# 3.3.6 Numérotation des versions



*esup-commons* résout le problème de la cohérence entre les versions de l'application et de la base sur laquelle elle s'appuie, en particulier dans les environnements cluster de nos portails.

## A quoi correspondent les numéros de version ?

Les numéros de version sont toujours de la forme **x.y.z-t**, maintenus dans le fichier **/properties/misc/application.xml** :

- **x** : le n° majeur de version, peut être changé en cas d'ajouts/changements de fonctionnalités importants, ou pour des refontes majeures des applications. Par convention, le changement de n° majeur peut impliquer pour l'utilisateur des manipulations manuelles pour la mise à jour, alors que toutes les autres mises à jour doivent se faire de manière (semi-) automatique.
- **y** : le n° mineur de version, qui correspond en général à un niveau de fonctionnalités. Pour un changement de n° mineur, l'exploitant est toujours invité à exécuter la commande upgrade-db, qui met à jour la structure de la base. C'est à cette occasion que le développeur peut faire exécuter des mises à jour supplémentaires, par exemple sur les données elles-mêmes, en allant modifier le *bean* **versionningService**.
- **z** : le n° de *patch*. Par convention, on conserve le n° de patch lorsque l'on ne modifie pas les fonctionnalités. à noter qu'une mise à jour pour un même n° de version mineur se fait de manière automatique, sans nécessiter l'appel de la commande upgrade-db.
- **t** : le n° de packaging, qui change seulement lorsque l'on modifie la documentation, le déploiement ou les fichiers d'exemple.

A chaque fois que l'on modifie la structure de la base, il suffit d'incrémenter le n° de version mineur : *esup-commons* force l'appel de upgrade-db (en lançant une exception) tant que le n° mineur de la base et celui de l'application ne sont pas les mêmes.
L'appel de la commande upgrade-db, en plus de mettre à jour la structure de la base de données, peut également être l'occasion de mettre à jour des données à l'aide de code Java exécuté par le *bean* **versionningService** (défini dans **META-INF/<appli>-domain-services-init.xml** du module domain-services de votre application), qui est appelé par la commande upgrade-db.
Note : le *bean* **versionningService** est également appelé de manière automatique par l'application lorsque seul le n° de *patch* diffère. Il est donc possible également dans ce cas d'exécuter du code de mise à jour des données.

## Comment faciliter les mises à jour ?

Distribuer une mise à jour d'un logiciel est en général chose facile, d'autant plus avec *esup-commons* car il suffit d'appeler une commande batch.
La mise à jour sur une plateforme de production est en général beaucoup plus délicate à cause de la nécessité de maintenir la cohérence des numéros de version de l'application et de la base de données. Ce point est résolu par le *bean* **versionningService**, comme montré plus haut.

## Comment ça marche

Les exemples données ici sont consultables dans l'application esup-example

### versionningService

**example-domain-services-init.xml** dans le répertoire **META-INF** du module **example-domain-services** :

```
<bean
 id="versionningService"
 class="org.esupportail.example.services.application.VersionningServiceImpl"
 parent="abstractDomainAwareBean" />
```

Ce fichier est dans le répertoire **META-INF** du module pour pouvoir être utilisé par différents autres modules. Typiquement la couche web et la couche batch.

A noter que ce bean étend **abstractDomainAwareBean** ce qui veut dire qu'il a accès au **DomainService** (ce qui lui permet notamment de savoir en quelle version est la base de données). De fait, il étend aussi **AbstractApplicationAwareBean** ce veut dire qu'il a accès au **ApplicationService** (définit via **example-domain-services-application.xml**, ce qui lui permet de savoir en quelle version est l'application). **VersionningService** saura donc, entre autres choses, vérifier la cohérence entre la base de données et l'application.

### Stockage de la version

La version de la base de données est stockée dans la base de données elle même. Pour manipuler cette version il y a du code métier, dao et un mapping objet/relationnel*.*

### mapping objet/relationnel

Le **VersionningService** manipule un objet ESUP-Commons qui est **Version**. Par contre, l'objet qui est stockée en base est un objet **VersionManager**. Cet objet est aussi un objet ESUP-Commons. Contrairement aux autres objets de votre applications qui ont besoin d'être enregistrés en base et qui peuvent être annotés directement avec des annotations JPA, **VersionManager** nécessite un fichier de mapping XML.

> ⚠ La version hibernate utilisée comme implémentation de JPA ne semble pas bien supporter les fichiers de mapping XML au format standardisé par JPA. Aussi nous utilisons un fichier de mapping XML spécifique à hibernate.

**VersionManager.hbm.xml** dans le répertoire **META-INF** de **example-dao** :

```
<hibernate-mapping package="org.esupportail.example.domain.beans">
 <class name="VersionManager">
  <id name="id">
   <generator class="native">
    <param name="sequence">VersionManager_id</param>
   </generator>
  </id>
  <property name="version" length="20" />
 </class>
</hibernate-mapping>
```

Note : Le fait le fichier hbm soit dans le classpath suffit

### Code DAO

Dans la mesure où n'est stockée que la version courante a 3 méthodes : **getVersionManager**, **addVersionManager** et **deleteVersionManager**

### Code métier

**getDatabaseVersion** qui permet de savoir en quelle version est la base de donnée

**updateDatabaseVersion** qui permet de supprimer la version astockée en base pour la remplacer par une nouvelle

## Vérification de version

> ⚠ Si l'application n'est pas en phase avec la version de la base de données alors on peut être amener à mettre à jour cette dernière. Cf. 3.3.3 Gestion de la structure de la base de données

La vérification de cohérence entre la version de la base de données et la version de l'application est faite pour chaque appel à la couche métier pouvant avoir un impact sur la base de données. Elle est faite via la méthode **checkVersion** du **VersionningService**. Cet appel est fait via APO et déclenchera une exception si l'application et la base de données ne sont pas en cohérence.

**example-domain-services-domain.xml** :

```
<aop:config>
 <aop:pointcut id="txMethods"
  expression="within(org.esupportail.*.domain.DomainServiceImpl)
              &amp;amp;&amp;amp; (execution(* add*(..)) || execution(* delete*(..)) || execution(* update*(..)))
              &amp;amp;&amp;amp; !execution(* *DatabaseVersion(..))" />
 <aop:aspect id="checkDbVersion" ref="versionningService">
  <aop:before pointcut-ref="txMethods" method="checkVersion" />
 </aop:aspect>
</aop:config>
```

A noter : Le point de coupe APO est un peu complexe. Il prend toutes les méthodes de la classe **DomainServiceImpl** qui commence par **add***, **delete*** et **update*** (soit toutes les méthodes qui peuvent avoir un impact sur la base de données) mais exclu les méthodes ***DatabaseVersion**. En effet, ces dernières peuvent aussi interagir avec la base de données pour mettre à jour la version de la base de données. Il est important de les exclure de la vérification pour éviter une boucle infinie.example-domain-services/src/main/resources/META-INF/example-domain-services-application.xml