Addressing Custom Requirements In OFBiz

Pre Gradle version

This page document the usage with Gradle, the pre-Gradle documentation is here: Addressing Custom Requirements In OFBiz

This is the most common requirement, where you have to customize implementation of framework in some areas you are using. In some cases we need to implement something or extend something in an existing(OOTB) OFBiz component which is very specific to a client and not generalized enough to put back in OFBiz trunk. So to achieve this we usually follow following strategies for testing(staging) and production systems:

Maintaining Patches

This strategy can be adopted in case when there are some small changes which have to be addressed, this can be either an extension in code for a service or enhancement in a form code or removal of some code etc. This strategy is also helpful and usually adopted for configuration settings e.g. If you want to enable email settings in general.properties or if you want to change a field sql-type for a specific database in fieldtype*.xml.

Creating a Patch

1. After making changes in any file either you can create a patch from eclipse or from console.
2. For creating a patch from eclipse you just need right click on a file if its a single file or on the root directory if multiple files are there then go to Team --> Create Patch. The patch created by this method needs a little extra concentration as you have to make sure the path to files in your patch is from ofbiz root dir and not the absolute path of your file system, otherwise it will not get applied from root directory, also you need to make sure eclipse properties are not there on the top of the patch, if its there needs to be removed.
3. Keep the patch file extension .patch. You can also refer to link
4. For creating a patch from console use following command on the root directory of files where the changes are:

```
svn diff > [file system path for patch file to be created with .patch extension]
```

Location of Patch File

1. In hot-deploy component directory patches should be maintained in proper order.
2. Create a patches dir at hot-deploy/customcomponent/patches and put all the patches here which are needed(common) in testing(staging) and production system.
3. Create a staging dir at hot-deploy/customcomponent/patches/staging and put all patches here which are needed in testing(staging) instance.
4. Create a production dir at hot-deploy/customcomponent/patches/production and put all patches here which are needed in production system only.

There are certain patches those you will only need to apply on staging box or only need to apply in production thats why we need to have different directories managed like above to differentiate properly and to avoid any confusion, when it comes to take system from testing to production environment. eg. two different baseUrl should be needed in content.properties file for these two different instances.

Applying a Patch

1. Applying a patch is again a thing can be done from eclipse or from console.
2. If you are not sure about it, always apply a patch from OFBiz root directory.
3. In Eclipse click on OFBiz root directory and got to Team --> Apply Patch, just browse the path to patch file and follow the instructions.
4. If the changes are in the patch are already there in files being patched then it will show you conflicts in the file which should be resolved before applying the patch.
5. For applying a patch from console use following command from OFBiz root directory:

```
patch -p0 < [patch file path]
```

6. While applying the patch if there is a conflict then system will prompt you to either postpone or continue patching, follow the instructions.
7. Once the patch is applied you don't need to apply it again until you replace the patched file from source repository or there are conflicts. In case of conflicts you need to update the patch file and apply it again.

Extending an Existing webapp Only

If you only need to change a few things such then follow the instructions How to Extend an existing component in customized application

Extending an Existing Component

You will find redundancy in the info you got from last point where you saw how to extend an existing webapp. Creating a new custom component for extending OOTB application is required in certain cases where there are heavy customization to be made and code to be kept manageable. You can take any of the approach that you find better. Both works good.

If a large set of code needs to be added or changed in a specific OFBiz OOTB component where you need to add new entities, services, screens, forms etc. then this strategy can be adopted and complete component can be overridden in hot-deploy directory, by creating a new component.

Following steps need to be performed for overriding an existing component:

1. Create custom component in hot-deploy directory by any name as you want to override, its just going be replica of existing OOTB OFBiz component structure with all the major directories and files required for setting up a component.
2. You can use ant createComponent target on OFBiz root directory to create this component in hot-deploy directory.
3. Only those files need to be added which you need to have in overriding the component, webapp. Rest of the resources are going to be used from OOTB component so if you have used gradlew createComponent target then you can safely delete some of the files e.g. if you are not going to write a service then services.xml file can be deleted, or if you are not going to add or extend an existing entity then you can delete entitymodel.xml file and their entries from ofbiz-component.xml file of custom component.
4. Basically this is the thin line between extending a webapp and a component. If you are going to have major changes in services and entities then creating a new component should be preferred where you will be extending the existing webapp.
5. e.g. If you want to have a separate component for adding or extending existing functionalities from catalog application which is a webapp in product component then you can have component in hot-deploy dir by extending the catalog webapp from product component.
6. Follow these steps:
    a. Make sure to properly override the webapp in ofbiz-component file of custom component as example given below:

```
<webapp name="catalog"
    title="Catalog"
    server="default-server"
    location="webapp/catalog"
    base-permission="OFBTOOLS,CATALOG"
    mount-point="/catalog"
    app-bar-display="true"/>
```

    b. Make sure to make entries of custom service defs(if-any), entity defs(if-any) and data files(if-any) in ofbiz-component file of custom component as example given below:

```
<entity-resource type="model" reader-name="main" loader="main" location="entitydef/entitymodel.xml"
/>
<entity-resource type="data" reader-name="seed" loader="main" location="data/*TypeData.xml"/>
<service-resource type="model" loader="main" location="servicedef/services.xml"/>
```

    c. Make sure to have same web.xml file from existing webapp.
    d. You can have new main-decorator defined for screens in webapp so if this is the case this needs to be changed in web.xml.
    e. Make sure to have controller.xml file in new webapp created and include the the controller from existing webapp and only include custom requests only rest will be taken care by included controller only, as shown bellow:

```
<include location="component://product/webapp/catalog/WEB-INF/controller.xml"/>
```

    f. Make sure to have error.jsp inside extended webapp.
    g. Make sure to have index.jsp file with redirect path.