

Objectif :

L'objectif de ce TP est de prendre en main [OSGi](#) et de maîtriser le processus de développement et de déploiement d'une application à base de services.

1 *Installation des outils.*

Vous allez volontairement partir des outils disponibles sur le Web afin de bien comprendre les différentes étapes de la mise en place du poste de travail.

Eclipse

Dans un premier temps, il s'agit (si ce n'est déjà fait), de télécharger une version d'eclipse. Afin de faciliter le développement, nous privilégierons une version d'Eclipse incluant le « Plug-in Development Environment (PDE)».

N.B. Si vous possédez déjà une version d'eclipse, créez un projet, faites un clic-droit et vérifiez la présence d'une option « Configurer > Convertir les projets en projets plug-in » dans le menu contextuel.

OSGi & Felix

Comme cela vous a été présenté, OSGi est un consortium ayant spécifié un environnement d'exécution, pour des applications ayant une architecture orientée services. Comme les spécifications ne font pas tout le travail, il faut aller chercher une implémentation, parmi d'autres, de ces spécifications. Nous travaillerons avec FELIX, car il est indépendant, et simple de prise en main. Equinox, autre implémentation d'OSGi existe nativement sous Eclipse, mais ne permet nécessairement pas de bien comprendre le processus de développement.

Téléchargez donc Félix : <http://felix.apache.org/site/downloads.cgi>.

Dezippez le répertoire et placez le à la racine de votre système de fichier (p.ex., [c:\felix](#)). Il y a un répertoire bundle où vous déposerez vos bundles développés.

Ouvrez une fenêtre de commande, placez vous dans le dossier d'installation de Felix, lancez Felix (`java -jar bin\felix.jar`). Si votre installation est correcte, la commande `ps` doit vous donner la liste des Bundles installés.

Les projets

Démarrez Eclipse. Créez 5 nouveaux projets Java, HelloService, HelloServiceFr, HelloServiceEn, HelloRequester et zLibs.

- Pour zLibs, créez un dossier « lib ». Importez-y la bibliothèque felix.jar contenue dans `c:\felix\bin`.
- Pour HelloService, HelloServiceFr, HelloServiceEn et HelloRequester : clic-droit sur le projet -> Configurer -> Convertir les Projets en Projets Plug-in.
- Pour HelloServiceFr, HelloServiceEn et HelloRequester : clic-droit sur le projet -> Build Path -> Configurer BuildPath
 - Libraries -> Add Jar et sélectionnez felix.jar contenu dans zLibs -> lib
 - Projets -> Add -> HelloService

Votre environnement est prêt, alors en avant !

2 Présentation de l'architecture

Nous allons déployer 4 bundles en tout.

- HelloService, chargé de fournir l'interface aux bundles d'implémentation,
- HelloServiceFr, qui sera l'implémentation française du « HelloService »,
- HelloServiceEn, qui sera l'implémentation anglaise du « HelloService »,
- HelloRequester, qui fera appel à ces différents services.

3 Codage du bundle HelloService

Le bundle HelloService est le plus simple des bundles. Il contient juste l'interface HelloService, et la publie afin que les autres bundles puissent y avoir accès.

Commencez par créer une interface HelloService, dans un package *osgi.hello*. Cette interface proposera une méthode *public String sayHello(String name)*, dont le paramètre sera le nom à qui dire bonjour. Et... c'est tout pour le code !

Il nous faut maintenant spécifier que ce bundle exporte le package « *osgi.hello* », contenant l'interface HelloService. Pour ce faire, nous allons compléter son manifest. Double-cliquez sur le fichier MANIFEST.MF dans le dossier META-INF du projet.

- Dans l'onglet *Runtime*
 - Ajoutez *osgi.hello* à la liste des « *Exported Package* »
 - Ajoutez « . » dans le *classPath*

Enregistrez, votre bundle est prêt à être packagé. Faites un clic-droit sur le dossier *src* de votre projet -> Export -> Java -> Jar File.

- Renseignez le « *Export destination* » avec « *C:\felix1.4\bundle\helloService.jar* »
- Cliquez *Next* 2 fois. Sélectionnez alors « *Use existing manifest from workspace* » et sélectionnez le manifest de votre projet.
- Finish. Vous venez de créer votre premier Bundle !

4 Implémentation française

L'interface étant en place, il nous faut développer une implémentation particulière; ici l'implémentation française.

Créez une classe HelloServiceFr dans un package *osgi.hello.fr*, implémentant HelloService. Complétez la méthode *sayHello* afin que son retour soit « *Bonjour <nom>* ».

Notre bundle, afin de pouvoir s'exécuter sur la plateforme OSGi nécessite un Activator. Ajoutez donc une classe *Activator* à votre projet, implémentant la classe *org.osgi.framework.BundleActivator*. Cet activator nous permettra, entre autre, de publier le service que nous offrons (HelloService) dans le contexte OSGi.

Dans la méthode *start* créez une instance de votre HelloService,

```
HelloService service = new HelloServiceFr();
```

et enregistrez le comme un service dans le contexte OSGi

```
reg = context.registerService(HelloService.class.getName(), service, null);
```

Dans la méthode *stop*, nous retirerons le service précédemment enregistré dans le contexte *osgi*.

Passons au manifest...

- Dans l'onglet *Overview* donnez la classe *Activator*(*osgi.hello.fr.Activator* en principe).
- Dans l'onglet *Dependencies* ajoutez à la liste « *Import Package* »
 - *org.osgi.framework* (Veillez à retirer la spécification de la version de package dans les propriétés.)
 - *osgi.hello*

Enregistrez, votre bundle est prêt à être packagé. Faites un clic-droit sur le dossier *src* de votre projet -> Export -> Java -> Jar File.

- Renseignez le « *Export destination* » avec « *C:\felix\bundle\helloServiceFr.jar* »
- Cliquez *Next* 2 fois. Sélectionnez alors « *Use existing manifest from workspace* » et sélectionnez le manifest de votre projet.
- *Finish*. Le bundle d'implémentation français est prêt.

5 Faites de même pour implémenter la version anglaise du service.

6 Création du HelloRequester

Nous avons donc potentiellement à notre disposition, un service qui nous offre la possibilité de dire bonjour, la langue étant dépendante de la / des implémentations actives. Essayons de les utiliser...

Créez une classe *Activator* dans un package *osgi.hello.req*, implémentant *BundleActivator*, ce sera notre classe utilisatrice de service. Vous profiterez de la méthode *start*, pour conserver une référence vers le contexte OSGi et pour lancer un thread (voir la classe *Thread*); la méthode *stop* vous servira pour stopper ce thread. Ce thread réalisera le code suivant:

```
while( ! end ){
    ServiceReference refs[];
    try {
        refs = context.getServiceReferences( null, "(objectClass=" +
HelloService.class.getName() + ")" );
        if(refs != null && refs.length != 0) {
            for(ServiceReference servRef : refs) {
                HelloService service = (HelloService)context.getService(servRef);
                System.out.println(service.sayHello("les étudiants"));
            }
            Thread.sleep(5000);
        } catch( InterruptedException ie) {
            e.printStackTrace();
        } catch( InvalidSyntaxException e) {
            e.printStackTrace();
        }
    }
}
```

Implémentez, prévoyez les actions de ce code.

Complétez le manifest et exportez votre bundle sous le nom *HelloRequester.jar* au même emplacement que les autres.

7 Lancement !

Nous allons maintenant vérifier que tous vos développements sont corrects, c'est à dire, si notre architecture réalise bien ce qu'on lui demande.

Allez dans votre fenêtre de Felix. Si vous l'avez fermée, ouvrez une fenêtre de commande, lancez `java -jar vin\felix.jar`.

Installons nos bundles. Dans le shell de Felix:

```
->install file:bundle\HelloService.jar  
->install file:bundle\HelloServiceFr.jar  
->install file:bundle\HelloServiceEn.jar  
->install file:bundle\HelloRequester.jar
```

```
-> ps  
START LEVEL 1  
ID State Level Name  
[ 0] [Active ] [ 0] System Bundle (1.4.0)  
[ 1] [Active ] [ 1] Apache Felix Shell Service (1.0.2)  
[ 2] [Active ] [ 1] Apache Felix Shell TUI (1.0.2)  
[ 3] [Active ] [ 1] Apache Felix Bundle Repository (1.2.1)  
[ 5] [Resolved] [ 1] HelloService (1.0.0)  
[ 6] [Resolved] [ 1] HelloServiceFr (1.0.0)  
[ 7] [Resolved] [ 1] HelloRequester (1.0.0)  
->
```

`ps` doit alors vous donner la fenêtre suivante: (+ la bundle anglais)

Maintenant, tapez les commandes suivantes en laissant 30 secondes entre chaque commande. Observez ce que produit chaque commande, commentez.

```
->start 5  
->start 6  
->start 7  
->....
```

Notez que le bundle exportant l'interface `HelloService` n'est jamais démarré !

Par curiosité, essayez de taper la commande `headers`, suivies ou non d'un numéro de bundle. Quelle utilité ?