

**FRAMEWORK LEGO
2.6.0
-
GUIDE UTILISATEUR**

Historiques des modifications

Numéro de version	Date de modification	Auteur des modifications	Paragraphe concerné	Nature des modifications
0.9	01/12/2003	Sébastien Charrier (EATE/IDVS/AIDV) Thierry Bey (EATE/IDVS/AIDV)	Tous	Document original
1.0	08/12/2003	Sébastien Charrier (EATE/IDVS/AIDV) Thierry Bey (EATE/IDVS/AIDV)	Tous	Complément d'informations
1.1	22/12/2003	Thierry Bey (EATE/IDVS/AIDV)	Tous	Corrections diverses
1.2	09/01/2004	Thierry Bey (EATE/IDVS/AIDV)	Tous	Modifications des balises xml du fichier fwk.xml
1.3	21/01/2004	Thierry Bey (EATE/IDVS/AIDV)	Chapitre 11	Installation de l'application exemple. Utilisateurs de test
1.4	30/01/2004	Thierry Bey (EATE/IDVS/AIDV)	Chapitre 11	Installation de l'application exemple.
1.5	11/02/2004	Thierry Bey (EATE/IDVS/AIDV)	Chapitre 3	Mécanisme d'externalisation des ressources et préconisation arborescence applicative.
1.6	05/03/2004	Thierry Bey (EATE/IDVS/AIDV)	Tous	Corrections diverses
1.7	10/04/2004	Thierry Bey (EATE/IDVS/AIDV) Johan Angeli (EATE/IDVS/AIDV) Jérémie Quinet (EATE/IDVS/AIDV)	Tous	Corrections diverses Organisation des classes de l'application exemple
1.8	11/05/2004	Jérémie Quinet (EATE/IDVS/AIDV)	Chapitre 8,10,11	Corrections diverses Inclusion du module d'administration dans une application Tag lib jspsecurity
1.9	30/07/2004	Jérémie Quinet (EATE/IDVS/AIDV)	Tous	Corrections diverses Gestion du timeout de session (chapitre 6.3)

2.0	24/09/2004	Johann Angeli (EATE/IDVS/AIDV)	Chapitre 8	Corrections diverses
2.1	27/09/2004	Thierry Bey (EATE/IDVS/AIDV) Jérémy Quinet (EATE/IDVS/AIDV)	Chapitres 6,7,10	Gestion des traces, fonctionnalité BasicAuthentication, file upload
2.2	08/11/2004	Fabrice Bellingard (EATE/IDVS/AIDV) Jérémy Quinet (EATE/IDVS/AIDV)	Chapitre 6.11 10.1.2	Informations sur Struts-Layout, informations sur la gestion des noms de groupe LDAP (notion de \$DOMAIN)
2.3	23/03/2004	Jérémy Quinet (EATE/IDVS/AIDV)	Chapitre 4.3	Ajout du chapitre
2.4	18/05/2005	Abdélali El Qoqui (EATE/IDVS/AIDV)	Chapitre 5.7.3, 6.7.5,11, 12	Ajout des chapitres Paramètres de services Templating (Velocity) Service de mail Taglib d'export
2.5	06/06/2005	Jérémy Quinet (EATE/IDVS/AIDV)	Chapitre 5, 13	Ajout/modification des chapitres Services multifonctions Accostage toolbox
2.6	06/09/2005	Abdélali El Qoqui (EATE/IDVS/AIDV)	Chapitre 6.7.5, 11, 14	Mise à jour de la documentation sur le tag d'exporter pour le RTF Velocity et le VTL Reporting JasperReports
2.7	12/01/2006	Abdélali El Qoqui (EATE/IDVS/AIDV)	Chapitre 12,15	Mise à jour Service de Mail (paramètres) Opérations CRUD
2.8	23/06/2006	Jérémy Quinet (EATE/IDVS)	Chapitre 10.1	Mise à jour de la gestion de sécurité (apparition des rôles ldap)
2.9	14/03/2007	Jérémy Quinet (EATE/IDVS)	Chapitre 4.1, 16	Suppression du chapitre concernant les informations générales. Ajout d'un chapitre sur les modifications à apporter en cas de migration de certains composants embarqués.
3.0	20/04/2007	Jérémy Quinet (EATE/IDVS)	Chapitre 13.3	Ajout des nouveaux attributs sur la configuration du transport http au niveau accostage Toolbox XML.
3.1	15/06/2007	Jérémy Quinet (EATE/IDVS)	Chapitre 16	Ajout du chapitre sur la limitation du nombre de sessions.
3.2	11/03/2010	Jérémy Quinet (IDVS)	Chapitre 17	Ajout du chapitre sur l'exécution asynchrone de BusinessService.

SOMMAIRE

1. OBJECTIF DU DOCUMENT.....	6
2. ARCHITECTURE GLOBALE	7
3. MODE OPERATOIRE	8
4. PERSISTANCE	10
4.1. OJB.....	10
4.2. Framework SQL	10
4.3. gestion du schema.....	15
5. SERVICES METIERS	16
5.1. Rôle.....	16
5.2. Architecture	16
5.3. Implémenter un service	17
5.4. Gestion transactionnelle.....	20
5.5. Appel d'un service métier.....	21
5.6. Appel inter-services	23
5.7. Configuration	23
5.8. BusinessService multifonctions	26
6. FRAMEWORK STRUTS	29
6.1. Introduction	29
6.2. Extension du framework Struts	30
6.3. RequestProcessor.....	30
6.4. Pages JSP	30
6.5. ActionForm.....	31
6.6. Actions	40
6.7. tagLibs	47
6.8. Struts-Tiles : solution de Templating.....	53
6.9. Configuration du Fichier Web.xml.....	57
6.10. Upload de fichiers	58
6.11. Bibliothèque Struts-Layout	59
7. TRACES.....	72
7.1. Traces applicatives	72
7.2. Traces JDBC.....	74
7.3. Administration du niveau de trace.....	75
8. MULTILINGUISME.....	76
8.1. Implémentation SGBDR standard	76
8.2. Implémentation SGBDR gerant l'état de traduction.....	78
8.3. Extension des Implémentations SGBDR fournies.....	80
8.4. Configuration	94
8.5. Intégration au framework struts.....	95
8.6. Module d'administration	95
9. EXCEPTIONS	97
10. SECURITE APPLICATIVE	98
10.1. Composants métier.....	98
10.2. Authentification	102

10.3.	basic authentication	104
10.4.	Habilitations	106
10.5.	gestionnaire de sécurité	109
11.	TEMPLATING VELOCITY	110
11.1.	Fonctionnement.....	110
11.2.	Utilisation avec lego	110
11.3.	Velocity Template Language (VTL)	111
11.4.	Exemples	116
12.	MAIL.....	119
12.1.	diagramme de classe	119
12.2.	Mail simple	119
12.3.	Mail HTML / Multipart	121
13.	ACCOSTAGE TOOLBOX XML.....	125
13.1.	introduction	125
13.2.	publication d'un BusinessService en tant que prise de service	125
13.3.	accès à une prise de service au travers d'un BusinessService	135
14.	REPORTING	146
14.1.	Introduction	146
14.2.	Utilisation	146
15.	OPERATIONS CRUD	151
15.1.	Service CRUD OJB	151
16.	LIMITATION DU NOMBRE DE SESSION J2EE	154
17.	EXECUTION ASYNCHRONE DE BUSINESSSERVICE	156
17.1.	Présentation	156
17.2.	Environnement Cible.....	156
17.3.	Fonctionnement.....	156
17.4.	Mise en place au niveau de la couche service	162
17.5.	Mise en place au niveau de la couche présentation.....	166
18.	MIGRATION	178
18.1.	Passage OJB 1.0.0 à 1.0.3.....	178
18.2.	Passage OJB 1.0.3 à 1.0.4.....	178
18.3.	Passage Struts 1.1 à 1.2.9	178
18.4.	Passage Struts-Layout 1.1 à 1.2	179
19.	ANNEXES	180
19.1.	Fichier web.xml	180
19.2.	Fichier struts-config.xml.....	181
19.3.	Fichier validation.xml	184
19.4.	Fichier tiles-defs.xml	186
19.5.	Fichier fwk.xml	187
19.6.	Fichier Datasource.jrxml.....	189

1. OBJECTIF DU DOCUMENT

Le framework JAVA va permettre aux projets de la DSIN de s'affranchir du développement d'un certain nombre de problématiques conceptuelles et techniques récurrentes. Ce socle technique a pour objectif également de fournir à des applications J2EE des réponses standardisées et cohérentes.

La solution se présente comme une librairie JAVA que les projets de la DSIN pourront intégrer à leur développement (composant réutilisable et mutualisable).

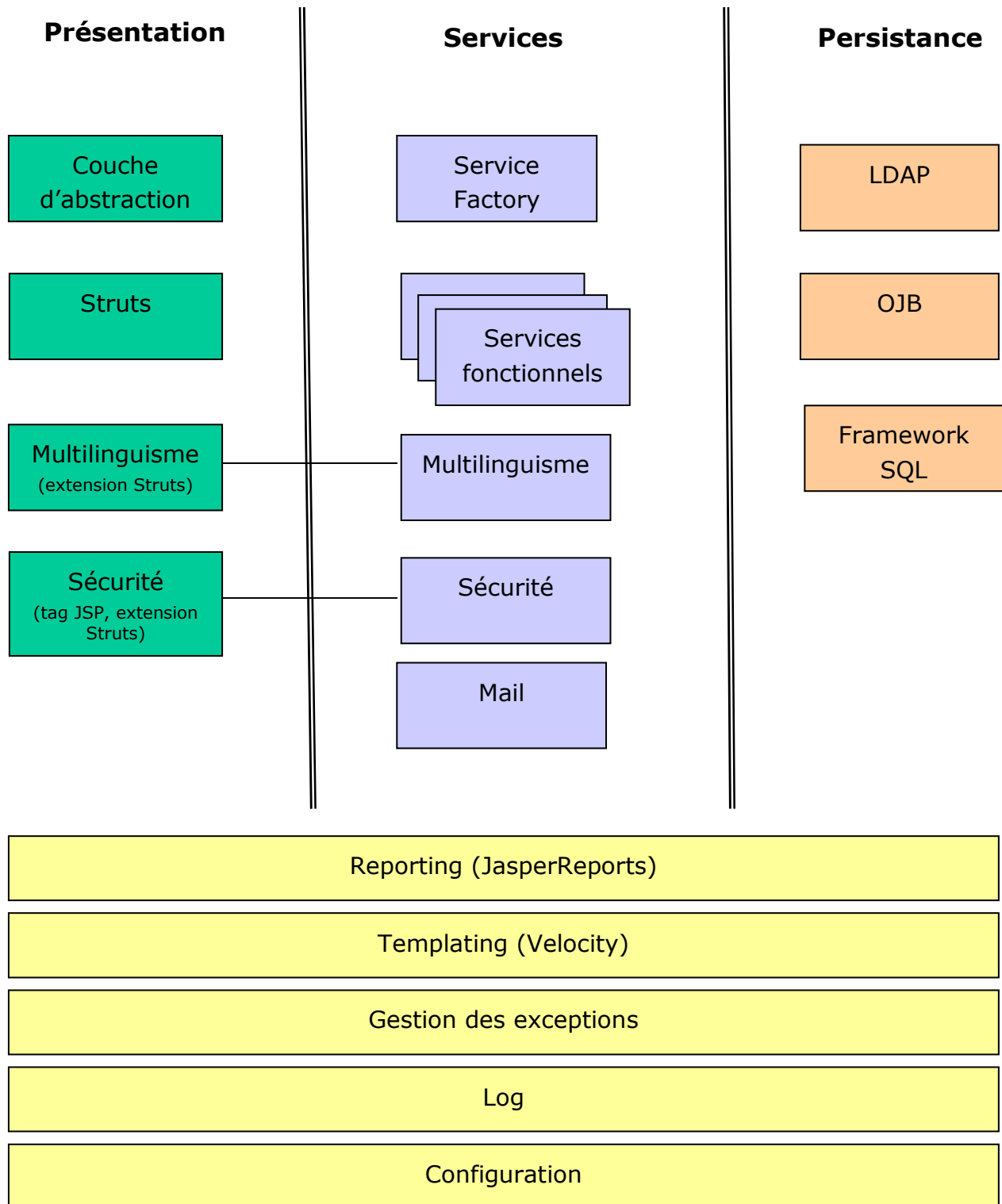
Le document décrit l'ensemble des fonctionnalités couvertes par le framework.

- Persistance (OJB, framework SQL)
- Services métiers
- Struts
- Traces
- Multilinguisme
- Exceptions
- Sécurité applicative
- Service de mail
- Templating Velocity
- Taglib d'export
- Reporting JasperReports

Pour chacune de ces fonctionnalités, le document s'attache à détailler le principe, le mode de fonctionnement ainsi que les préconisations d'utilisation et les restrictions d'utilisations lorsqu'elles existent. Des exemples viennent illustrer les différents items.

Enfin, le document décrit, de manière exhaustive, l'ensemble des fichiers nécessaires à l'intégration du framework dans un projet JAVA/J2EE. Une description du mode opératoire est proposée.

2. ARCHITECTURE GLOBALE



3. MODE OPERATOIRE

Pour démarrer le développement d'une application web basée sur le framework, voici les différentes étapes à suivre. A chaque étape sont associés les fichiers de configuration devant être ajoutés ou mis à jour.

On suppose que la modélisation UML et le MPD sont finalisées.

1. Génération des classes et interfaces Java des objets persistants découlant du diagramme de classes UML.
2. Configuration du mapping O/R avec OJB (fichiers *OJB.properties*, *repository.xml*).
3. Développement et configuration des services fonctionnels identifiés par les cas d'utilisation de la modélisation UML (fichier *fwk.xml*).
4. Mise en place du multilinguisme (mise à jour des fichiers *struts-config.xml* et *fwk.xml* et ajout du fichier *repository-i18n.xml*).
5. Développement des pages JSP, des beans *ActionForm* et des classes *Actions* du framework Struts. Configuration de la cinématique (fichiers *web.xml*, *struts-config.xml* et potentiellement *tiles-defs.xml* et *validation.xml*).
6. Mise en œuvre de la sécurité applicative (mise à jour des fichiers *struts-config.xml*, *fwk.xml* et pages JSP).

Le framework est livré sous forme d'un fichier ZIP qui contient l'arborescence suivante :

<javadoc>

```
<appAdmin>
<noyau>
<sql >
<struts>
```

<conf>

```
<cxl> -- fichiers de conf. De la Toolbox XML
```

```
<standard> -- fichiers de conf. LEGO
```

```
fwk.xml -- fichier de conf. à placer dans le classpath
```

```
fwk.xsd - schéma descriptif du fichier de conf. , permet de valider le contenu lors de l'édition du
fwk.xml avec un éditeur xml et contient aussi de la documentation sur les différents tags.
```

```
<ojb> -- fichiers de conf. OJB à placer dans le classpath
```

```
OJB.properties
repository.xml
repository.dtd
repository-user.xml
repository-database.xml
repository-internal.xml
repository-i18n.xml
```

```
<struts> -- fichiers de conf. Struts à placer dans le répertoire /WEB-INF
```

```
struts-config.xml
fwk-struts-config_1_2.dtd
...
```

<lib>

<ext> -- librairies externes (1) à placer dans le répertoire */WEB-INF/lib* suivant le besoin
<fwk> -- librairies du framework (2) à placer dans le répertoire */WEB-INF/lib*

- (1) castor-x.x.x.jar, clppkannuldap-x.x.x.jar, commons-beanutils-x.x.x.jar, commons-collections-x.x.x.jar, commons-dbcx.x.x.jar, commons-digester-x.x.x.jar, commons-fileupload-x.x.x.jar, commons-lang-x.x.x.jar, commons-logging-x.x.x.jar, commons-pool-x.x.x.jar, commons-validator-x.x.x.jar, jakarta-oro-x.x.x.jar, jcert-x.x.x.jar, jnet-x.x.x.jar, jsse-x.x.x.jar, log4j-x.x.x.jar, struts-x.x.x.jar, xercesImpl-x.x.x.jar, xmlParserAPIs-x.x.x.jar, p6spy-x.x.x.jar, db-obj-x.x.x.jar, saxpath-x.x.x.jar, jdom-x.x.x.jar .
- (2) fwk-app-admin-x.x.x.jar, fwk-noyau-x.x.x.jar, fwk-sql-x.x.x.jar, fwk-struts-x.x.x.jar, fwk-taglib-x.x.x.jar, fwk-commons-x.x.x.jar, fwk-vrn-x.x.x.jar.

- (3) un mécanisme est mis en oeuvre pour permettre l'externalisation des ressources. Un fichier *project.properties* peut-être créé et mis dans le CLASSPATH de l'application. La clé *ROOT* permet de définir un répertoire dans lequel les fichiers de configuration seront recherchés (répertoire dans le CLASSPATH ou répertoire présent dans un .jar présent dans le CLASSPATH).

Par défaut, le fichier *fwk.xml* est pris en compte par ce mécanisme et **doit** être présent dans le répertoire spécifié. La lecture des fichiers ressources de l'application par la méthode statique *getResource* sur la classe *ResourceLoader* (package *com.inetpsa.fwk.utils*) permet également de bénéficier du mécanisme mis en place.

Ex :

Contenu du fichier project.properties

ROOT=prd1

Arborescence disque

<config> répertoire présent dans le CLASSPATH de l'application

<prd1>

fwk.xml

<prd2>

fwk.xml

log.properties

Si aucun fichier *project.properties* n'est créé les fichiers de ressources présents dans le CLASSPATH de l'application seront pris en compte.

Le framework n'impose pas d'arborescence applicative.

Il est cependant recommandé de mettre en oeuvre un découpage applicatif pertinent : une application de type MVC manipule des objets de nature différente (actions, formulaires, services, objets métiers). Il est préférable de les positionner dans des packages séparés. Le découpage fonctionnel des packages peut-être complété par un découpage technique (ou inversement).

A titre d'exemple :

com.inetpsa.<prd>.clients.actions

com.inetpsa.<prd>.clients.forms

com.inetpsa.<prd>.clients.services

com.inetpsa.<prd>.clients.model

4. PERSISTANCE

Le framework propose une alternative entre 2 solutions de persistance : une 1^{ère} solution basée sur l'outil de mapping objet relationnel OJB et une 2^{ème} solution basée sur un framework JDBC classique.

La solution OJB est parfaitement adaptée dans les cas de requêtes simples : consultation, création, modification et suppression unitaire.

Par contre, dès qu'il s'agit de requêtes complexes mettant en jeu un grand nombre de jointures, OJB n'est pas forcément la meilleure solution. En effet, OJB ne manipule que des objets et non des données diverses provenant de plusieurs tables. Une solution de contournement à ce problème pourrait être de définir une Vue et de mapper la Vue à un objet OJB. Cette solution devra être étudiée en fonction de la typologie de l'application. L'autre alternative est l'utilisation du framework SQL dont l'utilisation est détaillée au §4.2.

Bien qu'il y ait 2 alternatives quant à la solution de persistance, **le seul moyen autorisé pour obtenir une connexion JDBC est de passer par OJB.** Cela revient à dire que toute application ayant accès à une base de données devra embarquer au minimum le framework OJB pour la gestion des connexions.

4.1. OJB

L'outil ObjectRelationalBridge (OJB) d'Apache est un framework de mapping Objet/Relationnel. **Il assure une gestion transparente de la persistance d'objets Java dans un SGBDR : toutes les requêtes SQL sont générées automatiquement par OJB.** Le développeur ne manipule plus que des objets Java au travers d'une API de persistance : la PersistenceBroker API.

Plus d'informations concernant l'utilisation générale d'OJB sont disponibles à l'adresse suivante : <http://db.apache.org/ojb/>.

4.2. FRAMEWORK SQL

4.2.1. DESCRIPTION

Le framework SQL n'a pas vocation à remplacer OJB. Il offre une alternative à OJB dans la réalisation de requêtes d'ordinaire complexes et difficiles à appréhender avec OJB en proposant une meilleure visibilité de la requête soumise à la base de données.

Chaque requête à la base sera représentée par un objet facilitant l'édition de ses propriétés (la requête sera directement éditée sous forme de chaîne de caractère).

Le framework SQL s'appuie sur les services OJB. La paramétrie est à la charge de la couche OJB.

- Il obéit aux mécanismes transactionnels fournis par OJB (commit/rollback).
- Il facilite la génération de jointures.
- Il prend en charge les lobs.
- Il réalise les requêtes à la base de données sous forme de PreparedStatement.

Ces spécificités autorisent une utilisation conjointe d'OJB et du framework SQL assurant une plus grande marge de manœuvre sur les techniques d'accès aux données.

4.2.2. LES OBJETS DU FRAMEWORK SQL

La gestion des erreurs au sein du framework SQL est réalisée par la classe **DBException**.

DBKEYWORD

Cet objet contient les mots clefs du framework SQL pouvant être utilisés dans une requête. Ces mots clefs seront remplacés à l'exécution par leurs valeurs pour l'utilisateur courant.

Exemple d'implémentation:

Considérons une requête SQL dont le schéma varie suivant certaines propriétés de l'utilisateur :
"SELECT EMPNO FROM "+**DBKeyword.DB_SCHEMA**+".EMP_PHOTO WHERE EMPNO = ?".

DBKeyword.DB_SCHEMA sera remplacé automatiquement pour cette requête par le résultat de l'exécution de la méthode **getDbSchema()** sur l'objet user lié à l'exécution.

QUERY

Un objet de type Query représente une requête SQL à exécuter dans la base de données. Cette requête peut contenir des paramètres variables, marqués par un point d'interrogation, qui seront renseignés à l'exécution par l'utilisateur à l'aide de la méthode addParameter(`java.lang.Object`).

Exemple de requête avec paramètres variables :

SELECT PICTURE FROM EMP_PHOTO WHERE EMPNO = ? AND PHOTO_FORMAT = ?

```

GetEmpPhoto query = new GetEmpPhoto ();           //Représente la requête ci-dessus
query.addParameter(emp.getEmpNo());                //L'ordre d'affectation des paramètres doit suivre
query.addParameter("gif");                          //l'ordre d'apparition des ? dans la requête.

QueryManager requestor = new QueryManager(user, broker, query);

```

L'utilisation des requêtes SQL est limitée aux 4 types de requête suivant (SELECT, INSERT, DELETE, UPDATE).

L'utilisateur aura à sa charge de définir pour chaque requête:

- Le nombre de paramètres variables (`getNbParameters()`).
- L'instruction SQL sans la clause WHERE (`getSQLQuery()`).
- La clause WHERE suivant le type de requête (`SelectQuery.getWhereClause()`, `DeleteQuery.getWhereClause()`, `UpdateQuery.getWhereClause()`).
- Le type d'objet à retourner si la requête est de type (`SelectQuery.getClassReturned()`).
- Si la requête peut renvoyer plusieurs résultats (`SelectQuery.isMultiRowsQuery()`).

SELECTQUERY

Un objet de type SelectQuery effectue une recherche dans la base de données.

Exemple d'implémentation:

Considérons la requête suivante: "SELECT EMPNO,PHOTO_FORMAT,PICTURE FROM EMP_PHOTO WHERE EMPNO = ? AND PHOTO_FORMAT = 'gif'".

```

public class GetEmpPhoto extends SelectQuery {

    public GetEmpPhoto() throws DBException {

```

```

    super();
    //La requête possède un paramètre variable.
    setNbParameters(1);
}

public Class getClassReturned() {
    return EmpPhoto.class;
}

public boolean isMultiRowsQuery() {
    return false;
}

public String getSQLQuery() {
    return "SELECT EMPNO,PHOTO_FORMAT,PICTURE FROM "+DBKeyword.DB_SCHEMA+" .EMP_PHOTO" ;
}

public String getWhereClause() {
    return "WHERE EMPNO = ? AND PHOTO_FORMAT = 'gif'";
}
}

```

INSERTQUERY

Un objet de type InsertQuery effectue une insertion dans la base de données.

Exemple d'implémentation:

Pour la requête suivante: "INSERT INTO EMPLOYEE
(EMPNO,FIRSTNME,LASTNAME,MIDINIT,EDLEVEL,BIRTHDATE) values (?, ?, ?, ?, ?, ?)".

```

public class InsertEmployee extends InsertQuery {

    public InsertEmployee() throws DBException {
        super();
        //La requête possède 6 paramètres variables.
        setNbParameters(6);
    }

    public String getSQLQuery() {
        return "INSERT INTO "+DBKeyword.DB_SCHEMA+" .EMPLOYEE EMPNO, FIRSTNME, LASTNAME,
MIDINIT, EDLEVEL,BIRTHDATE) values (?, ?, ?, ?, ?, ?)";
    }
}

```

UPDATEQUERY

Un objet de type UpdateQuery effectue une mise à jour dans la base de données.

Exemple d'implémentation:

Pour la requête suivante: "UPDATE EMPLOYEE SET MIDINIT = ? , SALARY = ? WHERE EMPNO = ?".

```

public class UpdateEmployee extends UpdateQuery {

```

```

public UpdateEmployee() throws DBException {
    super();
    //La requête possède 3 paramètres variables.
    setNbParameters(3);
}
public String getSQLQuery() {
    return "UPDATE "+DBKeyword.DB_SCHEMA+".EMPLOYEE SET MIDINIT = ? , SALARY = ?";
}
public String getWhereClause() {
    return "WHERE EMPNO = ?";
}
}

```

DELETEQUERY

Un objet de type DeleteQuery effectue une suppression dans la base de données.

Exemple d'implémentation:

Pour la requête suivante: "DELETE FROM EMPLOYEE WHERE EMPNO = ?".

```

public class DeleteEmployee extends DeleteQuery {

    public DeleteEmployee() throws DBException {
        super();
        //La requête possède un paramètre variable.
        setNbParameters(1);
    }

    public String getSQLQuery() {
        return "DELETE FROM "+DBKeyword.DB_SCHEMA+".EMPLOYEE ";
    }
    public String getWhereClause() {
        return "WHERE EMPNO = ?";
    }
}

```

QUERYMANAGER

Les objets, de type QueryManager ont pour fonction d'exécuter les requêtes SQL représentées par un objet de type Query.

Il travaille avec le PersistenceBroker fournit par OJB et avec l'utilisateur courant.

Exemple d'implémentation:

Pour la requête suivante GetEmpPhoto (voir ci-dessus).

```

public EmpPhoto getEmployeePhoto(PersistenceBroker broker, User user, Employee emp)
    throws DBException {

    GetEmpPhoto query = new GetEmpPhoto ();
    query.addParameter(emp.getEmpNo());
}

```

```

    QueryManager requestor = new QueryManager(user, broker, query);
    requestor.execute(user);

    return (EmpPhoto)requestor.getResult();
}

```

BLOB

Un objet Blob est utilisé pour réaliser le mapping avec les colonnes de type Blob de la base de données.

Cet objet doit être utilisé comme classe des colonnes de type Blob des objets DbObject.

CLOB

Un objet Clob est utilisé pour réaliser le mapping avec les colonnes de type Clob de la base de données. Cet objet doit être utilisé comme classe des colonnes de type Clob des objets DbObject.

DBOBJECT

Un objet DbObject correspond au mapping d'une ligne renvoyée par une requête SQL. Il peut correspondre au résultat de plusieurs requêtes SQL.

Exemple d'implémentation:

```

public class Employee extends DbObject {

    public Employee() throws DBException {
        //On définit la liste des colonnes de l'objet
        //Les colonnes sont contenues par un objet de type DbKeys
        keys.setKey("EMPNO", true, false, String.class);
        keys.setKey("FIRSTNME", false, true, String.class);
        keys.setKey("LASTNAME", false, true, String.class);
        keys.setKey("BIRTHDATE", false, true, Date.class);
    }

    public String getEmpNo() throws DBException {
        return (String) keys.getValue("EMPNO");
    }

    public void setEmpNo(String empno) throws DBException {
        keys.setValue("EMPNO", empno);
    }

    public String getFirstName() throws DBException {
        return (String) keys.getValue("FIRSTNME");
    }

    public void setFirstName(String firstName) throws DBException {
        keys.setValue("FIRSTNME", firstName);
    }

    public String getLastName() throws DBException {
        return (String) keys.getValue("LASTNAME");
    }

    public void setLastName(String lastName) throws DBException {
        keys.setValue("LASTNAME", lastName);
    }
}

```

4.3. GESTION DU SCHEMA

Lego propose une possibilité de gestion automatique du schéma utilisé pour l'accès à la base, que ce soit via le framework sql ou ojb.

FRAMEWORK SQL

Comme vu au [4.2.2](#) le framework sql peut utiliser comme schéma la valeur de la méthode **getDbSchema()** de l'objet user lié à l'exécution.

OJB

Un mécanisme a été mis en place afin que le schéma utilisé via ojb soit lui aussi la valeur provenant de la méthode **getDbSchema()** de l'objet user lié à l'exécution, pour que ce mécanisme fonctionne il faut utiliser un PersistentBroker obtenu via la méthode **getBroker(alias)** de la classe **BusinessService**.

Pour information le mécanisme mis en place travaille sur la modification au runtime des metadata du repository et l'affectation des metadata correctes aux PersistentBroker à partir de profiles.

IMPLEMENTATION PAR DEFAUT DE GETDBSCHEMA

L'implémentation par défaut de la méthode **getDbSchema()** sur la classe **com.inetpsa.fwk.security.beans.User** permet de récupérer la valeur de schéma à partir d'une propriété d'environnement JVM.

La clé de cette propriété est à positionner dans le fichier **fwk.xml** (via l'éditeur framework par exemple dans l'onglet « généralités ») de cette manière :

```
<fwk>
...
    <sgbd schemapropertykey="db2ofwk" />
</fwk>
```

Il faudra alors paramétrer une propriété système pour la jvm exécutant le code, exemple de positionnement que l'on retrouverait sur la ligne de commande :

```
-Ddb2ofwk=wja
```

« wja » étant ici le schéma à utiliser.

Il est possible, en surchargeant la classe **com.inetpsa.fwk.security.beans.User**, de réimplémenter la méthode **getDbSchema()** pour des besoins particuliers, par exemple fournir une valeur de schéma différente suivant la marque de l'utilisateur.

5. SERVICES METIERS

5.1. ROLE

Un service métier correspond à l'implémentation d'une fonctionnalité métier avec les règles de gestion associées. **Les services métier répondent ainsi à l'ensemble des cas d'utilisations (Use Cases) de l'application.**

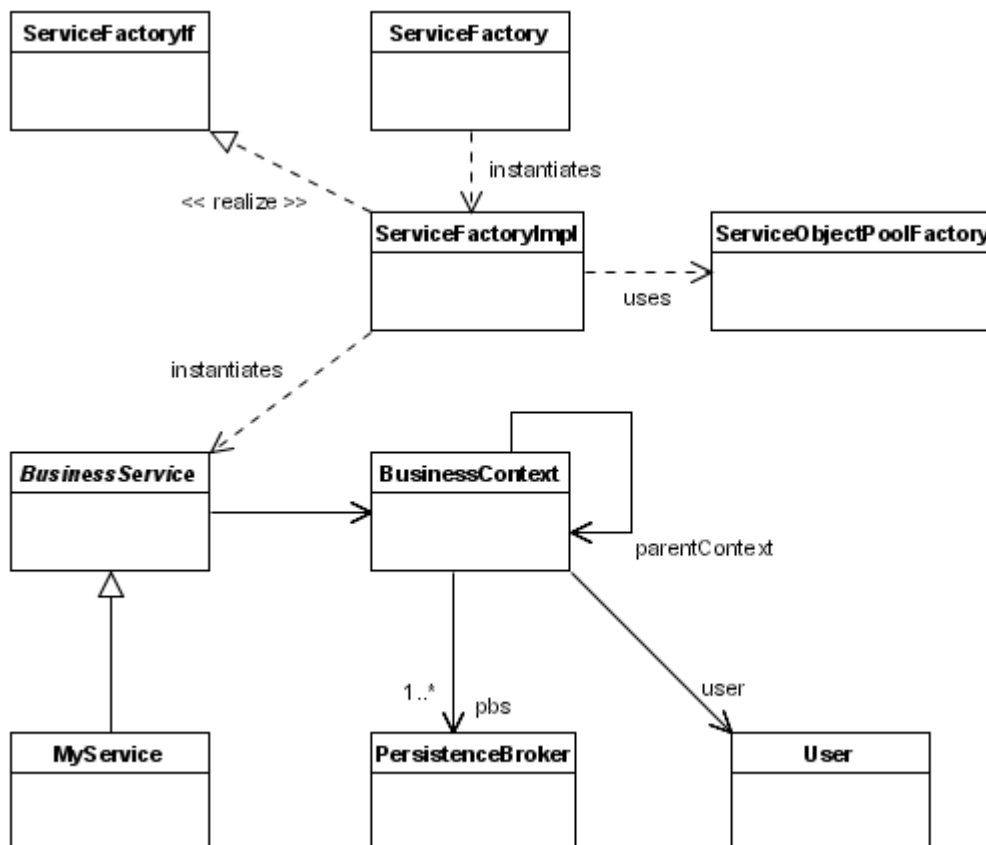
Si l'on prend l'exemple d'une application bancaire, on peut envisager les services métier suivants :

- Lecture de la liste des comptes d'un client,
- Lecture de la liste des opérations d'un compte client sur une période donnée,
- Enregistrement d'un virement entre 2 comptes,
- ...

Seul un service métier devrait avoir accès aux données persistantes de l'application. Il est capable également de gérer un contexte transactionnel (le contexte transactionnel est géré sur la connexion jdbc, et non pas lié au JTA)

5.2. ARCHITECTURE

Le diagramme ci-dessous présente les différentes classes et interfaces entrant en jeu dans la gestion des services fonctionnels du framework (package *com.inetpsa.fwk.service*).



Un Service métier est représenté par la classe abstraite *com.inetpsa.fwk.service.BusinessService*.

A chaque instance de service est associée un contexte. Ce contexte se présente sous la forme d'une instance de la classe *com.inetpsa.fwk.service.BusinessContext*. Il a pour rôle de maintenir des informations propres au service tels que l'utilisateur qui exécute le service ou bien encore les brokers d'OJB permettant d'effectuer les accès à la couche de persistance.

Un système de factory est présent afin de délivrer des instances de service fonctionnel. L'implémentation par défaut de la fabrique fournie par le framework (classe *ServiceFactoryImpl*) s'appuie sur un système de pooling d'instances (classe *ServiceObjectPoolFactory*) afin d'optimiser les performances. Cela permet d'éviter de nombreuses créations d'instances en mémoire.

5.3. IMPLEMENTER UN SERVICE

L'implémentation d'un service passe par la création d'une classe héritant de la classe abstraite *com.inetpsa.fwk.service.BusinessService*.

5.3.1. PARAMETRES D'ENTREE/SORTIE

Un service métier peut avoir des paramètres d'entrée/sortie. Ces paramètres doivent être identifiés grâce à une clé qu'il est conseillé de définir sous forme de constantes publiques. En effet, leurs valeurs devront être passées ou récupérées grâce à ces clés (cf. §5.5.2 et §5.5.4).

Exemple :

Paramètres d'entrée/sortie du service retournant une liste d'objets provenant de table de référence :

```
/**
 * Paramètre d'entree définissant le type de l'objet dont on veut
 * récupérer la liste (objet de type Class).
 */
public static final String IN_ELEMENT_CLASS = "IN_ELEMENT_CLASS";
/**
 * Paramètre d'entree définissant le nom d'un attribut de l'objet
 * sur lequel on veut effectuer un tri (objet de type String).
 */
public static final String IN_ORDER_BY_ATTR = "IN_ORDER_BY_ATTR";
/**
 * Paramètre d'entree définissant dans quel sens le tri doit
 * être effectué (objet de type Boolean).
 * Si Boolean.TRUE, le tri sera ascendant, sinon le tri sera descendant.
 * Ce paramètre n'a de sens que si le paramètre IN_ORDER_BY_ATTR est défini.
 */
public static final String IN_ORDER_BY_ASC = "IN_ORDER_BY_ASC";
/**
 * Paramètre de sortie contenant la liste des objets de référence
 * (Objet de type Collection).
 */
public static final String OUT_LISTE = "OUT_LISTE";
```

Il reste ensuite à la charge du développeur 4 méthodes à implémenter :

```
/**
 * Méthode contenant la logique d'initialisation du service.
 * @throws FwkException
 */
```

```

protected abstract void doInit() throws FwkException;

/**
 * Méthode contenant la logique d'exécution du service.
 * @throws FwkException
 */
protected abstract void doExecute() throws FwkException;

/**
 * Méthode contenant la logique de libération du service.
 * @throws FwkException
 */
protected abstract void doClose() throws FwkException;

/**
 * Renvoie un tableau de chaînes contenant en première position un identifiant
 * et en seconde position un mot de passe, ce couple permettant la création d'une
 * connexion à la base de donnée associée à l'alias avec les droits suffisant pour
 * effectuer les accès présent dans doExecute() et liés à cet alias.
 * @return String[]
 */
protected abstract String[] getServiceAccount(String alias);

```

5.3.2. METHODE DOINIT()

La méthode `doInit()` est invoquée à l'appel de la méthode `init()` du service. Cette méthode `init()` n'est appelée qu'une seule fois lors de l'instanciation du service, c'est-à-dire avant qu'il ne soit placé dans le pool. **Elle ne doit pas être appelée directement par l'application.**

5.3.3. METHODE DOEXECUTE()

La méthode `doExecute()` est invoquée à l'appel de la méthode `execute()` du service. Elle contient la logique métier et l'accès à la couche persistance via une instance du *PersistenceBroker* du framework OJB (au travers de la notion de *PersistentContext*).

L'obtention d'une instance d'un *PersistenceBroker* auprès d'un pool se fait en appelant la méthode `getPersistentContext()` présente dans la classe parente du service (à savoir la classe *BusinessService*) en fournissant en paramètre une instance de *OJBPersistentContextKey* (le constructeur prend en paramètre le nom d'un alias correspondant à une configuration d'accès base cf. **§Erreur ! Source du renvoi introuvable.**) ce qui permettra d'obtenir une instance de *IPersistentContext* de type OJB sur laquelle on dispose de la méthode `getBroker()`.

Exemple :

```

protected void doExecute() throws FwkException {
    OJBPersistentContext persistentContext = null;

    OJBPersistentContextKey key = new
    OJBPersistentContextKey(ALIAS,getServiceAccount(ALIAS));

    Class clazz = (Class) getInput(IN_ELEMENT_CLASS);
    String orderByAttribute = (String) getInput(IN_ORDER_BY_ATTR);
    Boolean orderByAscending = (Boolean) getInput(IN_ORDER_BY_ASC);

    try {
        persistentContext = (OJBPersistentContext) getPersistentContext(key);
    }
}

```

```

        PersistenceBroker broker = persistentContext.getBroker();

        QueryByCriteria q = new QueryByCriteria(clazz);
        if ((orderByAttribute != null) && (orderByAttribute.length() > 0)) {
            boolean asc = (orderByAscending == null) ? true :
orderByAscending.booleanValue();
            q.addOrderBy(orderByAttribute, asc);
        }
        Collection liste = broker.getCollectionByQuery(q);

        setOutput(OUT_LISTE, liste);

    } catch (Exception e) {
        e.printStackTrace();
        throw new ServiceTechnicalException(e);
    } finally {
        if (persistentContext != null) {
            closePersistentContext(key);
        }
    }
}

```

Remarques :

- Cette méthode renvoie exclusivement des exceptions de type *FwkException*. Cela signifie que toute autre exception doit être transformée en *FwkException* avant d'être propagée dans les couches supérieures.
- L'instance du *IPersistentContext* doit être libérée (afin de retourner quoi qu'il arrive au pool l'instance de PersistenceBroker sous-jacente). Cela se fait par l'appel de la méthode *closePersistentContext()* dans un bloc *finally* en mettant en paramètre l'instance d'*OJBPersistentContextKey* utilisé initialement.

5.3.4. METHODE DOCLOSE()

La méthode *doClose()* est invoquée à l'appel de la méthode *close()* du service. **Cette méthode *close()* doit être appelée explicitement dans un bloc *finally* (cf. §5.5.5).**

Toute ressource allouée au sein du service peut être libérée dans cette méthode.

5.3.5. METHODE GETSERVICEACCOUNT()

La méthode *getServiceAccount()* doit renvoyer un compte utilisateur d'accès au SGBDR qui servira pour l'obtention d'une connexion JDBC lors de l'appel à la méthode *getBroker()*. Ce compte doit être fonction de l'alias sélectionné passé en paramètre de la méthode.

On a vu qu'il était possible de définir ce compte utilisateur dans le fichier de configuration d'OJB *repository-database.xml* (cf. §**Erreur ! Source du renvoi introuvable.**). Cependant, l'utilisation au travers d'un *IPersistentContext* ne prendra en compte que les informations d'accréditations positionnées sur l'instance d' *OJBPersistentContextKey* (que l'on peut initialiser via un appel à *getServiceAccount()*). On peut toutefois implémenter la méthode *getServiceAccount()* pour qu'elle aille chercher la configuration du compte utilisateur définie dans le fichier repository. Cette implémentation est présentée ci-dessous en exemple.

Exemple :

```

protected String[] getServiceAccount(String alias) {
    String[] account = new String[2];

```

```

    ConnectionRepository conRep =
        MetadataManager.getInstance().connectionRepository();
    PBKey key = conRep.getStandardPBKeyForJcdAlias(alias);

    account[0] = key.getUser();
    account[1] = key.getPassword();

    return account;
}

```

5.3.6. PRECONISATIONS

Il est fortement recommandé de **définir une classe de base abstraite pour tous les objets « BusinessService » d'un même projet**. Cela permettra notamment de factoriser l'implémentation de la méthode `getServiceAccount()` qui dans la majorité des cas sera la même quelle que soit le service métier. Cette classe de base héritera de la classe du framework *BusinessService*.

Dans un souci d'homogénéisation, **le nom de ces classes Java devrait finir par le terme « Service », ou commencer par le terme « service » (mais ne pas mélanger les 2 notations)**.

5.4. GESTION TRANSACTIONNELLE

La gestion des transactions au sein des services métiers repose sur celle fournie par OJB. La classe *BusinessService* possède les 3 méthodes classiques de démarcation d'une transaction à savoir :

```

public void beginTransaction(IPersistentContextKey key) throws FwkException

public void commitTransaction(IPersistentContextKey key) throws FwkException

public void abortTransaction(IPersistentContextKey key) throws FwkException

```

Ces méthodes prennent toutes en paramètre un *IPersistentContextKey* en l'occurrence pour ojb l'implémentation *OJBPersistentContextKey* (dont le constructeur prend en paramètre le nom d'un alias correspondant à une configuration d'accès base cf. §**Erreur ! Source du renvoi introuvable.**).

Il ne faut donc pas utiliser les méthodes du même nom de la classe *PersistenceBroker*.

Exemple :

```

protected void doExecute() throws FwkException {
    OJBPersistentContext persistentContext = null;
    OJBPersistentContextKey key = new
    OJBPersistentContextKey(ALIAS,getServiceAccount(ALIAS));

    Intervention newIntervention = null ;

    try {
        persistentContext = (OJBPersistentContext) getPersistentContext(key);
        PersistenceBroker broker = persistentContext.getBroker();
        beginTransaction(key);
    }
}

```

```

...

broker.store(newIntervention);
commitTransaction(key);

} catch (Exception e) {
    e.printStackTrace();
    abortTransaction(key);
    throw new ServiceTechnicalException(e);
} finally {
    if (persistentContext != null) {
        closePersistentContext(key);
    }
}
}

```

Remarque :

Cette gestion transactionnelle ne supporte pas le commit à 2 phases. Le contexte transactionnel est géré sur la connexion jdbc, et non pas liée au JTA.

5.5. APPEL D'UN SERVICE METIER

L'invocation d'un service fonctionnel (ou métier) se fait en plusieurs étapes :

- Obtention d'une instance du service fonctionnel.
- Passage des paramètres d'entrée du service (optionnel).
- Exécution du service.
- Récupération des paramètres de sortie (optionnel).
- Libération des ressources.

5.5.1. OBTENTION D'UNE INSTANCE D'UN SERVICE FONCTIONNEL

Pour obtenir une instance d'un service métier, on doit faire appel à une classe jouant le rôle de fabrique (design-pattern « factory »). Cette classe fabrique implémente l'interface *com.inetpsa.fwk.service.ServiceFactoryIf*. Elle s'obtient en appelant la méthode statique *getInstance()* sur la classe *com.inetpsa.fwk.service.ServiceFactory*. Une instance d'un service métier ne doit donc pas être créée en appelant directement le constructeur de sa classe d'implémentation.

L'obtention d'une instance de service se fait en appelant la méthode *getService()* sur la classe d'implémentation de la fabrique. Cette méthode prend en paramètre les 2 informations suivantes :

- Le nom du service à instancier; Ce nom est défini par l'attribut *name* de l'élément *<service>* dans le fichier de configuration du framework *fwk.xml* (c.f. paragraphe §5.7).
- Un objet *User*.

```

BusinessService service = null;
User user;

try {
    // Récupération d'une instance du service fonctionnel
    // On suppose que le user est non null
    ServiceFactoryIf factory = ServiceFactory.getInstance();
    service = factory.getService(user, "myService");
}
...

```

Dans la plupart des cas, vous n'aurez pas à coder directement ces 2 lignes de code. En effet, l'appel d'un service métier se fait dans le cas d'une application web depuis une classe *FWKAction*. Cette classe dispose d'une méthode *getService()* prenant comme paramètre l'objet *HttpServletRequest* courant (pour retrouver l'utilisateur) et le nom du service. Pour plus de renseignements, cf. §6.6.

De même, dans le cas où l'on souhaiterait appeler un service métier depuis un autre service métier, la classe abstraite *BusinessService* dispose de la même manière que la classe *FWKAction*, une méthode *getService()*. Pour plus de renseignements, cf. §5.6.

5.5.2. PASSAGE DES PARAMETRES D'ENTREE DU SERVICE

Si le service métier nécessite le passage de paramètres d'entrée, il faut utiliser la méthode *setInput()* avec comme signature le nom du paramètre et l'objet correspondant à la valeur. Ce paramètre doit obligatoirement être de type objet (i.e. pas de type primitif).

```
...  
// Passage des paramètres d'entrée du service  
service.setInput( MyService.IN_PARAM_1, new Integer(15) );  
service.setInput( MyService.IN_PARAM_2, "toto" );  
...
```

5.5.3. EXECUTION DU SERVICE

L'exécution du service métier revient à appeler la méthode *execute()* sur l'instance du service.

```
...  
// Exécution du service  
service.execute();  
...
```

5.5.4. RECUPERATION DES PARAMETRES DE SORTIE

Si le service retourne des valeurs en sortie, ces valeurs peuvent être récupérées via la méthode *getOutput()* en passant en entrée le nom du paramètre de sortie.

```
...  
// Récupération des paramètres de sortie  
Collection liste = (Collection) service.getOutput( MyService.OUT_PARAM_1 );  
...
```

5.5.5. LIBERATION DES RESSOURCES

Enfin, il est très important de toujours libérer les ressources utilisées par le service en fin d'utilisation du service. Pour cela, on appelle la méthode *close()* sur le service dans un bloc *finally*.

```
} catch (FwkException e) {  
...  
}
```

```

    } finally {
        if (service != null) {
            try {
                service.close();
            } catch (FwkException e1) {
                e1.printStackTrace();
            }
        }
    }
}

```

5.6. APPEL INTER-SERVICES

Un service métier peut faire appel à un ou plusieurs autres services métiers. Pour ce faire, une méthode `getService()` existe au niveau de la classe `BusinessService` pour récupérer une instance du service que l'on veut appeler.

```
BusinessService getService(String name) throws FwkException
```

Pour rappel, le paramètre *name* est le nom du service défini par l'attribut *name* de l'élément `<service>` dans le fichier de configuration du framework *fwk.xml* (c.f. paragraphe §5.7).

Si un contexte transactionnel existe dans le service appelant, alors le service appelé fera partie intégrante de la transaction démarrée dans le service appelant. Pour cela, il est important d'utiliser les méthodes de démarcation transactionnelles fournies par la classe `BusinessService`.

Il n'est pas conseillé d'abuser des appels entre services métiers. Cela peut avoir des impacts négatifs sur les performances.

5.7. CONFIGURATION

La configuration des services métiers s'effectue dans le fichier *fwk.xml* en y déclarant un élément `<services>`. Dans cet élément, on retrouve l'élément `<factory>` permettant de configurer la fabrique d'instances de services métier et une liste d'éléments `<service>` permettant de configurer individuellement chaque service de l'application.

```

<services>
  <factory ... ></factory>
  <service ... > ... </service>
  <service ... > ... </service>
  ...
</services>

```

5.7.1. FABRIQUE

L'élément `<factory>` possède un attribut permettant de positionner l'implémentation de factory utilisée et plusieurs autres attributs pour gérer la paramétrie du pool.

Attribut	Description
<i>className</i>	Nom complet de la classe d'implémentation de la fabrique.

<i>maxActive</i>	contrôle le nombre maximum des objets (par clef) qui peuvent être utilisés par le pool en même temps. Si non positif, il n'y a aucune limite au nombre d'objets qui peuvent être en activité en même temps. Si maxActive est dépassé, le pool est épuisé.
<i>maxdle</i>	contrôle le nombre maximum des objets qui peuvent reposer le ralenti dans le pool (par clef) à tout moment. Quand la valeur est négative, il n'y a aucune limite au nombre d'objets en attente.
<i>whenExhaustedAction</i>	indique le comportement de la méthode borrowObject(java.lang.Object) lorsque l'on atteint les limites du pool: o Qd whenExhaustedAction est à 0 , borrowObject(java.lang.Object) déclenche une NoSuchElementException o Qd whenExhaustedAction est positionné à 1 , borrowObject(java.lang.Object) crée un nouvel objet et le retourne. o Qd whenExhaustedAction est positionné à 2, borrowObject(java.lang.Object) bloquera jusqu'à ce qu'un nouvel objet soit disponible
<i>timeBetweenEvictionRunsMillis</i>	indique combien de temps le thread d'éviction devrait attendre avant d'examiner i il y a des objets disponibles. Si la valeur est non positive, aucun thread ne sera lancé.
<i>minEvictableIdleTimeMillis</i>	indique la quantité de temps minimum qu'un objet peut rester disponible dans le pool avant qu'il soit éligible à l'expulsion. Quand non positif, aucun objet ne sera sorti du pool.

Exemple:

```
<factory className="com.inetpsa.fwk.service.ServiceFactoryImpl" maxActive="10"
...></factory>
```

5.7.2. SERVICE

Tout service utilisé dans l'application doit être déclaré dans le fichier *fwk.xml* par un élément *<service>*. Cet élément possède 3 attributs présentés dans le tableau ci-dessous.

Attribut	Description
<i>name</i>	Identifiant donné au service. Cet identifiant sert notamment pour la récupération d'une instance du service auprès de la factory de services. Cet attribut est case sensitive.
<i>Description</i>	Description fonctionnelle du service.
<i>className</i>	Nom complet de la classe d'implémentation du service.

L'élément *<service>* peut contenir des éléments fils *<condition>* permettant de définir les droits d'exécution de ce service en fonction des profils/rôles de l'utilisateur. Cette configuration est détaillée dans le paragraphe §10.4.1.

Exemple :

```
<services>
...
<service name="rechercherInterventions"
description="recherche multi criteres des interventions"
className="com.inetpsa.wja.service.RechercherInterventionsService">
...
</service>
<service name="creerIntervention"
```



```

description="création d'une intervention"
className="com.inetpsa.wja.service.CreerInterventionService">
...
</service>
...
</services>

```

5.7.3. PARAMETRES DECLARATIF DES SERVICES

Pour tout service, on peut déclarer les paramètres d'entrée à initialiser.
Ces paramètres sont définis au moyen de l'élément **<configurations>**.

Les valeurs des paramètres sont récupérées au niveau en tant que type String, les conversions de types restant à la charge du développeur.

On peut distinguer 2 familles de paramètres :

- **Paramètre mono valué** : correspond à un couple (clé, valeur) ;

```

<entry>
  <key>MAIL_HOST</key>
  <value>smtp.inetpsa.com</value>
</entry>

```

- **Paramètre multi valué** : correspond à un couple (clé, liste de valeurs), cette liste de valeurs étant une liste d'éléments **<value>** contenu dans un élément **<values>**.

```

<multipleentry> ❷
  <key>MAIL_RECIPIENTS</key>
  <values>
    <value>test@mpsa.com</value>
    <value>toto@mpsa.com</value>
    ...
  </values>
</ multipleentry >

```

Exemple :

Service de mail avec des entrées simples ❶ (serveur de mail & envoyeur) et une entrée multiple ❷ (liste de destinataires)

```

<services>
...
<service   name="mailer"
description="envoi d'un mail"
className="com.inetpsa.fwk.service.mail.SimpleMailService">
  <configurations>
    <entry> ❶
      <key>MAIL_HOST</key>
      <value>smtp.inetpsa.com</value>
    </entry>
    <entry>
      <key>MAIL_SENDER</key>
      <value>test@mpsa.com</value>
    </entry>
  </configurations>
</service>

```

```

        <entry>
            ...
        </entry>
        <multipleentry> ②
            <key>MAIL_RECIPIENTS</key>
            <values>
                <value>test@mpsa.com</value>
                <value>toto@mpsa.com</value>
                ...
            </values>
        </ multipleentry >

    </configurations>
    ...
</service>
...
</services>

```

Les valeurs des paramètres peuvent être récupérées dans un service au moyen des méthodes :

- Pour un paramètre mono valué :

```
protected String getParameterValue(String key) throws ServiceTechnicalException;
```

- Pour un paramètre multi valué :

```
protected Collection getParameterValues(String key) throws ServiceTechnicalException;
```

5.8. BUSINESSSERVICE MULTIFONCTIONS

En se basant sur un mécanisme proche de la gestion des DispatchAction au sein de Struts Lego 2 propose une implémentation de BusinessService capable de mettre en avant plusieurs points d'entrée, on peut donc maintenant créer une implémentation de BusinessService pour gérer plusieurs besoins métier qui seraient fonctionnellement re-groupables.

IMPLEMENTATION D'UN DISPATCHBUSINESSSERVICE

La classe de service implémentée n'hérite plus de [com.inetpsa.fwk.service.BusinessService](#) mais de [com.inetpsa.fwk.service.AbstractDispatchBusinessService](#) et la méthode doExecute n'est plus à implémenter puisque fournie par la classe [AbstractDispatchBusinessService](#), il reste ensuite à créer plusieurs méthodes sur la classe de service implémentée, chaque méthode correspondant à une *fonction* du service ayant une logique métier autonome.

Exemple :

```

public class serviceVehicule extends AbstractDispatchBusinessService {

    protected rechercherVehicule(){
        ...
    }

    protected creerVehicule(){

```

```

...
}

protected supprimerVehicule(){
...
}

protected String[] getServiceAccount(){
...
}
}

```

UTILISATION D'UN DISPATCHBUSINESSSERVICE

dans la mesure où un service multifonctions hérite de `BusinessService` il se comporte et s'utilise de la même manière qu'un service standard, la seule différence se situe au niveau de l'obtention de l'instance de service au travers de la factory : on ne demande plus une instance à partir d'un unique identifiant de `BusinessService` on lui associe *un nom de fonction*, exemple :

```

BusinessService service = null;
User user;

try {
    // Récupération d'une instance de service multifonctions
    // On suppose que le user est non null
    ServiceFactoryIf factory = ServiceFactory.getInstance();
    service = factory.getService(user, "serviceVehicule" ,
    "rechercherVehicule" );
    ...
}

```

Toutes les méthodes permettant précédemment de récupérer une instance de `BusinessService` (sur les actions struts ou les `BusinessService` eux-mêmes) ont donc été dupliquées avec une nouvelle signature prenant un paramètre supplémentaire : *un nom de fonction*.

Ensuite le service s'utilise de manière classique, exemple pour l'instance récupérée si dessus :

```

...
// Passage des paramètres d'entrée du service
service.setInput("immatriculation", "123PSA78");

// Exécution du service
service.execute();

// Récupération des paramètres de sortie
Collection liste = (Collection) service.getOutput("liste");
...

```

Sachant que l'exécution du `BusinessService` déclenchera ici la méthode `rechercherVehicule`, puisque l'instance obtenue auprès de la factory l'a été pour le service `serviceVehicule` de type multifonction et pour la fonction `rechercherVehicule`.

ASSOCIATION NOM DE FONCTION -> METHODE EXECUTEE

par défaut le nom de la méthode exécutée est le nom de la fonction demandée mais ce comportement peut au besoin être modifié, il faut alors surcharger sur la classe d'implémentation la méthode `getMethodName` en charge de fournir le nom de la méthode à exécuter en prenant en paramètre le nom de la fonction, exemple de l'implémentation par défaut :

```
protected String getMethodName(String functionName) throws FwkException{  
    return functionName ;  
}
```



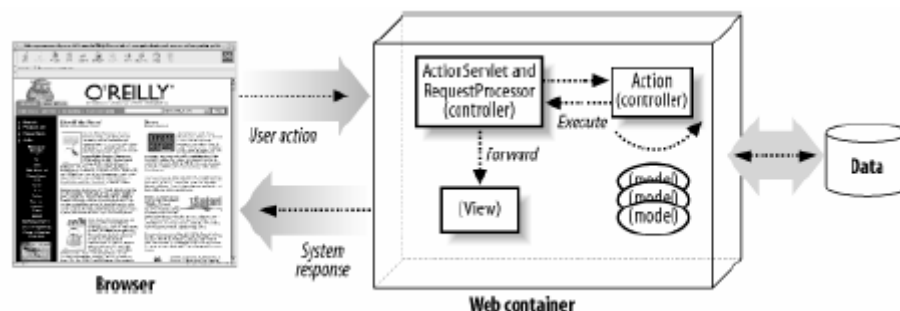
La sécurité étant toujours paramétrée au niveau du service il faut prendre garde à ne regrouper sur un `DispatchBusinessService` que des fonctions étant accessibles pour un même périmètre de sécurité.

6. FRAMEWORK STRUTS

La version de Struts intégrée au framework est la version Release 1.1. Le site officiel du framework Struts est situé à l'adresse <http://jakarta.apache.org/struts>.

6.1. INTRODUCTION

Le framework Struts implémente le modèle d'architecture MVC2 (Model-View-Controller) de Sun en fournissant d'une part une servlet contrôleur qui gère la cinématique de l'application et d'autre part des éléments de présentation tels que les taglibs JSP pour la couche d'affichage.



Ce framework met en œuvre tout un ensemble de composants dont voici les principaux :

Objet	Description
<i>ActionForward</i>	Objet interprété par le contrôleur permettant de faire le choix d'une Vue ou d'une classe Action.
<i>ActionForm</i>	Objet de la couche Vue permettant de sauvegarder les informations saisies par un utilisateur.
<i>ActionMapping</i>	Objet permettant d'associer une URI à une classe Action et définissant tout un ensemble de paramètres.
<i>ActionServlet / RequestProcessor</i>	Objets faisant partie de la couche Contrôleur qui reçoit et traite les requêtes des utilisateurs provenant de la couche de présentation (Vue).
<i>Classes Action</i>	Objets faisant partie de la couche Contrôleur qui interagit avec le Modèle pour requêter ou faire de la mise à jour de données et indique à l'ActionServlet, par l'intermédiaire d'un objet ActionForward, la prochaine vue à afficher.

Voici les étapes qui sont exécutées par le contrôleur à l'arrivée d'une requête :

1. Le contrôleur reçoit la requête et détermine grâce à son URI quelle est la classe *Action* associée à cette URI. Cette association est définie au moyen d'un *ActionMapping* dans le fichier de configuration *struts-config.xml*.
2. Le contrôleur exécute le code de l'*Action*. C'est lors de cette étape que les services fonctionnels peuvent être interrogés et par la même occasion les bases de données.
3. A la fin de l'exécution de l'*Action*, celle-ci envoie à l'*ActionServlet* un objet *ActionForward* indiquant la prochaine page JSP à afficher ou la prochaine *Action* à exécuter.
4. Finalement la page JSP récupère les données provenant du modèle par l'intermédiaire des tags JSP. Ces données ont été préalablement stockées lors de l'étape précédente sous forme d'attributs dans la requête ou dans la session.

6.2. EXTENSION DU FRAMEWORK STRUTS

Le framework PSA étend le framework Struts afin d'y intégrer les quelques spécificités telles que les fonctionnalités d'internationalisation, de sécurité, Les principales classes du framework Struts ont donc été dérivées et devront être utilisées en priorité.

Le tableau ci-dessous présente les classes du framework Struts qui ont été dérivées ainsi que les classes correspondantes dans le framework PSA.

Classe framework Struts	Classe dérivée framework PSA
<i>org.apache.struts.action.ActionServlet</i>	<i>com.inetpsa.fwk.struts.action.FWKActionServlet</i>
<i>org.apache.struts.tiles.TilesRequestProcessor</i>	<i>com.inetpsa.fwk.struts.action.FWKTilesRequestProcessor</i>
<i>org.apache.struts.action.RequestProcessor</i>	<i>com.inetpsa.fwk.struts.action.FWKRequestProcessor</i>
<i>org.apache.struts.action.ActionForm</i>	<i>com.inetpsa.fwk.struts.action.FWKActionForm</i>
<i>org.apache.struts.validator.ValidatorForm</i>	<i>com.inetpsa.fwk.struts.action.FWKValidatorForm</i>
<i>org.apache.struts.action.Action</i>	<i>com.inetpsa.fwk.struts.action.FWKAction</i>
<i>org.apache.struts.action.ActionMapping</i>	<i>com.inetpsa.fwk.struts.config.FWKActionMapping</i>
<i>org.apache.struts.config.ControllerConfig</i>	<i>com.inetpsa.fwk.struts.action.FWKControllerConfig</i>
<i>org.apache.struts.util.MessageResourceFactory</i>	<i>com.inetpsa.fwk.struts.messages.FWKMessageResourceFactory</i>
<i>org.apache.struts.util.MessageResource</i>	<i>com.inetpsa.fwk.struts.messages.FWKMessageResource</i>

Correspondance classes framework Struts / classes framework PSA

6.3. REQUESTPROCESSOR

La classe *RequestProcessor* est un des objets constituant la couche Contrôleur Struts. Comme il a été dit dans le paragraphe précédent, le framework PSA redéfinit cette classe fournie par Struts ainsi que la classe de configuration associée *ControllerConfig*. Pour bénéficier de ces extensions, il faut configurer l'élément `<controller>` du fichier *struts-config.xml* comme ci-dessous :

```
<controller className="com.inetpsa.fwk.struts.action.FWKControllerConfig"
  processorClass="com.inetpsa.fwk.struts.action.FWKTilesRequestProcessor"
  debug="0" contentType="text/html" inputForward="true">
  <set-property property="loginAction" value="login"/>
  <set-property property="timeoutForward" value="timeout"/>
</controller>
```

Le rôle de l'attribut *inputForward* est explicité au paragraphe §6.5.3.

Le rôle des propriétés *loginAction* et *timeoutForward* est explicité au paragraphe §10.4.2.

Le rôle de l'attribut *timeoutForward* est de paramétrer un forward global vers lequel est redirigé l'utilisateur lorsque sa session a expiré et que celui-ci soumet une nouvelle requête.

6.4. PAGES JSP

6.4.1. DECLARATION DES LIBRAIRIES DE TAGS JSP

Les librairies de tags JSP utilisées au sein de l'application web doivent être déclarées dans le fichier web.xml.

```
<taglib>
  <taglib-uri>/WEB-INF/struts-bean.tld</taglib-uri>
  <taglib-location>/WEB-INF/struts-bean.tld</taglib-location>
</taglib>
...
<taglib>
  <taglib-uri>/WEB-INF/struts-tiles.tld</taglib-uri>
  <taglib-location>/WEB-INF/struts-tiles.tld</taglib-location>
</taglib>
```

Les librairies des tags utilisés au sein d'une page JSP doivent être déclarées en tête de cette même page par la directive **taglib**. Dans le cas où une librairie n'est pas déclarée, les tags de cette librairie ne seront pas interprétés. Cependant, un oubli de ce type ne génère pas d'erreur ou d'exception. Il faut donc vérifier dès que l'on utilise un tag JSP si la librairie correspondante est déclarée en tête de la page JSP.

Exemple de déclaration de la librairie struts-html :

```
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
```

La valeur de l'attribut **uri** doit correspondre à la valeur donnée à l'élément taglib-uri dans le fichier web.xml.

L'attribut *prefix* permet de préciser le préfixe des tags à utiliser pour cette librairie. Un tag JSP est déclaré de la façon suivante : *<prefix : nom_tag>* (Ex : *<html:form>* pour le tag form).

6.4.2. PRECONISATIONS

L'utilisation des scriptlets (code Java inclus dans une page JSP) doit rester une exception. Il faut au maximum utiliser les tags JSP.

Les tags *<jsp:useBean>* et *<jsp:getProperty>* sont remplacés par l'utilisation des taglibs *struts-html* et *struts-bean*.

Toute requête souhaitant afficher une page de l'application doit obligatoirement passer par le contrôleur. Ce principe qu'il est nécessaire de respecter implique que tout lien dans une page JSP doit pointer non pas directement sur une autre page JSP mais sur une URI correspondant à une action Struts.

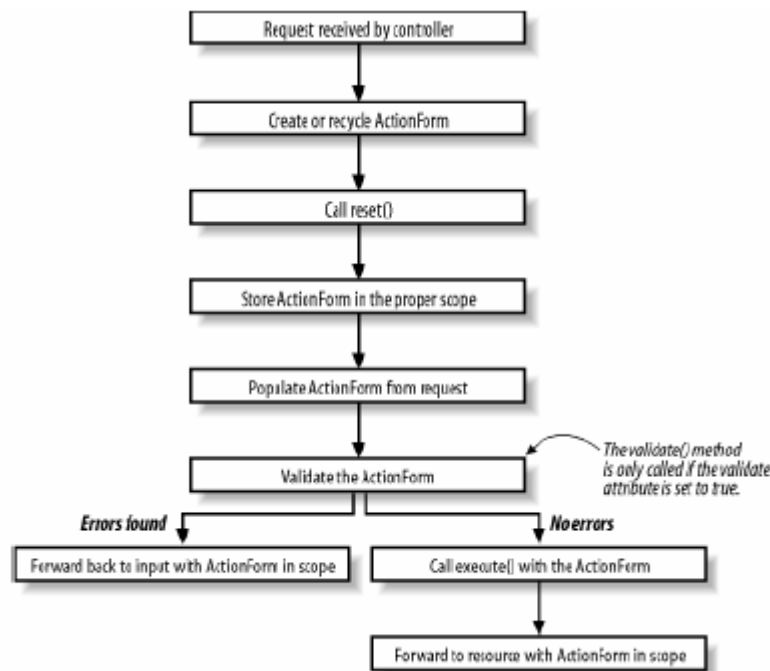
Le nom des pages JSP doit commencer par une minuscule.

6.5. ACTIONFORM

6.5.1. ROLE

Un *ActionForm* joue le rôle d'un buffer mémoire pour les données saisies dans un formulaire web par un utilisateur, chacun de ses attributs correspondant à un champ de saisie d'un formulaire. Il permet ainsi de valider syntaxiquement les données saisies avant qu'elles ne puissent être exploitées par l'Action et le service fonctionnel.

Pour bien mesurer le rôle d'un *ActionForm*, il est important de bien appréhender le cycle de vie d'un *ActionForm*. Le schéma ci-dessous montre l'enchaînement des actions orchestré par le contrôleur Struts.



Cycle de vie d'un *ActionForm*

1. Une requête arrive au contrôleur.
2. Si cette requête correspond à une action déclarant un *ActionForm*, une instance de cet *ActionForm* est récupérée en mémoire ou créée si elle n'existe pas. Cette instance est recherchée dans le scope (*session* ou *request*) spécifié par l'attribut *scope* de l'élément `<action>` dans le fichier *struts-config.xml*.
3. La méthode *reset()* de l'*ActionForm* est appelée. Elle permet de positionner si besoin est des valeurs par défaut pour les champs de saisie du formulaire.
4. L'instance de l'*ActionForm* est placée dans le scope indiqué par l'attribut *scope* de l'élément `<action>`.
5. Les attributs de l'*ActionForm* sont mis à jour à partir des paramètres contenus dans la requête HTTP. Pour chaque paramètre de la requête HTTP, le contrôleur appelle la méthode *set<Nom_param>(valeur_param)* de l'*ActionForm* si elle existe.
6. La méthode *validate()* de l'*ActionForm* est appelée si l'attribut *validate* de l'élément `<action>` est positionné à *true*.
7. Si la validation retourne des erreurs, le contrôleur fait suivre la requête HTTP vers une ressource (page JSP ou action) spécifiée par l'attribut *input* de l'élément `<action>` avec l'instance de l'*ActionForm* dans le scope indiqué. S'il n'y a pas d'erreurs, le contrôleur appelle la méthode *execute()* de l'action et forward la requête HTTP vers la ressource désignée par la valeur de retour de cette méthode *execute()*.

Exemple n°1 : Affichage d'une page contenant un formulaire.

1. L'utilisateur invoque l'url *http://localhost/wja/filtreIntervention.do* permettant d'afficher une page contenant un formulaire de recherche multicritères.

Cette url correspond à l'action struts suivante :


```
<action      path="/filtreInterventions"
              type="com.inetpsa.wja.ui.action.PreparerRechercheInterventionsAction"
              name="rechercheInterventionsForm" scope="request" validate="false">
    <forward name="success" path="/criteresRecherche.jsp"></forward>
</action>
```

2. Le bean d'id *rechercheInterventionsForm* est recherché dans le scope *request*. Il correspond à une instance de la classe *com.inetpsa.wja.ui.form.RechercheInterventionsForm* comme nous le précise l'élément *<form-bean>* suivant :

```
<form-bean  name="rechercheInterventionsForm"
              type="com.inetpsa.wja.ui.form.RechercheInterventionsForm">
</form-bean>
```

Dans notre exemple, nous supposons que le bean n'a pas été trouvé. Il est donc créé grâce au constructeur par défaut de l'ActionForm.

3. La méthode *reset()* est appelée sur l'ActionForm.
4. L'ActionForm est ensuite placé dans le scope *request*.
5. Pour chaque paramètre contenu dans la requête HTTP en cours, le contrôleur Struts vérifie si une méthode setter correspondante n'existe pas dans l'ActionForm. Si c'est le cas, cette méthode setter est appelée avec en paramètre la valeur du paramètre de la requête HTTP.
6. Le contrôleur Struts regarde ensuite la valeur de l'attribut *validate* de l'élément *<action>*. Si cet attribut a une valeur égale à *true* (valeur par défaut), la méthode *validate()* de l'ActionForm est invoquée. Dans notre exemple, la validation n'est pas effectuée.
7. Dans le cas où la validation ne générerait pas d'erreurs, le contrôleur exécute le code de l'Action.

Exemple n°2 : Action résultant d'une soumission d'un formulaire.

1. L'utilisateur invoque l'url *http://localhost/wja/searchIntervention.do* permettant d'afficher une page contenant un tableau résultat de la recherche multicritères.

Cette url correspond à l'action struts suivante :

```
<action      path="/searchInterventions"
              type="com.inetpsa.wja.ui.action.RechercherInterventionsAction"
              name="rechercheInterventionsForm" scope="request"
              validate="true" input="/criteresRecherche.jsp">
    <forward name="success" path="/resultatsRecherche.jsp"></forward>
</action>
```

2. Le bean d'id *rechercheInterventionsForm* est recherché dans le scope *request*. Dans notre exemple, le bean n'a pas été trouvé. Il est donc créé grâce au constructeur par défaut de l'ActionForm.
3. La méthode *reset()* est appelée sur l'ActionForm.
4. L'ActionForm est ensuite placé dans le scope *request*.
5. La requête HTTP contient les valeurs saisies par l'utilisateur dans le formulaire de recherche. Ces valeurs sont donc sauvegardées dans l'ActionForm.
6. La valeur de l'attribut *validate* de l'élément *<action>* étant à *true*, la méthode *validate()* de l'ActionForm est invoquée.

7. Si celle-ci renvoie *null* ou un objet *ActionErrors* vide, le code de l'Action est exécuté. Sinon, le contrôleur redirige l'utilisateur vers la ressource web indiquée par l'attribut *input* de l'élément `<action>`.

6.5.2. IMPLEMENTATION

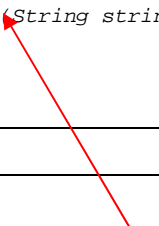
- Créer une nouvelle classe Java et la faire hériter soit de la classe **`com.inetpsa.fwk.struts.action.FWKValidatorForm`** si l'on désire utiliser le Struts-Validator pour la validation syntaxique (c.f. §6.5.4), soit de la classe **`com.inetpsa.fwk.struts.action.FWKActionForm`** si l'on souhaite implémenter « à la main » ces contrôles syntaxiques.
- Déclarer pour chaque champ de saisie (élément HTML `<input>`) du formulaire, un attribut privé de type *String* et générer les accesseurs (get et set) en tant que méthodes public. Les règles de nommage sont celles d'un composant JavaBean.

Remarques :

- Le nom de l'attribut doit correspondre au nom du champ de saisie du formulaire, c'est-à-dire à la valeur de l'attribut *property* du tag de la librairie *struts-html* correspondant.

```
public class LoginForm extends FWKActionForm {
    ...
    public String getUserId() { return userid; }
    public void setUserid(String string) {userid = string;}
    ...
}
```

```
<tr>
  <td align="right"><bean:message key="login.uid"/> </td>
  <td><html:text size="8" property="userid"/></td>
</tr>
```



- Pour les champs de saisie multivalués (ex : élément `<select>` à sélection multiple, *checkbox* ayant le même nom), il s'agira d'un attribut de type *String[]*.
- Surcharger la méthode *reset()*. Cette étape est optionnelle. Cette méthode permet de fixer des valeurs par défaut aux champs de saisie d'un formulaire.

Exemple:

```
public void reset(ActionMapping arg0, HttpServletRequest arg1) {
    super.reset(arg0, arg1);
    userid = "toto";
}
```

Particularité du champ de saisie de type *checkbox* :

Lorsqu'un formulaire contient une case à cocher et que l'*ActionForm* correspondant est placé en *session*, la méthode *reset()* doit remettre à *null* la valeur de l'attribut correspondant à la *checkbox*.

```
private String checkbox = null;

public void reset(ActionMapping arg0, HttpServletRequest arg1) {
    super.reset(arg0, arg1);
    checkbox = null;
}
```

Prenons l'exemple d'un formulaire contenant une case à cocher et un bouton submit. Si l'on coche la case et que l'on soumet le formulaire, l'attribut unique de l'ActionForm contiendra la valeur de la case à cocher. On suppose que la soumission du formulaire réaffiche la même page contenant le formulaire. Décochons maintenant la case et soumettons à nouveau le formulaire. Cette même page va se réafficher mais avec la case toujours cochée ce qui est contraire au résultat attendu. En effet, une case à cocher décochée ne transmet aucune valeur dans la requête HTTP et donc la méthode set de l'attribut de l'ActionForm ne sera pas appelée. Comme l'ActionForm est en session, il conserve l'ancienne valeur de la case à cocher, ce qui correspond à une case cochée. Il est donc nécessaire de remettre à null l'attribut de l'ActionForm pour ne plus prendre en compte l'ancienne valeur.

- Surcharger la méthode *validate()*. Cette étape est optionnelle. Cette méthode permet d'effectuer des validations syntaxiques sur les valeurs saisies par l'utilisateur dans le formulaire.

Exemple:

```
public ActionErrors validate(ActionMapping arg0, HttpServletRequest arg1) {
    ActionErrors errors = super.validate();
    if (errors == null) {
        errors = new ActionErrors();
    }

    if (StringUtils.isBlank(userid)) {
        errors.add(Globals.ERROR_KEY, new ActionError("errors.required", "uid"));
    }
    if (StringUtils.isBlank(password)) {
        errors.add(Globals.ERROR_KEY, new ActionError("errors.required", "pwd"));
    }

    return errors;
}
```

Remarques :

- Dans le cas où l'ActionForm hérite de la classe *FWKValidatorForm* (choix du Struts-Validator), la surcharge de cette méthode doit contenir impérativement l'appel à la méthode *super.validate()* sous peine de passer outre les contrôles définis dans le fichier *validation.xml*.

6.5.3. CONFIGURATION

Un *ActionForm* doit être déclaré dans le fichier de configuration *struts-config.xml* du framework Struts par un élément *<form-bean>*.

```
<struts-config>
...
<form-beans>
```

```
<form-bean name="loginForm" type="com.inetpsa.wja.ui.form.LoginForm"/>
...
</form-beans>
...
```

L'attribut **name** correspond à l'id donné au bean permettant de le retrouver dans un des 2 scopes suivants : *request* ou *session* (valeur par défaut).

L'attribut **type** correspond à la classe d'implémentation de l'ActionForm.

Comme il a été dit précédemment, lorsque la méthode *validate()* d'un ActionForm renvoie un objet ActionErrors non vide, le contrôleur redirige l'utilisateur vers la ressource web indiquée par l'attribut *input* de l'élément *<action>*. Cet attribut peut avoir comme valeur soit une page JSP, soit l'URI d'une autre action (ex : /login.do), soit enfin le nom d'un élément *<forward>* (local à l'action ou global).

Ce dernier choix (celui du *<forward>*) est préconisé car il permet de concentrer au niveau des éléments *<forward>* toutes les ressources vers lesquelles une action peut transférer la requête y compris celle en cas d'erreurs de validation. Cependant, pour ce cas précis, **il est nécessaire de configurer au préalable le contrôleur Struts, i.e. l'élément *<controller>*. Il faut fixer son attribut *inputForward* à *true*.**

Exemple :

```
<action path="/searchInterventions" name="rechercheInterventionsForm"
type="com.inetpsa.wja.ui.action.RechercherInterventionsAction"
scope="request" input="valErrors" validate="true">
  <forward name="success" path="/resultatsRecherche.jsp"></forward>
  <forward name="valErrors" path="/criteresRecherche.jsp"></forward>
</action>
...
<controller ... inputForward="true">
...
</controller>
```

6.5.4. STRUTS-VALIDATOR

Le mécanisme du Struts-Validator permet de définir dans un même fichier de configuration xml tous les contrôles syntaxiques sur les champs de saisie des formulaires de l'application. Ces contrôles syntaxiques suivant leur nature peuvent être effectués soit côté client par du code javascript, soit côté serveur, soit coté client et côté serveur. Le tableau ci-dessous présente les différents contrôles syntaxiques existants :

Contrôle	Description	Client	Serveur
required	Test si le champ est vide.	✓	✓
requiredIf	Test si le champ est vide uniquement si une condition portant sur un ou plusieurs autres champs du formulaire est remplie.		✓
maxlength	Test si la valeur du champ ne dépasse pas une certaine longueur.	✓	✓
mask	Test si la valeur du champ correspond à une expression régulière.	✓	✓
byte	Test si la valeur du champ est de type byte.	✓	✓
short	Test si la valeur du champ est de type short.	✓	✓
integer	Test si la valeur du champ est de type integer.	✓	✓

long	Test si la valeur du champ est de type long.		✓
float	Test si la valeur du champ est de type float.	✓	✓
double	Test si la valeur du champ est de type double.		✓
date	Test si la valeur du champ est de type date avec un certain pattern.	✓	✓
intRange	Test si la valeur du champ est de type integer et si elle est comprise entre une certaine valeur min et une certaine valeur max.	✓	✓
floatRange	Test si la valeur du champ est de type float et si elle est comprise entre une certaine valeur min et une certaine valeur max.	✓	✓
creditCard	Test si la valeur du champ correspond à un numéro de carte de crédit.	✓	✓
email	Test si la valeur du champ correspond à une adresse email.	✓	✓

Pour activer le Struts-Validator, il faut suivre les étapes suivantes :

1. Ajouter la **déclaration du plugin** dans le fichier *struts-config.xml*

```
<plug-in className="org.apache.struts.validator.ValidatorPlugIn">
  <set-property property="pathnames"
    value="/WEB-INF/validator-rules.xml,/WEB-INF/validation.xml"/>
</plug-in>
```

2. Les classes *ActionForm* doivent hériter de la classe ***com.inetpsa.struts.action.FWKValidatorForm***.
3. **Définir les contrôles à effectuer sur les champs de saisie des formulaires dans le fichier *validation.xml***. Ce fichier, ainsi que le fichier *validator-rules.xml* fourni avec Struts, doivent être placés dans le répertoire *WEB-INF* de la webapp.

Exemple:

```
<form-validation>
  <formset>
    <form name="loginForm">
      <field property="userid" depends="required">
        <msg name="required" key="errors.required"/>
        <arg0 name="required" key="login.uid"/>
      </field>
      <field property="password" depends="required">
        <msg name="required" key="errors.required"/>
        <arg0 name="required" key="login.pwd"/>
      </field>
    </form>
    <form name="rechercheInterventionsForm">
      <field property="codeIntervention" depends="integer">
        <msg name="integer" key="errors.integer"/>
        <arg0 name="maxlength" key="rchInt.codeInt"/>
      </field>
      <field property="client" depends="maxlength">
        <msg name="maxlength" key="errors.maxlength"/>
        <arg0 name="maxlength" key="rchInt.client"/>
        <arg1 name="maxlength" key="7" resource="false"/>
        <var var-name="maxlength" var-value="7"/>
      </field>
    </form>
  </formset>
</form-validation>
```

```
</form>
</formset>
</form-validation>
```

4. **Activer la validation côté serveur** au niveau des Actions Struts en positionnant l'attribut `validate` de l'élément `<action>` à `true` dans le fichier `struts-config.xml`.

Exemple :

```
<action path="/searchInterventions" name="rechercheInterventionsForm"
        scope="request" type="com.inetpsa.wja.ui.action.RechercherInterventionsAction"
        input="/criteresRecherche.jsp" validate="true">
    <forward name="success" path="/resultatsRecherche.jsp"></forward>
</action>
```

5. **Activer la validation côté client** en ajoutant dans la partie en-tête de la page JSP contenant le formulaire, le tag `<html:javascript>` et ajouter l'événement `onsubmit` sur le formulaire.

Exemple :

```
<head>
...
<html:javascript formName="loginForm" dynamicJavascript="true"
                 staticJavascript="false"/>
<script language="JavaScript1.1" src="<html:rewrite page="/staticJS.jsp"/>">
</script>
</head>
...
<html:form method="post" action="/authentifier"
           onsubmit="return validateLoginForm(this);">
```

Remarques :

- La page `staticJS.jsp` contient le code suivant :

```
<%@ page contentType="application/x-javascript" %>
<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<html:javascript dynamicJavascript="false" staticJavascript="true"/>
```

- Le nom de la méthode javascript à exécuter sur l'évènement `onsubmit` du formulaire se construit de la façon suivante : on ajoute au terme `validate` le nom de l'ActionForm (i.e. la valeur de l'attribut `name` de l'élément `<form-bean>` dans le fichier `struts-config.xml`) avec la 1^{ère} lettre en majuscule.

6.5.5. PRECONISATIONS

Les attributs d'un « ActionForm » doivent tous être de type String ou de type String[]. En effet, ils ont pour rôle de sauvegarder les valeurs des champs de saisie des formulaires web, or ces valeurs sont transmises côté serveur par le protocole HTTP que sous la forme de chaînes de caractères.

Dans un souci d'homogénéisation, **le nom de ces classes Java devra finir par le terme « Form »**.

En terme de validation syntaxique, **la solution Struts-Validator est préconisée** car elle permet une meilleure réutilisation des contrôles de premier niveau.

6.6. ACTIONS

6.6.1. ROLE

Un objet de type « Action » fait partie du Contrôleur dans le modèle MVC. Une Action joue donc un rôle de coordination entre d'une part la vue (pages JSP & ActionForm) et d'autre part le modèle (services fonctionnels & objets métiers).

Il s'agit d'une implémentation du design pattern *Command*.

On peut classer les actions en 3 groupes principaux :

- **Type 1** : Actions permettant de préparer l'affichage d'une page JSP ne contenant pas de formulaire.
- **Type 2** : Actions permettant de préparer l'affichage d'une page JSP contenant un formulaire.
- **Type 3** : Actions permettant de traiter les données provenant d'un formulaire.

6.6.2. IMPLEMENTATION

L'implémentation d'une classe Action se résume à 2 étapes :

- Créer une classe Java et la faire hériter de la classe ***com.inetpsa.fwk.struts.action.FWKAction***.
- **Implémenter la méthode *doExecute()***. Cette méthode doit renvoyer un objet *ActionForward* indiquant au contrôleur vers quelle ressource web (page JSP, action) il doit transmettre la requête.

Le déroulement classique d'une méthode *doExecute()* est décrit ci-dessous. Suivant le type de l'action, certaines étapes n'ont pas lieu d'être.

CONVERSION DES DONNEES DE PRESENTATION EN DONNEES METIER

Cette étape n'est valable que dans le cas d'une action de type 3. Cela signifie qu'elle est exécutée suite à la soumission d'un formulaire et donc à l'envoi de données de présentation (i.e. données saisies par l'utilisateur dans le formulaire et sauvegardées dans l'ActionForm). Ces données métier serviront comme paramètres d'entrée pour les services fonctionnels.

Les données métier, contrairement aux données de présentation, peuvent être de tout type : *String*, *Double*, *Date*, *Integer*, Pour tous les types autres que *String*, une conversion est donc nécessaire. **La classe de base *FWKAction* met à disposition du développeur des méthodes utilitaires permettant de faire ces conversions.**

```
Date convertStringToDate(String data)
Date convertStringToDate(String data, String pattern)
Double convertStringToDouble(String data)
Double convertStringToDouble(String data, String pattern)
...
```

APPEL D'UN SERVICE FONCTIONNEL

La récupération d'une instance d'un service fonctionnel se fait grâce à la méthode *getService()* présente dans la classe *FWKAction*. Cette méthode prend en entrée l'identifiant du service. Cet identifiant est défini dans le fichier de configuration du framework *fwk.xml* (c.f. paragraphe §5.7).


```
/**
 * Récupération d'un Service fonctionnel.
 * @param request Requête HTTP
 * @param serviceName Nom du service fonctionnel
 */
BusinessService getService(HttpServletRequest request, String serviceName)
```

Exemple d'appel du service fonctionnel « listerElementsReference » permettant de récupérer une liste d'éléments provenant d'une table de référence. Ce service prend en entrée 3 paramètres et restitue en sortie une collection d'objets.

```
BusinessService service = null;
Collection c = null;

try {
    // Récupération d'une instance du service fonctionnel
    service = getService(request, "listerElementsReference");

    // Passage des paramètres d'entrée du service
    service.setInput(
        ListerElementsReferenceService.IN_ELEMENT_CLASS,
        typeElement);
    service.setInput(
        ListerElementsReferenceService.IN_ORDER_BY_ATTR,
        attrOrderBy);
    service.setInput(
        ListerElementsReferenceService.IN_ORDER_BY_ASC,
        new Boolean(sortAsc));

    // Exécution du service
    service.execute();

    // Récupération des paramètres de sortie
    c =
        (Collection) service.getOutput(
            ListerElementsReferenceService.OUT_LISTE);

} catch (FwkException e) {
    throw e;
} finally {
    // Libération des ressources
    if (service != null) {
        service.close();
    }
}
```

Remarques :

- **Avant de vouloir invoquer un service fonctionnel, il est primordial de connaître son interface** à savoir ses différents paramètres d'entrée/sortie (nom, type, caractère optionnel) et les exceptions pouvant être potentiellement levées. Ces informations doivent être consignées dans la javadoc du service fonctionnel.
- Pour pouvoir être appelé, tout service fonctionnel doit être déclaré dans le fichier de configuration *fwk.xml*.

- **La libération des ressources doit se faire obligatoirement dans un bloc finally** pour éviter qu'en cas d'exception les ressources utilisées par le service ne soit jamais relâchées.

POSITIONNEMENT DES BEANS RESULTAT DANS LE SCOPE REQUEST OU LE SCOPE SESSION

Cette étape est valable surtout lorsque l'action invoque un service fonctionnel retournant une ou plusieurs valeurs. Pour mettre à disposition ces valeurs à la page JSP ou à une autre action, il est nécessaire de placer ces données dans la requête ou dans la session. Un cas typique est la récupération de listes de données de référence pour les afficher dans un formulaire en tant que liste déroulante.

Exemple:

```
public ActionForward doExecute(  
    ActionMapping arg0,  
    ActionForm arg1,  
    HttpServletRequest arg2,  
    HttpServletResponse arg3)  
    throws Exception {  
  
    ...  
  
    // Positionnement des listes de référence dans la requete HTTP  
    arg2.setAttribute(  
        "difficultes",  
        getCollectionRef(arg2, Difficulte.class, "id", true));  
    arg2.setAttribute(  
        "priorites",  
        getCollectionRef(arg2, Priorite.class, "id", true));  
  
    ...  
  
    // Positionnement de la liste des experts dans la requete HTTP  
    arg2.setAttribute("experts", getExperts(arg2));  
  
    ...  
}
```

Pour permettre une meilleure montée en charge, il faut veiller à ce que la session ait une taille raisonnable. Cette taille raisonnable est fonction des ressources du serveur.

MISE A JOUR DE L'ACTIONFORM.

Cette étape n'est valable que dans le cas d'une action de type 2. Il s'agit de positionner des valeurs par défaut aux champs de saisie du formulaire. A la différence de la méthode `reset()` de l'ActionForm qui a aussi pour rôle de positionner des valeurs par défaut aux champs de saisie d'un formulaire (cf. §6.5.2), les valeurs par défaut définies au sein de l'Action peuvent être des données métier résultant de l'appel à un service fonctionnel.

Exemple:

```
public ActionForward doExecute(  
    ActionMapping arg0,  
    ActionForm arg1,  
    HttpServletRequest arg2,  
    HttpServletResponse arg3)  
    throws Exception {
```

```

        CreateInterventionForm form = (CreateInterventionForm) arg1;
        ...

        // Positionnement du user authentifié comme expert par défaut
        User user = getUser(arg2);
        form.setExpertID(user.getId());
        ...

    }

```

REDIRECTION VERS UNE PAGE JSP OU UNE AUTRE ACTION.

La dernière étape qui doit être présente dans toute Action est d'indiquer au contrôleur vers quelle ressource web rediriger l'utilisateur.

La liste des ressources web qu'il est possible d'atteindre depuis l'action doit être définie au niveau du fichier de configuration *struts-config.xml* en définissant un ensemble d'éléments *<forward>*. Chaque élément *<forward>* est défini par un identifiant qui est la valeur de son attribut *name*.

Exemple :

```

<action path="/createIntervention" name="createInterventionForm" scope="request"
        type="com.inetpsa.wja.ui.action.CreateInterventionAction"
        validate="true" input="valErrors" roles="redacteurs">
    <forward name="success" path="/filtreInterventions.do"></forward>
    <forward name="valErrors" path="/afficherCreateIntForm.do"></forward>
    <forward name="failure" path="createInt"></forward>
</action>

```

Cette action définit 3 éléments *<forward>* :

- Le forward « success » pour le cas où l'action se déroulerait normalement.
- Le forward « valErrors » dans le cas où des erreurs sont détectées à la validation du formulaire.
- Le forward « failure » dans le cas où une exception est levée.

La méthode *mapping.findForward()* permet de renvoyer l'objet *ActionForward* correspondant en fonction de l'identifiant passé en paramètre.

Exemple:

```

public ActionForward doExecute(
    ActionMapping arg0,
    ActionForm arg1,
    HttpServletRequest arg2,
    HttpServletResponse arg3)
    throws Exception {

    String fwdName = "success";
    ...
    BusinessService service = null;

    try {
        ...
    }
}

```

```

        if (service != null)
            service.execute();
    } catch (BusinessException be) {
        fwdName = "failure";
        ActionMessages am = new ActionMessages();
        addExceptionMessages(am, be);
    } finally {
        if (service != null) {
            service.close();
        }
    }

    return arg0.findForward(fwdName);
}

```

6.6.3. PRECONISATIONS

Les classes Action se comportant comme des servlets, **l'usage des variables d'instances doit être proscrit ou tout du moins très limité**. En effet, l'instance d'une classe Action est multithreadé et peut donc être exécutée par plusieurs clients à la fois.

Une Action ne doit pas contenir de traitements métiers ni de règles de gestion. Ce rôle est dévolu au service fonctionnel.

Dans un souci d'homogénéisation, **le nom de ces classes Java devra finir par le terme « Action »**.

6.6.4. CONFIGURATION

Avant de configurer action par action, il est nécessaire de configurer l'élément `<action-mappings>` afin de pouvoir utiliser le nouvel attribut **profils** sur les éléments `<action>`. Pour de plus amples explications quant au rôle de cet attribut **profils**, cf. §10.4.2.

```

<action-mappings type="com.inetpsa.fwk.struts.config.FWKActionMapping">
...
</action-mappings>

```

La configuration d'une Action se fait en déclarant un élément `<action>` dans le fichier `struts-config.xml`.

Suivant le type de l'action, le nombre d'attributs de l'élément `<action>` requis varie. Pour rappel, les différents types possibles pour une action sont décrits au §6.6.1.

Attribut	Description	Type 1	Type 2	Type 3
path	Path relatif au contexte de la webapp désignant l'action. Ce path doit commencer par un '/' et ne doit pas contenir de caractère \.	✓	✓	✓
type	Nom de la classe Action. Elle doit hériter de la classe <code>com.inetpsa.fwk.struts.action.FwkAction</code>	✓	✓	✓
name	Nom de l'ActionForm associé à cette action		✓	✓

input	Nom du forward qui sera utilisé en cas d'erreurs de validation du formulaire. Si l'attribut validate est à true, cet attribut est obligatoire.	(✓)	(✓)
validate	Mettre à true pour autoriser la validation côté serveur. Valeur par défaut : true.	(✓) false	(✓)
scope	Scope (request ou session) sous lequel l'ActionForm est recherché et stocké. Valeur par défaut : session.	(✓)	(✓)
profils	Liste des profils autorisés à exécuter l'action.	(✓)	(✓) (✓)
roles	Liste des rôles autorisés à exécuter l'action.	(✓)	(✓) (✓)

✓ : attribut requis / (✓) : attribut optionnel

Exemple de configuration d'une action de type 1 :

Cette Action affiche une page contenant un tableau de données.

```
<action      path="/listInterventions"
            parameter="listeInt"
            type="com.inetpsa.wja.ui.action.AfficherListeIntAction">
    <forward name="liste" path="listeInt"></forward>
</action>
```

Exemple de configuration d'une action de type 2 :

Cette Action affiche une page contenant un formulaire d'authentification.

```
<action      path="/login"
            name="loginForm"
            scope="request"
            validate="false"
            type="com.inetpsa.wja.ui.action.PreparerLoginAction">
    <forward name="login" path="login"></forward>
</action>
```

Exemple de configuration d'une action de type 3 :

Cette Action permet d'invoquer un service fonctionnel effectuant une recherche multicritères et d'afficher les résultats.

```
<action      path="/searchInterventions"
            name="rechercheInterventionsForm"
            scope="request"
            type="com.inetpsa.wja.ui.action.RechercherInterventionsAction"
            input="valErrors">
    <forward name="success" path="listeInt"></forward>
    <forward name="valErrors" path="/filtreInterventions.do"></forward>
</action>
```

6.6.5. GESTION DES EXCEPTIONS

Un des rôles des classes Action est de traiter les exceptions métiers, c'est-à-dire les exceptions résultant de règles de gestion non validées dans la méthode *execute()* d'un service fonctionnel. Ces exceptions doivent dériver de la classe *BusinessException*.

Il faut donc déclarer un bloc **try catch** sur le code manipulant les services fonctionnels.

```
public ActionForward doExecute(
    ActionMapping arg0,
    ActionForm arg1,
    HttpServletRequest arg2,
    HttpServletResponse arg3)
    throws Exception {

    String fwdName = "success";
    BusinessService service = null;

    try {
        ...
        if (service != null)
            service.execute();

    } catch (BusinessException be) {
        fwdName = "failure";
        ActionErrors ae = new ActionErrors();
        addExceptionMessages(ae, be);
        saveErrors(arg2, ae);
    } finally {
        ...
    }

    return arg0.findForward(fwdName);
}
```

Le traitement d'une exception fonctionnelle se traite le plus souvent de la façon suivante :

- Un objet *ActionErrors* est créé. Il s'agit d'une liste d'objets *ActionError* correspondant chacun à un message d'erreur.
- On appelle la méthode *addExceptionMessages()* présente dans la classe *FWKAction* afin d'y placer le message d'erreur de l'exception dans la liste *ActionErrors*.
- On place l'objet *ActionErrors* sous forme d'attribut dans la requête grâce à la méthode *saveErrors()*.
- Enfin, on redirige la requête vers une page JSP (le plus souvent, il s'agira de la même page contenant le formulaire que l'on vient de soumettre) pour y afficher le(s) message(s) d'erreur grâce aux tags JSP `<html:messages>` ou `<html:errors>`.

```
<html:messages id="msg">
    <bean:write name="msg"/><br>
</html:messages>
```

ou

```
<html:errors/>
```

6.6.6. ACCES A LA SESSION

La classe abstraite *FWKAction* possède 3 méthodes utilitaires permettant de placer un objet en session, de récupérer un objet de la session et de supprimer un objet de la session.

```
/**
 * Mise en session d'un objet.
 * @param request Requête HTTP
 * @param key Cle de l'objet
 * @param obj Objet
 */
void putInSession(HttpServletRequest request, String key, Object obj)

/**
 * Récupération d'un objet en session.
 * @param request Requête HTTP
 * @param key Cle de l'objet
 * @return Objet
 */
Object getFromSession(HttpServletRequest request, String key)

/**
 * Suppression d'un objet de la session.
 * @param request Requête HTTP
 * @param key Cle de l'objet
 */
void removeFromSession(HttpServletRequest request, String key)
```

L'intérêt d'utiliser ces méthodes est qu'elles permettent de tracer les différentes actions effectuées autour de la session de chaque utilisateur. Pour cela, il faut autoriser l'affichage des traces de niveau INFO pour le logger désigné par la constante *FWKAction.LOGGERKEY_SESSIONTRACKER*. Pour de plus amples informations sur cette opération, c.f. paragraphe §7.

6.7. TAGLIBS

6.7.1. TAGLIB STRUTS-HTML

La librairie de tags « struts-html » contient entre autres les tags permettant de générer le code des éléments HTML d'un formulaire : <form>, <input>. Les autres tags de la librairie concernent d'autres éléments HTML tels que les liens, les images, etc. Le code généré est compatible HTML 4.01.

Vous trouverez la liste des tags de cette librairie ainsi que la documentation associée à l'URL suivante : <http://jakarta.apache.org/struts/userGuide/struts-html.html>.

6.7.2. TAGLIB STRUTS-BEAN

La librairie de tags « struts-bean » contient un ensemble de tags permettant de manipuler des beans dans n'importe quel scope (page, request, session, application) : définition de nouveaux beans, affichage des propriétés d'un bean, etc.

Vous trouverez la liste des tags de cette librairie ainsi que la documentation associée à l'URL suivante : <http://jakarta.apache.org/struts/userGuide/struts-bean.html>.

6.7.3. TAGLIB STRUTS-LOGIC

La librairie de tags « struts-logic » contient des tags conditionnels (permettant d'interpréter ou non le corps du tag en fonction d'une condition), un tag permettant d'itérer sur une collection et des tags permettant de naviguer dans l'application.

Vous trouverez la liste des tags de cette librairie ainsi que la documentation associée à l'URL suivante : <http://jakarta.apache.org/struts/userGuide/struts-logic.html>.

6.7.4. TAGLIB DISPLAY

La librairie de tags « display » permet l'affichage d'un tableau de données avec des fonctionnalités de pagination, de tri, de regroupement, etc.

Toutes les informations sur cette taglib se trouvent à l'url <http://displaytag.sourceforge.net>.

6.7.5. TAGLIB FWK

La librairie fwk contient les tags suivants :

Tag	Description
jspsecurity	Tag renvoyant une réponse HTTP 403 lorsqu'un utilisateur tente d'accéder directement à la page JSP sans passer par le contrôleur struts en utilisant un mapping vers la page.
onlinehelplink	Tag générant un lien hypertexte vers une page d'aide.
initblock	Tag permettant de définir une condition pour afficher ou non un bloc HTML. Cette condition est liée à un rôle ou à une liste de profils. A utiliser conjointement avec le tag block.
block	Tag définissant un bloc HTML à afficher ou non suivant une condition définie par le tag initblock. A utiliser conjointement avec le tag initblock.
exporter	Tag permettant de générer le code relatif à l'export d'un tableau dans divers formats (CSV, Excel, PDF, XML, HTML et RTF).
exporterTemplate	Tag permettant de définir les templates Velocity à utiliser (seulement pour CVS, XML et HTML). A utiliser conjointement avec le tag exporter.
exporterItem	Tag permettant de définir les colonnes à exporter. A utiliser conjointement avec le tag exporter.

TAG JSPSECURITY

Ce tag ne possède pas d'attributs. Pour de plus amples informations concernant son rôle et son utilisation, cf. §10.4.3.

TAG ONLINEHELPLINK

Ce tag permet de générer automatiquement un lien vers une page d'aide en ligne. L'URL du lien est générée par la méthode *doGenerateUrl()* de la classe spécifiée par l'attribut *className*.

Attribut	Obligatoire	Type	Description
<i>id</i>	Non	String	L'identifiant du lien d'aide.

className	Oui	String	Nom complet de la classe d'implémentation héritant de la classe <i>com.inetpsa.fwk.onlinehelp.beans.OnlineHelpUrlGenerator</i> .
-----------	-----	--------	--

Liste des attributs du tag `<fwk:onlinehelplink>`

Exemple :

- On crée une classe héritant de la classe abstraite *OnlineHelpUrlGenerator* et on implémente la méthode *doGenerateUrl()*. Cette méthode construit un lien qui aura comme action l'ouverture d'une popup affichant le contenu de la page d'aide relative à la page dans laquelle sera placé le tag JSP. Cette page d'aide est un fichier html placé dans un répertoire *help*.

```
public class WJAHelpURLGenerator extends OnlineHelpUrlGenerator {

    public WJAHelpURLGenerator() {
        super();
    }

    protected String doGenerateUrl() {
        return "<a href=\"javascript:winOpen( "
            + "\"http://"
            + getRequest().getServerName()
            + ":"
            + getRequest().getServerPort()
            + getRequest().getContextPath()
            + "/help/"
            + getPath()
            + ".html', 'helpPopup', 500, 400, true, false)\">Aide</a>";
    }
}
```

- On place dans la page JSP à l'endroit où l'on veut afficher le lien d'aide, le tag `<fwk:onlinehelplink>` :

```
<fwk:onlinehelplink className="com.inetpsa.wja.util.WJAHelpURLGenerator"/>
```

TAG INITBLOCK

Attribut	Obligatoire	Type	Description
<i>id</i>	Oui	String	L'identifiant du bloc html.
<i>role</i>	Oui, si l'attribut <i>profil</i> n'est pas renseigné	String	Le rôle que l'utilisateur doit avoir pour entrer dans le bloc html.
<i>profiles</i>	Oui, si l'attribut <i>rôle</i> n'est pas renseigné	String	La liste des profils que l'utilisateur doit avoir pour entrer dans le bloc html.
<i>operator</i>	Non	String	Permet de spécifier un opérateur AND ou OR entre les profils définis dans l'attribut <i>profiles</i> .

Liste des attributs du tag `<fwk:initblock>`

Exemple :

- On définit un bloc html *admin* lié au rôle applicatif *administrateur* :

```
<fwk:initblock id="admin" role="administrateur"/>
```

- On définit un bloc html *redacteur* lié aux profils *correcteur* et *lecteur* :

```
<fwk:initblock id="redacteur" profiles="correcteur,lecteur" />
```

TAG BLOCK

Attribut	Obligatoire	Type	Description
<i>id</i>	Oui	String	L'identifiant du bloc html
<i>allowed</i>	Oui	boolean	Indique si le contenu du bloc html doit être exécuté lorsque l'utilisateur répond ou non aux critères définis lors de l'initialisation du bloc

Liste des attributs du tag `<fwk:block>`

Exemple :

- On définit un bloc html à exécuter dans le cas où l'utilisateur possède le rôle applicatif *administrateur* :

```
<fwk:block id="admin" allowed="true">
    Vous êtes un administrateur!<BR>
</fwk:block>
```

- On définit un bloc html à exécuter dans le cas où l'utilisateur ne posséderait pas le rôle applicatif *administrateur* :

```
<fwk:block id="admin" allowed="false">
    Vous n'êtes pas un administrateur!<br>
</fwk:block>
```

TAG EXPORTER

Pour l'utilisation du tag d'export, il faut ajouter dans le fichier « **struts-config.xml** », les lignes suivantes :

```
<action path="/exporterAction"
    type="com.inetpsa.fwk.export.actions.ExporterAction">
</action>
```

Attribut	Obligatoire	Type	Description
<i>action</i>	Oui	String	Action Struts à appeler.
<i>name</i>	Oui	String	Nom de l'objet en session
<i>templateKey</i>	Oui	String	Clé d'internationalisation, contenant le code HTML à générer. Le message doit contenir les paramètres suivants : <ul style="list-style-type: none"> {0} : CSV {1} : Excel {2} : XML {3} : PDF {4} : HTML {5} : RTF
<i>charset</i>	Non	String	Charset à utiliser (par défaut UTF-8)
<i>type</i>	Oui	String	Liste des exports souhaités séparés par le caractère ','. Les types possibles sont : <ul style="list-style-type: none"> csv xml excel html pdf rtf Le type par défaut est CSV.
<i>className</i>	Non	String	Nom de la classe implémentation de l'interface : com.inetpsa.fwk.exporter.IExportAdditionalContext qui permet d'ajouter des objets additionnels dans le contexte Velocity.

Liste des attributs du tag `<fwk:exporter>`

Exemple de format pour **templateKey** :

- Pour un export au format CSV, Excel et XML

```
export.format.key=Export : <a href={0}><span>CSV</span></a> | <a href={1}><span>Excel</span></a>
</a> | <a href={2}><span>XML</span></a></a>
```

- Pour un export PDF et CVS

```
export.format.key=Format d'exports : <a href={3}><span>PDF</span></a> | <a href={0}><span>CSV</span></a></a>
```

Exemple :

Dans la page JSP :

```
<fwk:exporter      name="myList"      templateKey="export.format.key"      type="csv,excel"
action="exporterAction.do">
...
</fwk:exporter>
```

Attention : Le modèle templateKey, doit être en cohérence avec les options utilisées type= "csv,excel,..."

TAG EXPORTERTEMPLATE

Attribut	Obligatoire	Type	Description
csv	Non	String	Nom du template Velocity à utiliser pour l'export CSV.
xml	Non	String	Nom du template Velocity à utiliser pour l'export XML.
html	Non	String	Nom du template Velocity à utiliser pour l'export HTML.

Liste des attributs du tag <fwk:exporterTemplate>

Les templates Velocity par défaut sont les suivants :

- CSV

```
#foreach( $header in $headers )
#if ( $velocityCount > 1 ),$header#else$header#end
#end

#foreach( $bean in $exporterBean )
#set ( $i = $velocityCount - 1 )
#foreach( $property in $properties )
#set ( $j = $velocityCount - 1 )
#if ( $j > 0 ),$list.get($bean, $j)#else$list.get($bean, $j)#end
#end
#end
```

- XML

```
<?xml version="1.0" encoding="$exporter.getCharset()" ?>

<export>
#foreach( $bean in $exporterBean )
#set ( $i = $velocityCount - 1 )
<line>
#foreach( $property in $properties )
#set ( $j = $velocityCount - 1 )
<$property>$list.get($bean, $j)</$property>
#end
</line>
#end
</export>
```

- HTML

```
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=$exporter.getCharset()">
</head>
<body>
<table>
<tr>
#foreach( $header in $headers )
<td>$header</td>
#end
</tr>
#foreach( $bean in $exporterBean )
<tr>
```

```
#foreach( $property in $properties )
#set ($j = $velocityCount - 1)
    <td>${list.get($bean, $j)}</td>
#end
</tr>
#end
</table>
</body>
</html>
```

Rem. : Pour la définition des templates, il faut indiquer le chemin, le chargeur de templates cherchera dans le classloader.

Exemple :

```
<fwk:exporter name="myList" templateKey="export.csv" type="csv,xml" action="exporterAction.do">
  <fwk:exporterTemplate
    csv="com/inetpsa/prd/templates/MyCSV.vm"
    xml="com/inetpsa/prd/templates/MyXML.vm"
  />
  <fwk:exporterItem property="id"/>
  <fwk:exporterItem property="libelle"/>
</fwk:exporter>
```

TAG EXPORTERITEM

Attribut	Obligatoire	Type	Description
titleKey	Non	String	Clé de message d'internationalisation contenant le titre de la colonne (utilisé pour les entêtes).
property	Oui	String	Nom de l'attribut de l'objet à exporter

Liste des attributs du tag <fwk:exporterItem>

Exemple :

```
<fwk:exporter name="myList" templateKey="export.csv" csv="true" action="exporterAction.do">
  <fwk:exporterItem property="id"/>
  <fwk:exporterItem property="libelle"/>
</fwk:exporter>
```

6.8. STRUTS-TILES : SOLUTION DE TEMPLATING

Différentes solutions sont à notre disposition pour faire du templating en ce qui concerne l'interface graphique web :

- L'utilisation des frames HTML qui reste la solution la plus performante.
- L'utilisation du tag <jsp:include>
- L'utilisation des Tiles fourni dans Struts qui est très proche de la solution précédente. Cependant, elle apporte les avantages suivants :

- Les définitions des templates sont centralisées dans un seul fichier xml. Cela évite donc la multiplication des fichiers (un pour la définition du template et un autre pour le corps du template).
- Il existe un mécanisme d'héritage entre templates, ce qui permet une factorisation maximale.

6.8.1. ACTIVATION DES TILES DANS STRUTS

Afin d'exploiter les Tiles dans une application Struts, il faut tout d'abord **déclarer le plugin *TilesPlugin* dans le fichier *struts-config.xml***.

```
<plug-in className="org.apache.struts.tiles.TilesPlugin" >
  <set-property property="definitions-config" value="/WEB-INF/tiles-defs.xml" />
  <set-property property="definitions-parser-validate" value="true" />
</plug-in>
```

La propriété *definitions-config* permet de spécifier le chemin d'accès au fichier de configuration XML des définitions Tiles.

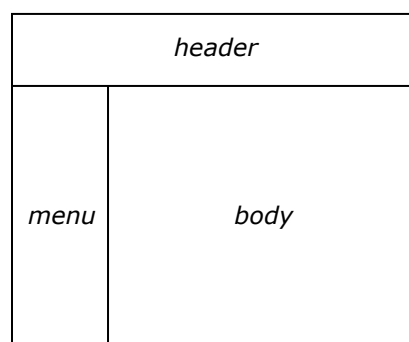
La deuxième condition nécessaire à l'utilisation des Tiles dans Struts est que **le contrôleur *RequestProcessor* doit hériter de la classe *org.apache.struts.tiles.TilesRequestProcessor***. Pour cela, le framework met à disposition la classe *FWKTilesRequestProcessor*. La configuration du contrôleur est alors la suivante.

```
<controller className="com.inetpsa.fwk.struts.action.FWKControllerConfig"
  processorClass="com.inetpsa.fwk.struts.action.FWKTilesRequestProcessor" ... >
  ...
</controller>
```

Enfin, il ne faut pas oublier de déclarer la librairie de tags *struts-tiles* dans le fichier *web.xml* ainsi qu'en début de toute page JSP utilisant cette librairie.

6.8.2. UTILISATION DES TILES

Pour illustrer l'utilisation des Tiles, nous allons prendre l'exemple d'un site ayant comme layout principal le layout suivant :



Ce layout comporte 3 zones de dimension fixe : un bandeau horizontal en haut (*header*), un bandeau vertical à gauche (*menu*) et la page principale (*body*). Il est représenté par la page *mainLayout.jsp* dont voici le code :

```

<%@ taglib uri="/WEB-INF/struts-html.tld" prefix="html" %>
<%@ taglib uri="/WEB-INF/struts-tiles.tld" prefix="tiles" %>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html:html>
<head>
  <html:base/>
  <title><tiles:getAsString name="title"/></title>
</head>
<body marginwidth="0" marginheight="0" topmargin="0" leftmargin="0">
  <div style="position: absolute; left:0px; top:0px; width: 1000px; height:70px">
    <tiles:insert attribute="header" flush="true"/>
  </div>
  <div style="position: absolute; left:0px; top:70px; width: 150px; height:500px">
    <tiles:insert attribute="menu" flush="true"/>
  </div>
  <div style="position: absolute; left:150px; top:70px; width:850px; height:500px;">
    <tiles:insert attribute="body" flush="true"/>
  </div>
</body>
</html:html>

```

Le tag `<tiles:insert>` permet d'insérer la valeur d'un attribut si ce dernier est de type page JSP ou de type définition.

Dans le cas d'un attribut de type String, il faut utiliser le tag `<tiles:getAsString>`.



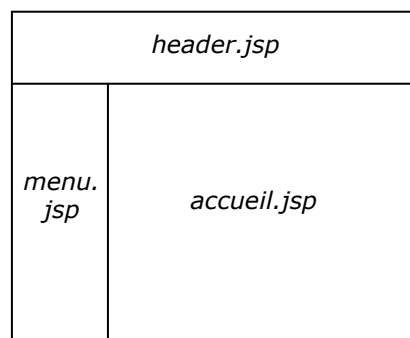
Le serveur WebSphere requiert la présence de l'attribut *flush* avec une valeur égale à *true* pour le tag `<tiles:insert>`.

Pour positionner des valeurs à ces attributs et donc constituer une page basée sur ce layout, il faut définir ce que l'on appelle une définition au sens Tiles. Une définition se déclare dans le fichier `/WEB-INF/tiles-defs.xml` (i.e. un des fichiers XML définis par la propriété *definitions-config* du plugin, cf. §6.8.1) de la manière suivante :

```

<definition name="accueil" path="/layouts/mainLayout.jsp">
  <put name="title" value="Ecran de bienvenue" />
  <put name="header" value="/header.jsp" />
  <put name="menu" value="/menu.jsp" />
  <put name="body" value="/accueil.jsp" />
</definition>

```



Les pages `header.jsp`, `menu.jsp` et `accueil.jsp` ne sont que des morceaux de pages. Elles ne contiennent pas de section `<head>` ni de section `<body>`.



Dans le cas où l'on souhaite donner une valeur null (ie : une String vide) à la valeur d'un attribut d'une Définition, il faut alors ajouter à l'élément `<put>` l'attribut *direct* avec une valeur égale à *true*.

Exemple:

```
<put name="menu" value="" direct="true"/>
```

6.8.3. FORWARD VERS UNE DEFINITION

La définition *accueil* représente désormais une page JSP assemblée de plusieurs sous pages JSP. Elle ne possède pourtant pas de chemin physique car elle est décrite au moyen d'un élément XML. Pour permettre de rediriger la requête vers une définition depuis une action Struts, **il faut indiquer dans le path d'un *forward* le nom de la définition.**

L'Action décrite ci-dessous est une *ActionForward* fournie par le framework Struts dont le rôle est d'invoquer la ressource spécifiée par l'attribut *parameter*. Dans notre cas, il s'agit d'afficher la définition *accueil*.

```
<action path="/accueil" parameter="accueil"
        type="org.apache.struts.actions.ForwardAction">
</action>
```

Autre exemple, la définition d'une page de login :

```
<definition name="login" path="/layouts/mainLayout.jsp">
  <put name="title" value="Ecran d'authentification" />
  <put name="header" value="/header.jsp" />
  <put name="menu" value="/menu.jsp" />
  <put name="body" value="/login.jsp" />
</definition>
```

et l'action correspondante pour afficher la page de login :

```
<action path="/login" ...>
  <forward name="login" path="login"></forward>
</action>
```



Un *global-forward* ne peut pointer vers une Définition. Seuls les `<forward>` inclus dans un élément `<action>` en sont capables.

6.8.4. HERITAGE ENTRE DEFINITIONS

Afin de permettre une factorisation maximale, une définition peut hériter d'une autre définition. Cela signifie que la définition fille aura comme attributs ceux de la définition mère avec la possibilité de les surcharger et même d'en ajouter de nouveaux.

On définit qu'une définition hérite d'une autre définition par le mot clé *extends*.

Exemple:

```
<!-- Main page layout used as a root for other page definitions -->
<definition name="mainLayout" path="/layouts/mainLayout.jsp">
    <put name="title" value="" />
    <put name="header" value="/header.jsp" />
    <put name="menu" value="" />
    <put name="body" value="" />
</definition>

<!-- Page layout used as a root for unsecured page definitions -->
<definition name="unsecuredPage" extends="mainLayout">
</definition>

<!-- Page layout used as a root for secured page definitions -->
<definition name="securedPage" extends="mainLayout">
    <put name="menu" value="/menu.jsp" />
</definition>

<!-- Login page definition -->
<definition name="login" extends="unsecuredPage">
    <put name="title" value="Ecran d'authentification" />
    <put name="body" value="/login.jsp" />
</definition>

<!-- Search page definition -->
<definition name="searchInt" extends="securedPage">
    <put name="title" value="Ecran de recherche mutli-critères" />
    <put name="body" value="/criteresRecherche.jsp" />
</definition>
```

Les définitions *mainLayout*, *unsecuredPage* et *securedPage* ne seront jamais invoqués directement par un *forward* car elles représentent uniquement des layouts et donc des pages incomplètes.

6.9. CONFIGURATION DU FICHIER WEB.XML

Les différentes configurations à effectuer dans le fichier web.xml sont les suivantes :

1. Déclaration de la servlet contrôleur du framework Struts.
2. Définition du mapping pour cette servlet.
3. Déclaration de la liste des taglibs utilisées au sein de la webapp.

DECLARATION DE LA SERVLET CONTROLEUR DU FRAMEWORK STRUTS

L'application web ne doit déclarer qu'une seule servlet par laquelle toutes les requêtes destinées à l'application passeront. La classe d'implémentation de cette servlet doit être la classe *org.apache.struts.action.ActionServlet* ou toute autre classe dérivée. Le framework dispose de la classe ***com.inetpsa.fwk.struts.action.FWKActionServlet***. On donne communément le nom *action* pour cette servlet mais ce n'est pas une obligation.

```
<servlet>
    <servlet-name>action</servlet-name>
    <servlet-class>com.inetpsa.fwk.struts.action.FWKActionServlet</servlet-class>
```

Deux paramètres d'initialisation de cette servlet doivent être renseignés :

Le paramètre *config* dont la valeur précise le chemin du fichier de configuration de Struts. Ce fichier doit être placé dans le répertoire *WEB-INF*.

```
<init-param>
  <param-name>config</param-name>
  <param-value>/WEB-INF/struts-config.xml</param-value>
</init-param>
```

Le paramètre *validate* de type booléen dont la valeur précise si oui ou non la validation syntaxique du fichier de configuration XML de Struts doit être effectuée. On la positionne à *false*.

```
<init-param>
  <param-name>validate</param-name>
  <param-value>false</param-value>
</init-param>
```

DEFINITION DU MAPPING DE LA SERVLET DU CONTROLEUR STRUTS

Les mapping des servlets présents dans ce fichier permettent au conteneur web de savoir vers quelle servlet il doit diriger la requête HTTP de la webapp et ce en fonction de l'URI de la requête.

Le mapping de la servlet *action* du contrôleur Struts définie précédemment est communément définie par l'URI « *.do ». Cela veut dire que toute requête dont l'URI aura une extension.do sera prise en compte par le contrôleur Struts. Là encore, ce n'est pas une obligation.

```
<servlet-mapping>
  <servlet-name>action</servlet-name>
  <url-pattern>*.do</url-pattern>
</servlet-mapping>
```

DECLARATION DE LA LISTE DES TAGLIBS UTILISEES AU SEIN DE LA WEBAPP

Cette configuration n'est pas propre à Struts mais à toute application web utilisant des taglibs JSP. Toute taglib utilisée dans l'application doit être déclarée dans ce fichier.

Exemple pour la taglib *struts-bean*

```
<taglib>
  <taglib-uri>/WEB-INF/struts-bean.tld</taglib-uri>
  <taglib-location>/WEB-INF/struts-bean.tld</taglib-location>
</taglib>
```

6.10. UPLOAD DE FICHIERS

Le framework LEGO propose une extension afin de palier aux différents problèmes rencontrés sur la gestion des charset sur certaine plate-forme.

MISE EN PLACE DE L'EXTENSION

La configuration se positionne au niveau du contrôleur Struts.

```
<controller ...>
<set-property property="multipartClass"
value="com.inetpsa.fwk.struts.fileupload.FWKMultiPartRequestHandler"/>
<set-property property="uploadHeaderEncodingCharset" value="UTF-8"/>
</controller>
```

multipartClass : Positionne le MultiPartRequestHandler du framework LEGO en lieu et place de celui proposé par défaut dans STRUTS.

uploadHeaderEncodingCharset : la propriété sert à indiquer le charset utilisé dans la requête http provenant du client web.

6.11. BIBLIOTHEQUE STRUTS-LAYOUT

Le framework LEGO est livré avec une bibliothèque de composants graphiques Web avancés : Struts-Layout.

Struts-Layout est en fait une surcouche open-source de Struts : il est possible de mêler tags Struts et tags Struts-Layout dans une même JSP. Là où Struts définit les tags « bas niveau » (faire des boucles, afficher un message internationalisé, tester des conditions, ...), Struts-Layout offre des possibilités d'affichage et de formatage des informations plus évoluées :

- définition et affichage de menus,
- gestion des onglets,
- affichage des tableaux complexes (avec tri côté client ou serveur, avec pagination pour les longues collections, avec champ input à l'intérieur, avec alternance de style pour les lignes du tableau, ...etc.),
- gestion des formulaires,
- définition et affichage d'arbres statiques ou dynamiques,
- possibilité de définir des « skins » pour les applications,
- définition d'un mécanisme de prise en compte des rôles,
- facilité de positionnement des composants graphiques grâce à des tags permettant de simplifier le layout (grille, ligne, colonne)
- ...etc.

Une partie des tags va être brièvement décrite dans ce chapitre de sorte à avoir une première prise en main de cette bibliothèque. Pour aller plus en profondeur dans l'utilisation de Struts-Layout, il est vivement conseillé d'aller consulter son site Web (notamment la partie Tutorial) :

<http://struts.application-servers.com>

De plus, les pages JSP d'exemple livrées avec la distribution de Struts-Layout sont un excellent moyen de comprendre le fonctionnement des tags.

6.11.1. LES TAGS DE POSITIONNEMENT

Struts-Layout dispose que quelques tags facilitant le positionnement des autres tags au sein d'une page Web. Ce sont des onglets, des panneaux, des grilles, des colonnes ou des lignes dans lesquelles viennent d'insérer les tags Struts-Layout.

LES PANELS

Un panneau est en fait un cadre qui peut avoir un titre, et qui va contenir un ensemble d'autres tags (qui peuvent eux-mêmes être des panels).

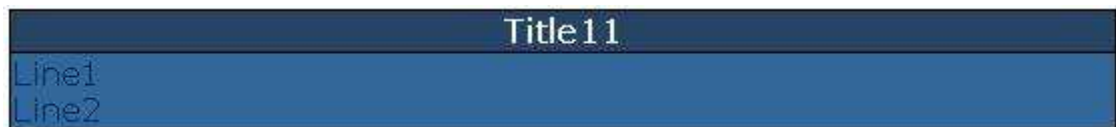
Attribut	Obligatoire	Type	Description
<i>arg0, arg1, arg2, arg3, arg4</i>	Non	String	Paramètres utilisés pour le titre du panneau.
<i>align</i>	Non	String	Spécifie l'alignement des éléments dans le panneau. Par défaut : « center ».
<i>key</i>	Non	String	Clé du message à utiliser pour le titre.
<i>styleClass</i>	Non	String	La classe CSS à utiliser.
<i>width</i>	Non	String	La largeur du panneau (absolue ou relative).

Liste (**partielle**) des attributs du tag `<layout:panel>`

Exemple :

```
<layout:panel styleClass="FORM" key="titleKey">
    <layout:message key="line1Key" />
    <layout:message key="line2Key" />
</layout:panel>
```

Rend :



Il est possible d'utiliser des images pour rendre le contour et le titre du panneau (par exemple pour avoir des coins ronds ou des contours particuliers). Consulter le site en ligne pour plus d'information : <http://struts.application-servers.com/doc/tags/panel.html> .

LES ONGLETS

Les onglets sont des panneaux particuliers : ils sont superposés et peuvent être affichés en cliquant sur une extension visible du panneau.

Address	Credit Card
Street1	street1.1
Street2	street1.2
City	city1
State	ST1
Zip Code	A1B-1C1
Country	US

Exemple d'onglets

La balise `<layout:tab>` est utilisée pour définir un panneau-onglet. La balise `<layout:tabs>` sert à regrouper un ensemble de panneaux-onglets. Ainsi, l'exemple ci-dessus correspondrait au code (partiel) ci-dessous :

```
<layout:tabs styleClass="MY_TABS" width="600px">
  <layout:tab key="message.address" width="120px">
    ...
  </layout:tab>
  <layout:tab key="message.creditCard" width="120px">
    ...
  </layout:tab>
</layout:panel>
```

Attribut	Obligatoire	Type	Description
<i>styleClass</i>	Non	String	La classe CSS à utiliser.
<i>selectedTabKeyName</i>	Non	String	Nom du paramètre (ou attribut) de requête qui contient la valeur de la clé de l'onglet qui doit être affiché en 1 ^{er} plan.
<i>width</i>	Non	String	La largeur des panneaux (absolue ou relative) contenus dans ce tag.

Liste des attributs du tag <layout:tabs>

Attribut	Obligatoire	Type	Description
<i>key</i>	Oui	String	Clé du message à utiliser pour le titre. C'est cette valeur qui est utilisée par l'attribut « selectedTabKeyName » de <layout:tabs>.
<i>forward, href, page</i>	Non	String	Si un de ces attributs est spécifié, alors le titre de l'onglet devient cliquable pour invoquer l'url correspondante.
<i>width</i>	Non	String	La largeur des onglets (pas des panneaux).

Liste des attributs du tag <layout:tab>



Si « forward », « href » ou « page » ne sont pas spécifiés, alors les panneaux et les onglet sont gérés en DHTML et sont imprimés dans la même page HTML. S'ils sont spécifiés, alors seul le panneau et l'onglet qui a été cliqué sont générés dans la page résultante.

LES GRILLES, LIGNES ET COLONNES

Ces quelques tags permettent de positionner finement les composants graphiques d'une page Web :

- **<layout:grid>** : permet de dessiner une grille où les composants sont insérés les uns à la suite des autres, en respectant un nombre de colonnes défini et en étant alignés verticalement et horizontalement.
- **<layout:column>** : permet d'insérer des composants verticalement (équivalent à une grille de 1 colonne)
- **<layout:row>** : permet d'insérer des composants sur une même ligne.
- **<layout:space>** : ajoute un espace vide.

Attribut	Obligatoire	Type	Description
<i>borderSpacing</i>	Non	Entier	Espace autour du bord de la grille. Par défaut : 0.
<i>cols</i>	Non	Entier	Nombre de colonnes. Par défaut : 2.
<i>space</i>	Non	booléen	Si false, il n'y aura pas d'espace entre chaque cellule.
<i>styleClass</i>	Non	String	La classe CSS à utiliser.
<i>width</i>	Non	String	La largeur de la grille.

Liste des attributs du tag <layout:grid>

Attribut	Obligatoire	Type	Description
<i>styleClass</i>	Non	String	La classe CSS à utiliser.

Liste des attributs du tag <layout:column>

Attribut	Obligatoire	Type	Description
<i>space</i>	Non	String	Si false, il n'y aura pas d'espace entre chaque cellule.

Liste des attributs du tag <layout:row>

Attribut	Obligatoire	Type	Description
<i>styleClass</i>	Non	String	La classe CSS à utiliser.

Liste des attributs du tag <layout:space>

6.11.2. GESTION DES FORMULAIRES

Struts-Layout facilite grandement le développement de formulaire Web.

Le code présenté ci-dessous (qui n'a pas d'intérêt fonctionnellement) ne contient aucune balise HTML ni Struts. Il permet cependant de rendre (en 19 lignes seulement) la page Web présentée juste après.

```
<layout:html>
  <layout:form action="myAction" styleClass="FORM">
    <layout:text key="search.name" property="name" styleClass="FIELD"/>
    <layout:text key="search.town" property="town" styleClass="FIELD"/>
    <layout:submit>
      <layout:message key="search.submit"/>
    </layout:submit>
  </layout:form>
</layout:html>
```

Les balises en gras sont celles qui assurent la gestion du formulaire HTML. (On ne s'occupera pas ici de la balise `<layout:html>` qui remplace de manière quasi transparente celle de Struts) Elles peuvent être regroupées en 3 groupes :

- **Le `<layout:form>`** : c'est l'équivalent du formulaire `<html:form>`, mais inclus dans un panel (d'où le bord noir).
- **Les champs input, comme `<layout:text>`** : ce sont les équivalent des input comme `<html:text>`, mais ils gèrent d'autres choses en plus (ils ont leur propre libellé, ils affichent une étoile si le champ est obligatoire, ils savent s'aligner proprement tout seul, ...etc)
- **Les boutons d'action, comme `<layout:submit>`** : ce sont les équivalent des boutons `<html:text>`, mais ont eux aussi des fonctionnalités en plus (comme le fait de pouvoir spécifier un nom de méthode pour une DispatchAction, donc la possibilité d'avoir facilement plusieurs boutons submit pour un formulaire).

LE FORMULAIRE

Le tag `<layout:form>` encapsule en fait le tag Struts `<html:layout>`. C'est pourquoi tous ses attributs ne sont pas décrits dans le tableau suivant. Seuls ceux qui ont été rajoutés y sont présentés.

Attribut	Obligatoire	Type	Description
<i>key</i>	Non	String	Clé du message à utiliser pour le titre du formulaire. Si cet attribut est spécifié, le formulaire sera affiché dans un panneau.
<i>reqCode</i>	Non	String	Nom de la méthode à appeler si l'action du formulaire est une DispatchAction.
<i>styleClass</i>	Non	String	La classe CSS à utiliser.
<i>onkeypress</i>	Non	String	Permet d'appeler du Javascript lorsqu'une touche est pressée.

Liste des attributs supplémentaires du tag `<layout:form>`

LES CHAMPS INPUT

Les champs input sont multiples :

- **`<layout:text>`** : champ input de type Text
- **`<layout:textarea>`** : champ input de type TextArea
- **`<layout:checkbox>`** : champ input de case à cocher
- **`<layout:password>`** : champ input de type mot de passe
- **`<layout:file>`** : champ input pour afficher une boîte de sélection de fichier
- **`<layout:date>`** : champ input de type Date
- **`<layout:radio>`** : champ input de type boutons radio
- **`<layout:select>`** : champ input de type liste déroulante
- **`<layout:option>`, `<layout:options>`, `<layout:optionsCollection>`** : permet de spécifier les éléments d'une liste déroulante
- **`<layout:radios>`** : comme `<layout:select>`, mais rend une liste de boutons radio

- **<layout:checkboxes>** : comme <layout:select>, mais rend une liste de case à cocher
- **<layout:detail>** : permet d'afficher le détail d'une ligne d'un tableau (cf. les collections et <layout:collectionDetail>).

Ces tags correspondent tous plus ou moins à des tags Struts équivalent. Ils ajoutent cependant certaines fonctionnalités intéressantes :

- Ils génèrent leur propre libellé
- Il est possible de rendre les tags non éditables
- Les tags savent afficher les erreurs qui sont liées à la propriété (du formulaire) qu'ils gèrent
- Ils peuvent afficher une étoile indiquant qu'ils doivent être renseignés. Cette étoile disparaît dynamiquement lorsque l'utilisateur frappe des caractères dans le champ, mais réapparaît si le champ est vidé alors qu'il est obligatoire.

Ces champs ont tout un ensemble d'attributs en commun qui est décrit ci-dessous. Pour connaître les attributs particuliers, se référer à la documentation en ligne.

Attribut	Obligatoire	Type	Description
<i>key</i>	Non	String	Clé du libellé du champ.
<i>arg0, arg1, arg2, arg3, arg4</i>	Non	String	Paramètres utilisés pour le libellé.
<i>name</i>	Non	String	Nom du bean (dans n'importe quel scope) pour lequel on veut éditer la propriété. Par défaut, il s'agit du bean Formulaire.
<i>property</i>	Oui	String	Propriété à consulter / éditer.
<i>isRequired</i>	Non	booléen	Si true, alors une étoile sera affichée à côté du champ.
<i>styleClass</i>	Non	String	La classe CSS à utiliser.
<i>policy</i>	Non	String	Nom des vérifications à effectuer pour afficher ou pas le champ. Cf. http://struts.application-servers.com/features/policy.html
<i>hint</i>	Non	String	Clé du message à afficher pour le tooltip du libellé.
<i>tooltip</i>	Non	String	Clé du message à afficher pour le tooltip du champ.
<i>layoutId</i>	Non	String	Identifiant HTML du tag englobant le composant. Cela permet notamment de le manipuler avec du Javascript ou du DHTML.
<i>onchange</i>	Non	String	Permet de mettre du Javascript appelé lorsque la valeur du champ change.
<i>value</i>	Non	String	Valeur initiale par défaut du champ.

Liste (**partielle**) des attributs communs des tags Input

LES BOUTONS D'ACTION

Plusieurs tags permettent de générer les boutons des formulaires.

Les tags **<layout:submit>**, **<layout:image>**, **<layout:cancel>** ou **<layout:reset>** correspondent aux tag Struts, avec des fonctionnalités en plus :

- Attribut **reqCode** : possibilité de spécifier la méthode de la DispatchAction à appeler lors de la l'envoi de la requête.
- Attribut **policy** : nom des vérifications à effectuer pour afficher ou pas le champ (comme pour les inputs).

Le tag **<layout:formActions>** permet de générer automatiquement les boutons Submit / Reset / Cancel. L'attribut **align** permet de contrôler leur positionnement (par défaut : « center »).

6.11.3. AFFICHAGE DES COLLECTIONS

Struts-Layout permet de gérer l'affichage de collection de données de manière simple. Il gère notamment les points suivants :

- Création de tableaux triables côté client et serveur
- Possibilité d'itérer sur 2 collections en même temps
- Alternance de styles (notamment couleurs) pour les lignes
- Utilisation de styles différents selon critères
- Possibilité de mettre des boutons radio ou des cases à cocher si la collection se situe dans un formulaire (permet de sélectionner des beans), ainsi que des champs input
- Fonctionnalités de pagination pour les collections longues, avec affichage personnalisable de l'état de la pagination
- Possibilité de mettre en surbrillance une ou des lignes qui correspondent à un critère défini
- Ecriture, dans les colonnes, de liens calculés sur la base d'actions Struts, de forward Struts, de pages ou de simples URL
- Affichage de certains détails d'une ligne dans des champs externes au tableau
- Formatage des données à afficher
- ...etc.

AFFICHER UNE COLLECTION DANS UN TABLEAU

L'affichage d'une collection Java dans un tableau est une chose rendue très aisée grâce à Struts-Layout. L'exemple ci-dessous (code et rendu correspondant) le prouve.

La collection Java sauvegardée en requête sous le nom **peopleList** contient un ensemble d'objets People (lesquels possèdent des méthodes **getFirstname()**, **getLastname()**, ...etc.).

```
<layout:collection name="peopleList" styleClass="ARRAY">
  <layout:collectionItem title="person.firstname" property="firstname" />
  <layout:collectionItem title="person.lastname" property="lastname" />
  <layout:collectionItem title="person.street" property="street" />
  <layout:collectionItem title="person.town" property="town" />
</layout:collection>
```

Firstname	Lastname	Street	Town
John	Smith	London Street	London
Steve	Smith	Baker Street	London
Bill	Smith	Cardigan Gardens	London

Les tags **<layout:collection>** et **<layout:collectionItem>** sont plus amplement décrits ci-dessous.

Attribut	Obligatoire	Type	Description
<i>name</i>	Non	String	Nom de la collection (dans n'importe quel scope) sur laquelle on souhaite itérer. Par défaut, il s'agit du bean Formulaire.
<i>property</i>	Non	String	Propriété du bean (représenté par "name") qui donne la collection à afficher.

<i>id</i>	Non	String	Identifiant utilisé pour référencer l'objet de la collection sur lequel on est en train de travailler à chaque itération.
<i>name2</i>	Non	String	Idem que "name", mais pour la 2 ^{nde} collection (si on souhaite itérer sur 2 collections)
<i>property2</i>	Non	String	Idem que "property", mais pour la 2 ^{nde} collection.
<i>id2</i>	Non	String	Idem que "id", mais pour la 2 ^{nde} collection
<i>indexId</i>	Non	String	Nom de l'objet qui va contenir l'index du bean sur lequel on est en train d'itérer.
<i>sortPictogram</i>	Non	String	Nom de l'image utilisé pour afficher le lien de tri dans l'entête du tableau.
<i>sortLabel</i>	Non	String	"Alt text" utilisé pour l'image décrite ci-dessus.
<i>sortAction</i>	Non	String	Action appelée pour le tri côté serveur : Struts-Layout en propose une (cf. "sort.do"). Pour avoir un tri en Javascript (côté client) : spécifier "client"
<i>sortParam</i>	Non	String	Nom du bean contenant la valeur du paramètre (optionnel) à rajouter à l'url de sortAction.
<i>styleClass</i>	Non	String	La classe CSS à utiliser pour le tableau.
<i>styleClass2</i>	Non	String	La classe CSS à utiliser pour l'alternance de style des lignes.
<i>selectName</i>	Non	String	Nom du bean utilisé pour récupérer les valeurs sélectionnées dans le tableau.
<i>selectProperty</i>	Non	String	Propriété du bean utilisée pour récupérer les valeurs sélectionnées dans le tableau.
<i>selectType</i>	Non	String	Permet de spécifier si la sélection de valeurs dans le tableau est unique ("radio") ou multiple ("checkbox").
<i>width</i>	Non	String	Largeur du tableau (absolue ou relative).
<i>align</i>	Non	String	Alignement du tableau. ("center" par défaut)
<i>offset</i>	Non	Entier	Index du 1 ^{er} objet (de la collection) qui sera affiché dans le tableau.
<i>length</i>	Non	Entier	Index du dernier objet affiché.

 Liste (*partielle*) des attributs de <layout:collection>


Attribut	Obligatoire	Type	Description
<i>action</i>	Non	String	Action Struts à utiliser pour rendre un lien sur le contenu de des éléments de la colonne.
<i>arg0, arg1</i>	Non	String	Paramètres utilisés pour le titre de l'entête de la colonne.
<i>filter</i>	Non	String	Si à false, ne filtre pas les caractères spécifiques HTML (comme les >, <, ...etc)
<i>forward</i>	Non	String	Forward Struts à utiliser pour rendre un lien sur le contenu de des éléments de la colonne.
<i>href</i>	Non	String	Référence à utiliser pour rendre un lien sur le contenu de des éléments de la colonne.
<i>name</i>	Non	String	Nom du bean à utiliser pour rendre le contenu de la colonne. Par défaut, il s'agit de l'objet en cours d'itération.
<i>onclick</i>	Non	String	Permet de mettre un événement Javascript sur l'éventuel lien généré pour le contenu de la colonne.

<i>page</i>	Non	String	Page à appeler pour le lien sur le contenu de des éléments de la colonne.
<i>param</i>	Non	String	Bean dont la valeur sera ajoutée à la fin du lien généré.
<i>paramId</i>	Non	String	Liste (séparée par des virgules) des noms de paramètres à rajouter à la fin du lien généré.
<i>paramName</i>	Non	String	Liste (séparée par des virgules) des beans contenant la valeur des paramètres à rajouter à la fin du lien généré.
<i>paramProperty</i>	Non	String	Liste (séparée par des virgules) des property des beans contenant la valeur des paramètres à rajouter à la fin du lien généré.
<i>property</i>	Non	String	Propriété du bean à utiliser pour rendre le contenu de la colonne.
<i>sortable</i>	Non	String	Si true, alors la colonne sera triable.
<i>styleClass</i>	Non	String	La classe CSS à utiliser pour rendre la colonne.
<i>style</i>	Non	String	Permet de surcharger les styles définis dans la classe styleClass. Doit finir par un ";".
<i>target</i>	Non	String	Frame cible.
<i>title</i>	Non	String	Clé du message à afficher comme entête de la colonne.
<i>type</i>	Non	String	Classe de formatage utilisée pour afficher le contenu. Cf. http://struts.application-servers.com/features/formatter.html
<i>title</i>	Non	String	Clé du message à afficher comme entête de la colonne.
<i>url</i>	Non	String	URL à utiliser pour rendre un lien sur le contenu de des éléments de la colonne.
<i>width</i>	Non	String	Largeur de la colonne (absolue ou relative).

Liste (**partielle**) des attributs de `<layout:collectionItem>`

AJOUTS DE CHAMPS INPUT DANS UN TABLEAU

Il est possible, pour une colonne, de mettre des champs input dans les cellules de sorte à pouvoir modifier un ensemble de bean de la collection affichée. Il faut pour cela que le tableau soit affiché dans un formulaire.

Identifiant	Titre	 Sujet
8	ppp	Struts-Layout
7	AAA	DemoServeurDeNews
6	News 5	DemoServeurDeNews
5	News 4	DemoServeurDeNews
4	News 3	DemoServeurDeNews
3	News 2	DemoServeurDeNews
2	News 1	DemoServeurDeNews
1	Pager	Struts-Layout
0	Sortable	Struts-Layout

Mettre à jour

Il s'agit du tag `<layout:collectionInput>`, qui remplace un simple `<layout:collectionItem>`.

Attribut	Obligatoire	Type	Description
<i>formName</i>	Non	String	Nom du formulaire utilisé pour renvoyer les valeurs des champs input.
<i>formProperty</i>	Non	String	Nom de la propriété indexée du formulaire utilisée pour récupérer les valeurs des champs input.
<i>maxlength</i>	Non	String	Longueur max du champ input.
<i>name</i>	Non	String	Nom du bean à utiliser pour rendre le contenu du champ. Par défaut, il s'agit de l'objet en cours d'itération.
<i>onchange</i>	Non	String	Permet de mettre un événement Javascript sur le champ input.
<i>policy</i>	Non	String	Nom des vérifications à effectuer pour afficher ou pas le champ.
<i>property</i>	Non	String	Propriété du bean à utiliser pour rendre le contenu du champ.
<i>size</i>	Non	String	Taille du champ input.
<i>styleClass</i>	Non	String	La classe CSS à utiliser pour rendre la colonne.
<i>style</i>	Non	String	Permet de surcharger les styles définis dans la classe styleClass. Doit finir par un ";".
<i>sortable</i>	Non	String	Si true, alors la colonne sera triable.
<i>title</i>	Non	String	Clé du message à afficher comme entête de la colonne.
<i>tooltip</i>	Non	String	Tooltip utilisé pour le champ.
<i>width</i>	Non	String	Largeur de la colonne (absolue ou relative).

Liste (**partielle**) des attributs de `<layout:collectionInput>`

AUTRES TAGS UTILES POUR LES COLLECTIONS

Les autres tags utiles pour améliorer le rendu des collections ou leur ajouter des fonctionnalités sont :

- **<collection:style>** : définition de critères de sélection pour donner des styles particuliers à certaines lignes
 - Par exemple : afficher les mails de priorité importante en rouge, et ceux de priorité faible en gris
- **<layout:pager>** : pagination du tableau pour les collections longues
- **<layout:pagerStatus>** : affichage de l'état de la pagination
- **<layout:collectionDetail>** : remplace un `<layout:collectionItem>`, à ceci près que l'information ne sera pas affichée dans une colonne mais dans un tag `<layout:detail>` situé en dehors du tableau (affichage dynamique en fonction de la position du curseur sur le tableau).

Pour ces tags, se référer à la documentation en ligne :

<http://struts.application-servers.com/doc/tags/collection.html>

6.11.4. AUTRES TAGS ET FONCTIONNALITES STRUTS-LAYOUT

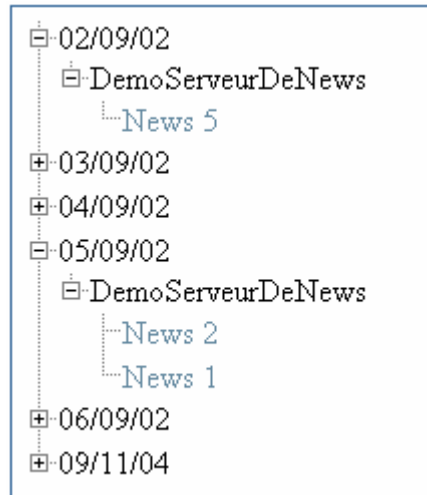
LES AUTRES TAGS

Les tags présentés précédemment sont les plus utilisés de la bibliothèques Struts-Layout. Cependant, beaucoup d'autres fonctionnalités sont disponibles via les tags suivants :

- **<layout:swap>** : offre la possibilité de sélectionner et désélectionner un ensemble de données comme suit :

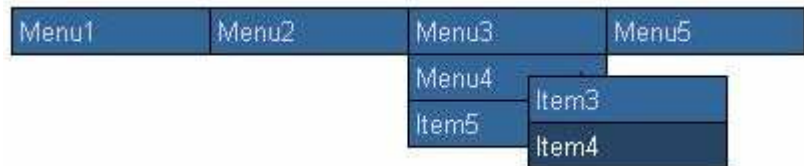
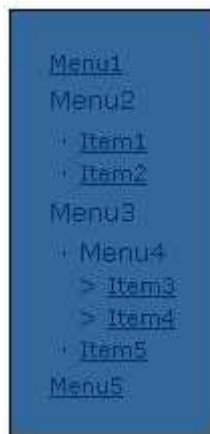
Titre	Sujet	<<	>>	Titre	Sujet
News 5	DemoServeurDeNews			News 1	DemoServeurDeNews
ppp	Struts-Layout			Pager	Struts-Layout
AAA	DemoServeurDeNews			News 3	DemoServeurDeNews
News 4	DemoServeurDeNews				
News 2	DemoServeurDeNews				
Sortable	Struts-Layout				

- **<layout:treeview>** : génération d'arbres statiques (description dans la JSP avec des tags) ou dynamiques (création de la structure côté serveur dans l'action Struts).
→ <http://struts.application-servers.com/doc/tags/treeview.html>



- **<layout:menu>**, **<layout:menuItem>**, **<layout:dynMenu>** : affichage de menus statiques (description dans la JSP avec des tags ou externalisation en XML) ou dynamiques (création côté serveur dans l'action Struts).

→ <http://struts.application-servers.com/doc/tags/menu.html>



- **<layout:crumbs>**, **<layout:crumb>** : permet de générer des éléments de navigation.

→ <http://struts.application-servers.com/doc/tags/crumbs.html>

[crumbs.home](#) > [crumbs.crumbs](#) > [crumbs.example](#)

- **<layout:write>** : comme le **<bean:write>**, avec la possibilité en plus de spécifier une classe pour le formatage du texte (via l'attribut "type"). De plus, ce tag sait se positionner dans un conteneur de Struts-Layout (panel, grid, ...).
- **<layout:message>** : identique au tag **<bean:message>**, excepté qu'il sait se positionner dans un conteneur de Struts-Layout (panel, grid, ...).
- **<layout:link>** : identique au tag **<html:link>**, excepté qu'il sait se positionner dans un conteneur de Struts-Layout (panel, grid, ...). De plus, ce tag génère un Javascript qui vérifie, au moment du clic sur le lien, s'il n'y a pas de formulaire (dans la page) possédant des données non sauvegardées.
- **<layout:policy>** : permet d'afficher ou pas le contenu de ce tag en fonction de vérifications à effectuer (notamment tout ce qui touche aux rôles).
→ Cf. <http://struts.application-servers.com/features/policy.html>

FONCTIONNALITES AVANCEES

En plus de tous les tags présentés précédemment, Struts-Layout offre des fonctionnalités avancées qui sont succinctement présentées ci-dessous.

INTERFACE SKINNABLE

Il est possible de définir plusieurs skins (i.e. plusieurs look) pour une même application basée sur Struts-Layout. Cela se passe au niveau des fichiers de configuration de Struts-Layout. Il y aura autant de fichier de configuration que de skins voulues. Les skins peuvent être changées dynamiquement, de manière programmatique.

→ <http://struts.application-servers.com/features/skin.html>

MODE CREATION DES JSP

Struts-Layout peut être configuré pour générer automatiquement des beans et des collections de beans de sorte à ne pas avoir à créer des actions pour maquetter ses écrans Web. Cependant, tous les composants Struts-Layout ne possèdent pas cette faculté.

→ <http://struts.application-servers.com/features/noerrormode.html>

MODE D’AFFICHAGE

Cette fonctionnalité est la plus avancée de Struts-Layout, mais aussi la plus complexe à prendre en main.

Elle permet de :

- spécifier dans quel mode d’affichage on souhaite positionner la future page qui sera affichée : mode édition, mode création ou mode consultation.
- spécifier pour chaque composant Struts-Layout de cette page quel doit être son comportement en fonction du mode d’affichage (caché ? visible ? éditable ? non éditable ? ...etc.). Il s’agit de l’attribut "**mode**" que l’on retrouve sur beaucoup de tags.

Cela permet de faire des écrans CRUD (Création, Recherche, Mise à jour, Destruction) dans 1 seule page JSP physique.

→ <http://struts.application-servers.com/features/displaymode.html>

7. TRACES

Les traces jouent un rôle très important en phase de débogage ainsi qu'en phase d'exploitation. Il faut donc s'astreindre, lors de la phase de développement à placer des traces dans le code aux endroits clés.

7.1. TRACES APPLICATIVES

7.1.1. UTILISATION DE L'API COMMONS-LOGGING

Les traces applicatives du framework sont basées sur la librairie open source *commons-logging* d'Apache (url : <http://jakarta.apache.org/commons>).

Cette librairie définit ce qu'on appelle des loggers. **Les loggers sont des objets permettant de générer des traces applicatives avec un certain niveau de gravité.** Ces objets implémentent l'interface *org.apache.commons.logging.Log*. La déclaration d'un logger se fait donc de la manière suivante :

```
import org.apache.commons.logging.LogFactory;
import org.apache.commons.logging.Log;
...

public class LoginAction ... {

    private static Log log =
        LogFactory.getLog("com.inetpsa.fwk.security.actions.LoginAction");

}
```

Le paramètre de la méthode statique *getLog()* définit le nom du logger. Il est conseillé de spécifier pour le nom du logger le nom complet de la classe dans laquelle il est défini. **Cela permet ainsi de filtrer les traces par rapport au nom des classes.**

Une fois un logger déclaré, on peut inclure une trace de niveau de gravité TRACE, DEBUG, INFO, WARN, ERROR ou FATAL grâce aux méthodes respectives *trace()*, *debug()*, *info()*, *warn()*, *error()* et *fatal()* disponibles sur l'instance du logger.

```
if (log.isDebugEnabled())
    log.debug(
        "On positionne la locale par défaut: "
        + locale
        + " pour "
        + request.getSession().getId());
```

A des fins d'optimisation, il est conseillé de tester au préalable si le niveau de gravité voulu pour le logger donné est activé suivant les filtres définis dans le fichier de configuration *fwk.xml*. Cela se fait par les méthodes du type *isDebugEnabled()*.

7.1.2. CONFIGURATION

La configuration des traces applicatives se base sur plusieurs fichiers :

- Le fichier *commons-logging.properties* définit la classe d'implémentation de la fabrique utilisée par la librairie *commons-logging* pour instancier les loggers générateurs des traces. Ce fichier doit impérativement être présent dans le classpath de l'application et contenir la ligne suivante :

```
org.apache.commons.logging.LogFactory=com.inetpsa.fwk.config.LogFactoryImpl
```

Il est livré dans les jars du framework et n'a donc pas besoin d'être positionné par le projet.

Le fichier *fwk.xml* et plus spécifiquement l'élément *<logs>*. Cet élément possède les attributs suivants :

Attribut	Description
<i>className</i>	Nom complet de la classe d'implémentation du système de logging. A choisir parmi les valeurs suivantes : <ul style="list-style-type: none"> <i>org.apache.common.logging.impl.SimpleLog</i> (SimpleLog) <i>org.apache.common.logging.impl.Log4jLogger</i> (Log4j) <i>org.apache.common.logging.impl.Jdk14Logger</i> (JDK 1.4)
<i>pattern</i>	Pattern servant à désigner les loggers pour lesquels les traces seront générées.
<i>level</i>	Niveau minimum des traces qui seront générées pour les loggers spécifiés par l'attribut <i>pattern</i> . A choisir parmi les valeurs suivantes (niveau de gravité du plus bas au plus haut) : <ul style="list-style-type: none"> trace debug info warn error fatal

Liste des attributs de l'élément <logs>

On peut affiner le filtre en ajoutant des éléments *<log>*. Cet élément possède les attributs suivants :

Attribut	Description
<i>name</i>	Nom d'un logger pour lesquels les traces seront générées.
<i>startsWith</i>	Pattern servant à désigner les loggers pour lesquels les traces seront générées. Seront activés les loggers dont le nom commence par la valeur de cet attribut.
<i>level</i>	Niveau minimum des traces qui seront générées pour le logger spécifié par l'attribut <i>name</i> ou le(s) logger(s) spécifié(s) par l'attribut <i>startsWith</i> . A choisir parmi les valeurs suivantes (niveau de gravité du plus bas au plu: haut) : <ul style="list-style-type: none"> trace debug info warn error fatal

Liste des attributs de l'élément <log>

Remarque:

Un élément *<log>* doit avoir soit l'attribut *name*, soit l'attribut *startsWith*.

Exemple:

L'exemple ci-dessous spécifie comme implémentation de logging le système *SimpleLog* fourni de base avec la librairie *commons-logging*. Tous les loggers dont le nom commence par *com.inetpsa* pourront générer des traces avec un niveau de gravité égal à *info*, *error* ou *fatal*.

De plus, tous les loggers dont le nom commence par *org.apache.commons* seront activés pour les niveaux *error* et *fatal*. Enfin les traces contenues dans la classe *FWKRequestProcessor* seront générées pour tous les niveaux de gravité au moins égal au niveau *debug*.

```
<logs      className="org.apache.commons.logging.impl.SimpleLog"      pattern="com.inetpsa"
level="info">
    <log startsWith="org.apache.commons" level="error" />
    <log name="com.inetpsa.fwk.struts.action.FWKRequestProcessor" level="debug" />
</logs>
```

7.2. TRACES JDBC

Un outil de mapping O/R tel qu'OJB génère automatiquement les requêtes JDBC. Le développeur n'a donc aucune maîtrise sur le code de ces requêtes. A des fins de débogage, il peut être important de connaître le code de ces requêtes. La librairie P6Spy offre la possibilité de tracer ces requêtes JDBC.

Cette librairie est composée d'un fichier archive *p6spy.jar* qui doit être placé dans le classpath de l'application.

Son utilisation se résume à 3 étapes :

1. Remplacer la classe d'implémentation du driver JDBC par celle de la librairie p6spy. Il s'agit de la classe *com.p6spy.engine.spy.P6SpyDriver*.
2. Mettre à jour le fichier *spy.properties*, fichier de configuration de cette librairie. La principale clé de ce fichier est la clé *realdriver* qui doit définir la classe d'implémentation du driver JDBC.

Exemple pour une base MySQL :

```
realdriver=com.mysql.jdbc.Driver
```

3. Ajouter aux paramètres de la JVM, le paramètre *p6.home* définissant le répertoire contenant le fichier *spy.properties*.

```
java ... -Dp6.home=c:\monAppli\WEB-INF\classes ...
```

Pour la plateforme WebSphere, il existe des spécificités d'installation qui sont expliquées en détail à l'url <http://www.p6spy.com/documentation/install.htm#websphere>.

7.3. ADMINISTRATION DU NIVEAU DE TRACE

7.3.1. PRESENTATION

Le framework propose un module d'administration permettant de modifier en temps réel le niveau des traces (définies dans le fichier `fwk.xml` comme précisé dans le paragraphe 7.1.2 ci-dessus). Cette fonctionnalité concerne toutes les instances de logger.

7.3.2. INCLUSION DU MODULE DANS UNE APPLICATION

Afin de bénéficier de ces fonctionnalités au sein d'une application, il est nécessaire de procéder à quelques manipulations, un ensemble de fichier à intégrer dans le projet web se trouvent dans le répertoire *adminLogs*.

Copier le répertoire *themes* à la racine de l'application web. Ce répertoire contient la feuille de style utilisée par les pages d'administration.

Copier le répertoire *images* à la racine de l'application web. Ce répertoire contient les images utilisées par les pages d'administration.

Copier le fichier *staticJS.jsp* et le répertoire *logs*, présents dans le répertoire *jsps*, à la racine de l'application web.

Ajouter dans le fichier *struts-config.xml* le contenu du fichier *AJOUT_struts-config.xml* présent dans le répertoire *xml*, il y a 2 parties à ajouter : des éléments *<form-bean>* et des éléments *<action>* . Pour chaque élément action on peut rajouter la sécurité nécessaire et correspondante au projet.

Si l'application est construite à base de Tiles (cf. §6.8), il sera alors nécessaire de mettre en place 1 définition supplémentaire pour intégrer les pages JSP du module d'administration au layout général de l'application. Il sera nécessaire de modifier dans le *struts-config.xml* les forward des actions **LogsAdminAffiche** et **LogsAdminUpdateLevels**.

Il faudra également modifier la page JSP pour retirer les lignes:

```
<html>
<head>
  <link rel="stylesheet" href="themes/adminlogs.css" type="text/css" />
</head>
<body>
```

Et :

```
</body>
</html>
```

Il ensuite faudra intégrer dans le layout tiles la feuille de style de l'administration (présente dans le répertoire *themes*) pour la définition.

8. MULTILINGUISME

La solution de multilinguisme du framework est basée sur l'interface *com.inetpsa.fwk.i18n.IServiceResources*. Cette interface offre les services suivants :

- Lecture/MAJ d'un message internationalisé (avec ou sans critères sur des attributs utilisateur),
- Import/Export des données,
- Gestion des locales.

Le framework fournit 2 implémentations de cette interface qui gèrent les messages internationalisés dans un SGBDR :

- Une implémentation de base qui recherche une ressource traduite uniquement sur une clé de traduction et une locale.
- Une implémentation héritant de celle de base et qui ajoute lors de la recherche de ressource traduite un critère portant sur un drapeau : traduit/non traduit afin de ne retourner que les ressources traduites.

8.1. IMPLEMENTATION SGBDR STANDARD

L'implémentation standard basée sur un SGBDR s'appuie sur 2 tables, une table de hiérarchie de locales et une table des traductions (la base doit être paramétrée en UTF8 dans le cas de charset spéciaux), la classe d'implémentation à positionner dans le fichier *fwk.xml* est [com.inetpsa.fwk.i18n.service.db.ServiceResourcesDB](#).

Cette implémentation utilise la couche de persistance OJB et nécessite l'ajout dans le classpath du fichier de mapping *repository-i18n.xml*. Ce fichier contient la définition d'un pool de connexions à la base de données stockant les données de traduction ainsi que les définitions du mapping pour les 2 tables décrites ci-dessous.

Il faut donc créer ces 2 tables dans la base de données puis mettre à jour le fichier de mapping *repository-i18n.xml* pour que le nom des tables et des champs soient corrects.

```
<jdbc-connection-descriptor
    jcd-alias="dbresources"
    default-connection="false"
    platform="MySQL"
    jdbc-level="2.0"
    driver="com.p6spy.engine.spy.P6SpyDriver"
    protocol="jdbc"
    subprotocol="mysql://localhost/WJA"
    dbalias=""
    username=""
    password=""
    eager-release="false"
    batch-mode="false"
    useAutoCommit="1"
    ignoreAutoCommitExceptions="false"
>

<connection-pool maxActive="2" validationQuery="" />
<sequence-manager
    className="org.apache.ojb.broker.util.sequence.SequenceManagerHighLowImpl">
    <attribute attribute-name="grabSize" attribute-value="20"/>
</sequence-manager>
</jdbc-connection-descriptor>
```

```
<class-descriptor class="com.inetpsa.fwk.il8n.service.Locale" table="LOCALES">
  <field-descriptor name="locale" column="LOCALE" jdbc-type="CHAR"
    length="5" primaryKey="true" />
  <field-descriptor name="parentLocale" column="PARENTLOCALE" jdbc-type="CHAR"
    length="5" />
</class-descriptor>
<class-descriptor class="com.inetpsa.fwk.il8n.service.Resource" table="RESOURCES">
  <field-descriptor name="locale" column="LOCALE" jdbc-type="CHAR" length="5"
    primaryKey="true" />
  <field-descriptor name="key" column="REFERENCE" jdbc-type="VARCHAR" length="254"
    primaryKey="true" />
  <field-descriptor name="value" column="VALUE" jdbc-type="VARCHAR" length="254" />
</class-descriptor>
```

Les attributs mis en gras dans le fichier ci-dessus ne doivent en aucun cas être modifiés.

Enfin, ne pas oublier de déclarer le lien vers ce fichier dans le celui de configuration principal d'OJB *repository.xml* (cf. §4.1).

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE descriptor-repository PUBLIC
  "-//Apache Software Foundation//DTD OJB Repository//EN"
  "repository.dtd"
[
  ...
  <!ENTITY il8n SYSTEM "repository_il8n.xml">
]>

<descriptor-repository version="1.0" isolation-level="read-uncommitted">
  ...
  &il8n;
</descriptor-repository>
```

8.1.1. TABLE DE HIERARCHIE

Cette table décrit une hiérarchie entre l'ensemble des langues supportées par l'application.

CHAMP	TYPE	PK	FK	NULL	VALEUR PAR DEFAUT
<i>Locale</i>	VARCHAR(5)	✓		x	''
<i>LocalePere</i>	VARCHAR(5)		✓	✓	Null

Une *Locale* est une chaîne de caractères qui respecte le pattern suivant : `<codeLangue>_<codePays>` où :

- `<codeLangue>` est le code de la langue sur 2 caractères,
- `<codePays>` est le code du pays sur 2 caractères.

Une *Locale* « racine » (*Locale* sans père) est identifiée par un champ *LocalePere* avec une valeur égale à *Null*.

Il peut y avoir plusieurs racines dans la table. Cependant, pour une même langue, il ne doit y avoir qu'une et une seule racine (i.e. 2 racines ne peuvent avoir en commun la même langue).

8.1.2. TABLE DES TRADUCTIONS

Cette table contient l'ensemble des traductions des libellés de l'application.

La taille des champ **Key** et **Value** peut être ajustée suivant les besoins de l'application. Dans ce cas (pour raison de cohérence), n'oubliez pas de répercuter la modification dans le fichier de mapping *repository-i18n.xml* d'OJB.

CHAMP	TYPE	PK	FK	NULL	VALEUR PAR DEFAUT
Locale	VARCHAR(5)	✓		X	''
Key	VARCHAR(254)	✓		X	''
Value	VARCHAR(254)			✓	Null

8.1.3. ALGORITHME DE RECHERCHE D'UNE TRADUCTION D'UN LIBELLE

On dispose des paramètres d'entrée suivants :

- Une locale,
- Une clé.

Dans le cas où la locale ne possède que la langue comme information, on recherche dans la table de hiérarchie la locale racine qui correspond à cette langue.

S'il existe un enregistrement dans la table des traductions dont la clé primaire correspond au couple (locale, clé).

Si la valeur de la traduction (champ Value) est non vide
on retourne cette valeur.

Sinon

Si la locale est présente dans la table HIERARCHIE et qu'elle a un père

Alors

On reprend le même traitement avec la locale père (1)

Sinon

Recherche dans la table HIERARCHIE de la racine correspondant à la langue de la locale

Si la locale trouvée est identique ou n'existe pas

Alors

Si la locale retrouvée à partir de la langue n'existe pas

Alors

On reprend le processus avec la langue par défaut

Sinon

Erreur (pas de traduction possible)

Sinon

La recherche est faite avec la locale qui correspond à la langue.

(1) Ex : si fr_CA n'est pas présente dans la table de traduction mais présente dans la table HIERARCHIE avec fr_FR comme père. La recherche de traduction se fera avec la clé fr_FR.

Si la locale n'est pas présente dans la table des hiérarchies, on reproduit le processus avec comme locale, celle qui est une racine et qui correspond au même code langue. Si cette dernière n'existe pas, on prend la locale par défaut.

8.2. IMPLEMENTATION SGBDR GERANT L'ETAT DE TRADUCTION

L'implémentation SGBDR permettant de gérer un état de traduction hérite de l'implémentation standard et donc s'appuie elle aussi sur les 2 tables : « hiérarchie de locales » et « traductions », la classe d'implémentation à positionner dans le fichier fwk.xml est [com.inetpsa.fwk.i18n.service.db.ServiceResourcesDBOT](#).

De la même façon elle utilise la couche de persistance OJB et nécessite l'ajout dans le classpath d'un fichier de mapping *repository-i18n.xml*.

Il faut donc créer les 2 tables dans la base de données puis mettre à jour le fichier de mapping *repository-i18n.xml* pour que le nom des tables et des champs soient corrects.

```
<jdbc-connection-descriptor
    jcd-alias="dbresources"
    default-connection="false"
    platform="MySQL"
    jdbc-level="2.0"
    driver="com.p6spy.engine.spy.P6SpyDriver"
    protocol="jdbc"
    subprotocol="mysql://localhost/WJA"
    dbalias=""
    username=""
    password=""
    eager-release="false"
    batch-mode="false"
    useAutoCommit="1"
    ignoreAutoCommitExceptions="false"
>

<connection-pool maxActive="2" validationQuery="" />
<sequence-manager
    className="org.apache.ojb.broker.util.sequence.SequenceManagerHighLowImpl">
    <attribute attribute-name="grabSize" attribute-value="20"/>
</sequence-manager>
</jdbc-connection-descriptor>

<class-descriptor class="com.inetpsa.fwk.i18n.service.Locale" table="LOCALES">
    <field-descriptor name="locale" column="LOCALE" jdbc-type="CHAR"
        length="5" primarykey="true" />
    <field-descriptor name="parentLocale" column="PARENTLOCALE" jdbc-type="CHAR"
        length="5"/>
</class-descriptor>

<class-descriptor class="com.inetpsa.fwk.i18n.service.Resource" table="RESOURCES">
    <field-descriptor name="locale" column="LOCALE" jdbc-type="CHAR" length="5"
        primarykey="true"/>
    <field-descriptor name="key" column="REFERENCE" jdbc-type="VARCHAR" length="254"
        primarykey="true" />
    <field-descriptor name="translated" column="TRANSLATED" jdbc-type="INTEGER"
        conversion="org.apache.ojb.broker.accesslayer.conversions.Boolean2IntFieldConversion"/>
    <field-descriptor name="value" column="VALUE" jdbc-type="VARCHAR" length="254"/>
</class-descriptor>
```

Les attributs mis en gras dans le fichier ci-dessus ne doivent en aucun cas être modifiés.

Enfin, ne pas oublier de déclarer le lien vers ce fichier dans celui de configuration principal d'OJB *repository.xml* (cf. §4.1).

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE descriptor-repository PUBLIC
    "-//Apache Software Foundation//DTD OJB Repository//EN"
    "repository.dtd"
[
```

```

...
<!ENTITY i18n SYSTEM "repository_i18n.xml">
]>

<descriptor-repository version="1.0" isolation-level="read-uncommitted">
    ...
    &i18n;
</descriptor-repository>

```

8.2.1. TABLE DE HIERARCHIE

La gestion est identique à celle de l'implémentation standard cf. [Table de hiérarchie](#)

8.2.2. TABLE DES TRADUCTIONS

Cette table contient l'ensemble des traductions des libellés de l'application.

La taille des champ **Key** et **Value** peut être ajustée suivant les besoins de l'application. Dans ce cas (pour raison de cohérence), n'oubliez pas de répercuter la modification dans le fichier de mapping *repository-i18n.xml* d'OJB.

CHAMP	TYPE	PK	FK	NULL	VALEUR PAR DEFAUT
<i>Locale</i>	VARCHAR(5)	✓		X	''
<i>Key</i>	VARCHAR(254)	✓		X	''
<i>Value</i>	VARCHAR(254)			✓	Null
<i>Translated</i>	INTEGER			X	0

8.2.3. ALGORITHME DE RECHERCHE D'UNE TRADUCTION D'UN LIBELLE

La gestion est identique à celle de l'implémentation standard cf. [Algorithme de recherche d'une traduction d'un libellé](#)

8.3. EXTENSION DES IMPLEMENTATIONS SGBDR FOURNIES

A partir de Lego 2.1 l'implémentation sgbdr de base met en place un mécanisme et des points d'entrée permettant de faire une extension afin de gérer des attributs utilisateurs supplémentaires en tant que critère de recherche.

Les modifications à mettre en place pour effectuer une extension complète va s'effectuer en 2 étapes :

- Création d'une nouvelle classe d'implémentation.
- Extension des classes présentes pour la gestion de la partie administration (si celle-ci est utilisée).

Les chapitres suivants vont détailler au travers d'un exemple le code à mettre en place afin d'effectuer une extension (le code correspondant à l'exemple est disponible à cette adresse : <http://gesbob.inetpsa.com/bob1/lego/download/app/exemplei18n.zip>), le critère mis en place au travers de cet exemple est de filtrer les recherches de traduction suivant la marque d'un utilisateur :

Si l'utilisateur n'a pas de marque on recherche la traduction commune à AP et AC.

Sinon

Si la traduction est disponible pour la marque (et la locale) de l'utilisateur

Alors

On retourne la traduction.

Sinon

On recherche la traduction commune à AP et AC.

La table des traductions va alors avoir cette typologie :

CHAMP	TYPE	PK	FK	NULL	VALEUR PAR DEFAUT
Locale	VARCHAR(5)	✓		X	''
Key	VARCHAR(254)	✓		X	''
Value	VARCHAR(254)			✓	Null
Brand	VARCHAR(5)			X	AC_AP

Les valeurs disponibles pour la marque sont :

- AC (marque citroën).
- AP (marque peugeot).
- AC_AP (traduction commune aux 2 marques).

8.3.1. EXTENSION DE COM.INETPSA.FWK...DB.SERVICERESOURCEDB

La mise en place va correspondre à étendre la classe [ServiceResourceDB](#) puis intervenir sur 4 points :

- Créer une classe de ressource étendant [com.inetpsa.fwk.i18n.service.Resource](#) et disposant du nouveau champ : **brand**.
- Créer une implémentation de [IServiceRessourcesDBExtension](#) liée à notre nouvelle implémentation de [IServiceResource](#) (sachant que [IServiceRessourcesDBExtension](#) définit les méthodes à mettre en place pour gérer les particularités de la nouvelle implémentation).
- Créer un constructeur par défaut pour notre implémentation de [IServiceResource](#) qui positionne au travers de la méthode **setExtension** une instance de notre implémentation de [IServiceRessourcesDBExtension](#).
- Créer une implémentation de [ImporterExporter](#) gérant la marque (en étendant [com.inetpsa.fwk.i18n.service.importexport.CSVImporterExporter](#)) et l'utiliser pour gérer l'import/export.

Exemple pour l'extension de [ServiceResourceDB](#) gérant la marque de l'utilisateur (elle contient en classe interne l'implémentation de [IServiceRessourcesDBExtension](#) et l'extension de [Resource](#) associées) :

```
import java.util.HashMap;
import java.util.Map;

import org.apache.commons.logging.Log;
import org.apache.commons.logging.LogFactory;
import org.apache.obj.broker.query.Criteria;

import com.inetpsa.clp.exception.LDAPException;
import com.inetpsa.fwk.i18n.service.Resource;
import com.inetpsa.fwk.i18n.service.db.ServiceResourcesDB;
import com.inetpsa.fwk.security.beans.User;

/**
 * Implémentation base de donnée du service de ressource i18n gérant la marque.
 */
public class ServiceResourcesDBNew extends ServiceResourcesDB {

    /**
     * Constructeur par défaut. Initialisant l'instance d'implémentation de
     * IServiceRessourcesDBExtension utilisée.
     */
    public ServiceResourcesDBNew() {
        setExtension(new BrandServiceRessourcesDBExtension());
    }

    /**
     * Implémentation de IServiceRessourcesDBExtension gérant la marque.
     */
    public class BrandServiceRessourcesDBExtension extends
        DefaultServiceRessourcesDBExtension {
        ...
    }

    // gestion de l'export
    ...

    /**
     * Cette classe définit une ressource traduite au niveau du système de
     * multilinguisme. <br>
     * La classe étend {@link Resource} et ajoute :
     * <ul>
     * <li>la marque associée à la ressource.</li>
     * </ul>
     */
    public static class BrandResource extends Resource {
        /**
         * serialVersionUID courant de la classe.
         */
        private static final long serialVersionUID = 3904957534556336438L;

        /**
         * Constante définissant les 2 marques AC et AP.
         */
        public static final String BRAND_AC_AP = "AC_AP";

        /**
         * Constante définissant la marque AP.
         */
    }
}
```

```
*/
public static final String BRAND_AP = "AP";

/**
 * Constante définissant la marque AC.
 */
public static final String BRAND_AC = "AC";

/**
 * Constructeur par défaut.
 */
public BrandResource() {
    super();
}

/**
 * Constructeur par copie.
 *
 * @param copy l'instance de MarqueResource à partir de laquelle
 *             initialiser cette instance.
 */
public BrandResource(BrandResource copy) {
    super(copy);
    setBrand(copy.getBrand());
}

/**
 * Marque lié à la ressource.
 */
private String brand;

/**
 * Retourne la marque.
 *
 * @return String
 */
public String getBrand() {
    return brand;
}

/**
 * Affecte la marque.
 *
 * @param brand la marque à affecter.
 */
public void setBrand(String brand) {
    this.brand = brand;
}

/**
 * Méthode d'utilitaire pour tester si la marque fournie en paramètre
 * est une marque valide.
 *
 * @param brand la marque à tester
 * @return boolean true si la marque est valide, false sinon
 */
public static boolean isBrandCorrect(String brand) {
    return BRAND_AP.equalsIgnoreCase(brand)
        || BRAND_AC.equalsIgnoreCase(brand)
        || BRAND_AC_AP.equalsIgnoreCase(brand);
}
```

```

    }
}
}

```

On remarque que l'extension effectuée de la classe `Resource` doit comporter un constructeur permettant d'effectuer une initialisation de l'instance courante à partir d'une instance du même type (on appelle le constructeur `super` pour que celui-ci initialise les champs qui sont propres à l'implémentation de la classe mère).

La classe `BrandServiceRessourcesDBExtension` étendant l'implémentation par défaut de `IServiceRessourcesDBExtension` (`DefaultServiceRessourcesDBExtension`) elle doit mettre en place plusieurs méthodes pour gérer les spécificités liées à l'ajout de notre critère sur la marque :

- La méthode `getResourceClass` qui retourne le type de classe de ressource utilisé par l'implémentation i18n courante.

```

/**
 * Implémentation de ServiceRessourcesDBExtension gérant la marque.
 *
 */
public class BrandServiceRessourcesDBExtension extends
    DefaultServiceRessourcesDBExtension {
    /**
     * Constante identifiant la clé correspondant à la marque dans les
     * {@link Map} d'attributs utilisé pour restreindre les recherches.
     */
    public static final String KEY_BRAND = "brand";

    /**
     * Constante identifiant la clé correspondant au flag indiquant qu'il ne
     * faut pas effectuer de recherche sur ac_ap si la traduction n'est pas
     * disponible pour la marque courante.
     */
    public static final String KEY_NO_AC_AP_CHECK = "NOACAP";

    /**
     * Constante définissant le nom du champ marque sur la classe Resource.
     */
    public static final String ATTRIBUTE_BRAND = "brand";

    /**
     * Le logger courant.
     */
    private Log log = LogFactory.getLog(getClass());

    /**
     *
     * Cf. méthode surchargée
     *
     * @see com.inetpsa.fwk.i18n.service.db.ServiceResourcesDB.IServiceRessourcesDBExtension#getResourceClass()
     */
    public Class getResourceClass() {
        return BrandResource.class;
    }
}

```

- La méthode `extractMapAttributesForUser` qui permet de retourner à partir d'un objet `User` une instance de `Map` contenant les données (chaîne de caractère) qui nous intéressent pour enrichir les critères de recherche : la marque dans notre cas, cette `Map` sera réutilisée dans d'autres méthodes (pour créer la requête entre autre).

```
/**
 *
 * Cf. méthode surchargée
 *
 *
 * @see
 * com.inetpsa.fwk.il8n.service.AbstractServiceResources.IServiceRessourcesDBExtension#extractMap
 * AttributesForUser(com.inetpsa.fwk.security.beans.User)
 */
public Map extractMapAttributesForUser(User user) {
    Map attributes = new HashMap();
    attributes.put(KEY_BRAND, getUserMarque(user));
    return attributes;
}
```

- La méthode `updateGetMessageFromBaseCriteria` qui va enrichir le `Criteria` utilisé pour récupérer en base une ressource à partir du(des) attribut(s) qui nous intéresse(nt) présent(s) dans la `Map` passée en paramètre.

```
/**
 *
 * Cf. méthode surchargée
 *
 *
 * @see
 * com.inetpsa.fwk.il8n.service.db.ServiceResourcesDB.IServiceRessourcesDBExtension#updateGetMess
 * ageFromBaseCriteria(org.apache.obj.broker.query.Criteria,
 *      java.util.Map)
 */
public void updateGetMessageFromBaseCriteria(Criteria criteres,
    Map attributes) {
    super.updateGetMessageFromBaseCriteria(criteres, attributes);

    String marque = (String) attributes.get(KEY_BRAND);

    if (!attributes.containsKey(KEY_NO_AC_AP_CHECK)
        && !BrandResource.BRAND_AC_AP.equalsIgnoreCase(marque)) {
        Criteria baseCriteres = criteres.copy(true, true, true);
        criteres.addEqualTo(ATTRIBUTE_BRAND, marque);
        baseCriteres.addEqualTo(ATTRIBUTE_BRAND,
            BrandResource.BRAND_AC_AP);
        criteres.addOrCriteria(baseCriteres);
    } else
        criteres.addEqualTo(ATTRIBUTE_BRAND, marque);
}
```

- La méthode `updateGetMessagesCriteria` qui va enrichir le `Criteria` utilisé pour récupérer en base des ressources à partir du(des) attribut(s) qui nous intéresse(nt) présent(s) dans la `Map` passée en paramètre.

```
/**
 *
 * Cf. méthode surchargée
 *
 * @see
 * com.inetpsa.fwk.i18n.service.db.ServiceResourcesDB.IServiceRessourcesDBExtension#updateGetMessagesCriteria(org.apache.obj.broker.query.Criteria,
 * java.util.Map)
 */
public void updateGetMessagesCriteria(Criteria criteres, Map attributes) {
    super.updateGetMessagesCriteria(criteres, attributes);

    if (attributes != null) {
        String marque = (String) attributes.get(KEY_BRAND);
        if (marque != null)
            criteres.addEqualTo(ATTRIBUTE_BRAND, marque);
    }
}
```

- La méthode `updateKeyWithAttributes` utilisée pour enrichir l'identifiant (unique) positionnée au niveau du cache (pour une locale) afin d'enregistrer une ressource, cet identifiant étant par défaut la clé de traduction, dans notre exemple nous y concaténons la marque.

```
/**
 *
 * Cf. méthode surchargée
 *
 * @see
 * com.inetpsa.fwk.i18n.service.db.ServiceResourcesDB.IServiceRessourcesDBExtension#updateKeyWithAttributes(java.lang.String,
 * java.util.Map)
 */
public String updateKeyWithAttributes(String key, Map attributes) {
    String marque = (String) attributes.get(KEY_BRAND);
    return super.updateKeyWithAttributes(key, attributes) + "<>"
        + marque;
}
```

Afin de pouvoir gérer correctement l'import/export il nous faut étendre la classe `com.inetpsa.fwk.i18n.service.importexport.CSVImporterExporter` avec 2 méthodes à redéfinir :

- `updateResourceWithExtraAttributes` qui met à jour une ressource à importer à partir des attributs (correspondant à notre implémentation) présents sur la ligne récupérée dans le fichier csv.
- `writeResourceToWriter` qui écrit dans le `BufferedWriter` passé en paramètre les attributs qui concernent notre implémentation pour l'instance de `Resource` fournie.

```
public class BrandCSVImporterExporter extends CSVImporterExporter {

    /**
     *
     * Cf. méthode surchargée
     *
     * @see
     com.inetpsa.fwk.il8n.service.importexport.CSVImporterExporter#updateResourceWithExtraAttribute
     s(com.inetpsa.fwk.il8n.service.Resource,
     *      java.lang.String)
     */
    protected void updateResourceWithExtraAttributes(Resource resource,
        String lineExtraAttributes, int lineNumber) throws FwkException {
        if (!ServiceResourcesDBNew.BrandResource
            .isBrandCorrect(lineExtraAttributes)) {
            throw new BadDataException(MessageFormat.format(
                "ligne ({0}) incorrecte, probleme sur la marque",
                new Object[] { new Integer(lineNumber) }));
        } else {
            ((ServiceResourcesDBNew.BrandResource) resource)
                .setBrand(lineExtraAttributes);
        }
    }

    /**
     *
     * Cf. méthode surchargée
     *
     * @see
     com.inetpsa.fwk.il8n.service.importexport.CSVImporterExporter#writeResourceToWriter(com.inetps
     a.fwk.il8n.service.Resource,
     *      java.io.BufferedWriter)
     */
    protected void writeResourceToWriter(Resource resource,
        BufferedWriter writer) throws FwkException {
        super.writeResourceToWriter(resource, writer);
        try {
            writer.write(SEPARATOR
                + ((ServiceResourcesDBNew.BrandResource) resource)
                    .getBrand());
        } catch (IOException e) {
            throw new FwkException(e);
        }
    }
}
```

Puis ajouter sur notre extension de [ServiceResourceDB](#) la gestion de l'import/export en redéfinissant [getImporterExporterClassForType](#) :

```

public class ServiceResourcesDBNew extends ServiceResourcesDB {

    /**
     * liste des identifiants de type d'implémentation gérés.
     */
    private static final String[] TYPES = { ImportExportTypeFactory.TYPE_CSV };

    /**
     * liste des classes d'implémentation d'import/export.
     */
    private static final String[] IMPL_CLASSES = {
        "com.inetpsa.fwk.impl.BrandCSVImporterExporter" };

    /**
     * Message d'erreur lorsqu'aucune classe d'implémentation n'est disponible
     * pour le type fourni.
     */
    private static final String ERROR_NO_IMPLEMENTATION_CLASS = "Aucune classe
d'implémentation n'est disponible pour le type {0}";

    /**
     * Map contenant les implémentations d'ImportExport à utiliser.
     */
    private static final Map IMPORTEXPORT_CLASSES = new HashMap();

    static {
        if (TYPES.length == IMPL_CLASSES.length)
            for (int i = 0; i < TYPES.length; i++) {
                IMPORTEXPORT_CLASSES.put(TYPES[i], IMPL_CLASSES[i]);
            }
    }

    ...

    public Class getImporterExporterClassForType(String type)
        throws FwkException {

        String className = (String) IMPORTEXPORT_CLASSES.get(type);

        if (className == null)
            throw new ImporterExporterException(MessageFormat.format(
                ERROR_NO_IMPLEMENTATION_CLASS, new Object[] { type }));
        else {
            try {
                return Class.forName(className);
            } catch (ClassNotFoundException e) {
                throw new ImporterExporterException(MessageFormat.format(
                    ERROR_NO_IMPLEMENTATION_CLASS, new Object[] { type },
                    e);
            }
        }
    }

    ...
}
    
```

8.3.2. EXTENSION DE L'ADMINISTRATION

Voici une copie d'écran de l'interface d'administration telle qu'elle va être pour la gestion de la marque dans notre exemple :

— Filtre sur libelle / locale —

Locale à traduire: Locale de base:

Filtre sur libelle: spécifique à la marque:

Libelle	Traduction	Traduction Initiale	Supprimer
admin.i18n.title.basetraduction	Traduction Initiale	Traduction Initiale	<input type="checkbox"/>
admin.i18n.title.delete	Supprimer	Supprimer	<input type="checkbox"/>
admin.i18n.title.libelle	Libelle	Libelle	<input type="checkbox"/>
admin.i18n.title.locale	Locale	Locale	<input type="checkbox"/>
admin.i18n.title.parentLocale	Locale Parente	Locale Parente	<input type="checkbox"/>
admin.i18n.title.traduction	Traduction	Traduction	<input type="checkbox"/>
admin.i18n.title.translated	Traduit	Traduit	<input type="checkbox"/>
appli.message.acueil	Accueil FR AC_AP	Accueil FR AC_AP	<input type="checkbox"/>
appli.message.menu	menu AC_AP fr_FR	menu AC_AP fr_FR	<input type="checkbox"/>
appli.title	title	title	<input type="checkbox"/>

1 2 >>

— Creation d'un nouveau libelle —

Libelle: Traduction: Locales: ☒ fr_FR (locale courante de traduction) ☐ en_US ☐ es_ES ☐ fr_BE ☐

spécifique à la marque:

On voit une combo box qui permet de choisir la marque des traductions courantes affichées dans la liste et en bas de l'écran pour la création d'une nouvelle traduction on voit que l'on a le choix de la marque pour laquelle on souhaite créer la traduction.

L'extension de l'administration va se faire à plusieurs niveaux :

- La classe `com.inetpsa.fwk.i18n.admin.actions.I18nUpdateFilterAction` à étendre avec la méthode `reload` à redéfinir, cette méthode indique à partir du formulaire passé en paramètre si la collection de traduction affichée couramment doit être rechargée (changement de filtre,...) :

```
protected boolean reload(I18nActionForm form) {
    boolean reload = super.reload(form);

    com.inetpsa.fwk.examples.forms.I18nActionForm currentFrom =
    (com.inetpsa.fwk.examples.forms.I18nActionForm) form;

    /**
     * On teste la valeur de marque courante par rapport à l'ancienne.
     */
    reload = reload
        || ((currentFrom.getBrand() != null && !currentFrom.getBrand()
            .equals(currentFrom.getOldBrand())));

    return reload;
}
```

Il y a un appel à la méthode `reload` de la classe parente afin de laisser l'implémentation standard indiquer si le rechargement doit avoir lieu suivant ses critères (filtre de recherche, locale courante,...) et ensuite on utilise l'opérateur booléen **OR** pour ajouter le test qui nous concerne (la marque liée aux traductions à afficher a t'elle changée).

- La classe `com.inetpsa.fwk.i18n.admin.actions.I18nAction` à étendre avec la méthode `getSearchAttributes` à redéfinir, cette méthode retourne une `Map` contenant des attributs de critères de recherche.

Cette instance de `Map` sera utilisée pour l'appel à la méthode `getMessages` de l'implémentation de `IServiceResource` qui retourne les ressources à lister sur la page d'administration (les clés utilisées aux niveau de la `Map` sont celles connues dans l'implémentation de `IServiceRessourcesDBExtension`) :

```
protected Map getSearchAttributes(I18nActionForm form) {
    com.inetpsa.fwk.examples.forms.I18nActionForm currentForm =
(com.inetpsa.fwk.examples.forms.I18nActionForm) form;
    Map attributes = super.getSearchAttributes(form);
    Boolean translated = null;

    String brand = currentForm.getBrand();

    /**
     * On s'assure que la valeur de marque qui va être stocké dans les
     * attributs est valide.
     */

    if (!ServiceResourcesDBNew.BrandResource.isBrandCorrect(brand))
        brand = ServiceResourcesDBNew.BrandResource.BRAND_AC_AP;

    attributes
        .put(
            ServiceResourcesDBNew.BrandServiceRessourcesDBExtension.KEY_BRAND,
            brand);

    return attributes;
}
```

La méthode `getTranslateAttributes` à redéfinir, cette méthode retourne une `Map` contenant des attributs de critères de recherche.

Les critères contenus dans cette `Map` seront utilisés pour récupérer les traduction présentes dans la 3^{ème} colonne du tableau au travers de l'appel à la méthode `getMessages` de l'implémentation de `IServiceResource`, celle-ci retournant les traductions à afficher sur la page d'administration (les clés utilisées aux niveau de la `Map` sont celles connues dans l'implémentation de `IServiceRessourcesDBExtension`) :

```
/**
 * Retourne une
 * {@link Map} contenant les attributs de recherche de texte traduit à afficher dans le
 * layout:collection.
 *
 * @param form le formulaire duquel extraire les attributs
 * @return Map
 */
protected Map getTranslateAttributes(I18nActionForm form) {
    Map attributes = super.getSearchAttributes(form);

    com.inetpsa.fwk.examples.forms.I18nActionForm currentForm =
(com.inetpsa.fwk.examples.forms.I18nActionForm) form;
    String brand = currentForm.getBrand();

    /**
     * On s'assure que la valeur de marque qui va être stocké dans les
     * attributs est valide.
     */

    if (!ServiceResourcesDBNew.BrandResource.isBrandCorrect(brand))
        brand = ServiceResourcesDBNew.BrandResource.BRAND_AC_AP;

    attributes
        .put(
            ServiceResourcesDBNew.BrandServiceRessourcesDBExtension.KEY_BRAND,
            brand);

    attributes
        .put(
            ServiceResourcesDBNew.BrandServiceRessourcesDBExtension.KEY_NO_AC_AP_CHECK,
            "");

    return attributes;
}
```

La classe [com.inetpsa.fwk.i18n.admin.actions.I18nUpdateCreateDeleteAction](#) à étendre avec la méthode [newResourceForUpdate](#) à redéfinir, cette méthode retourne une instance de [Resource](#) à mettre à jour, elle est initialisée à partir du formulaire passé en paramètre et de l'index de la ressource dans le formulaire :

```
protected Resource newResourceForUpdate(I18nActionForm form, int index)
    throws FwkException {
    Resource resource = null;

    com.inetpsa.fwk.examples.forms.I18nActionForm currentForm =
(com.inetpsa.fwk.examples.forms.I18nActionForm) form;
    String brand = currentForm.getBrand();

    if (ServiceResourcesDBNew.BrandResource.isBrandCorrect(brand)) {
        resource = super.newResourceForUpdate(form, index);
        if (resource != null)
            ((ServiceResourcesDBNew.BrandResource) resource)
                .setBrand(brand);
    }

    return resource;
}
```

La méthode **newResourceForCreate** à redéfinir, cette méthode retourne une instance de **Resource** à créer, elle est initialisée à partir du formulaire passé en paramètre et de l'index de la ressource dans le formulaire :

```
protected Resource newResourceForCreate(I18nActionForm form, int index)
    throws FwkException {
    Resource resource = super.newResourceForCreate(form, index);

    if (resource != null) {
        com.inetpsa.fwk.examples.forms.I18nActionForm currentForm =
(com.inetpsa.fwk.examples.forms.I18nActionForm) form;

        String brand = currentForm.getCreateBrand();

        /**
         * On s'assure que la valeur de marque qui va être stocké dans les
         * attributs est valide, s'il s'agit de ALL on positionne la marque
         * courante à AC et la méthode duplicateResourceForParams permettra
         * de créer l'instance pour AP.
         */
        if (BRAND_ALL.equalsIgnoreCase(brand))
            brand = ServiceResourcesDBNew.BrandResource.BRAND_AC;
        else if (!ServiceResourcesDBNew.BrandResource.isBrandCorrect(brand))
            brand = ServiceResourcesDBNew.BrandResource.BRAND_AC_AP;

        ((ServiceResourcesDBNew.BrandResource) resource).setBrand(brand);
    }

    return resource;
}
```

La méthode **newResourceForDelete** à redéfinir, cette méthode retourne une instance de **Resource** à supprimer, elle est initialisée à partir du formulaire passé en paramètre et de l'index de la ressource dans le formulaire :

```
protected Resource newResourceForDelete(I18nActionForm form, int index)
    throws FwkException {
    Resource resource = null;

    com.inetpsa.fwk.examples.forms.I18nActionForm currentForm =
(com.inetpsa.fwk.examples.forms.I18nActionForm) form;
    String brand = currentForm.getBrand();

    if (ServiceResourcesDBNew.BrandResource.isBrandCorrect(brand)) {
        resource = super.newResourceForDelete(form, index);
        if (resource != null)
            ((ServiceResourcesDBNew.BrandResource) resource)
                .setBrand(brand);
    }

    return resource;
}
```

La méthode **duplicateResourceForParams** à redéfinir, cette méthode est utilisée lorsque l'utilisateur a demandé à créer en une fois une ressource sous plusieurs valeurs de critères, exemple : il a choisi plusieurs locales de création : fr_FR **et** en_US **et** fr_BE, ... ou dans notre cas il a choisi de créer la ressource sous la marque AC **et** sous la marque AP. La méthode duplique l'instance de **Resource** fournie en paramètre à partir d'informations présentes dans le formulaire et liées à l'implémentation courante :

```
protected Collection duplicateResourceForParams(Resource resource,
        I18nActionForm form) {
    Collection result = null;

    com.inetpsa.fwk.examples.forms.I18nActionForm currentForm =
    (com.inetpsa.fwk.examples.forms.I18nActionForm) form;

    if (BRAND_ALL.equalsIgnoreCase(currentForm.getCreateBrand())) {
        result = new ArrayList();

        ServiceResourcesDBNew.BrandResource newResource =
        (ServiceResourcesDBNew.BrandResource) resource
            .clone();
        newResource.setBrand(ServiceResourcesDBNew.BrandResource.BRAND_AP);
        result.add(newResource);

        Collection resources = super.duplicateResourceForParams(resource,
            form);
        Iterator liste = resources.iterator();

        while (liste.hasNext()) {
            ServiceResourcesDBNew.BrandResource currentResource =
            (ServiceResourcesDBNew.BrandResource) liste
                .next();

            result.add(currentResource);

            /**
             * la marque AC est déjà dans la liste on ajoute la marque AP.
             */
            newResource = (ServiceResourcesDBNew.BrandResource) currentResource
                .clone();
            newResource
                .setBrand(ServiceResourcesDBNew.BrandResource.BRAND_AP);
            result.add(newResource);
        }
    } else
        result = super.duplicateResourceForParams(resource, form);

    return result;
}
```

8.4. CONFIGURATION

La configuration du service de multilinguisme s'effectue dans le fichier *fwk.xml* grâce à l'élément *<i18n>*. Cet élément se compose de 2 éléments fils à renseigner :

L'élément `<class>` correspondant à la classe d'implémentation du service. Cette classe doit implémenter l'interface `com.inetpsa.fwk.i18n.service.IServiceResources` ou étendre la classe `com.inetpsa.fwk.i18n.service.AbstractServiceResources`.

L'élément `<default_locale>` désigne la locale qui sera prise par défaut.

Exemple:

```
<i18n>
  <class>com.inetpsa.fwk.i18n.service.db.ServiceResourcesDB</class>
  <default_locale>fr_FR</default_locale>
</i18n>
```

8.5. INTEGRATION AU FRAMEWORK STRUTS

Le framework Struts propose comme implémentation par défaut pour le multilinguisme l'implémentation i18n classique, c'est-à-dire celle à base de fichiers propriétés. Son ouverture permet cependant d'y greffer sa propre implémentation.

C'est le but de la configuration suivante dans le fichier `struts-config.xml` au niveau de l'élément `<message-resources>`.

```
<message-resources factory="com.inetpsa.fwk.struts.messages.FWKMessageResourcesFactory"
  parameter="" null="true" />
```

8.6. MODULE D'ADMINISTRATION

Le module d'administration du multilinguisme possède 3 fonctionnalités présentes sur 3 pages JSP distinctes. Le tableau ci-dessous présente ces 3 fonctionnalités avec pour chacune d'entre elles la page JSP et l'action Struts principale associées.

Fonctionnalité	Description	Page JSP	Action
Traductions	Permet de créer/modifier/supprimer des libellés internationalisables et de les traduire dans les locales existantes.	i18n.jsp	I18nAdmin
Locales	Permet de créer/modifier/supprimer des locales.	i18n_locales.jsp	I18nAdminLocale
Import / Export	Permet d'importer et d'exporter des données.	i18nimportexport.jsp	I18nImportExportAction

8.6.1. INCLUSION DU MODULE DANS UNE APPLICATION

Afin de bénéficier de ces fonctionnalités au sein d'une application, il est nécessaire de procéder à quelques manipulations, un ensemble de fichier à intégrer dans le projet web se trouvent dans le répertoire `adminI18n`.

Copier le répertoire `themes` (présent dans le répertoire `commun`) à la racine de l'application web. Ce répertoire contient la feuille de style utilisée par les pages d'administration.

Copier le répertoire *images* (présent dans le répertoire *commun*) à la racine de l'application web. Ce répertoire contient les images utilisées par les pages d'administration.

Copier le fichier *staticJS.jsp* et le répertoire *i18n*, présents dans le répertoire *jsps* (correspondant à l'implémentation choisie : *Implementation attribut traduit* ou *Implementation standard*), à la racine de l'application web.

Ajouter dans le fichier *struts-config.xml* le contenu du fichier *AJOUT_struts-config.xml* présent dans le répertoire *xml* (correspondant à l'implémentation choisie : *Implementation attribut traduit* ou *Implementation standard*), il y a 2 parties à ajouter : des éléments *<form-bean>* et des éléments *<action>* . Pour chaque élément action on peut rajouter la sécurité nécessaire et correspondante au projet.

Ajouter dans le fichier *validation.xml* le contenu du fichier *AJOUT_validation.xml* présent dans le répertoire *xml* (correspondant à l'implémentation choisie : *Implementation attribut traduit* ou *Implementation standard*).

On peut ensuite démarrer l'application, aller avec un navigateur sur l'action **I18nImportExportAction.do**, puis déclencher l'import du fichier *export-init.csv* présent dans le répertoire *donnees init* (correspondant à l'implémentation choisie : *Implementation attribut traduit* ou *Implementation standard*) en positionnant comme charset : **Cp1252**.

Ceci permet d'initialiser les tables *i18n* avec la locale française et les quelques traductions (messages d'erreurs) liés à l'administration *i18n*.

Si l'application est construite à base de Tiles (cf. §6.8), il sera alors nécessaire de mettre en place 3 définitions supplémentaires pour intégrer les pages JSP du module d'administration au layout général de l'application. Il sera nécessaire de modifier dans le *struts-config.xml* les forward des actions **I18nAdmin**, **I18nAdminLocale** et **I18nImportExportAction** et il faudra également modifier les 3 pages JSP pour retirer les lignes:

```
<html>
<head>
  <link rel="stylesheet" href="themes/i18n.css" type="text/css" />
</head>
<body>
```

Et :

```
</body>
</html>
```

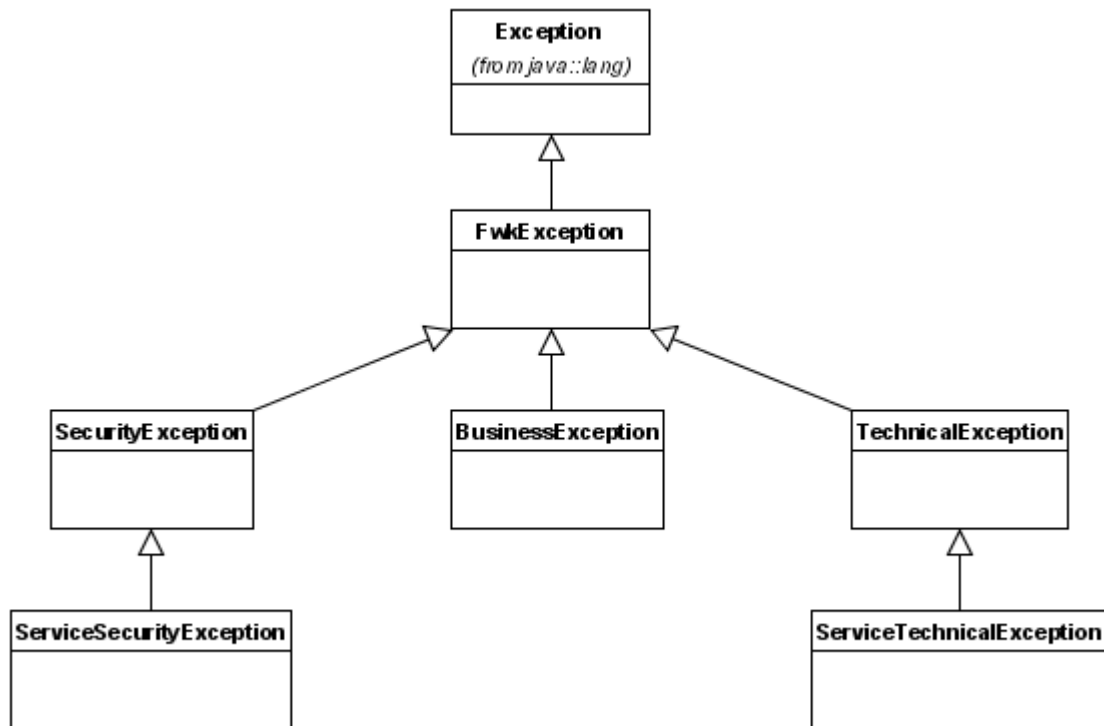
Il ensuite faudra intégrer dans le layout tiles la feuille de style de l'administration *i18n* (présente dans le répertoire *themes*) pour les 3 définitions.

Afin de pouvoir bénéficier de la pagination sur les pages d'administration il sera aussi nécessaire, si ce n'est déjà fait, de positionner dans le fichier *struts-config.xml* l'action de sort de Struts-Layout :

```
<action path="/sort" type="fr.improve.struts.taglib.layout.sort.SortAction"
scope="request" validate="false" />
```


9. EXCEPTIONS

Les exceptions propres au framework sont définies dans le package *com.inetpsa.fwk.exception*.



L'exception de base du framework est la classe *FwkException* qui hérite de la classe *java.lang.Exception*. Cette classe permet de stocker une exception de plus bas niveau, un message d'erreur ainsi qu'un identifiant de message.

Les exceptions du framework se déclinent en 3 grandes familles :

Les exceptions de type *SecurityException* qui sont déclenchées lors d'un problème lié à la sécurité applicative (informations d'authentification erronées, l'utilisateur n'a pas les droits pour exécuter telle action, ...). Le sous-type *ServiceSecurityException* permet de préciser l'origine de l'exception à savoir une classe de la couche Service.

Les exceptions de type *TechnicalException* qui sont déclenchées suite à des problèmes techniques inopinés (problème de base de données, bugs, ...). Le sous-type *ServiceTechnicalException* permet de préciser l'origine de l'exception à savoir une classe de la couche Service.

Les exceptions de type *BusinessException* qui sont déclenchées suite au non-respect de règles de gestion dans les services fonctionnels. Les exceptions fonctionnelles définies au sein d'une application devront hériter de cette exception.

10. SECURITE APPLICATIVE

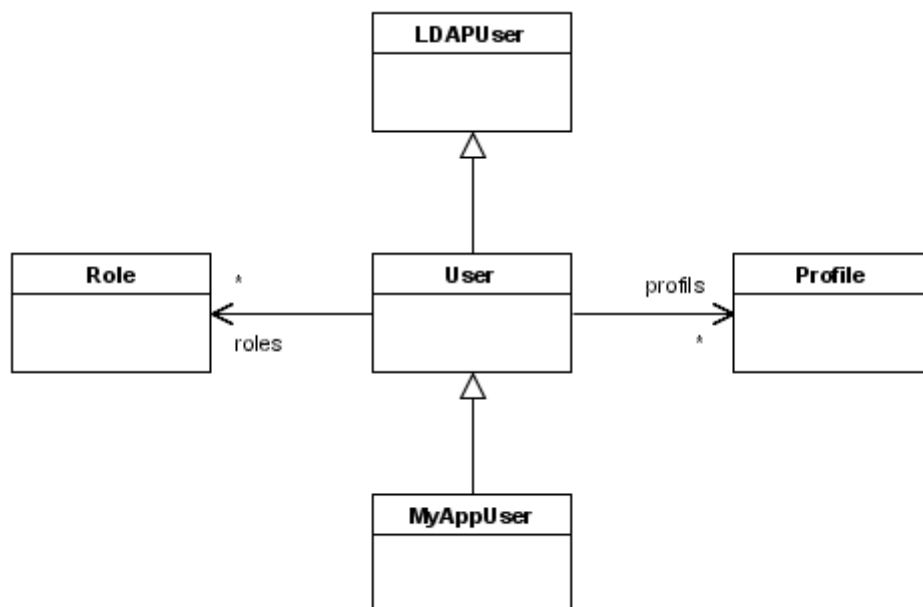
La solution de sécurité applicative proposée par le framework s'articule autour des 3 concepts suivants : l'utilisateur, le profil et le rôle.

Ce chapitre propose une description détaillée de chacun de ces 3 objets ainsi que la manière de les utiliser et les configurer pour la mise en œuvre d'une gestion des droits au sein d'une application.

10.1. COMPOSANTS METIER

10.1.1. UTILISATEUR

Le framework met à disposition la classe `com.inetpsa.fwk.security.beans.User` pour consigner les informations relatives à l'utilisateur de l'application. Cette classe hérite de la classe `com.inetpsa.clp.LDAPUser` faisant partie du composant LDAP. Il est conseillé de définir une sous-classe `MyAppUser` propre à l'application. Cela peut permettre de définir des méthodes qui seront utilisées pour déterminer les profils de l'utilisateur.



La classe d'implémentation de l'objet User doit être définie dans le fichier `fwk.xml` grâce à l'élément `<user>` :

```
<user className="com.inetpsa.fwk.security.beans.User"></user>
```

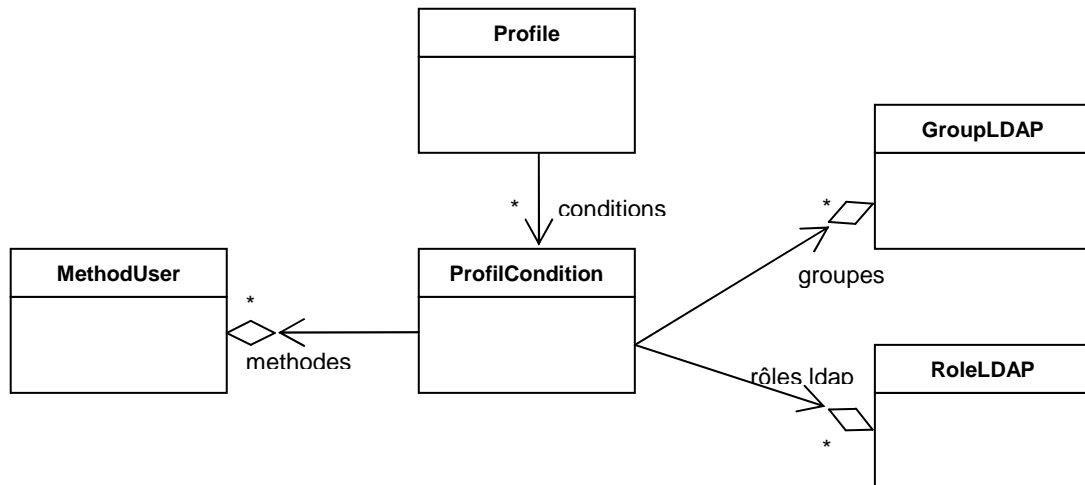
Un objet User est automatiquement placé dans la session utilisateur lors de la 1^{ère} requête à l'application sous la clé `Constants.SESSION_USER`. Celui-ci reste vide tant que l'utilisateur ne s'est pas authentifié. Il est possible de le vérifier en appelant la méthode `isAuthenticated()` qui retourne `false` si ce dernier n'est pas authentifié et `true` dans le cas contraire.

10.1.2. PROFIL

Un profil est déterminé par un ensemble de groupes LDAP et par un ensemble de méthodes définies sur la classe d'implémentation de l'objet utilisateur.

Pour déclarer un profil au sein de l'application, il faut créer une classe héritant de la classe du framework `com.inetpsa.fwk.security.beans.Profile`. Il faut ensuite ajouter un élément `<profil>` dans le fichier `fwk.xml`.

Le framework met à disposition un profil par défaut implémenté par la classe `com.inetpsa.fwk.security.beans.DefaultProfile`. **Ce profil par défaut est affecté automatiquement à tout utilisateur authentifié.** Ce profil n'a pas besoin d'être déclaré dans le fichier `fwk.xml`; il l'est implicitement. Son identifiant est «DefaultProfile» (valeur de la constante `com.inetpsa.fwk.security.beans.DefaultProfile.NAME`).



Un utilisateur se voit attribuer un profil si pour une condition du profil:

(toutes les méthodes du User déclarées dans la condition renvoient true)

ET

(l'utilisateur appartient à chacun des groupes LDAP déclarés dans la condition)

ET

(l'utilisateur possède chacun des rôles LDAP déclarés dans la condition)

Exemple :

D'après la configuration ci-dessous, les utilisateurs appartenant au groupe LDAP `ROLE.WJA.ADM` se verront attribuer le profil `redacteur` tandis que ceux du groupe LDAP `ROLE.FWL.ACCE.SORTANTS.PSA` se verront attribuer le profil `lecteur`.

```

<profiles>
  <profile name="redacteur" description="Profil Redacteur"
    className="com.inetpsa.wja.security.profil.ProfilExpert">
    <condition>
      <ldapGroups>
        <ldapGroup>ROLE.WJA.ADM</ldapGroup>
      </ldapGroups>
    </condition>
  </profile>
  <profile name="lecteur" description="Profil Lecteur"
    className="com.inetpsa.wja.security.profil.ProfilLecteur">
  </profile>
</profiles>
  
```

```
<condition>
  <ldapGroups>
    <ldapGroup>ROLE.FWL.ACCESSION.SORTANTS.PSA</ldapGroup>
  </ldapGroups>
</condition>
</profile>
</profiles>
```

Concernant la gestion de noms de groupes LDAP avec partie variable \$DOMAIN : on peut positionner dans la condition au niveau du nom du groupe ldap le joker * , ex :

```
<condition>
  <ldapGroups>
    <ldapGroup>ROLE.WJA.ADM.*</ldapGroup>
  </ldapGroups>
</condition>
```

Cette condition sera validée si l'utilisateur appartient à un groupe ldap commençant par «ROLE.WJA.ADM.» .

A noter que s'il y a nécessité d'exploiter la partie variable des groupes auxquels appartient l'utilisateur et qui correspondent à la règle, le framework fournit une collection(*java.util.Collection*) d'objets de type *com.inetpsa.clp.LDAPGroup* au niveau du constructeur de la classe *Profile* correspondante. Il est alors possible, dans ce constructeur, d'itérer sur la liste d'objets de type *com.inetpsa.clp.LDAPGroup* et d'extraire la partie variable du nom des groupes pour la stocker dans une « liste de \$DOMAIN » au niveau de la classe *Profile*.

Concernant la gestion de noms de rôles LDAP : on peut positionner dans la condition au niveau du nom du rôle ldap le joker * , ex :

```
<condition>
  <ldapRoles>
    <ldapRole>ROLE.WJA.*</ldapRole>
  </ldapRoles>
</condition>
```

Concernant l'utilisation de *MethodUser* au niveau d'une condition il faut procéder comme ceci dans le cas d'une méthode sans paramètre :

```
<condition>
  <userMethods>
    <userMethod name="isAdmin"/>
  </userMethods>
</condition>
```

Cette condition sera valide si la méthode suivante présente sur l'objet *User* retourne true :

```
public boolean isAdmin(String[] params){
...
}
```

Comme ceci dans le cas d'une méthode avec paramètre :

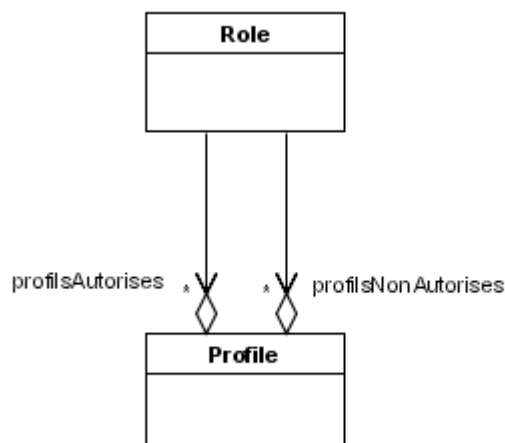
```
<condition>
  <userMethods>
    <userMethod name="isFromCountry">
      <params>
        <param>FR</param>
      </params>
    </userMethod>
  </userMethods>
</condition>
```

Cette condition sera valide si la méthode suivante présente sur l'objet *User* retourne true :

```
public boolean isFromCountry(String[] params){
if (params.length>0)
{
  String country = params[0] ;
  //country contiendra dans notre cas : FR
  ...
}else {
  ...
}
}
```

10.1.3. ROLE LEGO

Un rôle est défini par un ensemble de profils autorisés et un ensemble de profils non autorisés. Un utilisateur se voit donc attribué un rôle s'il possède tous les profils autorisés de ce rôle et s'il ne possède aucun profil non autorisé de ce rôle.



Le rôle est purement logique : il ne se matérialise pas par une classe mais seulement par l'élément `<role>` dans le fichier de configuration *fwk.xml*.

Exemple :

D'après la configuration ci-dessous, les utilisateurs ayant un profil rédacteur et un profil *lecteur* se verront attribuer le rôle *redacteurLecteur* tandis que ceux ayant uniquement le profil *redacteur* se verront attribuer le rôle *administrateur*.

```
<roles>
  <role name="redacteurLecteur">
    <authorized_profile>
      <profile>redacteur</profile>
      <profile>lecteur</profile>
    </authorized_profile>
    <non_authorized_profile/>
  </role>
  <role name="administrateur">
    <authorized_profile>
      <profile>redacteur</profile>
    </authorized_profile>
    <non_authorized_profile>
      <profile>lecteur</profile>
    </non_authorized_profile>
  </role>
</roles>
```

10.2. AUTHENTIFICATION

La phase d'authentification est la phase au cours de laquelle le serveur demande au client de décliner son identité, à savoir un identifiant et un mot de passe.

10.2.1. SECURITYMANAGER

La classe *com.inetpsa.fwk.security.service.SecurityManager* possède une méthode d'authentification utilisant le composant LDAP :

```
public static void authenticate(User user, String login, String password)
                                throws FwkException
```

Le serveur compare alors les données saisies par l'utilisateur à la banque de données stockées dans l'annuaire LDAP. Si le compte fourni par l'utilisateur ne correspond à aucun utilisateur connu de l'annuaire d'entreprise (identifiant et/ou mot de passe erronés), alors une *SecurityException* est levée. Dans le cas où l'authentification est réussie, l'utilisateur se voit affecter ses profils et les rôles correspondants.

10.2.2. ACTION DE LOGIN

Dans le cas où la phase d'authentification est implémentée par un formulaire web, le framework met à disposition une classe Action Struts prête à l'emploi pour la validation du formulaire d'authentification : *com.inetpsa.fwk.security.actions.LoginAction*.

Pour authentifier l'utilisateur, cette action fait appel au service *SecurityManager* vu précédemment. Une fois l'utilisateur authentifié, elle met à jour la locale courante :

- A partir de la valeur associée à l'attribut `com.inetpsa.fwk.constants.Constants.SESSION_PRECHOOSSED_LOCALE` si celui-ci est présent en session (permet au projet de proposer ou imposer un choix de locale à l'utilisateur).
- Ou si l'attribut n'est pas présent, la locale qui peut être construite à partir des propriétés de l'utilisateur (provenant de l'annuaire LDAP : langue et pays).
- Ou si les données utilisateur ne sont pas suffisantes pour construire la locale alors celle par défaut définie dans le `fwk.xml` est utilisée.

Pour l'utiliser au sein de l'application, il suffit d'ajouter dans les *action-mappings* du fichier *struts-config.xml* l'élément `<action>` suivant :

```
<action path="/authentifier" name="loginForm" validate="true" input="relogin"
        className="com.inetpsa.fwk.struts.action.FWKLoginActionMapping"
        scope="request" type="com.inetpsa.fwk.security.actions.LoginAction">
  <set-property property="loginInput" value="userid"/>
  <set-property property="passwordInput" value="password"/>
  <forward name="loginfinalsuccess" path="securedPage"></forward>
  <forward name="relogin" path="/login.do"></forward>
</action>
```

Les éléments listés ci-dessous sont à renseigner obligatoirement :

Élément de configuration	Valeur
Property loginInput	Nom du champ de saisie du formulaire d'authentification correspondant à l'identifiant de l'utilisateur.
Property passwordInput	Nom du champ de saisie du formulaire d'authentification correspondant au mot de passe.
Forward loginfinalsuccess	Chemin de la ressource web à invoquer en cas d'authentification réussie.
Forward relogin	Chemin de la ressource web à invoquer en cas d'authentification échouée.

La surcharge par le framework du RequestProcessor Struts permet d'offrir une fonctionnalité supplémentaire liée au login :

à savoir que si un utilisateur non authentifié essayait d'accéder à une action sécurisée alors le contrôleur struts le redirige vers l'action de login en ayant au préalable stocké les informations sur la requête initiale, puis une fois que la phase d'authentification s'est correctement déroulée l'utilisateur est redirigé vers l'action correspondant à sa demande initiale si la sécurité qui lui est appliquée le permet.

Les informations stockés par le RequestProcessor pour permettre le renvoi vers l'action émise en premier lieu sont propres à struts, à savoir le formulaire lié à l'action. Si des paramètres sont récupérés via un appel à `getParameter(...)` sur le *request* dans l'action redirigée alors ceux-ci ne seront pas disponibles après la phase d'authentification, de même pour toutes les informations du *request* liées à la requête http.

10.2.3. ACTION DE LOGOUT

Une action récurrente dans une application web nécessitant une phase d'authentification est l'action de déconnexion. Pour cela, le framework met à disposition une classe Action Struts prête à l'emploi : `com.inetpsa.fwk.security.actions.LogoutAction`.

Son action est l'invalidation de la session courante. Il en résulte qu'après avoir appelé cette action, l'utilisateur doit à nouveau s'authentifier pour exécuter toute action sécurisée.

Pour l'utiliser au sein de l'application, il suffit d'ajouter dans les *action-mappings* du fichier *struts-config.xml* l'élément `<action>` suivant :

```
<action path="/logout" type="com.inetpsa.fwk.security.actions.LogoutAction">
    <forward name="logoutsuccess" path="/accueil.do" redirect="true" ></forward>
</action>
```

Les éléments listés ci-dessous sont à renseigner obligatoirement :

Elément de configuration	Valeur
Forward logoutsuccess	Chemin de la ressource web à invoquer après la déconnexion. Positionner le paramètre <i>redirect</i> à <i>true</i> .

10.3. BASIC AUTHENTICATION

10.3.1. PRESENTATION GENERALE

Le framework LEGO propose deux fonctionnalités autour de la notion de Basic Authentication:

- **Authentification automatique** : le header de basic authentication, positionné par une application tierce (portail par exemple), est utilisé pour authentifier automatiquement l'utilisateur.
- **Authentification par code retour** : utilisation du code retour 401 pour demander l'authentification d'un client accédant à une ressource protégée.

10.3.2. AUTHENTIFICATION AUTOMATIQUE

MISE EN PLACE

La configuration se positionne au niveau du contrôleur Struts.

```
<controller ...>
<set-property property="basicAuthRealm" value="/gdr01"/>
<set-property property="timeoutWithBasicAuth " value=" timeoutbasic"/>
<set-property property=" basicAuthValidateForward " value=" invalidbasicauth"/>
</controller>
```

basicAuthRealm : permet de positionner le domaine sur lequel on souhaite gérer l'authentification.

timeoutWithBasicAuth : Si la propriété est présente, la valeur correspond à un forward global permettant d'afficher une page (différente de la page de timeout standard) à l'utilisateur lui spécifiant que la session est tombée en timeout.

basicAuthValidateForward : permet de statuer sur le comportement à adopter lorsqu'un client demande une ressource protégée sans être authentifié et se trouve dans un des deux cas suivant :

- le client est tombé en timeout de session et n'a jamais cliqué sur un lien pointant sur l'action Struts décrite ci-dessous.

- Le client a invalidé sa session par une action de logout.

Si la propriété n'est pas présente, le client sera ré-authentifié de manière transparente puis la ressource protégée fournie.

Si la propriété est présente, la valeur de celle-ci correspond à un forward global permettant d'afficher une page à l'utilisateur, elle peut par exemple être utilisée pour lui indiquer :

- Qu'il faut qu'il ferme son navigateur et en ouvre un nouveau pour effectuer une nouvelle phase d'authentification.
- Ou alors la page peut indiquer que s'il souhaite se ré-authentifier automatiquement à partir de ses informations (header BasicAuthentication), il peut cliquer sur un lien pointant sur une action STRUTS particulière (fournie par le framework).

L'action doit être paramétrée au niveau du fichier struts-config.xml de la manière suivante :

```
<action path="chemin spécifique à l'application"
        type="com.inetpsa.fwk.security.actions.BasicAuthTokenAction">
  <forward name="success" path="chemin spécifique à l'application ex: /accueilauth.do"
    redirect="true"/>
</action>
```

Une autre propriété est disponible :

```
<controller ...>
...
<set-property property="basicAuthActivate" value=" true"/>
...
</controller>
```

basicAuthActivate : indique que la gestion du header de BasicAuth est activée (*true*) ou inactivée (*false*), la propriété est optionnelle la valeur par défaut si elle n'est pas présente est *true*.

10.3.3. AUTHENTIFICATION PAR CODE RETOUR

MISE EN PLACE

La configuration se positionne au niveau du contrôleur Struts.

```
<controller ...>
<set-property property="basicAuthRealm" value="/gdr01"/>
<set-property property="timeoutWithBasicAuth" value=" timeoutbasic"/>
<set-property property=" basicAuthValidateForward " value=" invalidbasicauth"/>
<set-property property=" basicAuthLogin " value=" true"/>
<set-property property=" basicAuthRetry " value=" 2"/>
</controller>
```

Voir ci-dessus pour la description des propriétés **basicAuthRealm**, **timeoutWithBasicAuth** et **basicAuthValidateForward**.

BasicAuthLogin : indique au contrôleur Struts de gérer la demande d'authentification à un client via un code retour 401 et non au travers d'un formulaire HTML (cf. §10.4.2)

BasicAuthRetry : spécifie le nombre d'authentification demandée par le contrôleur (via l'envoi de code 401). Un code 403 sera envoyé à l'issue de ces demande pour spécifier un problème de sécurité.

10.4. HABILITATIONS

Une habilitation est le fait d'accorder à un utilisateur authentifié une permission qui peut être le droit d'exécuter un service métier ou encore le droit de visualiser telle page de l'application.

Il est possible de définir les habilitations à différents niveaux de l'architecture applicative :

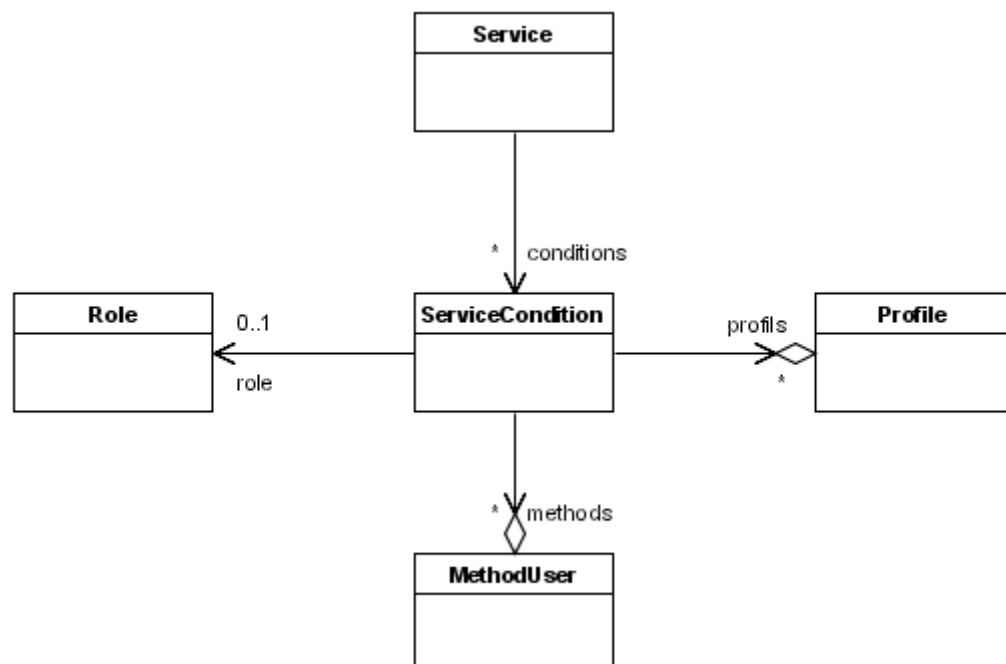
- Service fonctionnel,
- Action Struts,
- Page JSP.

Le niveau le plus important à sécuriser est celui du service fonctionnel. Les autres niveaux sont moins importants car ils n'accèdent pas directement aux données métiers sensibles au contraire des services fonctionnels.

10.4.1. SECURISER L'EXECUTION D'UN SERVICE FONCTIONNEL

La configuration des habilitations pour un service fonctionnel se fait au niveau du fichier *fwk.xml*. Pour chaque service défini dans ce fichier, il est possible de définir un ensemble de conditions à remplir pour pouvoir exécuter ce service. **Si aucune de ces conditions n'est remplie, la fabrique de services renverra une exception de type *ServiceSecurityException*.**

La constitution d'une condition est décrite par le diagramme de classes ci-dessous :



Une condition est remplie si :

(
 (un rôle est présent **ET** l'utilisateur possède ce rôle) **OU**
 (au moins un profil est présent **ET** l'utilisateur possède tous les profils) **OU**
 (aucun rôle et profil n'est présent)
)
ET
 (
 (une ou n méthodes du User sont présentes **ET** toutes ces méthodes renvoient true) **OU**
 (aucune méthode n'est présente)
)

Exemples :

```
<service name="rechercherInterventions" ... >
  <condition>
    <profiles>
      <profile>DefaultProfil</profile>
    </profiles>
  </condition>
</service>
<service name="creerIntervention" ... >
  <condition>
    <role>administrateur</role>
  </condition>
</service>
```

10.4.2. SECURISER L'EXECUTION D'UNE ACTION STRUTS

La configuration des habilitations pour une action Struts se fait au niveau d'un élément `<action>` du fichier `struts-config.xml`. Cet élément permet de déclarer 2 attributs `profils` et `roles` dont la valeur contiendra respectivement une liste de profils et une liste de rôles (chaque item de la liste doit être séparé par une virgule).

Le contrôleur Struts autorisera un utilisateur à exécuter une Action si :

(Les attributs `profils` et `roles` ne sont pas présents ou sont vides)
OU
 (
 (l'utilisateur est authentifié)
 ET
 (
 (l'attribut `roles` est présent et non vide) **ET**
 (l'utilisateur possède au moins un des rôles présents dans la liste de l'attribut `roles`)
)
 OU
 (
 (l'attribut `profils` est présent et non vide) **ET**
 (l'utilisateur possède au moins un des profils présents dans la liste de l'attribut `profils`)
)
)

)
)

Exemples :

```
<action path="/searchInterventions" ... profils="redacteur,lecteur">
    ...
</action>

ou

<action path="/searchInterventions" ... roles="allUsers">
    ...
</action>

<action path="/afficherCreateIntForm" ... profils="redacteur">
    ...
</action>
```

Dans le cas où l'utilisateur n'est pas authentifié et tente d'exécuter une Action sécurisée (i.e. l'attribut *roles* ou l'attribut *profils* est présent et non vide), il est automatiquement redirigé sur la page d'authentification de l'application. Cette page d'authentification est à spécifier par la propriété *loginAction* de l'élément *<controller>* du fichier *struts-config.xml* qui doit contenir comme valeur un forward global.

```
<controller className="com.inetpsa.fwk.struts.action.FWKControllerConfig"
    processorClass="com.inetpsa.fwk.struts.action.FWKTilesRequestProcessor"
    debug="0" contentType="text/html" inputForward="true">
    <set-property property="loginAction" value="login"/>
</controller>
```

Une fois l'utilisateur authentifié, le contrôleur exécutera l'action précédemment demandée par l'utilisateur si ce dernier possède les droits d'exécution, sinon il renverra une erreur HTTP 403.

10.4.3. SECURISER L'AFFICHAGE D'UNE PAGE JSP

Afin d'empêcher un utilisateur d'appeler directement une page JSP depuis un navigateur web sans passer par le contrôleur struts, il suffit d'ajouter en tête de la page JSP le tag *<fwk:jspsecurity>*.

Pour cela, il est conseillé de définir une page *security.jsp* dont le source serait le suivant :

```
<%@ taglib uri="http://fwk.org" prefix="fwk" %>
<fwk:jspsecurity/>
```

Puis, pour toute page JSP dont on veut autoriser l'accès uniquement via un mapping struts, on ajoute en toute 1^{ère} position de la page la ligne suivante :

```
<%@ include file="security.jsp"%>
```

Dans le cas où l'utilisateur accède directement à la page, ce tag a renvoie une erreur HTTP 403 au navigateur web.

10.5. GESTIONNAIRE DE SECURITE

A partir de lego 2.1 il est possible de remplacer par une nouvelle implémentation le gestionnaire de sécurité par défaut qui effectue l'authentification à partir de l'annuaire ldap d'entreprise et l'affectation des profils à partir des conditions définies dans le fichier fwk.xml.

Ceci se fait en 2 étapes :

10.5.1. CREER UNE IMPLEMENTATION DE COM.INETPSA.FWK.SECURITY.SERVICE.ISECURITYINITIALIZER

Il faut pour cela implémenter 4 méthodes :

- `boolean authenticate(String login, String password)` : qui effectue l'authentification et retourne true si le couple identifiant/mot de passe est correct, false sinon.
- `void initUser(User user, String login, String password)` : qui initialise l'utilisateur avec les données nécessaires, au minimum l'identifiant.
- `Collection getUserProfiles(User user)` : qui retourne une `Collection` d'instances de classes héritant de `com.inetpsa.fwk.security.beans.Profile` qui correspondent aux profils de cet utilisateur.
- `Collection getUserRoles(User user)` : qui retourne une `Collection` de `String` qui correspondent aux rôles de cet utilisateur.

Il est possible d'étendre l'implémentation par défaut : `com.inetpsa.fwk.security.service.SecurityManager.DefaultSecurityInitializer` pour bénéficier de tout ou partie de ses traitements.

10.5.2. POSITIONNER L'IMPLEMENTATION A UTILISER :

Ceci se fait au niveau du fichier fwk.xml dans le tag security il y a un tag securityinitializer à positionner :

```
<security>
...
<securityinitializer>
  <className>com.inetpsa.fwk.test.SecurityInitializer</className>
  <onlyMultipleProfiles>false</onlyMultipleProfiles>
</securityinitializer>
</security>
```

Il faut indiquer la classe d'implémentation de gestionnaire de sécurité. On dispose, au travers du tag `onlyMultipleProfiles`, de la possibilité d'indiquer au module d'authentification de lego de ne pas laisser accéder à l'application un utilisateur dont le couple identifiant/mot de passe est correct mais qui ne s'est vu attribué aucun profil de l'application (il ne dispose que du `DefaultProfile` dans le cas de l'implémentation par défaut de `ISecurityInitializer`).

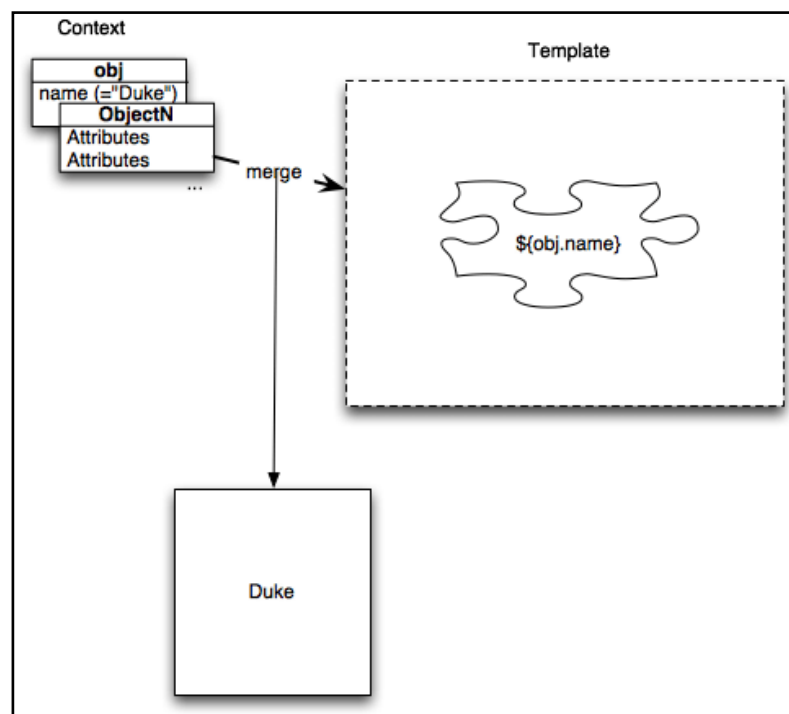
11. TEMPLATING VELOCITY

Velocity (<http://jakarta.apache.org/velocity>), projet de la communauté Apache Jakarta, est un moteur de substitution basé sur Java.

Velocity permet de faire référence à des méthodes définies dans du code Java dans des templates.

Velocity peut être utilisé pour générer tout type de document pouvant être généré à partir d'un template, par exemple, des états, des pages HTML, un mail, ...

11.1. FONCTIONNEMENT



Velocity utilise un contexte, dans lequel il y a tous les objets Java à utiliser dans le template.

Vous trouverez plus d'information dans le guide (en français) Velocity, à l'adresse suivante :

http://jakarta.apache.org/velocity/user-guide_fr.html

11.2. UTILISATION AVEC LEGO

L'utilisation de Velocity avec lego se fait au moyen des classes :

- **VelocityEngineFactory**

Cette classe permet de récupérer une instance d'objet *VelocityEngine*, avec des propriétés pouvant être surchargées par les utilisateurs.

- **AbstractVelocityEngineUtils**

Cette classe est une classe utilitaire permettant de lancer les substitutions sur les templates.

Cette classe possède des méthodes pouvant récupérer le document dans un objet de type **String**, ou de l'écrire directement dans un objet de type **Writer**.

1. instanciation de **VelocityEngineFactory**

```
VelocityEngineFactory factory = new VelocityEngineFactory();
```

2. Positionnement de diverses propriétés (si nécessaire)

```
Properties properties = new Properties();

properties.put(KEY_RESOURCE_LOADER, VALUE_RESOURCE_LOADER);
properties.put(KEY_RESOURCE_LOADER_CLASSPATH_CLASS, VALUE_RESOURCE_LOADER_CLASSPATH_CLASS);
...
properties.put(KEY_OUTPUT_ENCODING, "UTF-8");

factory.setVelocityProperties(properties);
```

3. Création du **VelocityEngine**

```
VelocityEngine engine = factory.createVelocityEngine();
```

On peut aussi surcharger des propriétés au niveau de l'objet *engine*, en utilisant la méthode `setProperty` :

```
engine.setProperty(KEY_OUTPUT_ENCODING, "ISO-8859-1");
```

4. Création d'un contexte pour mettre les objets nécessaires au template

```
Map context = new HashMap();

context.put("properties", properties);
context.put("headers", headers);
context.put("exporterBean", bean);
```

5. Utilisation ensuite de l'**AbstractVelocityEngineUtils**

```
String document = AbstractVelocityEngineUtils.mergeTemplate(engine, "myTemplate.vm",
encoding, context)
```

Pour plus de détails, se reporter à la documentation Javadoc de ces classes.

11.3. VELOCITY TEMPLATE LANGUAGE (VTL)

11.3.1. COMMENTAIRES

Les commentaires permettent d'inclure du texte descriptif qui sera ignoré par le moteur de substitution de Velocity.

Les commentaires permettent de se rappeler et d'expliquer à d'autres ce que font les instructions.

Il y a 3 types de commentaire :

1. **Commentaire de ligne** : commence par **##** et qui se termine à la fin de la ligne.

Ex. :

```
## Ceci est un commentaire d'une seule ligne
```

2. **Commentaire multi-lignes** : commence par **##** et se termine par **##**.

Ex. :

```
##
Ceci est un
Commentaire sur
plusieurs lignes
##
```

3. **Commentaire VTL** : Permet d'inscrire des informations comme l'auteur, la version du document. Ce commentaire commence par ******* et se termine par **##**.

Ex. :

```
***
Ceci est un commentaire VTL
@author
@version 1.5
##
```

11.3.2. REFERENCES

Il y a 3 types de références en VTL : les variables, les propriétés et les méthodes.

VARIABLES

Les variables doivent respectées l'expression régulière suivante :

\$ [!] [{] [a..z, A..Z] [a..z, A..Z, 0..9, -, _] [}]

La notation abrégée pour une variable consiste en un caractère "\$" initial suivi d'un *Identificateur* VTL. Un identificateur VTL doit commencer par une lettre (a .. z ou A .. Z). Le reste des caractères est limité aux types suivants:

- lettre (a .. z, A .. Z)
- chiffre (0 .. 9)
- tiret ("-")
- trait de soulignement ("_")

Voici quelques exemples de références valides en VTL:

```
$foo
$lastName
$last-name
$last_name
$lastName1
```

PROPRIETES

Les propriétés ont un format qui les distingue des variables.

La notation abrégée consiste en un caractère **\$** initial suivi d'un identifiant VTL, suivi d'un point (".") et d'un autre identifiant VTL. Quelques exemples de références valides de propriétés en VTL:

```
$customer.lastName
$purchase.Total
```

La propriété **\$customer.lastName** peut avoir deux significations :

- Elle peut signifier: "Regarde dans la table de hachage identifiée par **customer** et renvoie la valeur associée à la clé **lastName**.
- Elle peut aussi faire référence à une méthode (cf. ci-dessous), c'est-à-dire que **\$customer.Address** pourrait être une manière abrégée d'écrire **\$customer.getLastName()**.

METHODES

Une méthode est définie dans le code Java, elle peut effectuer un calcul ou retourner une valeur.

Les méthodes sont des références qui consistent en un caractère "\$", suivi d'un identifiant VTL, suivi d'un *corps de méthode*. Un corps de méthode VTL consiste en un identifiant VTL suivi du caractère parenthèse ouvrante ("("), éventuellement suivi d'une liste de paramètres, suivi du caractère parenthèse fermante (")").

```
$customer.getAddress()
$purchase.getTotal()
$page.setTitle( "My Home Page" )
$car.setColors( ["White", "Red", "Blue"] )
```

TYPES DE NOTATIONS

1. Notations abrégées

Le pattern des références en notation abrégée est :

\$ [a..z, A..Z][a..z, A..Z, 0..9, -, _]

La notation abrégée est la notation la plus utilisée, comme pour identifier les propriétés. Elle sert aux variables, aux propriétés et aux méthodes. Pour les méthodes, cela permet de référencer les propriétés au lieu des méthodes

Ex. :

<code>\$customer.getAddress()</code>	<code>\$customer.address</code>
<code>\$purchase.getTotal()</code>	<code>\$purchase.total</code>

2. Notations formelles

Le pattern des références en notation formelle est :

\$ [{][a..z, A..Z][a..z, A..Z, 0..9, -, _][}

Le nom de la référence est entourée des caractères { et }.

Dans certain, on peut être amené dans un template d'avoir une variable utilisé comme préfixe de substitution pour un mot, donc pour éviter une mauvaise interprétation de la référence on utilise cette notation formelle.

Ex. :

Dans le contexte Velocity, on a seulement l'objet référencé par **\$class.name**.

La classe **\$class.nameFactory** permet de récupérer une instance d'objet **\$class.name**.

Les sorties voulues sont :

La classe **PersonFactory** permet de récupérer une instance d'objet **Person**.

Ou

La classe *VehiculeFactory* permet de récupérer une instance d'objet *Vehicule*.

\$class.nameFactory n'est pas dans le contexte, donc pour éviter toute ambiguïté au niveau des références, l'écriture dans le template doit se faire de la manière suivante :

La classe ***#{class.name}***Factory permet de récupérer une instance d'objet *\$class.name*.

3. Notations silencieuses

Le pattern des références en notation silencieuse est :

\$ [!] [{ [a..z, A..Z] [a..z, A..Z, 0..9, -, _] [}]

Lorsque Velocity rencontre une référence non définie, son comportement par défaut est de laisser tel quel la référence.

Exemple :

```
<input type="text" name="email" value="$email"/>
```

Au premier chargement, la variable référencée par *\$email* n'a pas de valeur, donc si on veut l'initialiser à vide, il faut utiliser la notation silencieuse.

La notation silencieuse permet de contourner le comportement normal de Velocity, c'est-à-dire qu'au lieu d'utiliser *\$email* dans le template, il faut utiliser *#!email*.

Il est donc préférable d'utiliser cette notation :

```
<input type="text" name="email" value="#!email"/>
```

La notation formelle et la notation silencieuse peuvent être utilisées ensemble, comme illustré ci-dessous.

```
<input type="text" name="email" value="#{!email}"/>
```

11.3.3. ELEMENTS DU LANGAGE VTL

ASSIGNATION

L'assignation se fait au moyen de la directive **#set**.

La syntaxe est la suivante :

#set(VARIABLE = VALUE)

VALUE peut être du type suivant :

Une variable	<code>#set(\$name = \$newName)</code>
Une chaîne littérale	<code>#set(\$name = "monica")</code>
Une propriété	<code>#set(\$name = \$user.name)</code>
Une méthode	<code>#set(\$name = \$user.getName())</code>
Un nombre littéral	<code>#set(\$age = 25)</code>
Une liste	<code>#set(\$media.types = ["Phone", \$fax, "Mail"])</code>
Une expression arithmétique	<code>#set(\$result = \$value *5 -1)</code>

CONDITION

La condition se fait au moyen des directives **if / else /elseif**.

La syntaxe est la suivante :

```
#if( [condition] ) [output] [elseif( [condition] ) [output] ]* [else [output] ] #end
```

On peut mettre autant de directives **elseif** que l'on veut (0..n).

Condition : expression booléenne

- Egal à : **#if(\$foo == \$bar)**
- Supérieur à : **#if(\$foo > 42)**
- Inférieur à : **#if(\$foo < 42)**
- Supérieur ou égal à : **#if(\$foo >= 42)**
- Inférieur ou égal à : **#if(\$foo <= 42)**
- Complément (Not) : **#if(!\$foo)**
- ET logique : **#if(\$foo && \$bar)**
- OU logique : **#if(\$foo || \$bar)**

Output : expression VTL.

Ex. :

```

#if( $foo < 10 )
  <strong>Go North</strong>
#elseif( $foo == 10 )
  <strong>Go East</strong>
#elseif( $bar == 6 )
  <strong>Go South</strong>
#else
  <strong>Go West</strong>
#end

```

BOUCLE

Directive permettant de faire des itérations.

La syntaxe est la suivante :

```
#foreach( [condition] ) [output] #end
```

Velocity fournit un moyen simple de récupération la valeur du compteur de boucle : **\$velocityCount**.

Ex. :

```

<table>
#foreach( $customer in $customerList )
  <tr><td>$velocityCount</td><td>$customer.Name</td></tr>
#end
</table>

```

Par défaut, le compteur **\$velocityCount**, démarre à 1. Il peut être paramétré pour commencer à 0, dans le fichier *velocity.properties* :

```

# Default name of the loop counter variable
reference.directive.foreach.counter.name = velocityCount
# Default starting value of the loop counter variable
reference.directive.foreach.counter.initial.value = 1

```

INCLUDE

Permet d'inclure un fichier ou des fichiers qui ne seront pas interprétés.

Syntaxe :

#include (*filename* [, *filename1*]*)

Ex. :

```
#include( "licence.txt" )  
#include( "one.gif", "two.txt", "three.htm" )  
#include( "greetings.txt", $seasonalstock )
```

PARSE

Permet d'importer un fichier contenant du VTL et de l'interpréter.

Syntaxe :

#parse (*filename*)

filename : peut être une chaîne représentant un nom de fichier ou une variable.

Ex. :

```
#parse( "parsefoo.vm" )  
#parse( $templateName )
```

STOP

Élément permettant d'arrêter l'exécution du moteur de substitution.

Syntaxe :

#stop

Cette directive peut être utile pour le débogage.

11.4. EXEMPLES

- Exemple de génération d'un état HTML affichant une liste de produits et leur prix

```
<html>  
  <h1>Liste des produits</h1>  
  <table>  
    <tr>  
      <td>N°</td>  
      <td>Produit</td>  
      <td>Prix (€)</td>  
    </tr>
```

```
#foreach( $product in $productList )
    <tr>
        <td>$product.id</td>
        <td>$product.name</td>
        <td>$product.price</td>
    </tr>
#end
</table>
</html>
```

- **\$product** est un bean Java contenant les getters *getId()*, *getName()* et *getPrice()*. Le fait d'appeler *\$product.id* est équivalent à *\$product.getId()*.
- **\$productList** est collection d'objets Product.

Rem. : Pour les libellés, on peut utiliser un objet qui va contenir la chaîne à utiliser (ex. pour l'internationalisation).

Le résultat ressemblera, dans un navigateur, à :

Liste des produits		
N° Produit		Prix (€)
1	Product 1	7,73
2	Product 2	7,25
3	Product 3	14,53
4	Product 4	21,57
5	Product 5	6,03
6	Product 6	42,78
7	Product 7	54,86
8	Product 8	9,15
9	Product 9	8,14
10	Product 10	29,38

2. Exemple de mailing avec message au format texte

Bonjour **\$person.civility \$person.lastName \$person.firstName**,

Vous avez récemment acheté un véhicule chez nous : **\$person.vehicle.model**, de couleur **\${person.vehicle.color}**.

Pour tout renseignement, vous pouvez contacter la concession :

\$concession.name
\$concession.address
\$concession.zipCode \$concession.city

Tél. : **\$concession.phone**

Cordialement,

Le service Relation Client.

Ce qui donne :

Bonjour Monsieur DURAND Pierre,

Vous avez récemment acheté un véhicule chez nous : PEUGEOT 206 HDI 110, de couleur Jaune.

Pour tout renseignement, vous pouvez contacter la concession :

Peugeot Poissy
20 rue de la gare
78230 Poissy

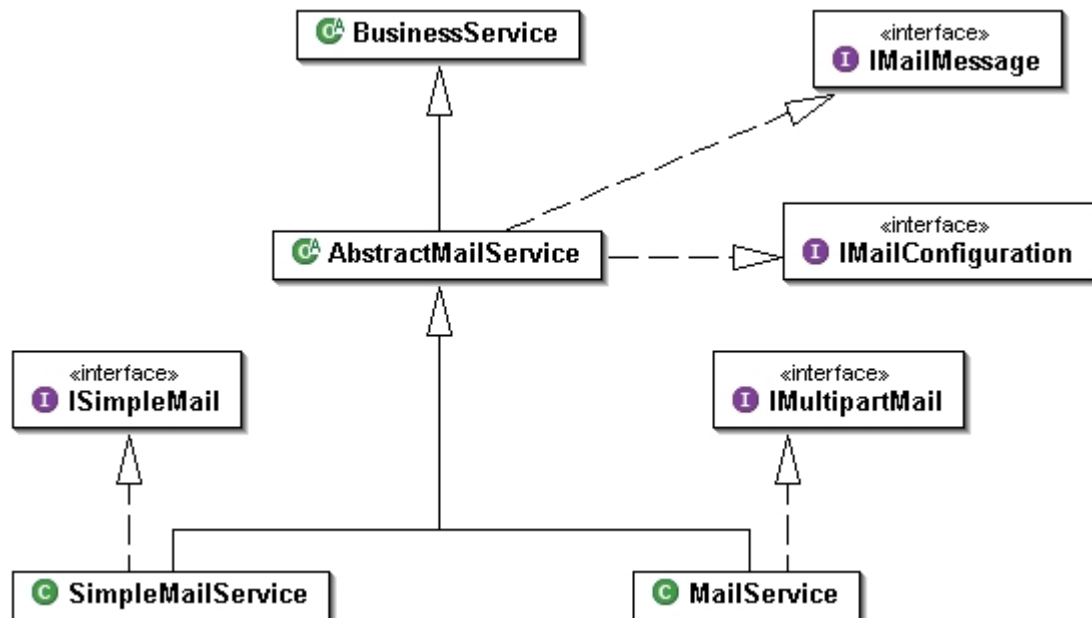
Tél. : 0139564523

Cordialement,

Le service Relation Client.

12. MAIL

12.1. DIAGRAMME DE CLASSE



12.2. MAIL SIMPLE

La classe d'implémentation de ce service est :

com.inetpsa.fwk.service.mail.SimpleMailService

12.2.1. PRESENTATION

Ce service permet de n'envoyer que des mails au format texte sans attachement.

Le message texte peut être généré à partir d'un template Velocity avec l'une des méthodes suivantes :

```

public void setMessageTemplate(String templateName, Map model) throws MailServiceException;

public void setMessageTemplate(String templateName, Map model, String path) throws MailServiceException;

public void setMessageTemplate(String templateName, String encoding, String outputEncoding, Map model) throws MailServiceException;

public void setMessageTemplate(String templateName, String encoding, String outputEncoding, Map model, String path) throws MailServiceException;
  
```

Les paramètres suivant peuvent être mis en paramètre dans le fichier *fwk.xml* :

- **MAIL_HOST** : Nom du serveur de mail (SMTP) ou adresse IP ;

- **MAIL_PORT** : Port SMTP du serveur de mail (par défaut 25) ;
- **MAIL_AUTH_USERNAME** : identifiant si l'accès SMTP nécessite une authentification ;
- **MAIL_AUTH_PASSWORD** : Mot de passe si l'accès SMTP nécessite une authentification ;
- **MAIL_SENDER** : L'adresse mail de l'expéditeur ;
- **MAIL_REPLYTO** : Les adresses mail de réponse ;
- **MAIL_RECIPIENTS** : La liste des adresses mail des destinataires ;
- **MAIL_RECIPIENTS_CC** : La liste des adresses mail des destinataires en copie ;
- **MAIL_RECIPIENTS_BCC** : La liste des adresses mail des destinataires en copie cachée ;
- **MAIL_SUBJECT** : L'objet du mail.

Cela permet d'initialiser le service de mail avec ces paramètres, et d'éviter de mettre dans le code ces informations.

12.2.2. VALIDATION

A l'exécution du service, une validation est faite sur les paramètres d'entrée, une liste contenant les codes d'erreur est retournée dans l'exception **MailServiceException**.

Les informations suivantes sont obligatoires :

- Serveur de mail
 - code d'erreur: **MailErrorKey.ERROR_KEY_MAIL_HOST_MANDATORY**
- L'adresse email de l'expéditeur
 - **MailErrorKey.ERROR_KEY_MAIL_SENDER_MANDATORY**
- Au moins un destinataire
 - **MailErrorKey.ERROR_KEY_MAIL_TO_MANDATORY**
- Sujet du mail
 - **MailErrorKey.ERROR_KEY_MAIL_SUBJECT_MANDATORY**
- Le message
 - **MailErrorKey.ERROR_KEY_MAIL_MESSAGE_MANDATORY**

12.2.3. EXEMPLES

Exemple 1 : Exemple de code d'invocation du service de mail simple (Mail minimum)

```
SimpleMailService mailService = (SimpleMailService) factory.getService(user, "...");

mailService.setMailHost("smtp.serveur.com");
mailService.setFrom("sender@serveur.com");
mailService.addTo("recipient@psa.com");
mailService.setSubject("Sujet du mail");
mailService.setMessage("Corps du message");

mailService.execute();
```

Exemple 2 : Exemple de code d'invocation du service de mail simple avec Velocity

```
SimpleMailService mailService = (SimpleMailService) factory.getService(user, "...");
```



```

mailService.setMailHost("smtp.serveur.com");
mailService.setFrom("sender@serveur.com");
mailService.addTo("recipient@psa.com");
mailService.setSubject("Sujet du mail");

Map model = new HashMap();
// Mettre les objets dans le contexte
model.put("...", ...);
model.put("...", ...);

mailService.setMessageTemplate("test.vm", model, "templates");

mailService.execute();

```

templates : correspond au répertoire contenant les templates Velocity.

test.vm : Fichier contenant le modèle.

12.3. MAIL HTML / MULTIPART

La classe d'implémentation de ce service est :

com.inetpsa.fwk.service.mail. MailService

12.3.1. PRESENTATION

Ce service permet d'envoyer des mails au format texte ou HTML avec ou sans attachement.

Comme pour le service de mail « Simple », le message texte ou HTML peut être généré à partir d'un template Velocity.

L'attachement de documents peut se faire en utilisant les méthodes suivantes :

- **addPart** : Ajout d'un « part » dans le mail

```

/**
 * Ajout d'une nouvelle partie dans le mail.
 *
 * @param content
 *           Le contenu.
 * @param contentType
 *           Le type de contenu (ex. <i>text/plain</i>, ...)
 *
 * @throws MailServiceException
 *           Erreur lors de l'ajout du <i>Part</i>
 */
public void addPart(String content, String contentType)
                    throws MailServiceException;

```

- **attach** : Attachement d'un fichier, en précisant la disposition.

```
/**
```

```

* Attache au mail un fichier localisé par son <b>URL</b>.
*
* @param url
*         l'URL du fichier (l'URL doit être valide !!!).
* @param name
*         Le nom du champs correspondant au fichier attaché.
* @param description
*         Une description pour l'attachement.
* @param disposition
*         La disposition de l'attachement <b><i>mixed</i></b> or
<b><i>inline</i></b>.
*         Utiliser les champs statiques
*         <code>IMultipartMail.INLINE</code> et
*         <code>IMultipartMail.ATTACHEMENT</code>
*
* @throws MailServiceException
*         Erreur lors de l'attachement du fichier
*/
public void attach(URL url, String name, String description, String
disposition) throws MailServiceException;

```

- **Embed** : Permet d'embarquer un contenu dans le corps du mail, chaque éléments embarqué possède un Content-ID.

```

/**
* Ajoute une URL dans le HTML.
*
* <p>
* Cette methode permet d'embarquer un fichier localisée par son URL dans le
* corps du mail. Cela permet, par exemple, d'ajouter directement des images
* dans le mail. Ces fichiers embarqués doivent être référencés par un code
* cid <code>cid:xxxxxx</code> dans l'URL, où xxxxxx est le <i>Content-
ID</i>
* retourné par cette méthode.
*
* <p>
* Exemple d'utilisation:<br>
* <code><pre>
*     MailService mailService = ...
*     ...
*     mailService.setHtmlMsg("<html><img src=cid:" +
*         embed(""file:/my/image.gif","image.gif") +
*         &gt;&lt;/html>&quot;);
*     ....
* </pre></code>
*
* @param url
*         l'URL du fichier.
* @param name
*         Le nom qui sera positionné dans un champ <i>header</i>.
*

```

```

* @return Une chaine contenant le <i>Content-ID</i> du fichier.
* @throws MailServiceException
*         si l'URL est invalide
*/
public String embed(URL url, String name) throws MailServiceException;

```

12.3.2. VALIDATION

Les validations sont les mêmes que celles effectuées par le mail simple (cf. 12.2.2)

12.3.3. EXEMPLES

Exemple 1 : Exemple de code d'invocation du service de mail Multipart (Mail minimum)

```

// Dans une action utilisez directement la méthode getService(...)
MailService mailService = (MailService) factory.getService(user, "...");

mailService.setMailHost("smtp.serveur.com");
mailService.setFrom("sender@serveur.com");
mailService.addTo("recipient@psa.com");
mailService.setSubject("Sujet du mail");
mailService.setMessage("Corps du message");

mailService.execute();

```

Exemple 2 : Exemple de code d'invocation du service de mail HTML

```

MailService mailService = (MailService) factory.getService(user, "...");

mailService.setMailHost("smtp.serveur.com");
mailService.setFrom("sender@serveur.com");
mailService.addTo("recipient@psa.com");
mailService.setSubject("Sujet du mail");

mailService.setHtmlMessage("<html>...</html>");

mailService.execute();

```

Exemple 3 : Exemple d'utilisation attachement de fichier

```

MailService mailService = (MailService) factory.getService(user, "...");
...
...
URL url = new URL("http://gesbob.inetpsa.com/bob1/lego/bandeauChartePSA/bandeau_fichiers/logo_site.jpg");
mailService.attach(url, "logo.jpg", "logo LEGO", MailService.ATTACHMENT);

mailService.execute();

```

Exemple 4 : Utilisation du embed

```

MailService mailService = (MailService) factory.getService(user, "...");

```

```
...  
...  
URL url = new URL("http://gesbob.inetpsa.com/bob1/lego/bandeauChartePSA/bandeau_fichiers/logo_site.jpg");  
String cid = mailService.embed(url, "Logo LEGO");  
  
String htmlMessage = "<html>The LEGO's logo - <img src=\"cid:\"+ cid + \"\"><html>";  
mailService.setHtmlMessage(htmlMessage);  
  
mailService.execute();
```

13. ACCOSTAGE TOOLBOX XML

13.1. INTRODUCTION

Les pré requis concernant ce chapitre sont de connaître la notion de [BusinessService](#) ainsi que les mécanismes mis en place par la toolbox xml.

A partir de la version 2.0 Lego propose un accostage de certains de ses mécanismes existants avec la toolbox xml, principalement au travers de l'apparition de deux nouvelles fonctionnalités :

- Mise à disposition d'un [BusinessService](#) Lego en tant que prise de service.
- Accès à une prise de service au travers d'un [BusinessService](#).

L'intérêt principal de la première fonctionnalité est de permettre de créer une seule entité métier (un [BusinessService](#)) qui va pouvoir être utilisée au sein de divers environnements qu'ils soient locaux (dans une action struts exécutée dans un serveur d'application, une jvm autonome pour un batch) ou distants (au travers d'un appel prise de service avec l'accostage toolbox xml). Cela permet de factoriser du code, de le maintenir plus facilement, et de le publier plus rapidement au travers d'une prise de service.

La seconde fonctionnalité se focalise sur l'homogénéité des structures de développement à mettre en place pour les équipes études. Au travers de Lego 1.x les développeurs ont pris l'habitude d'utiliser des [BusinessService](#) pour déclencher du code métier local, l'accostage va maintenant leur permettre d'utiliser des [BusinessService](#) pour accéder à du code métier s'exécutant sur un serveur distant au travers de la toolbox et ce de manière transparente en terme de programmation, offrant ainsi une continuité de développement que le métier soit géré localement ou à distance.

13.2. PUBLICATION D'UN BUSINESSSERVICE EN TANT QUE PRISE DE SERVICE

Avant de pouvoir publier en prise de service un [BusinessService](#) il va y avoir plusieurs points à prendre en compte, points liés à l'infrastructure toolbox xml utilisée pour la gestion des prises de services.

13.2.1. ETAPES A SUIVRE POUR PUBLIER UN BUSINESSSERVICE

La publication d'un [BusinessService](#) en tant que prise de service va se faire au travers du fichier fwk.xml qui va contenir toute la paramétrie nécessaire, pour ce faire nous allons y retrouver un nouveau tag [priseservices](#) :

```
<?xml version="1.0" encoding="UTF-8"?>
<fwk>
  <i18n>
    <class>com.inetpsa.fwk.i18n.service.db.ServiceResourcesDB</class>
    <default_locale>fr_FR</default_locale>
  </i18n>
  ...
  <services>
  ...
  </services>
  <priseservices>
  ...
  </priseservices>
  <sgbd schemapropertykey="db2ofwk" />
</fwk>
```

A l'intérieur de ce tag nous allons retrouver autant de tag **priseservice** que de **BusinessService** que nous souhaitons publier.

Le tag **priseservice** va contenir 4 blocs de configurations au travers des tags :

- **servicemapping**
- **inputoutputmapping**
- **exceptionmappings**
- **security**

CONFIGURATION SERVICEMAPPING

Le tag **servicemapping** va contenir la configuration permettant de relier un appel de prise de service à traiter avec le **BusinessService** à exécuter (et optionnellement la fonction à exécuter sur ce **BusinessService**).

Par exemple :

```
...
<priseservices>
  <priseservice>
    <servicemapping>
      <do>GetList</do>
      <servicename>RechercheCampagne</servicename>
    </servicemapping>
  </priseservice>
</priseservices>
...
```

Sur réception d'un appel prise de service avec le verbe **GetList** on déclenche l'exécution du service **RechercheCampagne**.

Ou :

```

...
<priseservices>
  <priseservice>
    <servicemapping>
      <do>GetList</do>
      <servicename>GestionCampagne</servicename>
      <function>recherche</function>
    </servicemapping>
    ...
  </priseservice>
</priseservices>

```

Sur réception d'un appel prise de service avec le verbe **GetList** on déclenche l'exécution de la fonction **recherche** du service **GestionCampagne**.

CONFIGURATION INPUTOUTPUTMAPPING

le paramétrage du tag **inputoutputmapping** va dépendre du type des entrée/sortie du **BusinessService** à publier, et donc du fait que le **BusinessService** ait été ou non créé dans l'optique d'être publié ultérieurement, les différents cas possibles sont détaillés dans les chapitres 13.2.2 et 13.2.3 .

CONFIGURATION EXCEPTIONMAPPINGS

La toolbox xml joint aux messages de réponse un code status indiquant quel est l'état de retour de la prise de service, certains code existent déjà (exemple : accès refusé -> code 42 , service indisponible -> code 51) et d'autres peuvent être utilisés par les projets créant des prises de services (il est alors utile de contacter l'équipe toolbox xml pour discuter de la création de nouveaux codes afin d'homogénéiser leur utilisation), sachant que les **BusinessService** travaillent eux avec des exception en cas de problème d'exécution il est nécessaire de pouvoir effectuer un **mapping exception levée -> code status à retourner**, ce qui est fait au travers du tag **exceptionmappings** .

Par exemple si notre service de recherche de campagne de rappel prend en entrée un numéro VIN et retourne une exception **com.inetpsa.fwk.example.IncorrectVinException** si celui-ci est incorrect alors on aura sachant que la toolbox propose le code 52 -> « invalid parameter » :

```

...
<priseservices>
  <priseservice>
    ...
    <exceptionmappings>
      <exceptionmapping>
        <exceptionclassname>com.inetpsa.fwk.example.IncorrectVinException</exceptionclassname>
        <code>52</code>
      </ exceptionmapping>
    </ exceptionmappings>
  </priseservice>
</priseservices>


```

Sur réception d'un appel prise de service avec un numéro VIN incorrect le [BusinessService](#) retournera une `IncorrectVinException` et le client de la prise de service recevra un code 52 -> « invalid parameter ».

Lego propose un mapping de base (qui ne peut être surchargé) pour un ensemble d'exceptions :

- la classe [com.inetpsa.fwk.service.priseservice.exception.UnkownDoException](#) (utilisée lorsque les [configuration servicemapping](#) définis ne permettent pas d'associer le `do` présent dans le message reçu avec un [BusinessService](#) à déclencher) sera mappée vers le code status **55**.
- la classe [com.inetpsa.fwk.exception.BadAuthentication](#) (utilisée lorsque les informations d'accréditation de l'utilisateur ne sont pas correctes) sera mappée vers le code status **41** (correspondant au niveau toolbox à la constante définie sur [IStatus.AUTHENTICATION_FAILED](#)).
- la classe [com.inetpsa.fwk.exception.SecurityException](#) (utilisée lorsque l'utilisateur ne dispose pas des droits suffisants pour exécuter le service) sera mappée vers le code status **42** (correspondant au niveau toolbox à la constante définie sur [IStatus.ACCESS_DENIED](#)).

Les définitions de mapping projets sont ensuite appliquées si l'exception ne correspond pas à une de celles définies ci-dessus, si aucune correspondance n'a été trouvée un code status **54** (correspondant au niveau toolbox à la constante définie sur [IStatus.SERVICE_FAILED](#)) est remonté.

	<p>Chaque définition de priseservice peut contenir des définitions d'exceptionmapping locales à cette prise de service, mais il est aussi possible de définir un mapping d'exception commun à toutes les prises de services au niveau du tag priseservices, exemple :</p> <pre><priseservices> ... <exceptionmappings> <exceptionmapping> <exceptionclassname>com.inetpsa.fwk.example.IncorrectVinException</exceptionclassname> <code>52</code> </ exceptionmapping> </ exceptionmappings> </priseservices></pre>
---	---

CONFIGURATION SECURITY

Lego permet d'appliquer une paramétrie de sécurité pour l'accès aux [BusinessService](#), le tag [security](#) va définir le comportement à adopter concernant la gestion de la sécurité lors du déclenchement d'un [BusinessService](#) suite à une demande de prise de service.

Pour ce faire il y a 3 tags à renseigner :

- [authenticated](#) avec comme valeur true ou false, si celui-ci est à false il n'est pas nécessaire pour le client de la prise de service de fournir le header contenant le mot de passe, l'identifiant seul suffit et aucune authentification n'a lieu, mais la sécurité d'accès au service (profils, ...) est quand même vérifiée.
- [anonymous](#) avec comme valeur true ou false, si celui-ci est à true le framework considère que l'accès à cette prise de service se fait de manière anonyme et donc aucune sécurité n'est vérifiée et aucune information sur l'utilisateur n'est disponible lors de l'exécution du service.
- [secured](#) avec comme valeur true ou false, si celui-ci est à false la sécurité mise en place sur le [BusinessService](#) ne sera pas vérifiée mais les informations utilisateur (objet User initialisé avec l'identifiant provenant les headers) seront disponibles lors de l'exécution.

Exemple d'une paramétrie nécessitant de fournir l'identifiant et le mot de passe en vue d'une authentification et pour laquelle la sécurité appliquée sur les [BusinessService](#) est vérifiée :

```
...
<priseservices>
  <priseservice>
    ...
    <security>
      <authenticated>true</authenticated>
      <anonymous>false</anonymous>
      <secured>true</secured>
    </security>
  </priseservice>
</priseservices>
...
```

CACHE UTILISATEUR

Lors d'un appel sécurisé à un [BusinessService](#) au travers d'une prise de service un objet de type [com.inetpsa.fwk.security.beans.User](#) sera instancié et initialisé à partir des informations présentes dans les headers de sécurité du message reçu par la toolbox, ce qui implique que pour chaque appel sécurisé à la prise de service on se retrouve avec une authentification et une vérification des droits d'un utilisateur. Pour des raisons de performances Lego 2.x propose donc la possibilité de mettre en place un cache sur les objets utilisateurs permettant ainsi de ne pas à chaque appel effectuer une initialisation de sécurité.

Ce cache va se paramétrer via un tag [usercache](#) à positionner dans [priseservices](#) avec 5 tags positionnables dont 1 obligatoire :

- [implclassname](#) : correspondant à la classe d'implémentation de gestion du cache à utiliser, lego en propose 2 : [com.inetpsa.fwk.service.priseservice.NoOpUserCache](#) qui est un gestionnaire n'effectuant pas de cache pour les objets [User](#) et [com.inetpsa.fwk.service.priseservice.DefaultUserCache](#) qui effectue du cache suivant la paramétrie définie par les tags ci-dessous.

Et 4 tags optionnels :

- [maxobjects](#) : nombre maximum d'objets [User](#) contenus dans le cache, quand ce nombre est atteint et qu'un nouvel objet doit être ajouté une politique *LastRecentlyUsed* est appliquée et c'est l'objet qui n'a pas été utilisé depuis la plus grande période de temps qui est supprimé, par défaut 200 objets.
- [timeout](#) : durée de vie (en secondes) d'un objet avant qu'il ne soit supprimé du cache (cela permet de « rafraîchir » les objets présents dans le cache vis-à-vis des droits présents dans l'annuaire), par défaut 120s.
- [objectsbetweenlaps](#) : nombre d'objets testés à chaque passe de vérification de timeout, par défaut 50 objets.
- [evictionsleeptime](#) : temps d'attente entre 2 vérifications de timeout, par défaut 10s.

PRISE DE SERVICE GEREE EN HTTP

Dans le cas d'une prise de service fournie en http il faut déclarer au niveau du fichier web.xml un servlet permettant de gérer les demandes :

```
<servlet>
    <servlet-name>FwkPriseServiceDispatcher</servlet-name>
    <servlet-class>com.inetpsa.fwk.service.priservice.FWKSoapDispatcherServlet</servlet-
class>
</servlet>
<servlet-mapping>
    <servlet-name>FwkPriseServiceDispatcher</servlet-name>
    <url-pattern>/priservice</url-pattern>
</servlet-mapping>
```

en positionnant le pattern utilisé pour répondre aux demandes de prise de service.

Un paramètre de configuration est disponible pour le servlet mis en place :

```
<init-param>
    <param-name>basicauth</param-name>
    <param-value>true</param-value>
</init-param>
```

Il indique au servlet de ne traiter que les appels de client ayant un header de BasicAuth contenant un couple login/mot de passe correct, il est possible d'effectuer des tests supplémentaires en étendant la classe FWKSoapDispatcherServlet et en redéfinissant la méthode :

```
boolean checkAccess(User user)
```

Par exemple vérifier que l'utilisateur appartient bien à un groupe ldap particulier et retourner *false* si ce n'est pas le cas comme dans l'exemple ci-dessous.

A noter que dans l'exemple il peut être intéressant d'activer les fonctionnalités de cache du composant java ldap psa (informations disponibles dans la documentation du composant) : lego ne cachant pas les informations liées au groupes ldap chaque appel de prise de service déclenchera l'exécution de :

```
user.getGroups();
```

Et donc la requête correspondante vers l'annuaire LDAP.

```

public class MySoapServlet extends FWKSoapDispatcherServlet {

    private LDAPGroup priseServiceGroup;

    public void init() throws ServletException {
        try {
            priseServiceGroup = new LDAPGroup();
            priseServiceGroup.setName("ROLE.PRD.PRISESERVICE");
        } catch (LDAPException e) {...//on lève une ServletException
        }
    }

    protected boolean checkAccess(User user) {
        boolean found = false;
        try {
            Enumeration groups = user.getGroups();
            while (groups.hasMoreElements() && !found)
                found = priseServiceGroup.equals((LDAPGroup) groups
                    .nextElement());
        } catch (LDAPException e) {...
        }
        return found;
    }
}

```

13.2.2. CAS D'UN BUSINESSSERVICE NON PREVU POUR LE BESOIN DE PUBLICATION

Pour rappel la toolbox xml travaille avec des schémas xml permettant de définir les messages de question et de réponse, les données échangées au travers de ces messages doivent donc correspondre à des schémas normalisés pouvant être retrouvés au sein du [repository XML](#) ou alors à des schémas définis conjointement avec l'équipe toolbox xml et qui seront à terme intégrés au sein du repository.

Il sera alors difficile d'effectuer de manière déclarative un mapping entre des données d'entrées du [BusinessService](#) qui seraient de type primitif (String , Integer, ...) ou alors de type complexe mais ne correspondant exactement avec aucun schéma défini (ou à définir) dans le repository.

Le framework propose alors aux projets de créer une classe implémentant [com.inetpsa.fwk.service.priseservice.ITransformerPrise](#) avec les 2 méthodes suivantes à créer :

- [fromPriseInputToBusinessServiceInput](#)([PsaActionInput](#) input,[BusinessService](#) service)
- [fromBusinessServiceOutputToPriseOutput](#)([BusinessService](#) service,[PsaResponse](#) response)

La première permettant d'initialiser les entrées d'un [BusinessService](#) à partir des paramètres présents dans le [PsaAction](#) provenant de la toolbox, on peut ainsi par exemple récupérer de la part de la toolbox xml des objets complexes correspondants à des schémas présent dans le repository puis initialiser des entrées primitives au niveau du [BusinessService](#).

La seconde permettant de récupérer les données

de sortie du [BusinessService](#) et d'initialiser les valeurs de réponse sur le [PsaResponse](#) géré par la toolbox.

On aura alors comme paramétrie par exemple :

```
...
<priseservices>
  <priseservice>
    ...
    <inputoutputmapping>
      <classmapping>
<transformerclassname>com.inetpsa.fwk.example.PriseServiceTransformerImpl</transformerclassna
me>
      </classmapping>
    </inputoutputmapping>
  </priseservice>
</priseservices>
...
```

13.2.3. CAS D'UN BUSINESSSERVICE PREVU POUR LE BESOIN DE PUBLICATION

Dans le cas d'un [BusinessService](#) développé dès le départ avec une ouverture potentielle vers l'extérieur les objets d'entrée/sortie du [BusinessService](#) correspondraient à des schémas présents dans le repository xml et donc le mapping avec les messages toolbox xml via du déclaratif est possible.

CONFIGURATION DES ENTREES

Le mapping entre les paramètres du message reçu et les entrées du [BusinessService](#) va pouvoir être effectué via 2 possibilités liées à la gestion par la Toolbox XML des paramètres reçus (avec ou sans matérialisation de l'objet [Parameter](#)):

- un mapping où l'on fourni le nom de la classe [Parameter](#) utilisée par la toolbox pour unmarshaler les paramètres ainsi qu'une série de couples : clé d'entrée BusinessService/attribut de l'objet [Parameter](#).

De cette manière pour chaque attribut lu sur l'objet [Parameter](#) (via le getter correspondant) la méthode [setInput](#) sera appelée sur le [BusinessService](#) en fournissant en entrée la clé paramétrée et l'objet récupéré au travers de l'attribut sur le [Parameter](#).

Exemple :

```

...
<priseservices>
  <priseservice>
    ...
    <inputoutputmapping>
      <declarativemapping>
        <input>
          <keyattributemappings>
            <containerclassname>com.inetpsa.fwk.test.Parameter</containerclassname>
            <keyattributemapping>
              <key>codevin</key>
              <attribute>vin</attribute>
            </keyattributemapping>
            <keyattributemapping>
              <key>langue</key>
              <attribute>langue</attribute>
            </keyattributemapping>
          </keyattributemappings>
        </input>
        ...
      </declarativemapping>
    </inputoutputmapping>
  </priseservice>
</priseservices>
...

```

Ici on aura 2 appels à la méthode **setInput** : un avec la clé **codevin** et l'objet correspondant à l'attribut **vin** de l'instance de **Parameter**, et un autre appel avec la clé **langue** et l'objet correspondant à l'attribut **langue** de l'instance de **Parameter**.

- Un mapping où l'on fourni une série de couples : clé d'entrée **BusinessService**/nom de classe correspondante.

De cette manière pour chaque clé on récupère au niveau de l'instance de **Parameter** l'objet dont la position correspond à celle de la clé dans la liste de clés, puis la méthode **setInput** sera appelée sur le **BusinessService**. L'ordre de la liste des clés paramétrées est donc important dans la mesure où celui-ci doit correspondre à l'ordre de la liste des objets paramètres sur le **Parameter**.

Exemple :

```
...
<priseservices>
  <priseservice>
    ...
    <inputoutputmapping>
      <declarativemapping>
        <input>
          <keyclassnamemappings>
            <keyclassnamemapping>
              <key>codevin</key>
              <classname>com.inetpsa.xml.produitprocess.vehicule.identification.VIN</classname>
            </keyclassnamemapping>
            <keyclassnamemapping>
              <key>langue</key>
              <classname>com.inetpsa.xml.structure.informatique.developpementetsi.referentielechange.Langue</classname>
            </keyclassnamemapping>
          </keyclassnamemappings>
        </input>
        ...
      </declarativemapping>
    </inputoutputmapping>
  </priseservice>
</priseservices>
...
```

Ici l'objet à la position 0 (de type com.inetpsa...VIN) de la liste des paramètres présents sur le [Parameter](#) sera utilisé pour effectuer un [setInput](#) sur le [BusinessService](#) avec la clé [codevin](#).

L'objet à la position 1 (de type com.inetpsa...Langue) de la liste des paramètres présents sur le [Parameter](#) sera utilisé pour effectuer un [setInput](#) sur le [BusinessService](#) avec la clé [langue](#).

CONFIGURATION DES SORTIES

Le mapping entre les données du message de réponse et les sorties du [BusinessService](#) va se faire au moyen d'une liste de clés de sortie du [BusinessService](#) qui correspondront (de manière ordonnée) à la liste des « valeurs » positionnées sur l'objet toolbox [PsaResponse](#).

Exemple :

```

...
<priseservices>
  <priseservice>
    ...
    <inputoutputmapping>
      <declarativemapping>
        ...
        <output>
          <keymapping>
            <key>campagnes</key>
          </keymapping>
        </output>
      </declarativemapping>
    </inputoutputmapping>
  </priseservice>
</priseservices>
...

```

Ici l'objet récupérable au travers de la méthode **getOutput** sur le **BusinessService** pour la clé **campagnes** sera ajouté à la liste des valeurs de l'instance de **PsaResponse**.



Si l'objet récupéré via la méthode **getOutput** est de type **Collection** alors ce n'est pas l'objet qui sera directement ajouté au **PsaResponse** mais les objets contenus dans la collection. Les objets ajoutés au **PsaResponse** devant pouvoir être « marshallé » via castor et donc être connus et correspondre à une structure définie au travers d'un schéma xml.

13.3. ACCES A UNE PRISE DE SERVICE AU TRAVERS D'UN BUSINESSSERVICE

Une nouvelle fonctionnalité offerte suite à l'accostage avec la toolbox xml est de pouvoir accéder à une prise de service de manière transparente (en terme de programmation) pour le développeur au travers de l'utilisation d'un **BusinessService**.

13.3.1. ETAPES A SUIVRE DEFINIR UN BUSINESSSERVICE POINTANT VERS UNE PRISE DE SERVICE

La définition d'un **BusinessService** « distant » se fait au travers d'un enrichissement de la définition courante d'un service notamment via l'apparition d'un nouveau tag : **distantmapping** utilisé au sein du tag **service**.

Dans le cas d'un **BusinessService** distant le type de la classe d'implémentation doit être :

com.inetpsa.fwk.service.priservice.DistantBusinessService

Cette implémentation étant fournie dans la distribution de Lego 2.x.

Exemple :

```

<fwk>
...
<services>
  <service name="servicedistant" description="..."
  classname="com.inetpsa.fwk.service.priservice.DistantBusinessService">
    ...
  </service>
</services>
...
</fwk>

```

La configuration vers la prise de service se fait au travers du tag **distantmapping** qui contient plusieurs blocs à paramétrer :

CONFIGURATION HEADERS

le tag **headers** va permettre de gérer les données contenues dans les headers du **IWrapper** utilisé par la toolbox xml, il y a 3 informations obligatoires à fournir et une optionnelle :

- **from** : identifiant de l'émetteur.
- **to** : identifiant du destinataire.
- **anonymous** : flag indiquant que les accréditations de l'utilisateur exécutant le service sont positionnées dans les headers de sécurité (valeur **false**) ou que le header de sécurité est vide (valeur **true**).
- **sendPassword** (optionel): flag indiquant que le mot de passe de l'utilisateur exécutant le service doit être positionné dans les headers de sécurité (valeur **true**) ou non (valeur **false**), si le tag n'est pas présent la valeur par défaut est **true**.

Exemple :

```

...
<services>
  <service name="servicedistant" description="..."
  classname="com.inetpsa.fwk.service.priservice.DistantBusinessService">
    <distantmapping>
      <headers>
        <from>application A</from>
        <to>application B</to>
        <anonymous>true</anonymous>
      </headers>
    ...
  </distantmapping>
</service>
</services>

```

CONFIGURATION TRANSPORT

le tag **transport** va contenir les données permettant de définir le type de transport à utiliser pour véhiculer les messages au travers toolbox xml, il y a 2 choix possibles : http et mqseries.

- Configuration d'un transport http :

Dans le cas du transport http il y a une information obligatoire à fournir :

- **recipienturl** : url http de la prise de service à laquelle on souhaite accéder.

Et plusieurs informations optionnelles :

- le compte de service utilisé pour s'authentifier via *BasicAuth*, constituée de 2 tags :
 - **login** : identifiant du compte de service.
 - **password** : mot de passe du compte de service.
- **requestcharset** : le charset envoyé dans le header HTTP *Content-type*.
- Les informations sur le proxy à utiliser pour émettre la requête :
 - **proxyhost** : le nom d'hôte du proxy.
 - **proxyport** : le port sur lequel accéder au proxy.
 - **proxyuser** : l'identifiant du compte de servitude pour s'authentifier au proxy.
 - **proxypassword** : le mot de passe du compte de servitude pour s'authentifier au proxy.

Exemples :

```
...
<services>
  <service name="servicedistant" description="..."
  classname="com.inetpsa.fwk.service.priservice.DistantBusinessService">
    <distantmapping>
      ...
      <transport>
        <http>
          <recipienturl>http://localhost:8080/testpriseservice/service</recipienturl>
        </http>
      </transport>
      ...
    </distantmapping>
  </service>
  .
  .
```

Ou :

```
...
  <transport>
    <http>
      <recipienturl>http://localhost:8080/testpriseservice/service</recipienturl>
      <requestcharset>utf-8</requestcharset>
      <proxy>
        <proxyhost>proxy-test.inetpsa.com</proxyhost>
        <proxyport>80</proxyport>
        <proxyuser>MZPFWK01</proxyuser>
        <proxypassword>test</proxypassword>
      </proxy>
    </http>
  </transport>
```

```
...  
<transport>  
  <http>  
    <recipienturl>http://localhost:8080/testpriseservice/service</recipienturl>  
    <securityaccount>  
      <login>identifiant compte service</login>  
      <password>mot passe compte service</password>  
    </securityaccount>  
  </http>  
</transport>  
...
```

- Configuration d'un transport mqseries :

Dans le cas du transport mqseries il y a des informations obligatoires à fournir :

- **oneway** : flag indiquant que la prise de service accédée est de type question seule (valeur **true**) ou de type question/réponse (valeur **false**).
- **queuemanagername** : le nom du gestionnaire de file d'attente mqseries auquel on accède.
- **requestqueue name** : le nom de la file d'attente dans laquelle sont déposés les messages de question.

Et des informations optionnelles :

- **persistenceenabled** : à utiliser en cas d'une prise de service en mode question seule, permet d'indiquer que les messages posés dans la file de question sont persistés (valeur à **true**) ou non persistés (valeur à **false**).
- **responsequeue name** : le nom de la file d'attente dans laquelle sont lus les messages de réponse (à positionner dans le cas d'une prise de service ayant **oneway** à **true**).
- **expirytimemillis** : le temps d'expiration des messages de question en millisecondes(par défaut à 30s).
- **waitinterval millis** : le délai d'attente d'un message de réponse en millisecondes(par défaut à 30s).

Exemple :

```
...
<services>
  <service name="servicedistant" description="..."
  classname="com.inetpsa.fwk.service.priservice.DistantBusinessService">
    <distantmapping>
      ...
      <transport>
        <mqseries>
          <oneway>false</oneway>
          <queuemanagename>JMS\LOCALHOSTA</queuemanagename>
          <requestqueue>JMS\LOCALHOSTA.QL.FWK.1</requestqueue>
          <responsequeue>JMS\LOCALHOSTA.QL.FWK.2</responsequeue>
          <expirytimemillis>30000</expirytimemillis>
          <waitinterval>30000</waitinterval>
        </mqseries>
      </transport>
    </distantmapping>
  </service>
</services>
...
```

CONFIGURATION DO <-> BUSINESSSERVICE/FONCTION

Le mapping du « service distant » vers la prise de service va pouvoir se faire de deux manières :

- le service n'est pas typé [DispatchBusinessService](#) et ne propose donc pas de fonctions, on va le lier avec un [do](#) unique et donc déclarer un seul tag [actionmapping](#), exemple :

```
...
<services>
  <service name="rechercheCampagnes" description="..."
  classname="com.inetpsa.fwk.service.priservice.DistantBusinessService">
    <distantmapping>
      ...
      <actionmappings>
        <actionmapping>
          <do>GetList</do>
        </actionmapping>
      </actionmappings>
    </distantmapping>
  </service>
</services>
...
```

- le service est typé multifonction et propose donc des fonctions, on déclare un tag **actionmapping** par fonction existante et lier chaque fonction avec un **do** utilisable au travers de la prise de service, exemple :

```
...
<services>
  <service name="gestionCampagnes" description="..."
  classname="com.inetpsa.fwk.service.priseservice.DistantBusinessService">
    <distantmapping>
    ...
      <actionmappings>
        <actionmapping>
          <do>GetList</do>
          <function>rechercherCampagne</function>
        </actionmapping>
        <actionmapping>
          <do>Create</do>
          <function>creerCampagne</function>
        </actionmapping>
      </actionmappings>
    ...
    </distantmapping>
  </service>
</services>
...
```

Dans l'exemple ci-dessus on voit que :

- la fonction **rechercherCampagne** du service **gestionCampagne** générera au niveau du message envoyé à la prise de service un **do** ayant comme valeur **GetList**.
- la fonction **creerCampagne** du service **gestionCampagne** générera au niveau du message envoyé à la prise de service un **do** ayant comme valeur **Create**.

CONFIGURATION DES ENTREES DE MANIERE DECLARATIVE

Pour chaque **actionmapping** déclaré il va être nécessaire de définir le mapping entre les entrées du **BusinessService** et les données mises dans le message de question envoyé à la prise de service.

Cela va se faire au moyen d'une liste de clés d'entrée du **BusinessService** qui correspondront (de manière ordonnée) à la liste des « paramètres » positionnés sur l'objet toolbox **Parameter**.

Exemple :

```

...
<services>
  <service name="rechercherCampagnes" description="..."
  classname="com.inetpsa.fwk.service.priservice.DistantBusinessService">
    <distantmapping>
      ...
      <actionmappings>
        <actionmapping>
          ...
          <inputoutput>
            <declarativemapping>
              <input>
                <keysmapping>
                  <key>vin</key>
                  <key>langue</key>
                </keysmapping>
              </input>
            ...
          </declarativemapping>
        </inputoutput>
      </actionmapping>
    </actionmappings>
  </distantmapping>
</service>
</services>

```

Dans l'exemple ci-dessus l'objet disponible en entrée du [BusinessService](#) pour la clé [vin](#) sera ajouté à la liste des paramètres du message de question puis l'objet disponible en entrée du [BusinessService](#) pour la clé [langue](#) sera ajouté à la liste des paramètres du message de question.



Si l'objet récupéré pour une clé d'entrée (via la méthode [getInput](#)) est de type [Collection](#) alors ce n'est pas l'objet qui sera directement ajouté au [Parameter](#) mais les objets contenus dans la collection. Les objets ajoutés au [Parameter](#) devant pouvoir être « marshallé » via castor et donc être connus et correspondre à une structure définie au travers d'un schéma xml.

CONFIGURATION DES SORTIES DE MANIERE DECLARATIVE

Pour chaque [actionmapping](#) déclaré il va être nécessaire de définir le mapping entre les données présentes dans le message de réponse reçu de la prise de service et les sorties du [BusinessService](#), cela peut se faire de 2 manières :

- un mapping où l'on fourni le nom de la classe [Value](#) utilisée par la toolbox pour « unmarshall » les valeurs ainsi qu'une série de couples : clé de sortie [BusinessService](#)/attribut de l'objet [Value](#).

De cette manière pour chaque attribut lu sur l'objet [Value](#) (via le getter correspondant) la méthode [setOutput](#) sera appelée sur le [BusinessService](#) en fournissant la clé paramétrée et l'objet récupéré au travers de l'attribut sur le [Value](#).

Exemple :

```

...
<services>
  <service name="rechercherCampagnes" description="..."
  classname="com.inetpsa.fwk.service.priservice.DistantBusinessService">
    <distantmapping>
    ...
      <actionmappings>
        <actionmapping>
        ...
          <inputoutput>
            <declarativemapping>
            ...
              <output>
                <keyattributemappings>
                  <containerclassname>com.inetpsa.fwk.test.Value</containerclassname>
                  <keyattributemapping>
                    <key>campagnes</key>
                    <attribute>listeCampagnes</attribute>
                  </keyattributemapping>
                </keyattributemappings>
              </output>
            ...
          </declarativemapping>
        </inputoutput>
      </actionmapping>
    </actionmappings>
  </distantmapping>
</service>
</services>

```

Ici on aura 1 appel à la méthode **setOutput** avec la clé **campagnes** et l'objet correspondant à l'attribut **listeCampagnes** de l'instance de **Value**.

- Un mapping où l'on fourni une série de couples : clé de sortie **BusinessService**/nom de classe correspondante.

De cette manière pour chaque clé on récupère au niveau de l'instance de **Value** l'objet dont la position correspond à celle de la clé dans la liste de clés, puis la méthode **setOutput** sera appelée sur le **BusinessService**. L'ordre de la liste des clés paramétrées est donc important dans la mesure où celui-ci doit correspondre à l'ordre de la liste des objets valeurs sur le **Value**.

Exemple :

```

...
<services>
  <service name="rechercherCampagnes" description="..."
  classname="com.inetpsa.fwk.service.priservice.DistantBusinessService">
    <distantmapping>
      ...
      <actionmappings>
        <actionmapping>
          ...
          <inputoutput>
            <declarativemapping>
              ...
              <output>
                <keyclassnamemappings>
                  <keyclassnamemapping>
                    <key>campagnes</key>
                </keyclassnamemappings>
                <classname>com.inetpsa.xml.commerce.apvtechnique.assistance.Campagnes</classname>
                </keyclassnamemapping>
              </keyclassnamemappings>
            </output>
          ...
        </declarativemapping>
      </inputoutput>
    </actionmapping>
  </actionmappings>
</service>
</services>
...

```

Dans cet exemple l'objet à la position 0 (de type com.inetpsa.xml...Campagnes) de la liste des valeurs présentes sur le **Value** sera utilisé pour effectuer un **setOutput** sur le **BusinessService** avec la clé **campagnes**.

CONFIGURATION DES ENTREES/SORTIES DANS LE CAS D'UNE UTILISATION D'OBJETS NON DEFINIS AU TRAVERS D'UN SCHEMA XML

Si les entrées/sorties du **BusinessService** sont basées sur des types primitifs (String , Integer, ...) ou alors des types complexes qui ne sont pas définis par un schéma xml (et donc non utilisable directement au travers de la toolbox xml) il sera alors difficile d'effectuer de manière déclarative le mapping.

Le framework propose alors aux projets de créer une classe implémentant **com.inetpsa.fwk.service.priservice.ITransformerService** avec les 2 méthodes suivantes à créer :

- **fromBusinessServiceInputToPriseInput** (**BusinessService** service, **PsaAction** action)
- **fromPriseOutputToBusinessServiceOutput** (**PsaResponseInput** response, **BusinessService** service)

La première permettant d'initialiser à partir des entrées d'un **BusinessService** les paramètres contenus dans le **PsaAction**, on peut ainsi par exemple prendre en entrée du **BusinessService** des

données primitives puis initialiser le [PsaAction](#) avec des objets complexes correspondants à des schémas présents dans le repository xml.

La seconde permettant de positionner les données de sortie du [BusinessService](#) à partir des valeurs de retour présentes sur le [PsaResponseInput](#) géré par la toolbox.

On aura alors comme paramétrie par exemple :

```
...
<services>
  <service name="rechercherCampagnes" description="..."
  classname="com.inetpsa.fwk.service.priservice.DistantBusinessService">
    <distantmapping>
    ...
      <actionmappings>
        <actionmapping>
        ...
          <inputoutput>
            <classmapping>
              <transformerclassname>com.inetpsa.fwk.example.ServiceTransformerImpl
            </transformerclassname>
            </classmapping>
          </inputoutput>
        </actionmapping>
      </actionmappings>
    </distantmapping>
  </service>
</services>
```

CONFIGURATION EXCEPTIONMAPPINGS

Comme vu précédemment la toolbox xml utilise pour la gestion d'erreur un code status indiquant quel est l'état de retour de la prise de service, alors que les [BusinessService](#) travaillent eux avec des exception en cas de problème d'exécution, il est donc nécessaire de pouvoir effectuer un **mapping code status retourné -> exception à lever**, ce qui est fait au travers du tag **exceptionmappings**.


Par exemple si la prise de service de recherche de campagne de rappel retourne un code 52 indiquant qu'un paramètre est invalide alors il peut-être intéressant de remonter une exception [com.inetpsa.fwk.example.InvalidParameterException](#) au niveau de l'exécution du service distant, ce qui donnerait la paramétrie:


```
...
<services>
  <service name="rechercherCampagnes" description="..."
  classname="com.inetpsa.fwk.service.priservice.DistantBusinessService">
    <distantmapping>
      ...
      <actionmappings>
        <actionmapping>
          ...
          <exceptionmappings>
            <exceptionmapping>
              <exceptionclassname>com.inetpsa.fwk.example.InvalidParameterException </exceptionclassname>
              <code>52</code>
            </ exceptionmapping>
          </ exceptionmappings>
        </actionmapping>
      </actionmappings>
    </distantmapping>
  </service>
</services>
...
```

Lego propose un mapping de base (qui ne peut être surchargé) pour un ensemble de code retour :

- le code status **41** (correspondant au niveau toolbox à la constante définie sur [IStatus.AUTHENTICATION_FAILED](#)) générera la levée d'une exception de type [com.inetpsa.fwk.exception.BadAuthentication](#) (utilisée lorsque les informations d'authentification de l'utilisateur ne sont pas correctes).
- le code status **42** (correspondant au niveau toolbox à la constante définie sur [IStatus.ACCESS_DENIED](#)) générera la levée d'une exception de type [com.inetpsa.fwk.exception.SecurityException](#) (utilisée lorsque l'utilisateur ne dispose pas des droits suffisants pour exécuter le service).

Si le code de retour ne correspond pas à une de ceux présents ci-dessus alors les définitions de mapping projets sont ensuite appliquées, si aucune correspondance n'a été trouvée une exception [com.inetpsa.fwk.exception.ServiceTechnicalException](#) est remontée avec comme message le texte associé au status dans la réponse venant de la prise de service.

	<p>Chaque définition de « service distant » peut contenir des définitions d'exceptionmapping locales à ce service, mais il est aussi possible de définir un mapping d'exception commun à toutes les « services distants » au niveau du tag services, exemple :</p> <pre><services> ... <exceptionmappings> <exceptionmapping> <exceptionclassname>com.inetpsa.fwk.example.InvalidParameterException </exceptionclassname> <code>52</code> </ exceptionmapping> </ exceptionmappings> </services></pre>
---	---

14. REPORTING

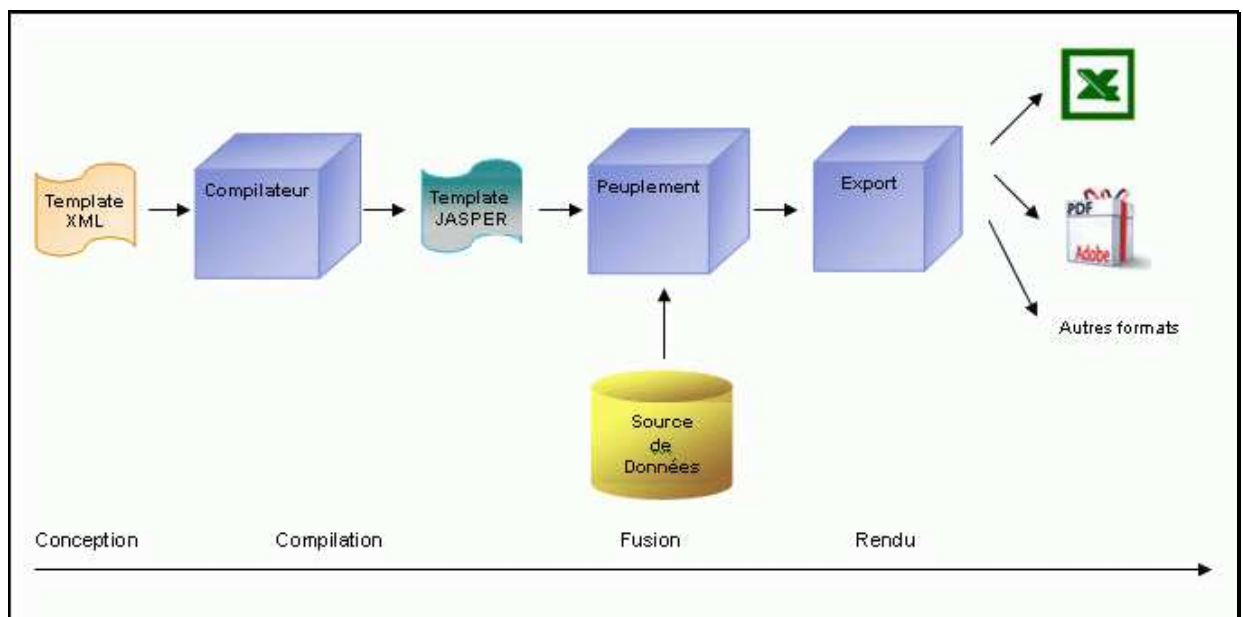
14.1. INTRODUCTION

Ce service fournit sous forme de classe utilitaire, permet de générer des rapports dans différents formats (CSV, PDF, HTML, XLS, XML, RTF et Texte).

Ce service de reporting utilise « **JasperReports** » (<http://jasperreports.sourceforge.net>), un projet opensource, entièrement écrit en Java

« **JasperReports** » utilise le formalisme « XML » pour représenter un état quelque soit son format de sortie., ce flux « XML » est nommé template.

Pendant la phase de génération, ce template « XML » sera compilé en un template « jasper » (équivalent à un fichier « class »). Ce dernier fusionnera avec une source de données pour la phase de peuplement, puis cette fusion sera véhiculée au gestionnaire d'export en charge du rendu demandé.



Fonctionnement de JasperReports

14.2. UTILISATION

14.2.1. CREATION DES RAPPORTS

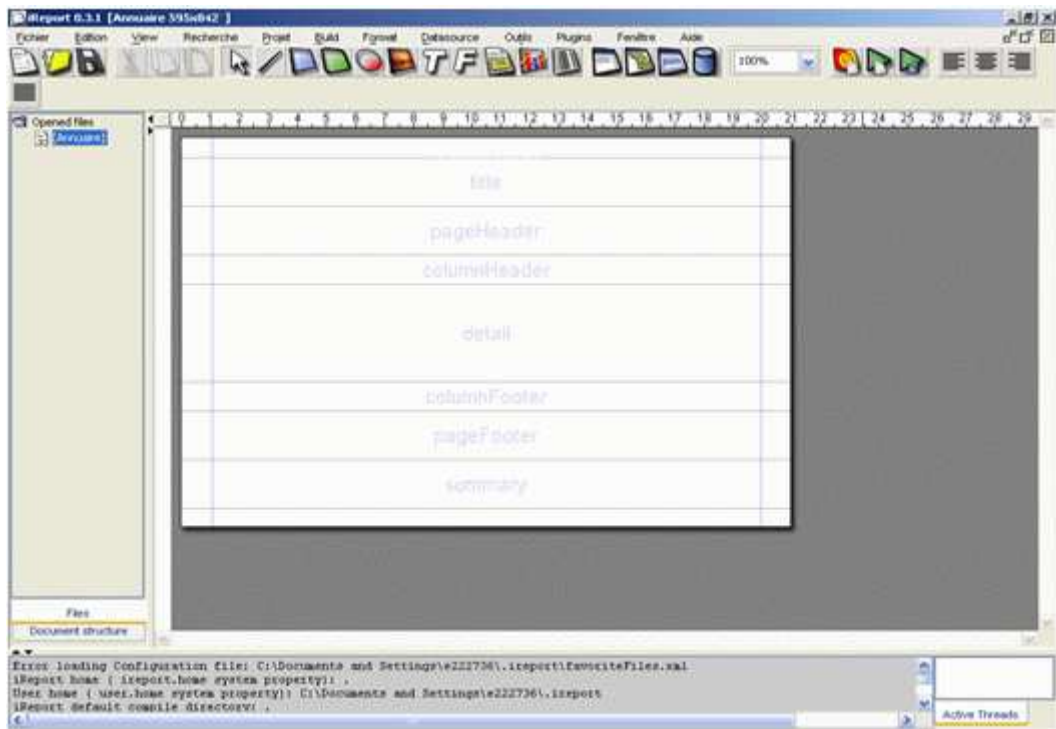
La partie génération du rapport (fichier « .jrxml ») et le rapport compilé (fichier « .jasper ») est indépendante de LEGO.

La création des rapports se fait par des outils tiers. Nous avons retenue l'outil iReport (<http://ireport.sourceforge.net>), une application JAVA/SWING opensource.

Par défaut l'état est découpé en bandes :

- **title** : Le contenu de cette bande est imprimé uniquement sur la première page de l'état.

- **pageHeader** : Le contenu de cette bande peut être imprimé sur l'ensemble des pages de l'état.
- **columnHeader** : Par défaut l'état possède une colonne principale, il est possible d'ajouter plusieurs colonne.
- **detail** : La bande détail est utilisée conjointement avec les groupes, elle affiche les valeurs d'itération d'un groupe.
- **columnFooter** : Représente la fin de la colonne.
- **pageFooter** : Le contenu de cette bande peut être imprimé sur l'ensemble des pages de l'état.
- **summary** : Résumé de l'état, s'imprime sur la dernière page de l'état.



Interface Swing iReport

14.2.2. CLASSE UTILITAIRE LEGO

LEGO fournit une classe utilitaire qui permet de charger un rapport à partir de différente source (fichier, flux, Url...), compiler un rapport, générer dans les formats XML, CSV, PDF, RTF, HTML et Texte, écrire le rapport dans un fichier, ... :

com.inetpsa.fwk.utils.jasper. AbstractJasperReportsUtils

```
public abstract class AbstractJasperReportsUtils {
    ...
    public static JasperReport loadReport(String report) throws Exception ;

    public static JasperReport loadReport(URL reportUrl) throws Exception ;

    public static JasperReport loadReport(InputStream inputStream) throws Exception ;

    public static JasperReport loadReport(File reportFile) throws Exception ;

    public static void renderAsCsv(JasperReport report, Map parameters, Object reportData,
        Writer writer) throws JRException ;
}
```

```

    public static void renderAsHtml(JasperReport report, Map parameters, Object reportData,
    Writer writer) throws JRException ;

    public static void renderAsHtml(JasperReport report, String imageURI, Map parameters,
    Object reportData, Writer writer) throws JRException ;

    public static void renderAsPdf(JasperReport report, Map parameters, Object reportData,
    OutputStream stream) throws JRException ;

    public static void renderAsXls(JasperReport report, Map parameters, Object reportData,
    OutputStream stream) throws JRException ;

    public static void renderAsXml(JasperReport report, Map parameters, Object reportData,
    OutputStream stream) throws JRException ;

    public static void renderAsRtf(JasperReport report, Map parameters, Object reportData,
    OutputStream stream) throws JRException ;

    public static void renderAsText(JasperReport report, Map parameters, Object reportData,
    OutputStream stream) throws JRException ;

    public static void writeToFile(String filename, byte[] content) throws IOException ;

    public static void writeToFile(String filename, StringWriter writer) throws
    IOException ;

    public static void writeToFile(String filename, String content) throws IOException ;

}

```

14.2.3. EXEMPLES D'UTILISATION

Le rapport source JasperReports a été mis en annexe (cf. 19.6).

Le rapport a besoin de certains paramètres et d'une source de données (implémentation de l'interface JRDataSource):

```

Map parameters = new HashMap();
parameters.put("ReportTitle", "Address Report");
parameters.put("DataFile", "CustomDataSourceFile");

```

PROGRAMME JAVA

- CSV

```

StringWriter writer = new StringWriter();
JasperReport report = AbstractJasperReportsUtils.loadReport("DataSourceReport.jasper");
AbstractJasperReportsUtils.renderAsCsv(report, parameters, customDataSrc, writer);
AbstractJasperReportsUtils.writeToFile("DataSourceReport.csv", writer);

```

- PDF

```

ByteArrayOutputStream os = new ByteArrayOutputStream();
JasperReport report = AbstractJasperReportsUtils.loadReport("DataSourceReport.jasper");
AbstractJasperReportsUtils.renderAsPdf(report, parameters, customDataSrc, os);
AbstractJasperReportsUtils.writeToFile("DataSourceReport .pdf", os.toByteArray());

```

PROJET WEB

Le code suivant est à mettre dans une action.

Exemple pour générer du PDF :

```
public class JasperTestAction extends FWKAction {  
    ...  
    ...  
    public ActionForward doExecute(ActionMapping mapping, ActionForm form,  
        HttpServletRequest request, HttpServletResponse response)  
        throws Exception {  
        ...  
        ❶ ServletContext context = getServlet().getServletContext();  
        File reportFile = new File(  
            context.getRealPath("/reports/DataSourceReport.jasper"));  
        ...  
        ❷ JasperReport report = AbstractJasperReportsUtils.loadReport(reportFile);  
  
        ❸ Map parameters = new HashMap();  
        parameters.put("ReportTitle", "Address Report");  
        parameters.put("BaseDir", reportFile.getParentFile());  
  
        ❹ CustomDataSource customDataSrc = ... ;  
  
        ❺ ByteArrayOutputStream os = new ByteArrayOutputStream();  
        AbstractJasperReportsUtils.renderAsPdf(report, parameters, customDataSrc, os);  
  
        ❻ byte[] bytes = os.toByteArray();  
        response.setContentType("application/pdf");  
        response.setContentLength(bytes.length);  
        ServletOutputStream ouputStream = response.getOutputStream();  
        ouputStream.write(bytes, 0, bytes.length);  
        ouputStream.flush();  
        ouputStream.close();  
  
        return null;  
    }  
}
```

❶ Récupération du ServletContext, pour récupérer le chemin réel du rapport.

❷ Chargement du rapport à partir du fichier.

❸ Construction de la Map, contenant les paramètres requis par le rapport.

Ex. Dans le rapport, les paramètres sont définis de la façon suivante :

```
<parameter name="ReportTitle" class="java.lang.String"></parameter>  
<parameter name="DataFile" class="java.lang.String"></parameter>
```

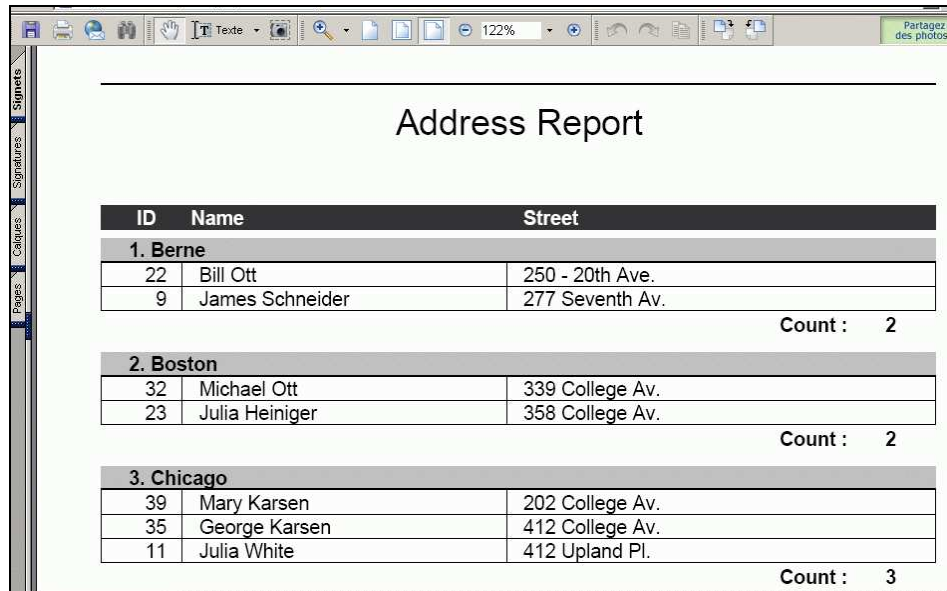
❹ Source de données

❺ Lancement de la génération du rapport au format PDF.

- On positionne dans la réponse, le type mime du contenu ("*application/pdf*") et la taille du fichier. Ensuite dans le flux de sortie de la réponse, on écrit le contenu fichier.

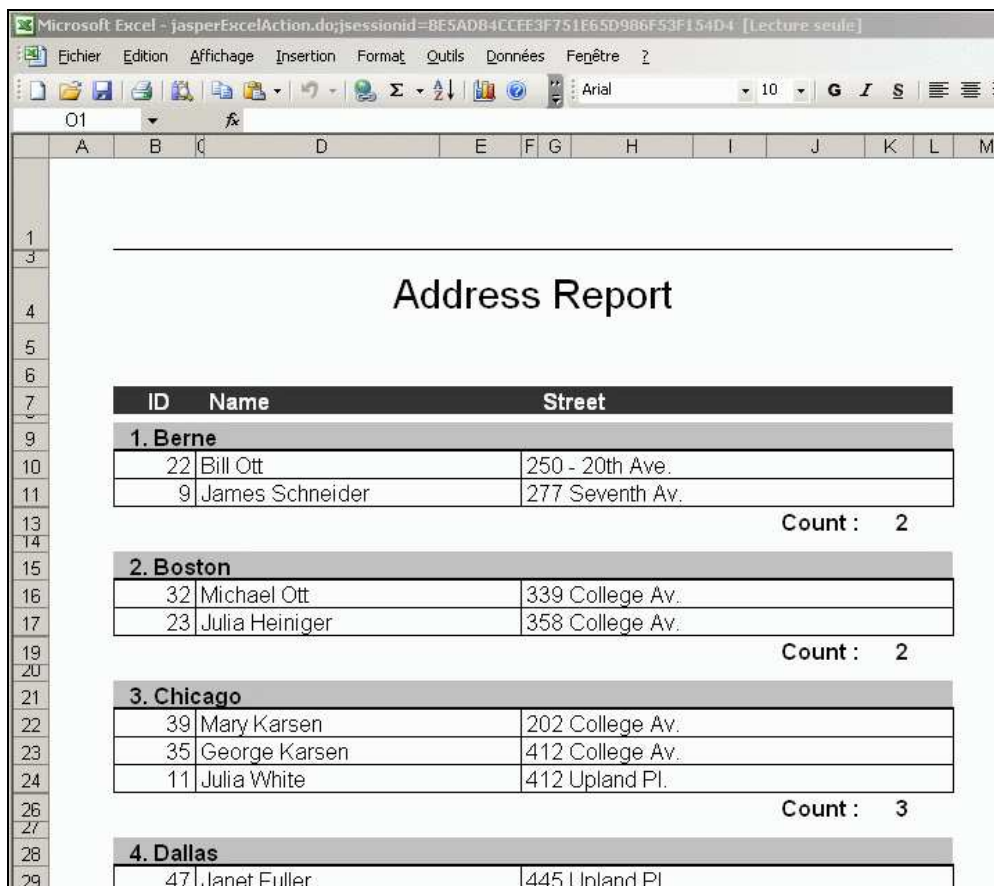
EXEMPLE DE SORTIE

- PDF



ID	Name	Street
1. Berne		
22	Bill Ott	250 - 20th Ave.
9	James Schneider	277 Seventh Av.
		Count : 2
2. Boston		
32	Michael Ott	339 College Av.
23	Julia Heiniger	358 College Av.
		Count : 2
3. Chicago		
39	Mary Karsen	202 College Av.
35	George Karsen	412 College Av.
11	Julia White	412 Upland Pl.
		Count : 3

- Excel



ID	Name	Street
1. Berne		
22	Bill Ott	250 - 20th Ave.
9	James Schneider	277 Seventh Av.
		Count : 2
2. Boston		
32	Michael Ott	339 College Av.
23	Julia Heiniger	358 College Av.
		Count : 2
3. Chicago		
39	Mary Karsen	202 College Av.
35	George Karsen	412 College Av.
11	Julia White	412 Upland Pl.
		Count : 3
4. Dallas		
47	Janet Fuller	445 Upland Pl.

15. OPERATIONS CRUD

CRUD est l'ensemble des opérations génériques :

- **C**reate : Création
- **R**esearch : Recherche
- **U**ppdate : Mise à jour
- **D**elete : Suppression

15.1. SERVICE CRUD OJB

Ce service est une implémentation des services CRUD pour le framework de persistance OJB :

`com.inetpsa.fwk.service.crud.ojb.OJBCrudService`

15.1.1. METHODES DISPONIBLES

Ce tableau reprend les paramètres d'entrées nécessaires en fonction de la méthode utilisée et les paramètres de sorties.

	ENTREE				SORTIE
	IN_CONTEXT_KEY	IN_CLASS	IN_OBJECT	IN_CRITERIA	OUT_OBJECT
create	X		X		
update	X		X		
delete	X		X		
deleteByQuery	X	X		X	
findObject	X	X		X	java.lang.Object
findObjects	X	X		X	java.util.Collection

Le nom des méthodes est défini au niveau de la classe OJBCrudService de la manière suivante :

```
/**
 * Nom de la méthode de création
 */
public static final String CREATE_METHOD = "create";

/**
 * Nom de la méthode de mise à jour
 */
public static final String UPDATE_METHOD = "update";

/**
 * Nom de la méthode de suppression
 */
public static final String DELETE_METHOD = "delete";
```

```

/**
 * Nom de la méthode de suppression avec critères
 */
public static final String DELETE_BY_QUERY_METHOD = "deleteByQuery";

/**
 * Nom de la méthode de recherche avec un objet en retour
 */
public static final String SEARCH_METHOD = "findObject";

/**
 * Nom de la méthode de recherche avec une collection d'objets en retour
 */
public static final String SEARCH_OBJECTS_METHOD = "findObjects";

```

15.1.2. UTILISATION

L'utilisation de ce service se fait de la même manière que les Business Service multifonctions ([cf. 5.8](#)).

Exemple : Méthode de création

objCrudService est le nom du service dans fichier mis dans le fichier **fwk.xml** :

```

<service name="objCrudService" description="Service CRUD OJB"
  className="com.inetpsa.fwk.service.crud.obj.OJBCrudService"/>

```

Code d'appel :

```

// Récupération d'une instance de Service
BusinessService objCRUDService = factory.getService(user, "objCrudService",
OJBCrudService.CREATE_METHOD);

// Construction du PersistentContextKey
IPersistentContextKey key = new OJBPersistentContextKey(ALIAS, getServiceAccount());

Person person = new Person("DUPOND", "Jean");

// Passage des paramètres d'entrée pour la création
objCRUDService.setInput(OJBCrudService.IN_CONTEXT_KEY, key);
objCRUDService.setInput(OJBCrudService.IN_OBJECT, person);

// Exécution du service
objCRUDService.execute();

```

Exemple : Méthode de recherche

```

// Récupération d'une instance de Service
BusinessService objCRUDService = factory.getService(user, "objCrudService",
OJBCrudService.SEARCH_OBJECTS_METHOD);

// Construction du PersistentContextKey
IPersistentContextKey key = new OJBPersistentContextKey(ALIAS, getServiceAccount());

```



```
// Construction des critères
Criteria criteria = new Criteria();
criteria.addLike("lastName", "A%");
criteria.add...

// Passage des paramètres d'entrée pour la création
objCRUDService.setInput(OJBCrudService.IN_CONTEXT_KEY, key);
objCRUDService.setInput(OJBCrudService.IN_CLASS, Person.class);
objCRUDService.setInput(OJBCrudService.IN_CRITERIA, criteria);

// Exécution du service
objCRUDService.execute();

// Récupération de la liste de résultat
Collection result = (Collection) objCRUDService.getOutput(OJBCrudService.OUT_OBJECT);
```

16. LIMITATION DU NOMBRE DE SESSION J2EE

A partir de la version 2.4.0 de LEGO une fonctionnalité a été ajoutée sur les contrôleurs Struts afin de pouvoir limiter le nombre de sessions authentifiées sur l'application.

Cette fonctionnalité permet de fixer un nombre maximum de sessions authentifiées pour lesquelles le contrôleur Struts autorisera l'exécution de requêtes vers des actions, au delà de ce nombre les clients se verront refuser l'accès et il sera possible de les rediriger vers une page d'information.

Une fois la limite atteinte aucune nouvelle session authentifiée ne sera acceptée tant qu'une session déjà autorisée ne tombe en timeout ou n'effectue un « logout ».

Les clients non authentifiés pourront quoi qu'il arrive exécuter les actions non sécurisés de l'application.

Il est à noter que cette limitation a lieu au sein de la JVM courante, en cas d'exécution de l'application sur de multiples JVM le nombre total de session autorisées sera donc égal à la limite fixée multipliée par le nombre de JVM. Ce nombre total pourra ou non être atteint suivant la répartition de la charge.



Il faut bien faire attention à la valeur limite positionnée dans la mesure où une fois la limite atteinte tous les nouveaux clients seront refusés. Et que donc avec une valeur trop faible et/ou une durée de session trop importante associée à une mauvaise utilisation de l'application par les utilisateurs (ne se déconnectent pas par exemple mais ferme le navigateur), **cela peut rapidement amener à refuser tout les nouveaux clients.**

De même il faut prendre en compte le fait que si la gestion BasicAuth est paramétrée pour réauthentifier automatiquement les utilisateurs (comportement par défaut) alors la réauthentification rajoutera directement la session dans la liste des sessions authentifiées, le seul moyen pour cet utilisateur de supprimer de la liste sa session authentifiée sera via timeout.

CONFIGURATION

La configuration liée au mécanisme de limitation est à positionner au niveau du fichier struts-config.xml via des tags set-property sous le tag controller. Plusieurs paramètres sont disponibles :

- **maxAuthSession** : cette propriété permet d'activer la limitation de session si elle est présente et > 0, elle correspond au nombre maximum de sessions authentifiées autorisées.
- **maxAuthForward** : correspond au nom du forward global vers lequel est redirigé un client web lorsque la limite est atteinte, **ce forward doit être paramétré en redirect="true"**.
- **invalidateSessionWhenMaxAuth** (optionnel, par défaut à True): booléen indiquant qu'il est nécessaire d'invalider une session déjà authentifiée qui accèderait au contrôleur alors que la limite est atteinte (ce cas de figure peut arriver par exemple quand l'authentification n'est pas directement gérée par la fonctionnalité de BasicAuth présente sur le RequestProcessor LEGO).

EXEMPLE DE MISE EN PLACE DANS LE FICHIER STRUTS-CONFIG.XML

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts-config PUBLIC "-//PSA idvs, fwk//DTD struts fwk Configuration 1.2//EN"
"http://java.inetpsa.com/lego/dtds/fwk-struts-config_1_2.dtd">

<struts-config>

...

    <!-- Global Forwards -->
    <global-forwards>
        <forward name="limituser" path="/displayLimit.do"></forward>
    </global-forwards>

...

    <!-- Action Mappings -->
    <action-mappings type="com.inetpsa.fwk.struts.config.FWKActionMapping">
        <action path="/displayLimit" type="org.apache.struts.actions.ForwardAction"
parameter="limit.jsp"/>
    </action-mappings>

...

    <controller
        className="com.inetpsa.fwk.struts.action.FWKControllerConfig"
        processorClass="com.inetpsa.fwk.struts.action.FWKRequestProcessor"
        contentType="text/html; charset=UTF-8"
        inputForward="true">

        <set-property property="maxAuthSession" value="20" />
        <set-property property="maxAuthForward" value="limituser" />
        <set-property property="invalidateSessionWhenMaxAuth" value="true" />

    </controller>

...

</struts-config>
```

Dans le cas présent, à partir de 20 sessions authentifiées les nouveaux clients seront redirigés vers la page `limit.jsp`.

17. EXECUTION ASYNCHRONE DE BUSINESSSERVICE

A partir de la version 2.6.0 LEGO propose une fonctionnalité permettant de déclencher de façon asynchrone l'exécution d'un BusinessService. Ce chapitre va décrire le fonctionnement ainsi que la mise en place de cette fonctionnalité qui s'intègre au niveau des couches service et présentation.

17.1. PRESENTATION

Le but premier de cette fonctionnalité est de permettre d'exécuter de façon asynchrone des traitements consommateurs en ressources cpu, et nécessitant un temps d'exécution important non compatible avec une utilisation « web » classique (protocole http qui est synchrone, présence de timeouts contraints au niveau des serveurs d'application et des équipements réseau, ...).

Comme nous le verrons elle offre aussi d'autres possibilités, par exemple :

- D'avoir un premier niveau de contrôle sur les ressources consommées : capacité à gérer un pool d'éléments exécutant les traitements et permettant de limiter le nombre de traitements actifs en parallèle.
- En cas de traitement générant du flux en résultat : possibilité de stocker le flux en mémoire jusqu'à une certaine limite puis sur le système de fichier passé cette limite.
- Possibilité de présenter à l'utilisateur une liste des traitements qui lui sont associés et leur état (en attente de traitement, en cours de traitement, traité, en erreur, ...).

17.2. ENVIRONNEMENT CIBLE

Cette fonctionnalité est prévue pour être utilisée dans les environnements suivants :

- JVM autonome (hors serveur d'application : batch, client lourd, ...).
- Sun Java System Web Server 6.x
- Websphere Application Server 6.1

17.3. FONCTIONNEMENT

Pour gérer cette fonctionnalité LEGO met en place 2 ensembles de mécanismes :

- Un premier au niveau de la couche service qui sera en charge de gérer l'exécution asynchrone du BusinessService ainsi que le cycle de vie et les fonctions associées (vérification de l'état d'un traitement, ...).
- Un second au niveau de la couche présentation pour faciliter l'intégration de cette exécution asynchrone dans un environnement web, notamment via l'utilisation d'une classe d'action struts particulière ainsi que de taglibs générant du code javascript (vérification d'état du traitement, redirection vers la page de résultat une fois le traitement terminé, ...).

17.3.1. ENTITES ET SCHEMA DE FONCTIONNEMENT POUR LA COUCHE SERVICE

Le mécanisme mis en place au niveau de la couche service va faire intervenir plusieurs entités, qui vont chacune avoir un rôle et un cycle de vie particulier :

ASYNCHRONOUSWORK

Correspond au « travail » qui va être exécuté de façon asynchrone, il contient les informations liés à celui-ci :

- Identifiant unique.

- Instance de BusinessService à exécuter de façon asynchrone.
- Etat du travail (en attente de traitement, en cours de traitement, terminé correctement, terminé avec erreur).
- Différents timing vis-à-vis de timeouts (timeout de prise en compte, d'exécution)

IASYNCHRONOUSWORKMANAGER / ASYNCHRONOUSWORKMANAGERIMPL / ASYNCHRONOUSWORKMANAGERFACTORY

Le « gestionnaire de travaux asynchrones » est l'entité en charge de la manipulation des AsynchronousWork (et du cycle de vie associé), elle permet de :

- Créer une instance de travail initialisée (identifiant, service associé).
- Démarrer le workflow d'exécution pour une instance de travail.
- Rechercher à partir d'un id une instance de travail existante (pour vérifier son état, ...).

IWORKER / ABSTRACTWORKER / THREADWORKER

Le « travailleur » est l'entité en charge de l'exécution du « travail », plus précisément du BusinessService qui lui est associé. L'implémentation [ThreadWorker](#) est prévue pour permettre l'exécution via une Thread locale à la JVM courante (obtenue via un [ThreadPoolExecutor](#)).

ISTREAMRESULT / DEFERREDFILESTREAMRESULT / DEFERREDFILESTREAMRESULTFACTORY

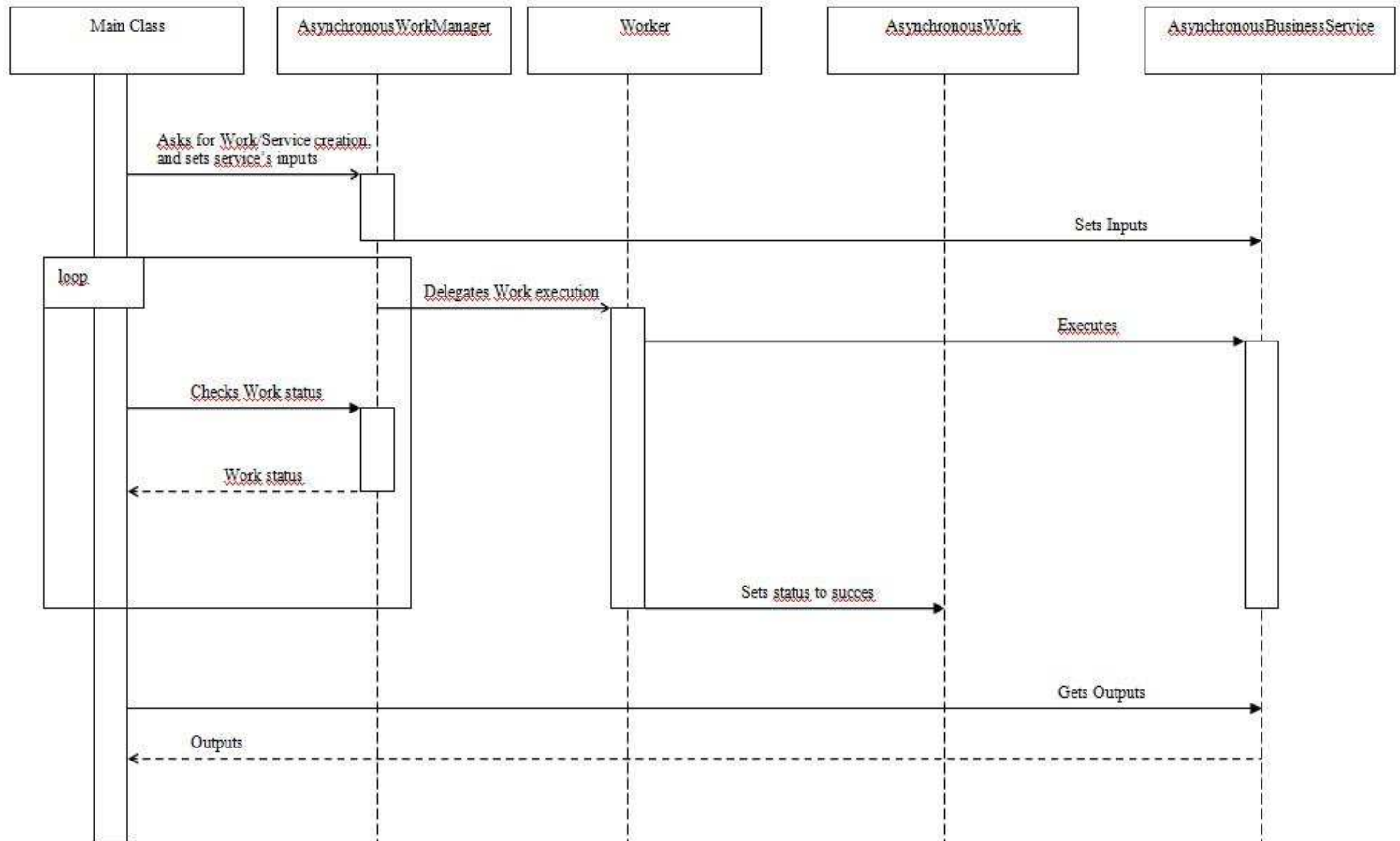
Ensemble de classes permettant de générer des résultats de type flux au niveau des BusinessService exécutés en asynchrone. Ainsi que de stocker ces résultats en mémoire jusqu'à une certaine taille, puis sur le système de fichier une fois cette taille dépassée, en attendant que le résultat soit récupéré.

IASYNCHRONOUSBUSINESSSERVICE / ABSTRACTASYNCHRONOUSBUSINESSSERVICE / ABSTRACTASYNCHRONOUSDISPATCHBUSINESSSERVICE

Classes d'implémentation de BusinessService spécifiques à l'exécution en mode asynchrone.

SCHEMA DESCRIPTIF D'EXECUTION

Le diagramme qui suit détaille un cas simplifié d'exécution asynchrone :



17.3.2. ENTITES ET SCHEMA DE FONCTIONNEMENT POUR LA COUCHE PRESENTATION

Les entités qui vont intervenir au niveau de la couche présentation sont les suivantes :

ABSTRACTASYNCHRONOUSACTION

Classe d'action à hériter afin de pouvoir mettre en place une exécution asynchrone en environnement Struts Lego. Elle va permettre de :

- Faciliter l'intégration de la fonctionnalité en environnement web, notamment via la possibilité de retourner l'état d'un travail à du code appelant (intégration avec les taglibs asynchrones Lego par ex., qui permettent de générer du code javascript pour effectuer des aller/retour client-serveur et prévenir l'utilisateur de l'état d'exécution de son travail asynchrone).
- Masquer l'interaction avec la gestion asynchrone de la couche service (création/gestion des entités « travail ») via un ensemble de méthodes.
- Faciliter la mise en place de résultats de type flux.

ASYNCHRONOUSWEBCONTEXT

Classe contenant les informations liées à un contexte d'exécution d'un travail asynchrone en environnement web :

- L'instance de travail.

- L'url de l'action associée.
- Map contenant les informations supplémentaires projets (par exemple un identifiant fonctionnel du travail asynchrone, ...).

Permet notamment au framework d'afficher à l'utilisateur des informations sur la liste des travaux (leur état : en cours, en attente, ...) et de lui proposer des liens pour venir récupérer/visualiser le résultat une fois l'exécution terminée.

ASYNCHRONOUSWORKTAG / ASYNCHRONOUSWORKFILETAG

Taglibs fournis par Lego et permettant de générer du code html/javascript afin de mettre en place de façon transparente une mécanique permettant :

- D'effectuer des aller/retour entre le navigateur client et le serveur d'application pour vérifier l'état d'un travail.
- Rafraîchir la page/partie de page (ou rediriger vers une autre page, fournir un fichier généré) lorsque le travail est dans un état « terminé » (correctement, avec erreur, en timeout).

SCHEMA DESCRIPTIF DE L'EXECUTION (CAS SIMPLE)

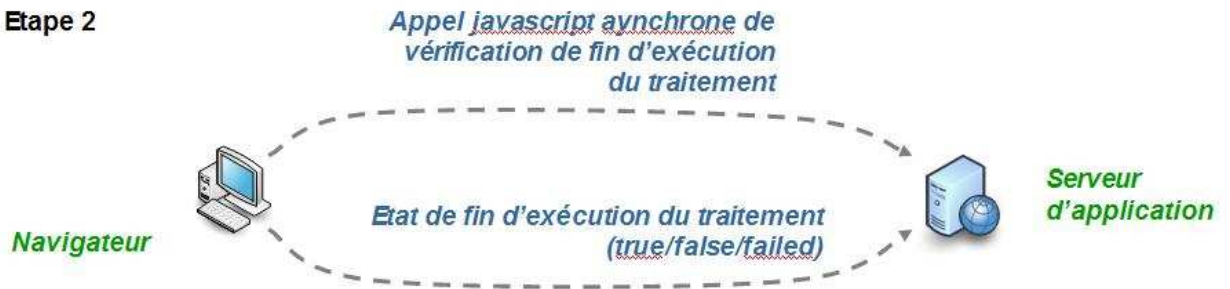
Dans un cas d'utilisation simple (l'utilisateur déclenche un traitement et attend le résultat) la gestion de l'exécution asynchrone au niveau de la couche présentation va se dérouler en 3 étapes décrites ci-dessous :



La première étape consiste en un appel du client web vers l'action struts asynchrone qui va déclencher la création du « travail » et le positionnement de celui-ci au niveau de la file d'attente des traitements asynchrones.

Le serveur retourne au client web la page d'attente qui va mettre en place un code javascript vérifiant à intervalle régulier l'état du traitement (de façon transparente via taglibs lego) :

Etape 2



La seconde étape est transparente pour l'utilisateur (et le développeur), il s'agit d'appels asynchrones entre le client web et le serveur d'application permettant de vérifier la fin d'exécution du traitement. Ces appels sont effectués à intervalle régulier (configurable, avec valeur par défaut 5s) via du code javascript (mis en place par les taglibs lego).

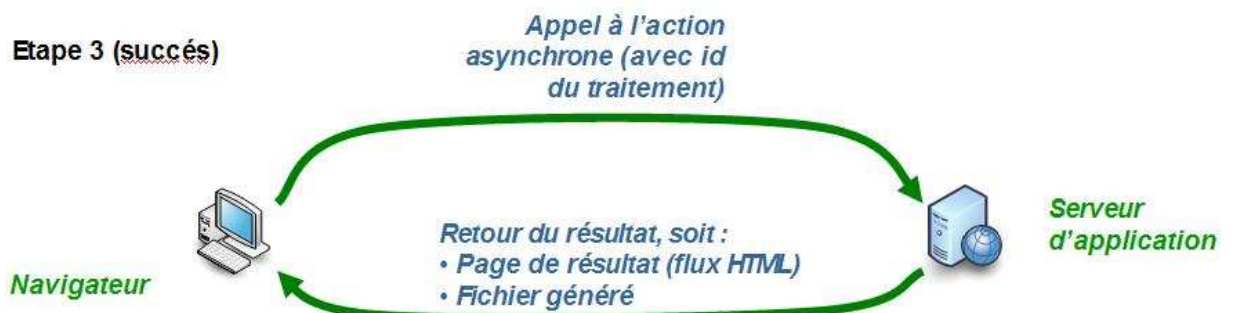
Une fois que le traitement n'est plus en attente d'exécution ni en cours d'exécution, le code javascript effectue un appel non asynchrone à l'action struts (en indiquant l'identifiant unique du traitement) correspondant à l'étape 3 :

Etape 3 (erreur)



Si le « travail » est en état **erreur** alors le serveur retourne la page d'erreur.

Etape 3 (succès)



Si le « travail » est en état **terminé avec succès** alors le serveur retourne suivant le besoin fonctionnel soit :

- La page HTML de résultat.
- Le fichier généré (pdf, image, ...).

SCHEMA DESCRIPTIF DE L'EXECUTION (CAS PLUS COMPLEXE)

Les mécanismes mis en place permettent de gérer des cas d'utilisation plus complexes, notamment celui d'un utilisateur qui déclenche un traitement et n'attend pas le résultat mais navigue ensuite sur d'autres pages de l'application.

Cette gestion nécessitera un peu plus d'écriture de code (action struts , jsp) de la part de l'équipe étude.

La première étape reste la même :

Etape 1



Appel à l'action asynchrone qui déclenche la création du « travail » et sa prise en compte, avec retour de la page d'attente.

L'utilisateur peut ensuite naviguer sur l'application :

Etape 2



Il sera nécessaire de mettre en place un mécanisme qui permettra de vérifier à chaque page demandé s'il existe des « travaux asynchrones » dont l'exécution est terminée.

Pour ce faire il est possible d'écrire une action struts (utilisant la classe [com.inetpsa.fwk.service.asynchronous.AsynchronousWorkManagerFactory](#)) ainsi qu'une JSP qui pourra être incluse dans le bandeau de l'application via un appel `<jsp:include>`, exemple :

```
...
... contenu du bandeau
...
<jsp:include flush="true" page="/AsynchronousInfos.do" />
```

Cette action struts peut récupérer les informations liées à l'état des « travaux » de l'utilisateur (via [AsynchronousWorkManagerFactory](#)) générer des données associées (messages pour l'utilisateur, ...) et positionner le tout dans l'`HttpServletRequest` avant de rediriger vers la JSP :

```

public class AsynchronousInfosAction extends FWKAction {

    public ActionForward doExecute(ActionMapping mapping, ActionForm form,
        HttpServletRequest request, HttpServletResponse response) throws Exception {

        Collection<AsynchronousWork> finishedWorks =
        AsynchronousWorkManagerFactory.getInstance().getFinishedWorks(getUser(request)
        );

        ... // traitement génération des informations à afficher à l'utilisateur

        request.setAttribute("currentWorksInfo", //informations générées);

        return mapping.findForward("result");
    }
}

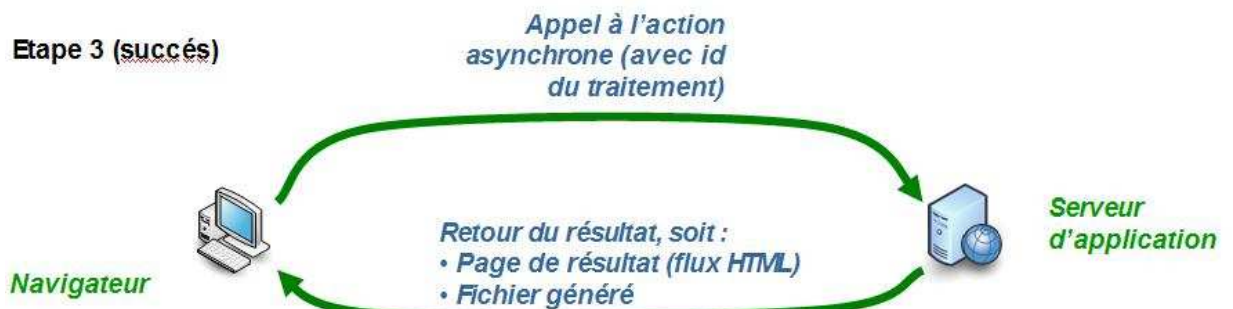
```

La page JSP pourra ensuite utiliser les informations pour afficher les messages à l'utilisateur directement dans le flux courant :

- travail terminé avec succès, avec le lien pour aller vers la page de résultat.
- Travail terminé avec erreur.

Il est aussi possible au niveau de la page JSP de ne pas afficher les informations dans le flux courant mais de mettre en place un code javascript ouvrant une popup qui contiendra les informations citées ci-dessus.

Ce mécanisme permettra à l'utilisateur de déclencher un traitement, de continuer à naviguer sur le reste de l'application et être prévenu de la fin d'exécution de ce traitement en cours de navigation, il pourra alors cliquer sur le lien de résultat qui lui est proposé et passer à l'étape suivante :



Qui est d'afficher le résultat lié au travail.

17.4. MISE EN PLACE AU NIVEAU DE LA COUCHE SERVICE

17.4.1. CONFIGURATION GENERALE

L'utilisation de la fonctionnalité d'exécution asynchrone de BusinessService nécessite la présence dans le fichier fwk.xml d'une configuration générale liée à gestion des travaux. Elle va se positionner via

l'utilisation d'un tag `asynchronousWorkManager` présent sous le tag `services` et contenir toute la paramétrie liée au gestionnaire d'exécution asynchrone :

CONFIGURATION DU COMPORTEMENT PAR DEFAULT

Exemple :

```
...
<services>
  <factory className="com.inetpsa.fwk.service.ServiceFactoryImpl" />

  <asynchronousWorkManager>
    <asynchLimit>10</asynchLimit>
    <executionTimeout>60000</executionTimeout>
    <expirationTimeout>60000</expirationTimeout>
    <tempFileDirectory>C:\temp\testasync</tempFileDirectory>
    <streamResultMaxMemSize>4000</streamResultMaxMemSize>
    <cleanThreadWaitTime>1000</cleanThreadWaitTime>
    <defaultWorkerType>Thread</defaultWorkerType>
    <workerTypes>
      <workerType>
        <name>Thread</name>
        <implementationClass>
com.inetpsa.fwk.service.asynchronous.worker.ThreadWorker
        </implementationClass>
        <parameters></parameters>
      </workerType>
    </workerTypes>
  </asynchronousWorkManager>
...
</services>
```

- **asynchLimit** : le nombre maximum de traitements asynchrones exécutés en parallèle, tous types de traitements confondus.
- **executionTimeout** : temps maximal (en ms) autorisé pour un travail asynchrone avant passage en état *timeout*. *Ce temps est décompté à partir du moment où le travail est créé.*



Dans la mesure où il est impossible en java de stopper l'exécution d'une Thread le travail est passé en état *timeout* mais l'exécution de celui-ci continue. Il faut donc bien faire attention à maîtriser le code exécuté en asynchrone : que ce soit sur le temps de traitement ou **sur les risques de boucle infinie**.

- **expirationTimeout** : temps (en ms) correspondant à la durée pendant laquelle le résultat d'un travail asynchrone est disponible une fois l'exécution de celui-ci terminée. Ce temps est décompté à partir du moment où le travail passe de l'état *en cours d'exécution* à *terminé*. Une fois ce délai dépassé le travail et son résultat ne sont plus disponibles, ce mécanisme permet d'éviter de surcharger la mémoire en limitant la durée de vie d'un résultat.
- **tempFileDirectory** : répertoire temporaire dans lequel peuvent être stockés les résultats de travaux asynchrones de type flux (génération de fichier pdf par exemple) si leur taille est considérée comme incompatible avec un stockage en mémoire.
- **streamResultMaxMemSize** : Taille maximale autorisée en mémoire pour un résultat de travail asynchrone de type flux, une fois cette taille dépassée le flux n'est plus stocké en mémoire mais sur système de fichier dans le répertoire configuré ci-dessus.

- **cleanThreadWaitTime** : Temps d'attente (en ms) entre 2 passages de la Thread chargée de surveiller l'expiration des travaux asynchrones et du nettoyage des listes les référençant. Par défaut positionné à 1s et à ne pas modifier sauf discussion préalable avec l'équipe LEGO.
- **defaultWorkerType** : identifiant du « type d'exécuteur de travail » utilisé par défaut.

CONFIGURATION DES WORKERTYPE

La définition des « types d'exécuteurs de travail » s'effectue via un tag **workerTypes** présent sous le tag **asynchronousWorkManager**.

LEGO fournit une implémentation gérant l'exécution au sein de la JVM courante via un pool de Thread (au travers du ThreadPoolExecutor), la définition correspondante est positionnée dans le fichier fwk.xml avec comme identifiant **Thread** qui est aussi paramétré comme implémentation par défaut :

```
...
<services>
  ...

  <asynchronousWorkManager>
    ...
    <defaultWorkerType>Thread</defaultWorkerType>
    <workerTypes>
      <workerType>
        <name>Thread</name>
        <implementationClass>
com.inetpsa.fwk.service.asynchronous.worker.ThreadWorker
        </implementationClass>
        <parameters></parameters>
      </workerType>
    </workerTypes>
  </asynchronousWorkManager>
...
</services>
```

Actuellement seule l'implémentation basée sur l'utilisation du ThreadPoolExecutor est disponible.

17.4.2. IMPLEMENTATION/DECLARATION DE BUSINESSSERVICE ASYNCHRONE

IMPLEMENTATION

La mise en place d'une implémentation de BusinessService asynchrone passe par la création d'une classe de service héritant de **AbstractAsynchronousBusinessService** et se limite à ce point si le service ne gère pas un retour de type flux, exemple :

```
...
public class TestAsyncService extends AbstractAsynchronousBusinessService {
...
}
```

Le contenu de la classe de BusinessService (méthode doExecute,...) reste identique à une exécution synchrone.

CAS D'UN RETOUR DE TYPE FLUX

Si les données de retour du [BusinessService](#) sont de type flux (génération de fichier pdf par exemple), il est nécessaire d'utiliser une instance de [DeferredFileStreamResult](#) au niveau de la méthode doExecute pour gérer le résultat. En dehors de ce point la mise en place du service reste identique à une exécution synchrone.

Exemple :

```
...
public class TestAsyncService extends AbstractAsynchronousBusinessService {
...
    protected void doExecute() throws FwkException {

        /**
         * On récupère le flux de sortie "asynchrone" (mémoire ou
         * système de fichier suivant la configuration
         * et la taille résultat).
         */
        DeferredFileStreamResult result = getDeferredFileStreamResult();

        try {
            // On utilise result comme OutputStream dans lequel
            // générer le flux
            ...
            ...
            ...
        } catch (IOException e) {
            ...
        } finally {
            result.close();
        }

        setOutput("RESULT", result);
    }
}
```

On obtient une instance de [DeferredFileStreamResult](#) que l'on utilise comme OutputStream dans lequel générer le flux de sortie, on ferme ce [DeferredFileStreamResult](#) et on le positionne comme résultat de sortie du [BusinessService](#).

DECLARATION

La déclaration s'effectue de la façon habituelle au niveau du fichier fwk.xml :

```
<service className="com.inetpsa.fwk.test.services.TestAsyncService"
    name="asynchSearch">
</service>
```

Il est possible de surcharger pour un service donné 2 paramètres de la configuration générale :

`asynchLimit` et `streamResultMaxMemSize`, la valeur alors positionnée ne sera prise en compte que pour l'exécution de ce service :

```
<service className="com.inetpsa.fwk.test.services.TestAsyncService"
  name="testAsynch">
  <configurations>
    <entry>
      <key>asynchLimit</key>
      <value>2</value>
    </entry>
    <entry>
      <key>streamResultMaxMemSize</key>
      <value>1024</value>
    </entry>
  </configurations>
</service>
```

Comme défini ci-dessus le service asynchrone `testAsynch` ne pourra avoir que 2 exécutions en cours en parallèle et les résultats de sortie de type flux seront stockés sur le système de fichier à partir d'une taille de 1024 ko, alors que pour les autres services la configuration générale s'appliquera.

17.5. MISE EN PLACE AU NIVEAU DE LA COUCHE PRESENTATION

17.5.1. IMPLEMENTATION / DECLARATION D'ACTION STRUTS

IMPLEMENTATION

La mise en place au niveau action struts nécessite d'hériter d'une classe spécifique :

`com.inetpsa.fwk.struts.actions.asynchronous.AbstractAsynchronousAction`

Et d'implémenter les méthodes abstraites associées :

- `getAsynchronousBusinessServiceName` : retourne l'identifiant (tel que défini dans le fichier `fwk.xml`) du `BusinessService` à exécuter en asynchrone via l'action. Exemple :

```
protected String getAsynchronousBusinessServiceName() {
    return "asynchSearch";
}
```

- `getAsynchronousBusinessServiceFunctionName` : retourne l'identifiant de la fonction à exécuter sur le `BusinessService` en cas de service de type `DispatchBusinessService`, ou `null` sinon.
- `prepareExecute` : méthode permettant de préparer le `BusinessService` asynchrone à son exécution, on peut au niveau de cette méthode positionner les données d'entrée du `BusinessService` (à partir de données présentes sur l'`ActionForm` par exemple), il est aussi

possible de positionner sur le contexte asynchrone des données propres au projet et caractérisant le « travail asynchrone » (par exemple un identifiant ou un type fonctionnel qui pourra ensuite être utilisé sur les pages affichant l'état du « travail »). Exemple :

```
public void prepareExecute(ActionMapping mapping, ActionForm form,
    HttpServletRequest request, HttpServletResponse response,
    BusinessService service, AsynchronousWebContext context) throws
FwkException {
    SearchForm currentForm = (SearchForm)form;
    service.setInput("searchName", currentForm.getName());

    context.getInformations().put("type", "Traitement de Recherche");
}
```

- **prepareResult** : méthode permettant de récupérer le résultat d'exécution du **BusinessService** et d'effectuer le nécessaire (par ex. le positionner au niveau de l'**HttpServletRequest**). Cette méthode doit retourner une instance d'**ActionForward** correspondant à la page de résultat à retourner au client web (ou *null* s'il n'y en a pas par exemple dans le cas d'une génération de flux). Exemple :

```
public ActionForward prepareResult(ActionMapping mapping, ActionForm form,
    HttpServletRequest request, HttpServletResponse response,
    BusinessService service) throws FwkException {
    request.setAttribute("resultat", service.getOutput("liste"));

    return mapping.findForward("success");
}
```

- **onWait** : méthode permettant de retourner une instance d'**ActionForward** correspondant à la page d'attente à afficher au client web tant que l'exécution du **BusinessService** n'est pas terminée. Exemple :

```
public ActionForward onWait(ActionMapping mapping, HttpServletRequest arg1,
    HttpServletResponse arg2) throws FwkException {
    return mapping.findForward("wait");
}
```

- **onError** : méthode permettant d'effectuer la gestion nécessaire en cas d'échec de l'exécution du **BusinessService** . Cette méthode doit retourner une instance d'**ActionForward** correspondant à la page d'erreur à afficher au client web. Exemple :

```

public ActionForward onError(ActionMapping mapping, ActionForm form,
    HttpServletRequest request, HttpServletResponse response,
    FwkException exception) throws FwkException {

    ActionMessages errors = new ActionMessages();

    if (exception instanceof TimeoutException) {
        errors.add(ActionMessages.GLOBAL_MESSAGE, new
    ActionMessage("error.timeout"));
    } else {
        errors.add(ActionMessages.GLOBAL_MESSAGE, new
    ActionMessage("error.failed"));
    }

    saveErrors(request, errors);

    return mapping.findForward("error");
}

```

DECLARATION

La déclaration s'effectue de la façon habituelle au niveau du fichier struts-config.xml :

```

<action path="/asynchSearchByName"
    type="com.inetpsa.fwk.test.AsynchSearchAction">
    <forward name="wait" path="/waitSearch.do" redirect="false"/>
    <forward name="success" path="/resultSearch.do" redirect="false"/>
    <forward name="error" path="/errorSearch.do" redirect="false"/>
</action>

```

17.5.2. PREPARATION A LA MISE EN PLACE DES JSP

Les mécanismes intervenant au niveau de la couche présentation sont basés en partie sur l'utilisation de javascript et nécessitent de charger des fichiers js au niveau des pages utilisant les taglibs LEGO.

Les fichiers js à utiliser sont disponibles au niveau de la distribution lego ([site lego2](#) > rubrique téléchargement/Distribution) dans le répertoire `conf/asynch/js`.

Ces fichiers devront être chargés par le navigateur au niveau des pages utilisant les taglib asynchrones lego.

Si tiles est utilisé cela peut facilement être fait en créant une définition spécialisée pour les pages « asynchrones », exemple :


```

<!-- Main page layout used as a root for other page definitions -->
<definition name="mainLayout" path="/mainLayout.jsp">
    <put name="title" value="title" />
    <put name="header" value="/header.jsp" />
    <put name="menu" value="/menu.jsp" />
    <put name="body" value="" direct="true" />
    <putList name="scripts"></putList>
    <putList name="themes"></putList>
</definition>

<!-- Layout for asynch pages, which includes specific JS files -->
<definition name="asynchLayout" extends="mainLayout">
    <putList name="scripts">
        <add value="js/prototype.js" />
        <add value="js/fwkasynch.js" />
    </putList>
</definition>

...

<definition name="cas1.result" extends="asynchLayout">
    <put name="title" value="cas1.title" />
    <put name="body" value="/cas1_result.jsp" />
</definition>

...

```

Avec au niveau de la page mainLayout.jsp :

```

...
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<logic:notEmpty name="scripts">
    <logic:iterate id="script" name="scripts">
        <script type="text/JavaScript" src="<bean:write
name="script"/>"></script>
    </logic:iterate>
</logic:notEmpty>
...
</head>
...

```

17.5.3. ECRITURE JSP ET UTILISATION DES TAGLIBS

Il est possible d'utiliser tiles avec le mécanisme asynchrone mis en place par LEGO au niveau de la couche présentation mais pour des raisons de simplicité les exemples qui suivent utilisent directement des jsp en résultat des actions struts.

Lego propose 2 taglibs afin de faciliter la mise en place des jsp et des mécanismes permettant d'effectuer les aller/retour entre le client web et le serveur pour vérifier l'état d'exécution d'un « travail asynchrone » et afficher les informations nécessaires à l'utilisateur (message d'attente, résultat d'exécution, message d'erreur).

- **asynchronousWork** : permet de mettre en place de façon transparente le mécanisme javascript permettant d'effectuer les aller/retour avec le serveur pour vérifier l'état

d'exécution d'un « travail asynchrone », ainsi que d'afficher le contenu de son body sur condition vis-à-vis de l'état d'exécution. Une fois l'exécution du travail terminé le code javascript mis en place peut recharger tout ou partie de la page afin d'afficher le résultat.

- **asynchronousWorkFile** : permet d'effectuer le même traitement que **asynchronousWork** en y ajoutant des spécificités liées à un résultat de type flux (image, pdf, ...).

En cas d'exécution asynchrone l'information affichable à l'utilisateur est découpée en 3 parties :

- Message d'attente.
- Page de résultat.
- Message d'erreur en cas de problème d'exécution.

La façon de mettre en place les tags va dépendre du découpage choisi dans le positionnement des informations citées ci-dessus, nous allons étudier les différents cas possibles :

CAS 1 : TOUTES LES INFORMATIONS DANS LA MEME PAGE

Il est possible d'utiliser le taglib **asynchronousWork** plusieurs fois dans la même page JSP et de faire gérer les 3 informations (message d'attente, résultat, message d'erreur) dans une seule page.

Exemple d'une page nommée **cas1.jsp** :

```
...
<%@ taglib uri="/WEB-INF/fwktag.tld" prefix="fwk"%>
...
<fwk:asynchronousWork condition="completed">
...
    <layout:pager maxPageItems="2">
        <layout:collection name="resultat" id="row" indexId="index">
            <layout:collectionItem property="firstname" width="20%" />
...
        </layout:collection>
    </layout:pager>
...
</fwk:asynchronousWork>

<fwk:asynchronousWork condition="failed">
    <html:errors />
</fwk:asynchronousWork>

<fwk:asynchronousWork>
    
</fwk:asynchronousWork>
...
```

Dans le cas où on utilise plusieurs fois le tag **asynchronousWork** dans une même page avec des conditions différentes l'ordre d'apparition est important il faut d'abord positionner le tag en condition *travail complété*, puis celui en condition *travail en erreur*, et enfin le tag en condition *traitement non terminé*.

Pour l'exemple ci-dessus lors du premier appel à la page après exécution de l'action asynchrone seul le body du dernier tag (sans condition explicitée et donc correspondant à un état *traitement non terminé*) sera évalué, les 2 autres conditions : *completed* et *failed* n'étant pas atteintes.

Ce tag permet donc d'afficher un message d'attente à l'utilisateur (ici un gif animé par exemple) et de mettre en place le mécanisme javascript effectuant les a/r avec le serveur pour vérifier l'état d'exécution. Si l'état d'exécution change (terminé avec résultat ou alors en erreur) alors le

navigateur sera redirigé vers l'action struts qui réaffichera cette page et cette fois-ci le body de l'un des 2 premiers tags sera pris en compte.

Les définitions d'action Struts associées seraient :

```
<action path="/cas1"
        type="com.inetpsa.fwk.test.AsynchCas1Action">
    <forward name="result" path="/resultCas1.do" redirect="false"/>
</action>

<action path="/resultCas1" parameter="/cas1.jsp"
        type="org.apache.struts.actions.ForwardAction" />
```

Il est important de bien faire attention à ce que le forward au niveau de l'action struts asynchrone (ici avec le path /cas1) ne redirige pas directement vers la jsp mais vers une action positionnée en frontal de la jsp.

La gestion des forward au niveau de la classe d'action struts serait :

```
public ActionForward prepareResult(ActionMapping mapping, ActionForm form,
    HttpServletRequest request, HttpServletResponse response,
    BusinessService service) throws FwkException {
    ...
    return mapping.findForward("result");
}

public ActionForward onWait(ActionMapping mapping, HttpServletRequest arg1,
    HttpServletResponse arg2) throws FwkException {
    return mapping.findForward("result");
}

public ActionForward onError(ActionMapping mapping, ActionForm form,
    HttpServletRequest request, HttpServletResponse response,
    FwkException exception) throws FwkException {
    ...
    return mapping.findForward("result");
}
```

CAS 2 : CHAQUE INFORMATION DANS UNE PAGE SEPARÉE (RECHARGEMENT COMPLET DES PAGES)

A l'inverse du cas 1 il est possible de n'utiliser le taglib `asynchronousWork` qu'une fois sur la page d'attente, et ainsi avoir 3 pages différentes

Exemple :

Page nommée `cas2_wait.jsp` :

```

...
<%@ taglib uri="/WEB-INF/fwktag.tld" prefix="fwk"%>
...

<fwk:asynchronousWork>
    
</fwk:asynchronousWork>

...

```

Page nommée `cas2_error.jsp` :

```

...

<html:errors />

...

```

Page nommée `cas2_result.jsp` :

```

...
<layout:pager maxPageItems="2">
    <layout:collection name="resultat" id="row" indexId="index">
        <layout:collectionItem property="firstname" width="20%" />
    </layout:collection>
</layout:pager>
...

```

Les définitions d'action Struts associées seraient :

```

<action path="/cas2"
        type="com.inetpsa.fwk.test.AsynchCas2Action">
    <forward name="wait" path="/waitCas2.do" redirect="false"/>
    <forward name="error" path="/errorCas2.do" redirect="false"/>
    <forward name="result" path="/resultCas2.do" redirect="false"/>
</action>

<action path="/waitCas2" parameter="/cas2_wait.jsp"
        type="org.apache.struts.actions.ForwardAction" />

<action path="/errorCas2" parameter="/cas2_error.jsp"
        type="org.apache.struts.actions.ForwardAction" />

<action path="/resultCas2" parameter="/cas2_result.jsp"
        type="org.apache.struts.actions.ForwardAction" />

```

Il est important de bien faire attention à ce que les forwards au niveau de l'action struts asynchrone (ici avec le path `/cas2`) ne redirige pas directement vers la jsp mais vers une action positionnée en frontal de la jsp.

La gestion des forwards au niveau de la classe d'action struts serait :

```
public ActionForward prepareResult(ActionMapping mapping, ActionForm form,
    HttpServletRequest request, HttpServletResponse response,
    BusinessService service) throws FwkException {
...
    return mapping.findForward("result");
}

public ActionForward onWait(ActionMapping mapping, HttpServletRequest arg1,
    HttpServletResponse arg2) throws FwkException {
    return mapping.findForward("wait");
}

public ActionForward onError(ActionMapping mapping, ActionForm form,
    HttpServletRequest request, HttpServletResponse response,
    FwkException exception) throws FwkException {
...
    return mapping.findForward("error");
}
```

CAS 3 : CHAQUE INFORMATION DANS UNE PAGE SEPARÉE (RECHARGEMENT PARTIEL DES PAGES)

Il est possible comme dans le cas 2 de séparer les informations dans 3 pages différentes, et de mettre en place un rechargement partiel de la page pour prévenir l'utilisateur que le traitement a changé d'état.

Exemple :

Page nommée [cas3_wait.jsp](#) :

```

...
<%@ taglib uri="/WEB-INF/fwktag.tld" prefix="fwk"%>
...

<fwk:asynchronousWork asynchRefreshResult="true">
    
</fwk:asynchronousWork>

...

```

Cette page met en place tout le layout de l'application (menus, bandeau, ...) et affiche un message d'attente à l'utilisateur via le tag `asynchronousWork`. L'attribut `asynchRefreshResult` positionné à `true` permettra de ne pas rafraichir tout le contenu de la page une fois le travail terminé mais de remplacer le contenu html du body du tag par la page de résultat.

Page nommée `cas3_error.jsp` :

```
<html:errors />
```

Cette page ne doit pas mettre en place de layout (menus, bandeau) dans la mesure où son contenu remplacera celui qui était dans le body du tag `asynchronousWork` de la page `cas3_wait.jsp`.

Page nommée `cas2_result.jsp` :

```

<layout:pager maxPageItems="2">
    <layout:collection name="resultat" id="row" indexId="index">
        <layout:collectionItem property="firstname" width="20%" />
    ...
    </layout:collection>
</layout:pager>

```

Cette page ne doit pas mettre en place de layout (menus, bandeau) dans la mesure où son contenu remplacera celui qui était dans le body du tag `asynchronousWork` de la page `cas3_wait.jsp`.

Les définitions d'action Struts associées seraient :

```

<action path="/cas3"
    type="com.inetpsa.fwk.test.AsynchCas3Action">
    <forward name="wait" path="/waitCas3.do" redirect="false"/>
    <forward name="error" path="/errorCas3.do" redirect="false"/>
    <forward name="result" path="/resultCas3.do" redirect="false"/>
</action>

<action path="/waitCas3" parameter="/cas3_wait.jsp"
    type="org.apache.struts.actions.ForwardAction" />

<action path="/errorCas3" parameter="/cas3_error.jsp"
    type="org.apache.struts.actions.ForwardAction" />

<action path="/resultCas3" parameter="/cas3_result.jsp"
    type="org.apache.struts.actions.ForwardAction" />

```

Il est important de bien faire attention à ce que les forwards au niveau de l'action struts asynchrone (ici avec le path /cas3) ne redirige pas directement vers la jsp mais vers une action positionnée en frontal de la jsp.

La gestion des forwards au niveau de la classe d'action struts serait :

```
public ActionForward prepareResult(ActionMapping mapping, ActionForm form,
    HttpServletRequest request, HttpServletResponse response,
    BusinessService service) throws FwkException {
...
    return mapping.findForward("result");
}

public ActionForward onWait(ActionMapping mapping, HttpServletRequest arg1,
    HttpServletResponse arg2) throws FwkException {
    return mapping.findForward("wait");
}

public ActionForward onError(ActionMapping mapping, ActionForm form,
    HttpServletRequest request, HttpServletResponse response,
    FwkException exception) throws FwkException {
...
    return mapping.findForward("error");
}
```

CAS 4 : GENERATION DE FLUX

La mise en place des JSP et du taglib pour une génération de flux s'approche du cas 3 vu précédemment avec une différence sur la gestion de la page de résultat.

Exemple :

Page nommée [casFichier_wait.jsp](#) :

```

...
<%@ taglib uri="/WEB-INF/fwktag.tld" prefix="fwk"%>
...

<fwk:asynchronousWorkFile resultUrl="resultLoadFile.do">
    
</fwk:asynchronousWorkFile>

...

```

Cette page met en place tout le layout de l'application (menus, bandeau, ...) et affiche un message d'attente à l'utilisateur via le tag `asynchronousWorkFile`. L'attribut `resultUrl` s'il est présent correspond à une action générant un flux HTML qui sera utilisé pour remplacer le contenu html du body du tag après que le flux résultat ait été téléchargé par le client web.

Page nommée `casFichier_error.jsp` :

```
<html:errors />
```

Cette page ne doit pas mettre en place de layout (menus, bandeau) dans la mesure où son contenu remplacera celui qui était dans le body du tag `asynchronousWorkFile` de la page `casFichier_wait.jsp`. Elle sera appelée en cas de problème d'exécution.

Page nommée `casFichier_result.jsp` :

```
fichier préparé.
```

Cette page ne doit pas mettre en place de layout (menus, bandeau) dans la mesure où son contenu remplacera celui qui était dans le body du tag `asynchronousWorkFile` de la page `casFichier_wait.jsp`.

Les définitions d'action Struts associées seraient :

```

<action path="/casFichier"
        type="com.inetpsa.fwk.test.AsynchCas3Action">
    <forward name="wait" path="/waitCas3.do" redirect="false"/>
    <forward name="error" path="/errorCas3.do" redirect="false"/>
    <forward name="result" path="/resultCas3.do" redirect="false"/>
</action>

<action path="/waitCas3" parameter="/cas3_wait.jsp"
        type="org.apache.struts.actions.ForwardAction" />

<action path="/errorCas3" parameter="/cas3_error.jsp"
        type="org.apache.struts.actions.ForwardAction" />

<action path="/resultCas3" parameter="/cas3_result.jsp"
        type="org.apache.struts.actions.ForwardAction" />

```

Il est important de bien faire attention à ce que les forwards au niveau de l'action struts asynchrone (ici avec le path /cas3) ne redirige pas directement vers la jsp mais vers une action positionnée en frontal de la jsp.

La gestion des forwards au niveau de la classe d'action struts serait :


```
public ActionForward prepareResult(ActionMapping mapping, ActionForm form,
HttpServletRequest request, HttpServletResponse response,
    BusinessService service) throws FwkException {
...
    return mapping.findForward("result");
}

public ActionForward onWait(ActionMapping mapping, HttpServletRequest arg1,
HttpServletResponse arg2) throws FwkException {
    return mapping.findForward("wait");
}

public ActionForward onError(ActionMapping mapping, ActionForm form,
HttpServletRequest request, HttpServletResponse response,
    FwkException exception) throws FwkException {
...
    return mapping.findForward("error");
}
```

18. MIGRATION

Lors de certaines montées de version LEGO il peut arriver que des composants Open Source embarqués montent aussi de version et qu'il soit alors potentiellement nécessaire d'effectuer une migration portant sur leur mise en place, exemple :

- A partir de LEGO 2.2.0 la version d'OJB utilisée n'est plus la 1.0.3 mais la 1.0.4
- A partir de LEGO 2.3.0 Struts 1.2.9 remplace la version 1.1 et Struts-Layout 1.2 remplace la version 1.1

Ce chapitre regroupe les points importants à prendre en compte lorsqu'une telle montée de version est à effectuer.

18.1. PASSAGE OJB 1.0.0 A 1.0.3

La release note OJB 1.0.3 est disponible [ici](#).

Il y a eu des changements sur la gestion du cache (plus de détails sont disponibles [ici](#)). Notamment la notion de *CacheFilter* qui n'est plus disponible telle quelle, et l'attribut *cacheable* n'est plus à utiliser sur le *ClassDescriptor* : un tag *object-cache* fait son apparition et peut-être positionné au niveau du *ClassDescriptor*.

18.2. PASSAGE OJB 1.0.3 A 1.0.4

La release note OJB 1.0.4 est disponible [ici](#).

Lors d'une montée de version d'OJB vers la 1.0.4 il est nécessaire de suivre plusieurs étapes :

- Remplacement de l'ancienne dtd par la nouvelle.
- Remplacement du fichier *OJB.properties* par celui correspondant à la version 1.0.4, et remise à jour du fichier par rapport aux modifications potentiellement effectuées (type de *ConnectionFactoryClass*, ...).
- Remplacer le fichier *repository_internal.xml* par celui provenant de la distribution LEGO.

18.3. PASSAGE STRUTS 1.1 A 1.2.9

La release note qui concerne Struts 1.2.9 est disponible [ici](#), celle concernant la 1.2.4 (version qui fait le pont entre 1.1 et 1.2.x) est disponible [ici](#).

Le passage de struts 1.1 à 1.2.9 nécessite des modifications portant sur plusieurs points :

18.3.1. MODIFICATIONS DIRECTEMENT LIEES A STRUTS

- Supprimer le fichier *fwk-struts-config_1_1.dtd* du répertoire *web-inf* (le nouveau fichier *fwk-struts-config_1_2.dtd* est présent dans le jar *fwk-strutsX.X.X.jar*) puis éditer le fichier *struts-config.xml* pour pointer vers la nouvelle version :

```
<!DOCTYPE struts-config PUBLIC "-//PSA idvs, fwk//DTD struts fwk
Configuration 1.2//EN" "http://java.inetpsa.com/lego/dtds/fwk-struts-
config_1_2.dtd">
```

Il est à noter que l'attribut debug du tag controller n'existe plus.

- Comme précisé dans la release note struts des fonctions indiquées comme étant *deprecated* en 1.1 ont été supprimées, notamment les attributs **name** et **type** du taglib **html:form** ce qui implique de les supprimer, et si du code javascript utilisait la valeur de l'attribut name pour accéder au form alors il faut modifier ce code.

18.3.2. MODIFICATIONS LIEES AU PLUGIN VALIDATOR

- Remplacer le fichier validator-rules.xml par le nouveau (présent dans la distribution LEGO).
- Editer le fichier validation.xml pour ajouter/mettre à jour le Doctype vers la bonne dtd :

```
<!DOCTYPE form-validation PUBLIC
"-//Apache Software Foundation//DTD Commons Validator Rules
Configuration 1.1.3//EN"
"http://jakarta.apache.org/commons/dtds/validator_1_1_3.dtd">
```

- S'assurer que tous les tags **html:javascript** présents dans les pages jsps pointent bien vers une définition de form dans le fichier validation.xml (cela ne posait pas de problème précédemment si ce n'était pas le cas).

18.3.3. MODIFICATIONS LIEES AU PLUGIN TILES

Editer le fichier tiles-defs.xml et s'assurer que l'on ait :

```
<!DOCTYPE tiles-definitions PUBLIC
"-//Apache Software Foundation//DTD Tiles Configuration 1.1//EN"
"http://jakarta.apache.org/struts/dtds/tiles-config_1_1.dtd">
```

18.4. PASSAGE STRUTS-LAYOUT 1.1 A 1.2

- Si le fichier Struts-Layout.properties par défaut n'est pas utilisé (ou surchargé), alors il faut partir du contenu du nouveau fichier et effectuer les modifications précédemment apportées : on peut par exemple désormais avoir des images de tris différentes suivant le tri sélectionné,
- Remplacer les ressources statiques liées à Struts-Layout par les nouvelles (répertoires config et images).

19. ANNEXES

19.1. FICHER WEB.XML

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc./DTD Web Application 2.2/EN"
"http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<web-app id="WebApp">
    <display-name>Suivi des interventions</display-name>
    <servlet>
        <servlet-name>action</servlet-name>
        <servlet-class>com.inetpsa.fwk.struts.action.FWKActionServlet</servlet-class>
        <init-param>
            <param-name>config</param-name>
            <param-value>/WEB-INF/struts-config.xml</param-value>
        </init-param>
        <init-param>
            <param-name>debug</param-name>
            <param-value>2</param-value>
        </init-param>
        <init-param>
            <param-name>detail</param-name>
            <param-value>2</param-value>
        </init-param>
        <init-param>
            <param-name>validate</param-name>
            <param-value>>false</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>action</servlet-name>
        <url-pattern>*.do</url-pattern>
    </servlet-mapping>
    <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
    </welcome-file-list>
    <taglib>
        <taglib-uri>/WEB-INF/struts-bean.tld</taglib-uri>
        <taglib-location>/WEB-INF/struts-bean.tld</taglib-location>
    </taglib>
    <taglib>
        <taglib-uri>/WEB-INF/struts-html.tld</taglib-uri>
        <taglib-location>/WEB-INF/struts-html.tld</taglib-location>
    </taglib>
    <taglib>
        <taglib-uri>/WEB-INF/struts-logic.tld</taglib-uri>
        <taglib-location>/WEB-INF/struts-logic.tld</taglib-location>
    </taglib>
    <taglib>
        <taglib-uri>/WEB-INF/struts-template.tld</taglib-uri>
        <taglib-location>/WEB-INF/struts-template.tld</taglib-location>
    </taglib>
    <taglib>
        <taglib-uri>/WEB-INF/struts-nested.tld</taglib-uri>
        <taglib-location>/WEB-INF/struts-nested.tld</taglib-location>
    </taglib>

```

```

</taglib>
<taglib>
    <taglib-uri>/WEB-INF/struts-tiles.tld</taglib-uri>
    <taglib-location>/WEB-INF/struts-tiles.tld</taglib-location>
</taglib>
<taglib>
    <taglib-uri>/WEB-INF/displaytag.tld</taglib-uri>
    <taglib-location>/WEB-INF/displaytag.tld</taglib-location>
</taglib>
<taglib>
    <taglib-uri>http://fwk.org</taglib-uri>
    <taglib-location>/WEB-INF/fwktag.tld</taglib-location>
</taglib>
</webapp>

```

19.2. FICHER STRUTS-CONFIG.XML

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts-config PUBLIC "-//PSA idvs, fwk//DTD struts fwk Configuration 1.2//EN"
"http://java.inetpsa.com/lego/dtds/fwk-struts-config_1_2.dtd">

<struts-config>

    <!-- Form Beans -->
    <form-beans>
        <form-bean name="loginForm" type="com.inetpsa.wja.ui.form.LoginForm"></form-bean>
        <form-bean name="rechercheInterventionsForm"
type="com.inetpsa.wja.ui.form.RechercheInterventionsForm"></form-bean>
        <form-bean name="createInterventionForm"
type="com.inetpsa.wja.ui.form.CreateInterventionForm"></form-bean>
        <form-bean name="I18nActionForm"
type="com.inetpsa.fwk.i18n.admin.forms.I18nActionForm"></form-bean>
        <form-bean name="I18nLocaleUpdateDeleteForm"
type="com.inetpsa.fwk.i18n.admin.forms.I18nLocaleForm"></form-bean>
        <form-bean name="I18nLocaleCreateForm"
type="com.inetpsa.fwk.i18n.admin.forms.I18nLocaleForm"></form-bean>
        <form-bean name="I18nImportExportActionForm"
type="com.inetpsa.fwk.i18n.admin.forms.I18nImportExportActionForm"></form-bean>
    </form-beans>

    <!-- Global Exceptions -->
    <global-exceptions>
        <exception handler="com.inetpsa.wja.security.SecurityExceptionHandler" key=""
type="com.inetpsa.fwk.exception.SecurityException"></exception>
        <exception path="error" key=""
type="com.inetpsa.fwk.exception.TechnicalException"></exception>
    </global-exceptions>

    <!-- Global Forwards -->
    <global-forwards>
        <forward name="accueil" path="/accueil.do"></forward>
        <forward name="login" path="/login.do"></forward>
        <forward name="logout" path="/logout.do"></forward>
        <forward name="search" path="/filtreInterventions.do"></forward>
        <forward name="create" path="/afficherCreateIntForm.do"></forward>
        <forward name="i18n" path="/I18nAdmin.do"></forward>
        <forward name="i18nLocales" path="/I18nAdminLocale.do"></forward>
        <forward name="i18nImportExport" path="/I18nImportExportAction.do"></forward>
        <forward name="timeout" path="timeout"/>
    </global-forwards>

```

```

</global-forwards>

<!-- Action Mappings -->
<action-mappings type="com.inetpsa.fwk.struts.config.FWKActionMapping">
    <action path="/accueil" parameter="accueil"
type="org.apache.struts.actions.ForwardAction"></action>
    <action path="/login" name="loginForm" scope="request"
type="com.inetpsa.wja.ui.action.PreparerLoginAction" validate="false">
        <forward name="login" path="login"></forward>
    </action>
    <action path="/authentifier" name="loginForm" validate="true" input="relogin"
className="com.inetpsa.fwk.struts.action.FWKLoginActionMapping" scope="request"
type="com.inetpsa.fwk.security.actions.LoginAction">
        <set-property property="loginInput" value="userid" />
        <set-property property="passwordInput" value="password" />
        <forward name="loginfinalsuccess" path="securedPage"></forward>
        <forward name="relogin" path="/login.do"></forward>
    </action>
    <action path="/logout" type="com.inetpsa.fwk.security.actions.LogoutAction" >
        <forward name="logoutsuccess" path="/accueil.do" redirect="true"></forward>
    </action>

    <action path="/filtreInterventions" name="rechercheInterventionsForm"
scope="request" validate="false"
type="com.inetpsa.wja.ui.action.PreparerRechercheInterventionsAction"
profils="redacteur,lecteur,administrateur">
        <forward name="sucess" path="searchInt"></forward>
    </action>
    <action path="/searchInterventions" name="rechercheInterventionsForm"
scope="request" type="com.inetpsa.wja.ui.action.RechercherInterventionsAction"
input="valErrors" profils="redacteur,lecteur,administrateur">
        <forward name="sucess" path="listeInt"></forward>
        <forward name="valErrors" path="/filtreInterventions.do"></forward>
    </action>
    <action path="/listInterventions" parameter="listeInt"
type="org.apache.struts.actions.ForwardAction"
profils="redacteur,lecteur,administrateur"></action>

    <action path="/afficherCreateIntForm" name="createInterventionForm" scope="request"
validate="false" type="com.inetpsa.wja.ui.action.PreparerCreateInterventionAction"
profils="redacteur,administrateur">
        <forward name="sucess" path="createInt"></forward>
    </action>
    <action path="/createIntervention" name="createInterventionForm" scope="request"
validate="true" input="valErrors" type="com.inetpsa.wja.ui.action.CreateInterventionAction"
profils="redacteur,administrateur">
        <forward name="sucess" path="/filtreInterventions.do"
redirect="true"></forward>
        <forward name="valErrors" path="/afficherCreateIntForm.do"></forward>
        <forward name="failure" path="/afficherCreateIntForm.do"></forward>
    </action>

    <action path="/I18nAdmin" name="I18nActionForm"
scope="session" type="com.inetpsa.fwk.i18n.admin.actions.I18nAction"
profils="administrateur">
        <forward name="success" path="i18n"></forward>
    </action>

    <action path="/I18nAdminUpdateFilter" name="I18nActionForm"
scope="session" type="com.inetpsa.fwk.i18n.admin.actions.I18nUpdateFilterAction"
profils="administrateur">
        <forward name="success" path="/I18nAdmin.do"></forward>
    </action>

```

```

        <action path="/I18nAdminUpdateCreateDelete" name="I18nActionForm"
            scope="session"
            type="com.inetpsa.fwk.i18n.admin.actions.I18nUpdateCreateDeleteAction"
            profils="administrateur">
            <forward name="success" path="/I18nAdmin.do"></forward>
            <forward name="erreurs" path="/I18nAdmin.do"></forward>
        </action>

        <action path="/I18nAdminLocale" name="I18nLocaleForm"
            scope="request" validate="false"
            type="com.inetpsa.fwk.i18n.admin.actions.I18NAfficheLocaleAction" profils="administrateur">
            <forward name="success" path="i18nlocales"></forward>
        </action>
        <action path="/I18nAdminCreateLocale" validate="true" name="I18nLocaleCreateForm"
            scope="request" input="erreurs"
            type="com.inetpsa.fwk.i18n.admin.actions.I18NCreateLocaleAction" profils="administrateur">
            <forward name="success" path="/I18nAdminLocale.do"></forward>
            <forward name="erreurs" path="/I18nAdminLocale.do"></forward>
        </action>
        <action path="/I18nAdminUpdateDeleteLocale" validate="true"
            name="I18nLocaleUpdateDeleteForm"
            scope="request" input="erreurs"
            type="com.inetpsa.fwk.i18n.admin.actions.I18NUpdateDeleteLocaleAction"
            profils="administrateur">
            <forward name="success" path="/I18nAdminLocale.do"></forward>
            <forward name="erreurs" path="/I18nAdminLocale.do"></forward>
        </action>

        <action path="/I18nImportExportAction" name="I18nImportExportActionForm"
            scope="request" type="com.inetpsa.fwk.i18n.admin.actions.I18nImportExportAction"
            profils="administrateur">
            <forward name="success" path="i18nimportexport"></forward>
        </action>
        <action path="/I18nExportAction" name="I18nImportExportActionForm"
            scope="request" type="com.inetpsa.fwk.i18n.admin.actions.I18nExportAction"
            profils="administrateur">
        </action>
        <action path="/erreur" validate="false" forward="/jsp/error.jsp"/>

        <action path="/Test" type="com.inetpsa.wja.ui.action.TestAction" >
        </action>

    </action-mappings>

    <!-- Controller -->
    <controller className="com.inetpsa.fwk.struts.action.FWKControllerConfig"
        processorClass="com.inetpsa.fwk.struts.action.FWKtilesRequestProcessor" contentType="text/html;
        charset=UTF-8" inputForward="true">
        <set-property property="loginAction" value="login" />
        <set-property property="timeoutForward" value="timeout" />
    </controller>

    <!-- Message Resources -->
    <message-resources parameter=""
        factory="com.inetpsa.fwk.struts.messages.FWKMessageResourcesFactory" null="true" />

    <!-- Plugins -->
    <plug-in className="org.apache.struts.validator.ValidatorPlugIn">
        <set-property property="pathnames" value="/WEB-INF/validator-rules.xml,/WEB-
        INF/validation.xml" />
    </plug-in>

```

```

<plug-in className="org.apache.struts.tiles.TilesPlugin">
  <set-property property="definitions-config" value="/WEB-INF/tiles-defs.xml" />
  <set-property property="definitions-parser-validate" value="true" />
</plug-in>

</struts-config>

```

19.3. FICHER VALIDATION.XML

```

<!DOCTYPE form-validation PUBLIC
"-//Apache Software Foundation//DTD Commons Validator Rules Configuration 1.1.3//EN"
"http://jakarta.apache.org/commons/dtds/validator_1_1_3.dtd">

<form-validation>
  <formset>
    <constant>
      <constant-name>patternDate</constant-name>
      <constant-value>dd/MM/yyyy</constant-value>
    </constant>
    <form name="loginForm">
      <field property="userid" depends="required">
        <arg0 name="required" key="login.uid"/>
      </field>
      <field property="password" depends="required">
        <arg0 name="required" key="login.pwd"/>
      </field>
    </form>
    <form name="rechercheInterventionsForm">
      <field property="codeIntervention" depends="integer">
        <arg0 name="maxlength" key="codeInt"/>
      </field>
      <field property="client" depends="maxlength">
        <arg0 name="maxlength" key="client"/>
        <arg1 name="maxlength" key="7" resource="false"/>
        <var var-name="maxlength" var-value="7"/>
      </field>
    </form>
    <form name="createInterventionForm">
      <field property="difficulte" depends="required">
        <arg0 key="difficulte"/>
      </field>
      <field property="statutIncident" depends="required">
        <arg0 key="statutInc"/>
      </field>
      <field property="statutIntervention" depends="required">
        <arg0 key="statutInt"/>
      </field>
      <field property="codeProjet" depends="required">
        <arg0 key="codePrj"/>
      </field>
      <field property="typeIntervention" depends="required">
        <arg0 key="typeInt"/>
      </field>
      <field property="client" depends="required">
        <arg0 key="client"/>
      </field>
      <field property="expertID" depends="required">
        <arg0 key="expert"/>
      </field>
    </form>
  </formset>
</form-validation>

```



```

</field>
<field property="dateDdeInt" depends="required, date">
  <arg0 key="dtDdeInt" />
  <var>
    <var-name>datePatternStrict</var-name>
    <var-value>${patternDate}</var-value>
  </var>
</field>
<field property="dateReceptCCT" depends="date">
  <arg0 key="dtRcptCCT" />
  <var>
    <var-name>datePatternStrict</var-name>
    <var-value>${patternDate}</var-value>
  </var>
</field>
<field property="dateFinPrevue" depends="date">
  <arg0 key="dtFinPrevue" />
  <var>
    <var-name>datePatternStrict</var-name>
    <var-value>${patternDate}</var-value>
  </var>
</field>
<field property="dateFinReelle" depends="date">
  <arg0 key="dtFinReelle" />
  <var>
    <var-name>datePatternStrict</var-name>
    <var-value>${patternDate}</var-value>
  </var>
</field>
<field property="tmpsResolution" depends="integer">
  <arg0 key="tmpsResol" />
</field>
</form>

<form name="I18nLocaleCreateForm">
  <field property="newLocale" depends="required,mask">
    <arg0 name="required" key="locale.newlocale" />
    <msg name="mask" key="locale.badnewlocale" />
    <var>
      <var-name>mask</var-name>
      <var-value>[a-z]{2}_[A-Z]{2}</var-value>
    </var>
  </field>
  <field property="copyFromLocale" depends="required,mask">
    <msg name="mask" key="locale.badfromlocale" />
    <var>
      <var-name>mask</var-name>
      <var-value>[a-z]{2}_[A-Z]{2}</var-value>
    </var>
  </field>
  <field property="parentLocale" depends="mask">
    <msg name="mask" key="locale.badparentlocale" />
    <var>
      <var-name>mask</var-name>
      <var-value>[a-z]{2}_[A-Z]{2}</var-value>
    </var>
  </field>
</form>
<form name="/I18nAdminUpdateDeleteLocale">
</form>
<form name="/I18nExportAction">

```

```

        </form>
    </formset>
</form-validation>

```

19.4. FICHIER TILES-DEFS.XML

```

<!DOCTYPE tiles-definitions PUBLIC
    "-//Apache Software Foundation//DTD Tiles Configuration//EN"
    "http://jakarta.apache.org/struts/dtds/tiles-config_1_1.dtd">

<!-- Definitions for Tiles documentation -->
<tiles-definitions>

    <!-- Main page layout used as a root for other page definitions -->
    <definition name="mainLayout" path="/mainLayout.jsp">
        <put name="title" value="title" />
        <put name="header" value="/header.jsp" />
        <put name="menu" value="" direct="true" />
        <put name="body" value="" direct="true" />
        <putList name="scripts">
            <add value="scripts/site.js" />
        </putList>
    </definition>

    <definition name="unsecuredPage" extends="mainLayout" path="/nomenulayout.jsp">
    </definition>

    <definition name="securedPage" extends="mainLayout">
        <put name="menu" value="/menu.jsp" />
    </definition>

    <definition name="accueil" extends="unsecuredPage">
        <put name="title" value="accueil.title" />
        <put name="body" value="/accueil.jsp" />
    </definition>

    <definition name="login" extends="unsecuredPage">
        <put name="title" value="login.title" />
        <put name="body" value="/login.jsp" />
    </definition>

    <definition name="timeout" extends="unsecuredPage">
        <put name="title" value="title" />
        <put name="body" value="/timeout.jsp" />
    </definition>

    <definition name="searchInt" extends="securedPage">
        <put name="title" value="rchInt.title" />
        <put name="body" value="/criteresRecherche.jsp" />
    </definition>

    <definition name="listeInt" extends="securedPage">
        <put name="title" value="lstInt.title" />
        <put name="body" value="/resultatsRecherche.jsp" />
        <putList name="themes">
            <add value="themes/screen.css" />
        </putList>
    </definition>

```

```
<definition name="createInt" extends="securedPage">
  <put name="title" value="createInt.title" />
  <put name="body" value="/createIntervention.jsp" />
</definition>

<definition name="i18n" extends="securedPage">
  <put name="body" value="/i18n/i18n.jsp" />
  <putList name="themes">
    <add value="themes/screen.css" />
    <add value="themes/i18n.css" />
  </putList>
</definition>

<definition name="i18nlocales" extends="securedPage">
  <put name="body" value="/i18n/i18n_locales.jsp" />
  <putList name="themes">
    <add value="themes/screen.css" />
    <add value="themes/i18n.css" />
  </putList>
</definition>

<definition name="i18nimportexport" extends="securedPage">
  <put name="body" value="/i18n/i18nimportexport.jsp" />
  <putList name="themes">
    <add value="themes/screen.css" />
    <add value="themes/i18n.css" />
  </putList>
</definition>

<definition name="error" extends="unsecuredPage">
  <put name="title" value="title" />
  <put name="body" value="/error.jsp" />
</definition>
</tiles-definitions>
```

19.5. FICHER FWK.XML

```
<?xml version="1.0" encoding="UTF-8"?>
<fwk>
  <i18n>
    <class>com.inetpsa.fwk.i18n.service.db.ServiceResourcesDB</class>
    <default_locale>fr_FR</default_locale>
  </i18n>
  <logs className="org.apache.commons.logging.impl.SimpleLog" pattern="com.inetpsa"
level="info" />
  <security>
    <directory>
      <server>annuaire.inetpsa.com</server>
      <port>389</port>
      <userid />
      <password />
      <pool>
        <cnxmax>100</cnxmax>
        <cnxmin>50</cnxmin>
        <cnxttl>120000</cnxttl>
        <timewaitmax>5000</timewaitmax>
        <active>true</active>
      </pool>
    </directory>
  </security>
</fwk>
```

```

</directory>
<user className="com.inetpsa.fwk.security.beans.User" />
<profiles>
    <profile name="administrateur" description="Profil Administrateur"
className="com.inetpsa.wja.security.profil.ProfilAdministrateur">
        <condition>
            <ldapGroups>
                <ldapGroup>ROLE.FWK.ADM</ldapGroup>
            </ldapGroups>
            <userMethods />
        </condition>
    </profile>
    <profile name="lecteur" description="Profil Lecteur"
className="com.inetpsa.wja.security.profil.ProfilLecteur">
        <condition>
            <ldapGroups>
                <ldapGroup>ROLE.FWK.LECTEUR</ldapGroup>
            </ldapGroups>
            <userMethods />
        </condition>
    </profile>
    <profile name="redacteur" description="Profil Redacteur"
className="com.inetpsa.wja.security.profil.ProfilExpert">
        <condition>
            <ldapGroups>
                <ldapGroup>ROLE.FWK.REDACTEUR</ldapGroup>
            </ldapGroups>
            <userMethods />
        </condition>
    </profile>
</profiles>
<roles>
    <role name="administrateur" description="">
        <authorized_profiles>
            <profile>administrateur</profile>
        </authorized_profiles>
        <non_authorized_profiles />
    </role>
    <role name="redacteurLecteur" description="">
        <authorized_profiles>
            <profile>redacteur</profile>
            <profile>lecteur</profile>
        </authorized_profiles>
        <non_authorized_profiles />
    </role>
</roles>
</security>
<services>
    <factory className="com.inetpsa.fwk.service.ServiceFactoryImpl" />
    <service name="creerIntervention" description="création d'une intervention"
className="com.inetpsa.wja.service.CreerInterventionService">
        <condition>
            <role>administrateur</role>
            <profiles />
            <userMethods />
        </condition>
        <condition>
            <profiles>
                <profile>redacteur</profile>
            </profiles>
            <userMethods />
        </condition>
    </service>
</services>

```

```

</service>
<service name="listerElementsReference" description="recuperation d'une liste
d'elements de reference" className="com.inetpsa.wja.service.ListerElementsReferenceService">
  <condition>
    <profiles>
      <profile>DefaultProfile</profile>
    </profiles>
    <userMethods />
  </condition>
</service>
<service name="listerExperts" description="recuperation de la liste des experts"
className="com.inetpsa.wja.service.ListerExpertsService">
  <condition>
    <profiles>
      <profile>DefaultProfile</profile>
    </profiles>
    <userMethods />
  </condition>
</service>
<service name="rechercherInterventions" description="recherche multi criteres des
interventions" className="com.inetpsa.wja.service.RechercherInterventionsService">
  <condition>
    <profiles>
      <profile>DefaultProfile</profile>
    </profiles>
    <userMethods />
  </condition>
</service>
</services>
</fwk>

```

19.6. FICHER DATASOURCE.JRXML

```

<?xml version="1.0" encoding="UTF-8"?>
<!-- Created using JasperAssistant (http://www.jasperassistant.com) -->
<!DOCTYPE jasperReport PUBLIC "-//JasperReports//DTD Report Design//EN"
"http://jasperreports.sourceforge.net/dtds/jasperreport.dtd">
<jasperReport name="DataSourceReport" pageWidth="595" pageHeight="842" columnWidth="515"
leftMargin="40" rightMargin="40" topMargin="50" bottomMargin="50">
  <reportFont name="Arial_Normal" isDefault="true" fontName="Arial" size="12" isBold="false"
isItalic="false" isUnderline="false" isStrikeThrough="false" pdfFontName="Helvetica"
pdfEncoding="Cp1252" isPdfEmbedded="false"/>
  <reportFont name="Arial_Bold" isDefault="false" fontName="Arial" size="12" isBold="true"
isItalic="false" isUnderline="false" isStrikeThrough="false" pdfFontName="Helvetica-Bold"
pdfEncoding="Cp1252" isPdfEmbedded="false"/>
  <reportFont name="Arial_Italic" isDefault="false" fontName="Arial" size="12"
isBold="false" isItalic="true" isUnderline="false" isStrikeThrough="false"
pdfFontName="Helvetica-Oblique" pdfEncoding="Cp1252" isPdfEmbedded="false"/>
  <parameter name="ReportTitle" class="java.lang.String">
  </parameter>
  <parameter name="DataFile" class="java.lang.String">
  </parameter>
  <field name="id" class="java.lang.Integer">
  </field>
  <field name="name" class="java.lang.String">
  </field>
  <field name="street" class="java.lang.String">
  </field>
  <field name="the_city" class="java.lang.String">
  </field>

```

```

        <fieldDescription>me.me.city</fieldDescription>
    </field>
    <variable      name="CityNumber"      class="java.lang.Integer"      incrementType="Group"
incrementGroup="CityGroup"  calculation="Count">
        <variableExpression><![CDATA[Boolean.TRUE]]></variableExpression>
    </variable>
    <group name="CityGroup" minHeightToStartNewPage="60">
        <groupExpression><![CDATA[$F{the_city}]]></groupExpression>
        <groupHeader>
            <band height="20">
                <textField evaluationTime="Group" evaluationGroup="CityGroup">
                    <reportElement mode="Opaque" x="0" y="5" width="515" height="15"
backcolor="#c0c0c0"/>
                    <box leftPadding="10" bottomBorder="1Point"/>
                    <textElement>
                        <font reportFont="Arial_Bold"/>
                    </textElement>
                    <textFieldExpression class="java.lang.String"><![CDATA[" " +
String.valueOf($V{CityNumber}) + ". " + String.valueOf($F{the_city}]]></textFieldExpression>
                </textField>
            </band>
        </groupHeader>
        <groupFooter>
            <band height="20">
                <staticText>
                    <reportElement x="400" y="1" width="60" height="15"/>
                    <textElement textAlignment="Right">
                        <font reportFont="Arial_Bold"/>
                    </textElement>
                    <text><![CDATA[Count :]]></text>
                </staticText>
                <textField>
                    <reportElement x="460" y="1" width="30" height="15"/>
                    <textElement textAlignment="Right">
                        <font reportFont="Arial_Bold"/>
                    </textElement>
                    <textFieldExpression
class="java.lang.Integer"><![CDATA[$V{CityGroup_COUNT}]]></textFieldExpression>
                </textField>
            </band>
        </groupFooter>
    </group>
    <title>
        <band height="70">
            <line>
                <reportElement x="0" y="0" width="515" height="1"/>
                <graphicElement/>
            </line>
            <textField isBlankWhenNull="true">
                <reportElement x="0" y="10" width="515" height="30"/>
                <textElement textAlignment="Center">
                    <font reportFont="Arial_Normal" size="22"/>
                </textElement>
                <textFieldExpression
class="java.lang.String"><![CDATA[$P{ReportTitle}]]></textFieldExpression>
            </textField>
            <textField isBlankWhenNull="true">
                <reportElement x="0" y="40" width="515" height="20"/>
                <textElement textAlignment="Center">
                    <font reportFont="Arial_Normal" size="14"/>
                </textElement>
            </textField>
        </band>
    </title>

```

```

        <textFieldExpression
class="java.lang.String"><![CDATA[${P{DataFile}}]></textFieldExpression>
        </textField>
    </band>
</title>
<pageHeader>
    <band height="20">
        <rectangle>
            <reportElement x="0" y="5" width="515" height="15" forecolor="#333333"
backcolor="#333333"/>
            <graphicElement/>
        </rectangle>
        <staticText>
            <reportElement mode="Opaque" x="0" y="5" width="55" height="15"
forecolor="ffffff" backcolor="#333333"/>
            <textElement textAlignment="Center">
                <font reportFont="Arial_Bold"/>
            </textElement>
            <text><![CDATA[ID]]></text>
        </staticText>
        <staticText>
            <reportElement mode="Opaque" x="55" y="5" width="205" height="15"
forecolor="ffffff" backcolor="#333333"/>
            <textElement>
                <font reportFont="Arial_Bold"/>
            </textElement>
            <text><![CDATA[Name]]></text>
        </staticText>
        <staticText>
            <reportElement mode="Opaque" x="260" y="5" width="255" height="15"
forecolor="ffffff" backcolor="#333333"/>
            <textElement>
                <font reportFont="Arial_Bold"/>
            </textElement>
            <text><![CDATA[Street]]></text>
        </staticText>
    </band>
</pageHeader>
<detail>
    <band height="15">
        <textField>
            <reportElement x="0" y="0" width="50" height="15"/>
            <box leftBorder="Thin" bottomBorder="Thin" leftPadding="10"
rightPadding="10"/>
            <textElement textAlignment="Right"/>
            <textFieldExpression
class="java.lang.Integer"><![CDATA[${F{id}}]></textFieldExpression>
        </textField>
        <textField isStretchWithOverflow="true">
            <reportElement positionType="Float" x="50" y="0" width="200"
height="15"/>
            <box leftBorder="Thin" bottomBorder="Thin" leftPadding="10"
rightPadding="10"/>
            <textElement/>
            <textFieldExpression
class="java.lang.String"><![CDATA[${F{name}}]></textFieldExpression>
        </textField>
        <textField isStretchWithOverflow="true">
            <reportElement positionType="Float" x="250" y="0" width="265"
height="15"/>
            <box leftBorder="Thin" bottomBorder="Thin" rightBorder="Thin"
leftPadding="10" rightPadding="10"/>
            <textElement/>

```

```

        <textFieldExpression
class="java.lang.String"><![CDATA[{$F{street}}]]></textFieldExpression>
    </textField>
</band>
</detail>
<pageFooter>
    <band height="40">
        <line>
            <reportElement x="0" y="10" width="515" height="1"/>
            <graphicElement/>
        </line>
        <textField>
            <reportElement x="200" y="20" width="80" height="15"/>
            <textElement textAlignment="Right"/>
            <textFieldExpression class="java.lang.String"><![CDATA["Page " +
String.valueOf($V{PAGE_NUMBER}) + " of" ]]></textFieldExpression>
        </textField>
        <textField evaluationTime="Report">
            <reportElement x="280" y="20" width="75" height="15"/>
            <textElement/>
            <textFieldExpression class="java.lang.String"><![CDATA[" " +
String.valueOf($V{PAGE_NUMBER}) ]]></textFieldExpression>
        </textField>
    </band>
</pageFooter>
<lastPageFooter>
    <band height="60">
        <textField>
            <reportElement x="0" y="10" width="515" height="15"/>
            <textElement textAlignment="Center"/>
            <textFieldExpression class="java.lang.String"><![CDATA["There were " +
String.valueOf($V{REPORT_COUNT}) +
" address records on this report." ]]></textFieldExpression>
        </textField>
        <line>
            <reportElement x="0" y="30" width="515" height="1"/>
            <graphicElement/>
        </line>
        <textField>
            <reportElement x="200" y="40" width="80" height="15"/>
            <textElement textAlignment="Right"/>
            <textFieldExpression class="java.lang.String"><![CDATA["Page " +
String.valueOf($V{PAGE_NUMBER}) + " of" ]]></textFieldExpression>
        </textField>
        <textField evaluationTime="Report">
            <reportElement x="280" y="40" width="75" height="15"/>
            <textElement/>
            <textFieldExpression class="java.lang.String"><![CDATA[" " +
String.valueOf($V{PAGE_NUMBER}) ]]></textFieldExpression>
        </textField>
    </band>
</lastPageFooter>
</jasperReport>

```