria=criteria);
}
```

evictEntity

Evict entity objects from session. The argument can be one persistence entity or an array of entities

Returns

- This function returns *void*

Arguments

Key	Type	Required	Default	Description
entity	any	Yes	---	

Examples

```
// evict one entity
ORMService.evictEntity( entity );

// evict an array of entities
entities = [ user1, user2 ];
ORMService.evictEntity( entities );
```

getAll

Retrieve all the instances from the passed in entity name using the id argument if specified. The id can be a list of IDs or an array of IDs or none to retrieve all. If the id is not found or returns null the array position will have an empty string in it in the specified order

Returns

- This function returns *array* of entities found.

Arguments

Key	Type	Required	Default	Description
entityName	string	Yes	---	
id	any	No	---	

Examples

```
// Get all user entities
users = ORMService.getAll("User");
// Get all the following users by id's
users = ORMService.getAll("User", "1,2,3");
// Get all the following users by id's as array
users = ORMService.getAll("User", [1,2,3,4,5]);
```

save

Save an entity using hibernate transactions or not. You can optionally flush the session also.

Returns

- This function returns *void*

Arguments

Key	Type	Required	Default	Description
entity	any	Yes	---	
forceInsert	boolean	No	false	
flush	boolean	No	false	
transactional	boolean	No	true	Use hibernate transactions or not

Examples

```
var user = ormService.new("User");
populateModel(user);
ormService.save(user);

// Save with immediate flush
var user = ormService.new(entityName="User", lastName="Majano");
ormService.save(entity=user, flush=true);
```

saveAll

Saves an array of passed entities in specified order and transaction safe

Returns

- This function returns *void*

Arguments

Key	Type	Required	Default	Description
entities	array	Yes	---	The array of entities to persist
forceInsert	boolean	No	false	
flush	boolean	No	false	

Examples

```
var user = ormService.new("User");
populateModel(user);
var user2 = ormService.new("User");
populateModel(user);
ormService.saveAll( [user1, user2] );
```

getQueryCacheRegion**Returns**

- This function returns *string*

Examples

```
// Give the name of the cache region used for this service
<cfoutput>#ormservice.getQueryCacheRegion()#</cfoutput>
```

count

Return the count of instances in the DB for the given entity name. You can also pass an optional where statement that can filter the count. Ex: count('User', 'age > 40 AND name="joe"). You can even use named or positional parameters with this method: Ex: count('User', 'age > ? AND name = ?', [40, 'joe'])

Returns

- This function returns *numeric*

Arguments

Key	Type	Required	Default	Description
entityName	string	Yes	---	

where	string	No		
params	any	No	structnew()	Named or positional parameters

Examples

```
// Get the count of instances for all books
ormService.count("Book");
// Get the count for users with age above 40 and named Bob
ormService.count("User", "age > 40 AND name='Bob'");
// Get the count for users with passed in positional parameters
ormService.count("User", "age > ? AND name=?", [40, 'Bob']);
// Get the count for users with passed in named parameters
ormService.count("Post", "title like :title and year = :year", {title:"coldbox", year:"2007"});
```

sessionContains

Checks if the current session contains the passed in entity

Returns

- This function returns *boolean*

Arguments

Key	Type	Required	Default	Description
entity	any	Yes	---	

Examples

```
function checkSomething( any User ){
// check if User is already in session
if( NOT ormService.sessionContains( arguments.User ) ){
// Not in hibernate session, so merge it in.
ormService.merge( arguments.User );
}
}
```

delete

Delete an entity using hibernate transactions. The entity argument can be a single entity or an array of entities. You can optionally flush the session also after committing

Returns

- This function returns *void*

Arguments

Key	Type	Required	Default	Description
entity	any	Yes	---	
flush	boolean	No	false	

Examples

```
var post = ormService.get(1);
ormService.delete( post );

// Delete a flush immediately
ormService.delete( post, true );
```

deleteAll

Deletes all the entity records found in the database in a transaction safe matter and returns the number of records removed

Returns

- This function returns *numeric*

Arguments

Key	Type	Required	Default	Description
entityName	string	Yes	---	The entity to purge
flush	boolean	No	false	

Examples

```
ormService.deleteAll("Tags");
```

getTableNames

Returns the table name of the passed in entity

Returns

- This function returns *string*

Arguments

Key	Type	Required	Default	Description
entityName	string	Yes	---	

Examples

```
var persistedTable = ormService.getTableNames( "Category" );
```

setQueryCacheRegion

Override the name of the default cache region name used for secondary level caching or coldbox caching.

Returns

- This function returns *void*

Arguments

Key	Type	Required	Default	Description
queryCacheRegion	string	Yes	---	

Examples

```
ormService.setQueryCacheRegion( 'MyAwesomeUserCache' );
```

findWhere

Find one entity (or null if not found) according to a criteria structure ex: findWhere(entityName="Category", {category="Training"}), findWhere(entityName="Users",{age=40,retired=false});

Returns

- This function returns *any*

Arguments

Key	Type	Required	Default	Description
entityName	string	Yes	---	
criteria	struct	Yes	---	A structure of criteria to filter on

Examples

```
// Find a category according to the named value pairs I pass into this method
var category = ormService.findWhere(entityName="Category", isActive=true, label="Training");
var user = ormService.findWhere(entityName="User", isActive=true, username=rc.username,password=rc.password);
```

getKey

Returns the key (id field) of a given entity, either simple or composite keys. If the key is a simple pk then it will return a string, if it is a composite key then it returns an array

Returns

- This function returns *any*

Arguments

Key	Type	Required	Default	Description
entityName	string	Yes	---	

Examples

```
var pkField = ormService.getKey( "User" );
```

merge

Merge an entity or array of entities back into the session

Returns

- This function returns *void*

Arguments

Key	Type	Required	Default	Description
entity	any	Yes	---	

Examples

```
// merge a single entity back
ormService.merge( userEntity );
// merge an array of entities
collection = [entity1,entity2,entity3];
ormService.merge( collection );
```

evictQueries

Evict all queries in the default cache or the cache region that is passed in.

Returns

- This function returns *void*

Arguments

Key	Type	Required	Default	Description
cacheName	string	No	---	

Examples

```
// evict queries that are in the default hibernate cache
ormService.evictQueries();
// evict queries for this service
ormService.evictQueries( ormService.getQueryCacheRegion() );
// evict queries for my artists
ormService.evictQueries( "MyArtists" );
```

findAll

Find all the entities for the specified query, named or positional arguments or by an example entity

Returns

- This function returns *array*

Arguments

Key	Type	Required	Default	Description
query	string	No	---	
params	any	No	[runtime expression]	
offset	numeric	No	0	
max	numeric	No	0	
example	any	No	---	

Examples

```
// find all blog posts
ormService.findAll("Post");
// with a positional parameters
ormService.findAll("from Post as p where p.author=?",['Luis Majano']);
// 10 posts from Luis Majano starting from 5th post ordered by release date
ormService.findAll("from Post as p where p.author=? order by p.releaseDate",['Luis majano'],offset=5,max=10);

// Using paging params
var query = "from Post as p where p.author='Luis Majano' order by p.releaseDate"
// first 20 posts
ormService.findAll(query=query,max=20)
// 20 posts starting from my 15th entry
ormService.findAll(query=query,max=20,offset=15);

// examples with named parameters
ormService.findAll("from Post as p where p.author=:author", {author:'Luis Majano'})
ormService.findAll("from Post as p where p.author=:author", {author:'Luis Majano'}, max=20, offset=5);

// query by example
user = ormService.new(entityName="User",firstName="Luis");
ormService.findAll( example=user );
```

deleteByQuery

Delete by using an HQL query and iterating via the results, it is not performing a delete query but it actually is a select query that should retrieve objects to remove

Returns

- This function returns *void*

Arguments

Key	Type	Required	Default	Description
query	string	Yes	---	
params	any	No	---	
max	numeric	No	0	
offset	numeric	No	0	
flush	boolean	No	false	

Examples

```
// delete all blog posts
ormService.deleteByQuery("from Post");
// delete query with positional parameters
ormService.deleteByQuery("from Post as b where b.author=? and b.isActive = :active",['Luis Majano',false]);

// Use query options
var query = "from User as u where u.isActive=false order by u.creationDate desc";
// first 20 stale inactive users
ormService.deleteByQuery(query=query,max=20);
// 20 posts starting from my 15th entry
ormService.deleteByQuery(query=query,max=20,offset=15,flush=true);

// examples with named parameters
ormService.deleteByQuery("from Post as p where p.author=:author", {author:'Luis Majano'})
```

isSessionDirty

Checks if the session contains dirty objects that are awaiting persistence

Returns

- This function returns *boolean*

Examples

```
// Check if by this point we have a dirty session, then flush it
if( ormService.isSessionDirty() ){
    ORMFlush();
}
```

deleteByID

Delete using an entity name and an incoming id, you can also flush the session if needed. The ID can be a single ID or an array of ID's to batch delete using hibernate DLM style deletes. The function also returns the number of records deleted.

Returns

- This function returns *numeric*

Arguments

Key	Type	Required	Default	Description
entityName	string	Yes	---	The name of the entity to delete
id	any	Yes	---	A single ID or array of IDs
flush	boolean	No	false	

Examples

```
// just delete
count = ormService.deleteByID("User",1);

// delete and flush
count = ormService.deleteByID("User",4,true);

// Delete several records, or at least try
count = ormService.deleteByID("User",[1,2,3,4]);
```

setQueryCaching

Turn on/off the usage of secondary caching level

Returns

- This function returns *void*

Arguments

Key	Type	Required	Default	Description
setQueryCaching	boolean	Yes	---	

Examples

```
ormService.setUseQueryCaching( true );
```

new

Get a new entity object by entity name. You can also pass in a structure called properties that will be used to populate the new entity with or you can use optional named parameters to call setters within the new entity to have shorthand population.

This method will also execute the [ORM Event Handler's postNew\(\)](#) method call if the event handling property has been enabled in the service. This in turn will fire the *ORMPostNew* interceptors that you can build upon.

Returns

- This function returns *any*

Arguments

Key	Type	Required	Default	Description
entityName	any	true	---	
properties	struct	false	{}	A structure of name-value pairs to populate the new entity with

Examples

```
// return empty post entity
var post = ormService.new("Post");

// create a new user entity with pre-defined params
var user = ormService.new(entityName="User", firstName="Luis", lastName="Majano", age="32", awesome=true);

// Create a new user entity with properties binded
var user = ormService.new("User", {fname="Luis", lname="Majano", cool=false, awesome=true});
```

getSessionStatistics

Information about the first-level (session) cache for the current session

Returns

- This function returns *struct*

Examples

```
// Let's get the session statistics
stats = ormService.getSessionStatistics;

// Lets output it
<cfoutput>
collection count: #stats.collectionCount# <br/>
collection keys: #stats.collectionKeys# <br/>
entity count: #stats.entityCount# <br/>
entity keys: #stats.entityKeys#
</cfoutput>
```

list

List all of the instances of the passed in entity class name. You can pass in several optional arguments like a struct of filtering criteria, a sortOrder string, offset, max, ignorecase, and timeout. Caching for the list is based on the useQueryCaching class property and the cachename property is based on the queryCacheRegion class property.

Returns

- This function returns *any*

Arguments

Key	Type	Required	Default	Description
entityName	string	Yes	---	
criteria	struct	No	[runtime expression]	
sortOrder	string	No		
offset	numeric	No	0	
max	numeric	No	0	
timeout	numeric	No	0	
ignoreCase	boolean	No	false	
asQuery	boolean	No	true	

Examples

```
users = ormService.list(entityName="User",max=20,offset=10,asQuery=false);
users = ormService.list(entityName="Art",timeout=10);
users = ormService.list("User",{isActive=false},"lastName, firstName");
users = ormService.list("Comment",{postID=rc.postID},"createdDate desc");
```

createService

Create a virtual abstract service for a specific entity. Basically a new service layer that inherits from the *BaseORMService* object but no need to pass in entity names, they are bound to the entity name passed here.

Returns

- This function returns *VirtualEntityService*

Arguments

Key	Type	Required	Default	Description
entityName	string	Yes	---	
useQueryCaching	boolean	No	Same as BaseService	
queryCacheRegion	string	No	Same as BaseService	

Examples

```
userService = ormService.createService("User");
userService = ormService.createService("User", true);
userService = ormService.createService("User", true, "MyFunkyUserCache");

// Remember you can use virtual entity services by autowiring them in via our DSL
component{
    property name="userService" inject="entityService:User";
    property name="postService" inject="entityService:Post";
}
```

findAllWhere

Find all entities according to criteria structure. Ex: `findAllWhere(entityName="Category", {category="Training"})`, `findAllWhere(entityName="Users", {age=40.retired=true})`;

Returns

- This function returns *array*

Arguments

Key	Type	Required	Default	Description
entityName	string	Yes	---	
criteria	struct	Yes	---	A structure of criteria to filter on

Examples

```
posts = ormService.findAllWhere(entityName="Post", author="Luis Majano");
users = ormService.findAllWhere(entityName="User", isActive=true);
artists = ormService.findAllWhere(entityName="Artist", isActive=true, artist="Monet");
```

countWhere

Returns the count by passing name value pairs as arguments to this function. One mandatory argument is to pass the 'entityName'. The rest of the arguments are used in the where class using AND notation and parameterized. Ex: `countWhere(entityName="User",age="20")`;

Returns

- This function returns *numeric*

Arguments

Key	Type	Required	Default	Description
entityName	string	Yes	---	

Examples

```
found = ormService.countWhere(entityName="Artist", artist="Monet");
found = ormService.countWhere(entityName="Post", author="Luis Majano");
found = ormService.countWhere(entityName="User", isActive=false, married=true);
```

exists

Checks if the given entityName and id exists in the database

Returns

- This function returns *boolean*

Arguments

Key	Type	Required	Default	Description
entityName	any	Yes	---	
id	any	Yes	---	

Examples

```
if( ormService.exists("Account",123) ){
    // do something
}
```

evict

Evict an entity from session, the id can be a string or structure for the primary key You can also pass in a collection name to evict from the collection

Returns

- This function returns *void*

Arguments

Key	Type	Required	Default	Description
entityName	string	Yes	---	
collectionName	string	No	---	
id	any	No	---	

Examples

```
ormService.evict(entityName="Account",account.getID());
ormService.evict(entityName="Account");
ormService.evict(entityName="Account", collectionName="MyAccounts");
```

findIt

Finds and returns the first result for the given query or null if no entity was found. You can either use the query and params combination or send in an example entity to find.

Returns

- This function returns *any*

Arguments

Key	Type	Required	Default	Description
query	string	No	---	
params	any	No	[runtime expression]	
example	any	No	---	

Examples

```
// My First Post
ormService.findIt(*from Post as p where p.author='Luis Majano');
// With positional parameters
ormService.findIt(*from Post as p where p.author=?, ['Luis Majano']);
// with a named parameter (since 0.5)
ormService.findIt(*from Post as p where p.author=:author and p.isActive=:active, { author="Luis Majano",active=true } );
// By Example
book = ormService.new(entityName="Book", author="Luis Majano");
ormService.findIt( example=book );
```

deleteWhere

Deletes entities by using name value pairs as arguments to this function. One mandatory argument is to pass the 'entityName'. The rest of the arguments are used in the where clause using AND notation and parameterized. Ex: deleteWhere(entityName="User",age="4",isActive=true);

Returns

- This function returns *numeric*

Arguments

Key	Type	Required	Default	Description
entityName	string	Yes	---	

Examples

```
ormService.deleteWhere(entityName="User", isActive=true, age=10);
ormService.deleteWhere(entityName="Account", id="40");
ormService.deleteWhere(entityName="Book", isReleased=true, author="Luis Majano");
```

executeQuery

Allows the execution of HQL queries using several nice arguments and returns either an array of entities or a query as specified by the asQuery argument. The params filtering can be using named or positional.

Returns

- This function returns *any*

Arguments

Key	Type	Required	Default	Description
query	string	Yes	---	
params	any	No	[runtime expression]	
offset	numeric	No	0	
max	numeric	No	0	
timeout	numeric	No	0	
asQuery	boolean	No	true	

Examples

```
// simple query
ormService.executeQuery( "select distinct a.accountID from Account a" );
// using with list of parameters
ormService.executeQuery( "select distinct e.employeeID from Employee e where e.department = ? and e.created > ?", ['IS','01/01/2010'] );
// same query but with paging
ormService.executeQuery( "select distinct e.employeeID from Employee e where e.department = ? and e.created > ?", ['IS','01/01/2010'],1,30);
// same query but with named params and paging
ormService.executeQuery( "select distinct e.employeeID from Employee e where e.department = :dep and e.created > :created", {dep='Accounting',created='01/01/2010'},10,20);
// GET FUNKY!!
```

refresh

Refresh the state of an entity or array of entities from the database

Returns

- This function returns *void*

Arguments

Key	Type	Required	Default	Description
entity	any	Yes	---	

Examples

```
var user = storage.getVar("UserSession");
ormService.refresh( user );
var users = [user1,user2,user3];
ormService.refresh( users );
```

getPropertyNames

Returns the persisted Property Names of the entity in array format

Returns

- This function returns *array*

Arguments

Key	Type	Required	Default	Description
entityName	string	Yes	---	

Examples

```
var properties = ormService.getPropertyNames("User");
```

get

Get an entity using a primary key, if the id is not found this method returns null. You can also pass an id = 0 and the service will return to you a new entity.

Returns

- This function returns *any*

Arguments

Key	Type	Required	Default	Description
entityName	string	Yes	---	
id	any	Yes	---	

Examples

```
var account = ormService.get("Account",1);
var account = ormService.get("Account",4);
var newAccount = ormService.get("Account",0);
```

clear

Clear the session removes all the entities that are loaded or created in the session. This clears the first level cache and removes the objects that are not yet saved to the database.

Returns

- This function returns *void*

Examples

```
ormService.clear();
```

getUseQueryCaching**Returns**

- This function returns *boolean*

Examples

```
Using Caching: #ormService.getUseQueryCaching()#
```

populate

Populate an entity with a structure of name-value pairs. Make sure the names of the properties match the keys in the structure.

Returns

- Void

Arguments

Key	Type	Required	Default	Description
target	any	Yes	---	The entity to populate
memento	struct	Yes	---	The structure of name-value pairs to try to populate the entity with
scope	string	No		Use scope injection instead of setter injection, no need of setters, just tell us what scope to inject to
trustedSetter	Boolean	No	false	Do not check if the setter exists, just call it, great for usage with <code>onMissingMethod()</code> and virtual properties
include	string	No		A list of keys to ONLY include in the population
exclude	string	No		A list of keys to exclude from the population

Examples

```
var user = ormService.populate( ormService.new("User"), data );
// populate with includes only
var user = ormService.populate( ormService.new("User"), data, "fname,lname,email" );
//populate with excludes
var user = ormService.populate(target=ormService.new("User"),memento=data,exclude="id,setup,total" );
```

populateFromJSON

Populate an entity with a JSON structure packet. Make sure the names of the properties match the keys in the structure.

Returns

- Void

Arguments

Key	Type	Required	Default	Description
target	any	Yes	---	The entity to populate
JSONString	string	Yes	---	The JSON packet to use for population
scope	string	No		Use scope injection instead of setter injection, no need of setters, just tell us what scope to inject to
trustedSetter	Boolean	No	false	Do not check if the setter exists, just call it, great for usage with <code>onMissingMethod()</code> and virtual properties
include	string	No		A list of keys to ONLY include in the population
exclude	string	No		A list of keys to exclude from the population

Examples

```
var user = ormService.populateFromJSON( ormService.new("User"), jsonString );
```

populateFromXML

Populate an entity with an XML packet. Make sure the names of the elements match the keys in the structure.

Returns

- Void

Arguments

Key	Type	Required	Default	Description
target	any	Yes	---	The entity to populate
xml	any	Yes	---	The xml string or xml document object to populate with
root	string	false		The xml root node to start the population with, by default it uses the XMLRoot.
scope	string	No		Use scope injection instead of setter injection, no need of setters, just tell us what scope to inject to

trustedSetter	Boolean	No	false	Do not check if the setter exists, just call it, great for usage with onMissingMethod() and virtual properties
include	string	No		A list of keys to ONLY include in the population
exclude	string	No		A list of keys to exclude from the population

Examples

```
var user = ormService.populateFromXML( ormService.new("User"), xml, "User");
```

populateFromQuery

Populate an entity with a query object. Make sure the names of the columns match the keys in the object.

Returns

- Void

Arguments

Key	Type	Required	Default	Description
target	any	Yes	---	The entity to populate
qry	query	Yes	---	The query to populate with
rowNumber	numeric	false	1	The row to use to populate with.
scope	string	No	---	Use scope injection instead of setter injection, no need of setters, just tell us what scope to inject to
trustedSetter	Boolean	No	false	Do not check if the setter exists, just call it, great for usage with onMissingMethod() and virtual properties
include	string	No	---	A list of columns to ONLY include in the population
exclude	string	No	---	A list of columns to exclude from the population

Examples

```
var user = ormService.populateFromQuery( ormService.new("User"), list("User",{id=4}) );
```