# 2009 Implementers Group Meeting Program How to use the API

## Services Overview

3 layer architecture

- DB layer
- Data access layer (Hibernate)
- Services
    - Patient
    - User
    - Obs
    - (10-15 svcs)

## Abstraction

Hibernate abstracts the implementation - do not need to code to the db. Map getters and setters to table fields

Each services has a DAO dedicated to it.

- Examples:
    - UserDAO
    - PatientDAO
- Typical DAO Methods:
    - Save
    - Get
    - Delete

The service methods are more complex

# Class Organization

OpenMRS API jar has the services, DAO's.

Web API jar has only the controllers needed for the web app.

- When you write a module you need to access items from Web API.
- 90% of the time you don't need to use the Web API; instead, you'll be accessing methods from the OpenMRs API.

If you are starting your own module, you may reference the objects already in the OpenMRS API. But if your module refers to its own objects, you need your own dao's and hibernate mapping module

Tips for Build.xml

- package-all - creates openmrs-api, wenapi, and test api jars
- test api jar packages the helper classes/objects to make tests easier

# Examples

Services are in org.openmrs.api package.
UserService has basic methods that act on simple openmrs objects, as opposed to table columns. UserServiceImpl is its implementation.

Controllers are where access of services happens.

UserFormController

- Example creating a user and persisting: call the api method for that object - Object.Save
- Takes 2 params - the object and a passsword string (for the user). If we're updating it can be null.
- Usually save methods do not take more than one object - this is not a typical case.

```
        public List<User> getListofUsers() {
        UserService userService = Context.getUserService;
        Role role = userService.getRole("provider");
        List<User> users = userService.getUsersInRole("provider");
        return users;
        }
```

Spring injects the specific implementation into the service (such as UserService) at startup.

In openmrs-trunk, applicationContext-service.xml has the setup config for Spring.

```
        <bean id="context" class="org.openmrs.api.context.Context">
        <property name="serviceContext"><ref bean="serviceContext"/><
/property>
        <property name="contextDAO"><ref bean="contextDAO"/></property>
        </bean>
```

This bean called BasicModuleFormController is of type BasicModuleFormController. Spring at startup instantiates this class, calls this setter, and passes its associated string

If you want to create a service, start from the module start page at http://modules.openmrs.org/modules/ - see the Example code - http://dev.openmrs.org/browser/openmrs-modules - to copy/paste code snippets. On modules wiki page at http://openmrs.org/wiki/Modules, see the Overall Module Structure listing.

Calling the moduleService.

SessionFactory - used by Hibernate

# Demo of creating a module

_ Code examples are incomplete and probably quite wrong... _

> **Best practice:** Extend BaseOpenmrsService in order to get the onStartup and shurdown in case something is added to BaseOpenmrsService .

```
public interface BasicModuleService extends BaseOpenmrsService {
// copy the class and package from the xml definition

private Log log = LogFactory.getLog(this.getClass());

public void onStartup
// flip back to the applicationContext.xml and see what else we demand to
have
public void setExampleField(ClassInApplicationContext) {
   // TODO
}

public List<>Users getUsers (String role) {
   if () {
   return this.dao...
   }
}
```

Must implement the DAO method.

> **Best practice:** Return should be an interface.

Creating the interface for the DAO

```
public interface BuildModuleDAO

// going back to applicationContext.xml - setter is specified.

public class HibernateBuildModule implements ModuleDAO {

private SessionFacory sessionFactory = null;
public void setSessionFactory sf () {
this.sessionfactory = sf;
}

public List<User> getUser (String role, Object obj, Object obj2, Object obj

Criteria criteria - sessionFacotry.getCurrentService().createCriteria();
criteria.addExpression.eq("roles",'');
criteria.addRestrictions.isNotNull("username");

// allows this object to be reused across different service methods
if (obj != null) {
criteria......

return criteria list();

}
```

Criteria in hibernate is a way to build sql objects in hibernate - user Java objects

Opened up the controller

The service upon startup is in the Context. Address it thusly:

```
BasicModuleService myService = (BasicModuleService) Context.getService
(BasicModuleService.class);
List<User> = myService.getMyFancyListbyRole("provider");
```

Should not cache this service - if it's unloaded and reloaded - length of the object is the length of time of the module. Service itself may cache things. When reloaded the cache is cleared up.

# AOP

Allows your module to affect the services of another module or of core.
"Anytime a Person method is called, call my method"

AOP only works on the entire class.
Your service will be called on every get, delete, etc.

3 different types of AOP

- Before service
- After
- Around (before and after)

If you are doing an around advice, use the matches method

# OpenMRS Docs

Best place to find a list of methods - attach source to your project.(Demo on how to do this in Eclipse. Browse to attach source)

Javadocs are at doc.openmrs.org.