Valraiso
8 place Colbert
69001 Lyon
Tel : +33 427 193 163

Documentation de la séquence d'appels web services à mettre en oeuvre pour procéder à la validation d'un panier de cours de ski ESF

Date	Version	Auteur	Commentaire
29 septembre 2009	Première Version	Cédric Marcone	
22 octobre 2010	Version 2	Cédric Marcone	Mise à jour de la section 12.2.2
13 décembre 2010	Version 3	Cédric Marcone	Mise à jour des URL des WS OSP
19 janvier 2011	Version 4	Cédric Marcone	Documentation de la vente en compte : Ajout des section 13, 13.1 et 13.2

La vente en ligne de cours de ski des ESF s'appuie sur un système générique de traitement des paniers, des commandes, des clients et des transactions.

Ce système s'appelle OSP (Online Selling Plateform). Il expose ses fonctionnalités via des web services que nous allons utiliser pour procéder à la gestion du paiement.

Ce document est organiser autour de blocs de pseudo-code décrivant, étape par étape, ce qu'il est nécessaire de mettre en oeuvre pour réaliser le paiement.

Ce document part du postulat qu'un panier est déjà constitué via les web services ESF de la vente en ligne.

Nous allons utiliser les méthodes de cinq web services :

Web service ESF permettant la lecture de la commande client

	Web Service	Référence pseudo-code
Service	CommandeWebService	esfCommandeWebService
WSDL	http://www.vente-en-ligne-esf.com/esfvelws/services/Commande?wsdl	

Web service OSP permettant la gestion de sessions OSP

	Web Service	Référence pseudo-code
Service	SessionWebService	ospSessionWebService
WSDL	https://transaction.valraiso.biz/osp/services/Session?wsdl	

Web service OSP permettant la gestion de paniers OSP

	Web Service	Référence pseudo-code
Service	BasketWebService	ospBasketWebService
WSDL	https://transaction.valraiso.biz/osp/services/Basket?wsdl	

Web service OSP permettant la gestion d'utilisateurs OSP

	Web Service	Référence pseudo-code
Service	UserWebService	ospUserWebService
WSDL	https://transaction.valraiso.biz/osp/services/User?wsdl	

Web service OSP permettant la gestion de boutiques OSP

	Web Service	Référence pseudo-code
Service	StoreWebService	ospStoreWebService
WSDL	https://transaction.valraiso.biz/osp/services/Store?wsdl	

Web service OSP permettant la gestion de commandes OSP

	Web Service	Référence pseudo-code
Service	OrderWebService	ospOrderWebService
WSDL	https://transaction.valraiso.biz/osp/services/Order?wsdl	

Préambule

Dans les exemples ci-après, on utilise une variable nommée «esfSessionId».

Cette variable correspond à l'identifiant de session ESF obtenu lors de la construction du panier via les API exposées sous forme de web service par la plateforme de vente en ligne des ESF.

```
String esfSessionId; // id de session ESF obtenu lors de la création d'une commande
```

1. Enregistrement de la commande dans la base de donnée ESF

La première étape est primordiale. Elle consiste en l'enregistrement de la commande dans la base de donnée ESF.

Cet enregistrement se fait en invoquant la méthode saveCommande du CommandeWebService ESF.

```
String commandeid = esfCommandeWebService.saveCommande (esfSessionId);
```

2. Ouverture de la session OSP

Une session doit être établie sur la plateforme OSP avant de procéder à la synchronisation de la commande ESF.

Une session OSP est obtenu dans le contexte d'un marchand.

Chaque ESF étant un marchand à part entière, on doit particulariser l'ouverture de la session avec un login et mot de passe.

Le login est la concaténation de la chaîne «Esf» et du code école.
Le mot de passe est le même pour chaque école : «m0n1t3uR».

Le code école est l'identifiant de l'objet de classe Ecole récupéré à partir du CommandeWebService ESF en appelant la méthode getEcole.

L'ouverture de la session OSP se fait en invoquant la méthode open du SessionWebService OSP.

```
Ecole  ecole         = esfCommandeWebService.getEcole (esfSessionId);
String ecoleid       = ecole.getId();
```

```
String storeid       = "Esf" + ecoleid;
String storePassword = "m0n1t3uR";

String ospSessionId  = ospSessionWebService.open (storeid, storePassword);
```

3. Transfert du client ESF vers OSP

Le client correspondant au client ESF doit être créé ou maintenu à jour dans la base OSP.

- On doit tester son existence côté OSP
- S'il existe, on doit l'authentifier, puis mettre à jour ses données
- S'il n'existe pas, on doit le créer

3.1 Récupération du client ESF

On récupère le client ESF qui s'est signé lors du processus de construction de la commande avec les web services ESF.

L'objet de classe Client est renvoyé par la méthode getClient du CommandeWebService ESF.

```
Client client = esfCommandeWebService.getClient (esfSessionId);
```

3.2 Test de l'existence du client dans la base OSP

Il est tout d'abord nécessaire de tester l'existence de l'utilisateur côté OSP afin de déterminer la conduite à tenir.

La vérification s'effectue sur la propriété «login» de l'objet de classe Client récupéré à l'étape précédente.

Pour accéder à cette propriété, on utilise l'accesseur getLogin.

On invoque ensuite la méthode exist sur le UserWebService OSP. Cette méthode renvoie un booléen indiquant si le client existe déjà côté OSP.

```
String  clientLogin    = client.getLogin();
boolean ospUsersExists = ospUserWebService.exist (ospSessionId, clientLogin);
```

3.3 Authentification d'un user existant

Si, à l'étape précédente, on a déterminé que le client existait déjà, il devient nécessaire de l'authentifier dans le contexte de la session OSP ouverte sur la plateforme OSP.

On a déjà obtenu le login du client à l'étape précédente, il faut maintenant obtenir son mot de passe.

Il est accessible via la propriété «password» de l'objet de classe Client récupéré à l'étape précédente.

Pour accéder à cette propriété, on utilise l'accesseur getPassword.

On invoque la méthode signIn sur le UserWebService OSP, en passant en paramètre le login et le mot de passe du client.

On doit ensuite récupérer l'objet de classe User correspondant au client ESF dans la plateforme OSP.

On invoque ensuite la méthode getConnectedUser sur le UserWebService OSP. Cette méthode renvoie l'objet User correspondant.

```
User ospUser = null;

if (ospUsersExists)
{
      String clientPassword = client.getPassword();

      ospUserWebService.signIn (ospSessionId, clientLogin, clientPassword);

      ospUser = ospUserWebService.getConnectedUser (ospSessionId);
}
```

3.4 Récupération des informations client à transmettre à OSP

Dans tous les cas, que le user existe déjà ou non, on va avoir besoin de certaines informations client issue de la commande ESF.

Ces informations vont servir, soit à la création de l'utilisateur, soit à la mise à jour de ses données personnelles.

Les données nécessaires sont : le nom, le prénom, l'e-mail, le champ adresse1, le champ adresse2, le code postal, la ville, le pays, son numéro de téléphone mobile, son numéro de téléphone fixe ainsi que son numéro de téléphone en station.

Ces données sont des propriétés de l'objet de classe Client obtenu à l'étape 3.1. Afin d'obtenir ces données, on utilise les accesseurs correspondant dans l'instance (getNom, getPrenom, getEmail, getAdresse1, getAdress2, getCodePostal, getVille, getPaysLibelle, getTelephoneMobile, getTelephoneStation et getTelephoneFixe).

```
String nom               = client.getNom();
String prenom            = client.getPrenom();
String email             = client.getEmail();
String adresse1          = client.getAdresse1();
String adresse2          = client.getAdresse2();
String codePostal        = client.getCodePostal();
String ville             = client.getVille();
String telephoneDomicile = client.getTelephoneDomicile();
String telephoneStation  = client.getTelephoneStation();
String telephoneMobile   = client.getTelephoneMobile();
String paysLibelle       = client.getPays().getNom();
```

3.5 Construction des attributs de client

Un User OSP contient des données intrinsèques et des données variables (liste d'attributs, ou couple nom / valeur).

A cette étape nous allons créer la liste de tous les attributs variables de l'utilisateur.

Un attribut doit être instancié pour chaque propriété variable et son nom et sa valeur doivent être renseignés.

Chaque nouvel attribut doit être ajouté à la collection des attributs des attributs de l'utilisateur.

Les attributs devant être valorisés dans le contexte de la vente en ligne ESF sont : adresse1, adresse2, codePostal, ville, pays, telephoneDomicile, telephoneStation et telephoneMobile.

Les valeurs correspondantes ont été extraites à l'étape précédente.

```
List<Attribute> clientAttributes = new ArrayList<Attribute>();

clientAttributes.add (new Attribute ("adresse1",          adresse1));
clientAttributes.add (new Attribute ("adresse2",          adresse2));
clientAttributes.add (new Attribute ("codepostal",        codePostal));
clientAttributes.add (new Attribute ("ville",             ville));
clientAttributes.add (new Attribute ("telephoneDomicile", telephoneDomicile));
clientAttributes.add (new Attribute ("telephoneStation",  telephoneStation));
clientAttributes.add (new Attribute ("telephoneMobile",   telephoneMobile));
clientAttributes.add (new Attribute ("pays",              paysLibelle));
```

3.6 Soumission de l'utilisateur à OSP

Une fois que les attributs de l'utilisateur OSP sont prêt, il nous reste à soumettre cet utilisateur en prenant en compte les deux cas possibles :

- la création si l'utilisateur n'existe pas
- ou sa mise à jour s'il est déjà présent du côté OSP.

La création de l'utilisateur est conditionnée par la nullité de la variable ospUser.

En effet, à l'étape 3.3 nous avons récupéré l'objet de classe User correspondant à notre utilisateur.

Si celui était null, il est nécessaire de recréer un utilisateur, sinon, on le met à jour.

3.6.1 Création de l'utilisateur OSP

La création commence par l'instanciation d'un nouvel objet de classe User.

Il faut ensuite valoriser ses propriétés intrinsèques via les cinq mutateurs correspondant : setLastname, setFirstname, setLogin, setPassword et setEmail.

Puis, on invoque la méthode subscribe sur le UserWebService OSP, en passant en paramètre le user nouvellement créé et la liste d'attributs de client créée à l'étape 3.5.

On doit ensuite récupérer l'utilsateur OSP nouvellement créé. Pour ce faire, on appelle la méthode getConnectedUser du UserWebService OSP.

```
if (ospUser == null)
{
      ospUser = new User();

      ospUser.setLastname  (nom);
      ospUser.setFirstname (prenom);
      ospUser.setLogin     (clientLogin);
      ospUser.setPassword  (clientPassword);
      ospUser.setEmail     (email);

      ospUserWebService.subscribe (ospSessionId, ospUser, clientAttributes);

      ospUser = ospUserWebService.getConnectedUser (ospSessionId);
}
```

3.6.2 Mise à jour de l'utilisateur OSP

La mise à jour consiste tout d'abord à valoriser les propriétés intrinsèques de l'utilisateur récupérer à l'étape 3.3.

Ces propriétés sont valorisées par les mutateurs setLastname, setFirstname et setEmail.

On invoque la méthode modifyUser sur le UserWebService OSP, en passant en paramètre le user et la liste d'attributs de client créée à l'étape 3.5.

```
else
{
      ospUser.setLastname  (nom);
      ospUser.setFirstname (prenom);
      ospUser.setEmail     (email);

      ospUserWebService.modifyUser (ospSessionId, ospUser, clientAttributes);
}
```

4. Préparation du panier OSP

On va travailler avec un panier OSP, on doit le préparer en fonction de son statut.Ce statut peut évoluer au fur et à mesure des interactions utilisateur, qui peut refuser de payer, ou encore modifier sa commande.

Lors de ces interactions, le statut du panier associé à la commande change. En fonction de ce statut, il faut préparer le panier à recevoir les infos de commande.

On récupère tout d'abord le panier associé à la session OSP en invoquant la méthode getBasket du BasketWebService OSP.

On récupère ensuite le statut courant du panier via l'accesseur getStatus sur l'objet de classe Basket.

- Si le panier est en statut freezed, refused ou timeout, on doit le «unfreezé», c'est à dire indiquer qu'on accepte qu'il soit mis à jour (appel à la méthode unfreeze du BasketWebService OSP).

  On doit ensuite le vider de son contenu avant de le renseigner de nouveau (appel à la méthode clear du BasketWebService OSP).

- Si le panier est en statut initialised, il suffit de le le vider de son contenu avant de le renseigner de nouveau (appel à la méthode clear du BasketWebService OSP).

- Enfin, dans tous les autres cas, on ferme le panier, ce qui a pour conséquence dans créer un nouveau (appel à la méthode close du BasketWebService OSP).

```
Basket basket = ospBasketWebService.getBasket (ospSessionId);

if (basket != null)
{
      String status = basket.getStatus();

      if (status.equals ("freezed") ||
          status.equals ("refused") ||
          status.equals ("timeout"))
      {
            ospBasketWebService.unfreeze (ospSessionId);
            ospBasketWebService.clear (ospSessionId);
      }
      else if (status.equals ("initialised"))
      {
            ospBasketWebService.clear (ospSessionId);
      }
      else
      {
            ospBasketWebService.close (ospSessionId);
      }
}
```

5. Récupération des lignes de commandes ESF à injécter dans OSP

On va devoir synchroniser le contenu des lignes de commande ESF avec OSP.

Pour récupérer la collection des sélections, il faut invoquer la méthode findAllSelections du CommandeWebService ESF.

```
WebSelection[] selections = esfCommandeWebService.findAllSelections (esfSessionId);
```

6. Création des lignes de commande OSP

Il faut ensuite itérer sur chacune des sélections de la collection récupérée à l'étape précédente.

A chaque itération on créer un nouvel objet de classe OrderLine ainsi qu'une collection d'attributs d'OrderLine (OrderLineAttributes).

On réalise ensuite, pour chaque itération, l'implémentation des étapes 6.1 à 6.10.

```
for (WebSelection selection : selections)
{
      OrderLine                orderLine  = new OrderLine();
      List<OrderLineAttribute> attributes = new ArrayList<OrderLineAttribute>();

      // Implémentation des traitements décrits dans les étapes 6.1 à 6.10
}
```

6.1 Construction de l'attribut libellé

Le libellé de la ligne de commande est le libellé du cours choisi dans la sélection courante. On doit donc tout d'abord récupérer le cours dans la sélection, puis son libellé.

On construit ensuite un nouvel objet de classe OrderLineAttribute. Puis, on valorise ses propriétés name, type et expression via les mutateurs correspondants (setName, setType et setExpression).

La valeur de l'attribut est stockée comme une constante dans son expression.

Il faut enfin ajouter l'objet à la collection des attributs de ligne de commande.

```
WebCours cours   = selection.getCours();
String   libelle = cours.getLibelle();

OrderLineAttribute attribute = new OrderLineAttribute();

attribute.setName       ("libelle");
attribute.setType       ("java.lang.String");
attribute.setExpression (libelle);

attributes.add (attribute);
```

6.2 Construction de l'attribut prix

Le prix de la ligne de commande est celui du cours choisi dans la sélection courante. Il est exprimé en centime, ce pourquoi on définie que l'expression de l'attribut doit être interprété comme un entier long.

On construit ensuite un nouvel objet de classe OrderLineAttribute. Puis, on valorise ses propriétés name, type et expression via les mutateurs correspondants (setName, setType et setExpression).

La valeur de l'attribut est stockée comme une constante dans son expression.

Il faut enfin ajouter l'objet à la collection des attributs de ligne de commande.

```
long   prix       = cours.getPrix();
String prixString = String.valueOf (prix);

OrderLineAttribute attribute = new OrderLineAttribute();

attribute.setName       ("prix");
attribute.setType       ("java.lang.Long");
attribute.setExpression (prixString);

attributes.add (attribute);
```

6.3 Construction de l'attribut pratique

La pratique (ou spécialité) est directement rattaché à la sélection courante. On souhaite récupérer son libellé.

On construit ensuite un nouvel objet de classe OrderLineAttribute. Puis, on valorise ses propriétés name, type et expression via les mutateurs correspondants (setName, setType et setExpression).

La valeur de l'attribut est stockée comme une constante dans son expression.

Il faut enfin ajouter l'objet à la collection des attributs de ligne de commande.

```
Pratique pratique        = selection.getPratique();
String   pratiqueLibelle = pratique.getLibelle();

OrderLineAttribute attribute = new OrderLineAttribute();

attribute.setName       ("pratique");
attribute.setType       ("java.lang.String");
attribute.setExpression (pratiqueLibelle);

attributes.add (attribute);
```

6.4 Construction de l'attribut niveau

Le niveau d'enseignement est directement rattaché à la sélection courante. On souhaite récupérer son libellé.

On construit ensuite un nouvel objet de classe OrderLineAttribute. Puis, on valorise ses propriétés name, type et expression via les mutateurs correspondants (setName, setType et setExpression).

La valeur de l'attribut est stockée comme une constante dans son expression.

Il faut enfin ajouter l'objet à la collection des attributs de ligne de commande.

```
Niveau niveau        = selection.getNiveau();
String niveauLibelle = niveau.getLibelle();

OrderLineAttribute attribute = new OrderLineAttribute();

attribute.setName       ("niveau");
attribute.setType       ("java.lang.String");
attribute.setExpression (niveauLibelle);

attributes.add (attribute);
```

6.5 Construction de l'attribut rendez-vous

Le rendez-vous est directement rattaché à la sélection courante. On souhaite récupérer son libellé.

On construit ensuite un nouvel objet de classe OrderLineAttribute. Puis, on valorise ses propriétés name, type et expression via les mutateurs correspondants (setName, setType et setExpression).

La valeur de l'attribut est stockée comme une constante dans son expression.

Il faut enfin ajouter l'objet à la collection des attributs de ligne de commande.

```
RendezVous rdv        = selection.getRendezVous();
String     rdvLibelle = rdv.getLibelle();

OrderLineAttribute attribute = new OrderLineAttribute();

attribute.setName       ("rdv");
attribute.setType       ("java.lang.String");
attribute.setExpression (rdvLibelle);

attributes.add (attribute);
```

6.6 Construction de l'attribut souhait moniteur

Le souhait moniteur est directement rattaché à la sélection courante.

On construit ensuite un nouvel objet de classe OrderLineAttribute. Puis, on valorise ses propriétés name, type et expression via les mutateurs correspondants (setName, setType et setExpression).

La valeur de l'attribut est stockée comme une constante dans son expression.

Il faut enfin ajouter l'objet à la collection des attributs de ligne de commande.

```
String souhaitMoniteur = selection.getSouhaitMoniteur();

OrderLineAttribute attribute = new OrderLineAttribute();

attribute.setName       ("souhait_moniteur");
attribute.setType       ("java.lang.String");
attribute.setExpression (souhaitMoniteur);

attributes.add (attribute);
```

6.7 Construction de l'attribut langue de l'élève

La langue de l'élève est directement rattaché à la sélection courante. On souhaite récupérer son libellé.

On construit ensuite un nouvel objet de classe OrderLineAttribute. Puis, on valorise ses propriétés name, type et expression via les mutateurs correspondants (setName, setType et setExpression).

La valeur de l'attribut est stockée comme une constante dans son expression.

Il faut enfin ajouter l'objet à la collection des attributs de ligne de commande.

```
LangueCours langueCours        = selection.getLangueCours();
String      langueCoursLibelle = langueCours.getLibelle();

OrderLineAttribute attribute = new OrderLineAttribute();

attribute.setName       ("eleve_langue");
attribute.setType       ("java.lang.String");
attribute.setExpression (langueCoursLibelle);

attributes.add (attribute);
```

6.8 Construction des attributs de l'élève

Les attributs eleve_nom, eleve_prenom et eleve_naissance dépendent de l'objet de classe Eleve rattaché à la sélection courante.

```
Eleve eleve = selection.getEleves()[0];
```

6.8.1 Construction de l'attribut nom de l'élève

Le nom de l'élève est directement rattaché à l'élève.

On construit ensuite un nouvel objet de classe OrderLineAttribute. Puis, on valorise ses propriétés name, type et expression via les mutateurs correspondants (setName, setType et setExpression).

La valeur de l'attribut est stockée comme une constante dans son expression.

Il faut enfin ajouter l'objet à la collection des attributs de ligne de commande.

```
String eleveNom = eleve.getNom();

OrderLineAttribute attribute = new OrderLineAttribute();

attribute.setName       ("eleve_nom");
attribute.setType       ("java.lang.String");
attribute.setExpression (eleveNom);

attributes.add (attribute);
```

6.8.2 Construction de l'attribut prénom de l'élève

Le prénom de l'élève est directement rattaché à l'élève.

On construit ensuite un nouvel objet de classe OrderLineAttribute. Puis, on valorise ses propriétés name, type et expression via les mutateurs correspondants (setName, setType et setExpression).

La valeur de l'attribut est stockée comme une constante dans son expression.

Il faut enfin ajouter l'objet à la collection des attributs de ligne de commande.

```
String elevePrenom = eleve.getPrenom();

OrderLineAttribute attribute = new OrderLineAttribute();

attribute.setName       ("eleve_prenom");
attribute.setType       ("java.lang.String");
attribute.setExpression (elevePrenom);

attributes.add (attribute);
```

6.8.3 Construction de l'attribut date de naissance de l'élève

La date de naissance de l'élève est directement rattaché à l'élève.

On construit ensuite un nouvel objet de classe OrderLineAttribute. Puis, on valorise ses propriétés name, type et expression via les mutateurs correspondants (setName, setType et setExpression).

La valeur de l'attribut est stockée comme une constante dans son expression.

Il faut enfin ajouter l'objet à la collection des attributs de ligne de commande.

```
SimpleDateFormat dateNaissanceFormat = new SimpleDateFormat ("dd/MM/yyyy");
Date             dateNaissance       = eleve.getDateNaissance();
String           dateNaissanceString = dateNaissanceFormat.format (dateNaissance);

OrderLineAttribute attribute = new OrderLineAttribute();

attribute.setName       ("eleve_naissance");
attribute.setType       ("java.lang.String");
attribute.setExpression (eleveNaissance);

attributes.add (attribute);
```

6.9 Construction des attributs d'horaires de la sélection

Les horaires choisies sont accessible, indirectement, via le cours de la sélection courante.

Nous allons voir comment construire la liste d'horaire, puis comment l'exploiter avant de la transformer en attribut de ligne de commande.

6.9.1 Extraction de la liste d'horaires

Les horaires sont indirectement accessibles car il y a deux cas d'usages. Soit nous sommes en présence d'un cours composé, soit d'un cours simple.

- Si le cours est un cours composé, on doit récupérer les cours sous-jacents (la composition).

  Puis, pour chaque cours composant, on doit extraire la première prestation, qui porte ses propres horaires.

  Il faut ensuite concaténer l'ensemble de ces listes d'horaires, pour n'en faire plus qu'une.

- Si le cours est un cours simple, on récupère directement la liste d'horaire de la première prestation rattachée au cours.

```
Horaire[]  horairesListe    = null;
WebCours[] compositionCours = cours.getCompositionCours();
boolean    coursCompose     = (compositionCours.length > 0);

if (coursCompose) // Cours compose
{
      List<Horaire> horaires = new ArrayList<Horaire>();
      for (WebCours composant : compositionCours)
      {
            WebPrestation prestation = composant.getPrestations()[0];
            for (Horaire horaire : prestation.getListeHoraire())
            {
                  horaires.add (horaire);
            }
      }
      horairesListe = horaires.toArray();
}
else // Cours simple
{
      WebPrestation prestation = cours.getPrestations()[0];
      horairesListe = prestation.getListeHoraire();
}
```

6.9.2 Construction de chaque horaire

Chacune des horaires de la liste doit ensuite être stockée dans la ligne de commande OSP sous forme d'attribut.

Le nom de l'attribut est de la forme horaire1, horaire2... La valeur de l'attribut doit être formatée à partir des propriétés de l'horaire.

On construit ensuite un nouvel objet de classe OrderLineAttribute. Puis, on valorise ses propriétés name, type et expression via les mutateurs correspondants (setName, setType et setExpression).

La valeur de l'attribut est stockée comme une constante dans son expression.

Il faut enfin ajouter l'objet à la collection des attributs de ligne de commande.

```
SimpleDateFormat horaireDateFormat  = new SimpleDateFormat ("dd/MM/yyyy");
SimpleDateFormat horaireHeureFormat = new SimpleDateFormat ("hh:mi");

for (int i = 0; i < horairesListe.length; i++)
{
      Horaire horaire       = horairesListe[i];
      Date    debut         = horaire.getDebut();
      Date    fin           = horaire.getFin();
      String  horaireName   = "horaire" + (i+1);
      String  horaireValue  = "";

      horaireValue += "le "  + horaireDateFormat.format  (debut);
      horaireValue += " de " + horaireHeureFormat.format (debut);
      horaireValue += " à "  + horaireHeureFormat.format (fin);

      OrderLineAttribute attribute = new OrderLineAttribute();

      attribute.setName       (horaireName);
      attribute.setType       ("java.lang.String");
      attribute.setExpression (horaireValue);

      attributes.add (attribute);
}
```

6.9.3. Insertion de l'attribut spécifiant le nombre d'horaires

On doit ensuite spécifier le nombre d'horaire attaché à cette ligne de commande.

On construit ensuite un nouvel objet de classe OrderLineAttribute. Puis, on valorise ses propriétés name, type et expression via les mutateurs correspondants (setName, setType et setExpression).

La valeur de l'attribut est stockée comme une constante dans son expression.

Il faut enfin ajouter l'objet à la collection des attributs de ligne de commande.

```
int    nbHoraires       = horairesListe.length;
String nbHorairesString = String.valueOf (nbHoraires);

OrderLineAttribute attribute = new OrderLineAttribute();

attribute.setName       ("nbhoraires");
```

```
attribute.setType       ("java.lang.String");
attribute.setExpression (nbHorairesString);

attributes.add (attribute);
```

6.10. Construction de la ligne de commande à partir des attributs

On doit ensuite associer la liste d'attributs à la ligne de commande (via le mutateur setAttributes).

On l'affecte ensuite au groupe de lignes de commande numéro 1. Dans ce cas, on utilise pas d'autres groupes.

Puis, on fixe l'expression à évaluer, c'est à dire la valeur de la ligne.
C'est l'expression ${prix}, qui signifie que la valeur de la ligne est portée par l'attribut prix.

Enfin, il faut ajouter la ligne de commande à la commande OSP.

```
orderLine.setAttributes (attributes);
orderLine.setExpression ("${prix}");
orderLine.setGroup (1);

ospBasketWebService.addLine (ospSessionId, storeid, orderLine);
```

7. Construction de la ligne de total dans le panier OSP

Une fois que chaque ligne de commande a été ajoutée au panier côté OSP, il est nécessaire de rajouter une dernière ligne : la ligne de total.

C'est sur la base de cette ligne que sera calculé le montant total du panier et donc, le montant de la transaction qui sera exécutée ultérieurement.

Elle a pour but de faire la somme de toutes les lignes de commande de son groupe (le goupe 1).

```
OrderLine orderLine = new OrderLine();

orderLine.setTotal (true);
orderLine.setPayable (true);
orderLine.setGroup (1);

ospBasketWebService.addLine (ospSessionId, storeid, orderLine);
```

8. Récupération de la commande OSP dans le panier

Un panier OSP est multi-commande. En effet, un panier peut contenir autant de commandes que de marchands impliqués dans la transaction.

Dans les étapes suivantes, on va travailler au niveau de la commande et non plus au niveau du panier. On va donc rechercher dans la collection des commandes du panier la commande correspondant à la boutique de notre marchand (via le storeid de l'école récupéré à l'étape 2).

```
Order   order  = null;
Basket  basket = ospBasketWebService.getBasket (ospSessionId);
Order[] orders = basket.getOrders();

for (Order basketOrder : orders)
{
      Store store = basketOrder.getStore();

      if (store.getId().equals (storeid))
      {
            order = basketOrder;
      }
}
```

9. Ajout des attributs de la commande OSP

Nous avons créé une commande, ainsi que les lignes de commande qui la compose dans le contexte OSP.

Il nous reste à ajouter les attributs au niveau de la commande elle même.

Pour cela nous allons avoir besoin de l'identifiant de la commande côté OSP.

```
int ospOrderId = order.getId();
```

9.1 Création de l'attribut de commande OSP "order_id"

La commande OSP a un numéro en propre. Afin de le surcharger et d'utiliser le numéro de commande de la commande ESF, il nous faut ajouter un attribut order_id.

On construit ensuite un nouvel objet de classe Attribute. Puis, on valorise ses propriétés name et value via les mutateurs correspondants (setName et setValue).

Il faut enfin ajouter l'objet à la commande via un appel à la méthode addAttribute du web service OrderWebService d'OSP.

```
Attribute attribute = new Attribute();

attribute.setName  ("order_id");
attribute.setValue (commandeid);

ospOrderWebService.addAttribute (ospSessionId, ospOrderId, attribute);
```

9.2 Ajout de l'attribut nom de la résidence

On doit associer le libellé de la résidence choisie par le client.

On accède à la résidence via un appel à la méthode getResidence du CommandeWebService ESF, puis à son libellé par un appel à l'accesseur getLibelle sur l'objet de classe Residence.

On construit ensuite un nouvel objet de classe Attribute. Puis, on valorise ses propriétés name et value via les mutateurs correspondants (setName et setValue).

Il faut enfin ajouter l'objet à la commande via un appel à la méthode addAttribute du web service OrderWebService d'OSP.

```
Residence residence        = esfCommandeWebService.getResidence (esfSessionId);
String    residenceLibelle = residence.getLibelle();

if (residence != null)
{
      Attribute attribute = new Attribute();

      attribute.setName  ("residence");
      attribute.setValue (residenceLibelle);

      ospOrderWebService.addAttribute (ospSessionId, ospOrderId, attribute);

}
```

9.3 Ajout dans la commande OSP du texte d'envoi de billet

On doit associer le paramètre booléen «envoiBillet» à la commande. On peut le récupérer indirectement par l'objet de classe Ecole récupérer à l'étape 2.

On appelle l'accesseur getParametres sur l'objet de classe Ecole pour avoir la propriété envoiBillet.

On construit ensuite un nouvel objet de classe Attribute. Puis, on valorise ses propriétés name et value via les mutateurs correspondants (setName et setValue).

Il faut enfin ajouter l'objet à la commande via un appel à la méthode addAttribute du web service OrderWebService d'OSP.

```
EcoleParametres ecoleParametres   = ecole.getParametres();
boolean         envoiBillet       = ecoleParametres.isEnvoiBillets();
String          envoiBilletString = String.valueOf (envoiBillet);

Attribute attribute = new Attribute();

attribute.setName  ("envoi_billet");
attribute.setValue (envoiBilletString);

ospOrderWebService.addAttribute (ospSessionId, ospOrderId, attribute);
```

9.4 Ajout dans la commande OSP de l'identifiant de l'école

On doit associer l'identifiant de l'école à la commande OSP.

L'identifiant de l'école (ecoleid) a été récupéré à l'étape 2.

On construit ensuite un nouvel objet de classe Attribute. Puis, on valorise ses propriétés name et value via les mutateurs correspondants (setName et setValue).

Il faut enfin ajouter l'objet à la commande via un appel à la méthode addAttribute du web service OrderWebService d'OSP.

```
Attribute attribute = new Attribute();

attribute.setName  ("ecole_id");
attribute.setValue (ecoleid);

ospOrderWebService.addAttribute (ospSessionId, ospOrderId, attribute);
```

9.5 Ajout dans la commande OSP du nom de l'école

On doit associer le nom de l'école à la commande OSP. L'ecole a été récupéré à l'étape 2, on accède directement à son nom via l'accesseur getNom sur l'objet de classe Ecole.

On construit ensuite un nouvel objet de classe Attribute. Puis, on valorise ses propriétés name et value via les mutateurs correspondants (setName et setValue).

Il faut enfin ajouter l'objet à la commande via un appel à la méthode addAttribute du web service OrderWebService d'OSP.

```
String nomEcole = ecole.getNom();

Attribute attribute = new Attribute();

attribute.setName  ("ecole_nom");
attribute.setValue (nomEcole);

ospOrderWebService.addAttribute (ospSessionId, ospOrderId, attribute);
```

9.6 Ajout à la commande OSP du nom d l'hôte réalisant la commande

On doit associer le nom de l'hote ayant passé la commande ESF à la commande OSP.

On récupère l'hote par un appel à la méthode getHote du CommandeWebService ESF.

Puis, on accède directement à son nom via l'accesseur getNom sur l'objet de classe Hote.

On construit ensuite un nouvel objet de classe Attribute. Puis, on valorise ses propriétés name et value via les mutateurs correspondants (setName et setValue).

Il faut enfin ajouter l'objet à la commande via un appel à la méthode addAttribute du web service OrderWebService d'OSP.

```
Hote   hote    = esfCommandeWebService.getHote (esfSessionId);
String nomHote = hote.getNom();

if (hote != null)
{
      Attribute attribute = new Attribute();

      attribute.setName  ("via_hote");
      attribute.setValue (nomHote);

      ospOrderWebService.addAttribute (ospSessionId, ospOrderId, attribute);
}
```

10. Preparation du panier OSP pour le paiement

Avant de procéder au paiement, il faut demander à OSP d'enregistrer le panier. On doit lui signifier cette demande en appellant la méthode saveForCurrentUser du BasketWebService OSP.

Puis, il faut indiquer à OSP que plus aucune modifiaction n'est acceptée sur ce panier : on le lock. On lui indique en appellent la méthode freeze du BasketWebService OSP.

```
ospBasketWebService.saveForCurrentUser (ospSessionId);
ospBasketWebService.freeze (ospSessionId);
```

11. Récupération des moyens de paiement

Toutes les écoles n'acceptent pas les mêmes moyens de paiement. Il faut donc propoer au client uniquement les moyens acceptés.

Les moyens de paiement par défaut sont CB, VISA et MASTERCARD.

Si une école n'accepte pas strictement cette liste de moyens de paiements, elle est surchargée via un attribut de store OSP nommé payment_cardlist.

Il faut donc récupérer les attributs de store via un appel à la méthode getAttributes du storeWebService OSP.

Puis, il faut itérer sur cette collection d'attribut pour rechercher celui dont le nom est payment_cardlist.

Si on retrouve cet attribut, sa valeur doit remplacer la valeur par défaut.

La liste est en réalité une chaîne de caractères, utilisant la virgule comme séparateur. Il faut donc procéder à l'extraction des types de moyen de paiements en découpant la chaîne selon ce séparateur.

Cette liste de moyens de paiement doit être utilisée dans le formulaire permettant à l'utilisateur de fournir ses informations de carte bancaire.

```
String defaultCardTypes = "CB,VISA,MASTERCARD";

Attribute[] storeAttributes = ospStoreWebService.getAttributes (ospSessionId);

for (Attribute storeAttribute : storeAttributes)
{
```

```
      String storeAttributeName  = storeAttribute.getName();
      String storeAttributeValue = storeAttribute.getValue();

      if (storeAttributeName.equals ("payment_cardlist"))
      {
            defaultCardTypes = storeAttributeValue; // ex : VISA,MASTERCARD,AMEX
      }
}

String[] cardTypes = defaultCardTypes.split (",");
```

12. Traitement du paiement

Une fois sur le formulaire de paiement, le client peut soit saisir ses données de carte bancaire, soit annulé son paiement pour abandonner ou modifier sa réservation.

Les actions à mener dans les deux cas diffèrent.

12.1 Le client annule son paiement

Si le client annule son paiement, il est nécessaire de le spécifier à OSP en invoquant la méthode validationCanceled du BsketWebService OSP.

```
ospBasketWebService.validationCanceled (ospSessionId);
```

12.2 Le client valide son paiement

Si le client valide son paiement, on doit tout d'abord collecter ses informations puis exécuter la transaction bancaire.

12.2.1 Création de la carte

Nous allons devoir instancier un objet de classe Card et alimenter ses propriétés à partir des informations collectées.

Les propriétés sont :

- le type de carte, valorisé par le mutateur setType
- le numéro de carte, valorisé par le mutateur setNumber
- le mois de validité, valorisé par le mutateur setMonth
- l'année de validité, valorisée par le mutateur setYear
- le cryptogramme, valorisé par le mutateur setComplementaryCode

Cet objet de classe Card doit être ajouté à une nouvelle collection de Card pour être utilisé lors de la transaction.

A titre d'information, plusieurs cartes peuvent être spécifiées lorsque le panier est multi-marchand. En effet, plusieurs transactions doivent être générées (une par marchand) et si le client utilise une e-card, les informations ne sont valides que pour une transaction : il est alors nécessaire de lui demander autant de numéro que d'Order dans le Basket OSP.

```
List<Card> cards = new ArrayList<Card>();

Card card = new Card();

String cardType   = "VISA";
String cardNumber = "1111222233334444";
int    cardMonth  = 12;
int    cardYear   = 2010;
String cardCCode  = "123";

card.setType              (cardType);
card.setNumber            (cardNumber);
card.setMonth             (cardMonth);
card.setYear              (cardYear);
card.setComplementaryCode (cardCCode);

cards.add (card);
```

12.2.2 Transaction bancaire

Une fois les informations de carte de paiement récupérées, on peut procéder à la transaction bancaire.

On invoque la méthode validateByPayment du BasketWebService OSP en lui passant en paramètre la liste de cartes.

Cette méthode retourne un booléen indiquant si la transaction a pu avoir lieu.

```
WebSessionInfo info = new WebSessionInfo();
info.setSessionId (ospSessionId);

String shareToken = ospSessionWebService.shareWithPaymentSite(info);
info = ospSessionWebService.decodeSessionInfo (shareToken, "valraiso",
"b739cf068adf3e90d57008c895f0203a");

boolean transactionDone = ospBasketWebService.validateByPayment (ospSessionId, cards);
```

12.2.3 Récupération du résultat de la transaction

Si la transaction a eu lieu, on doit vérifier si elle a été accepté par la banque.

On appelle la méthode getPaymentInfo du OrderWebService OSP pour notre commande. Cette méthode retourne un objet de classe PaymentInfo qui contient les informations relative à notre transaction.

Il contient notamment une propriété accepted accessible via l'accesseur isAccepted qui permet de savoir si la transaction a été autorisée.

```
if (transactionDone)
{
    PaymentInfo info     = ospOrderWebService.getPaymentInfo (ospSessionId, ospOrderId);
    boolean     accepted = info.isAccepted();

    // Traitement effectué à l'étape suivante
}
```

12.2.4 Traitement a effectuer si le paiement a été autorisé

Si la transaction a été acceptée, il nous reste à notifier la vente en ligne ESF afin que la commande soit prise en compte.

La notification s'effectue par un appel à la méthode validateCommande du CommandeWebService ESF.

Cette méthode prend en paramètre deux d'informations présentes dans l'objet de classe PaymentInfo récupéré à l'étape précédente :

- le numéro d'autorisation bancaire, accessible via l'accesseur getAuthoristationId
- le certificat de transaction, accessible via l'accesseur getTransactionCertificate

```
if (accepted)
{
      String authorisationid  = info.getAuthorisationId();
      String certificate      = info.getTransactionCertificate();
      String ospOrderIdString = String.valueOf (ospOrderId);

      esfCommandeWebService.validateCommande (esfSessionId,
                                              commandeid,
                                              ospOrderIdString,
                                              authorisationid,
                                              certificate);
}
```

13. Traitement de la commande en mode délégation (vente en compte)

Cette section n'est applicable que si vous ne souhaitez pas que le paiement induise un versement direct du montant de la commande sur le compte bancaire de l'ESF.

C'est une alternative à la section 12.2.

13.1 Identification du partenaire réalisant la vente en compte

Tout d'abord, afin que l'ESF pour laquelle vous faites une vente puisse garder la trace du fait que cette commande est établie en «vente en compte», il est nécessaire de réaliser une étape supplémentaire juste avant l'étape 1 de ce document.

L'identifiant (login) et le mot de passe de l'intermédiaire seront déterminés et affectés ESF par ESF : vous devrez donc, de votre côté, gérer une table avec ces données.

Voici les champs nécessaires à ce stockage :

Nom de colonne	Type	Nullable
ecoleid	varchar(6)	not null
login	varchar(32)	not null
password	varchar(32)	not null

Le code école utilisé est le code banalisé, fournit par le SNMSF, permettant l'identification d'une école.

Exemple :

- 998 : Ecole de test
- 419 : Les ménuires

```
String  intermediaireLogin = "yyyy";
String  intermediairePassword = "zzzz";

boolean success = esfCommandeWebService.setIntermediaire (esfSessionid,
                                                           intermediaireLogin,
                                                           intermediairePassword);
```

13.2 Finalisation de la transaction

Dans le cas de la vente en compte, à la place des sections 12.2, 12.2.1, 12.2.2, 12.2.3 et 12.2.4, il faut valider la commande par délégation.

On partage dans un premier temps la session OSP avec l'intermédiaire, puis on la décode avec ses identifiants avant de la valider par délégation.

```
Hote           hote                  = esfCommandeWebService.getHote (esfSessionId);
Intermediaire  intermediaire         = hote.getIntermediaire();
String         intermediaireId       = intermediaire.getId();
String         intermediaireLogin    = intermediaire.getLogin();
String         intermediairePassword = intermediaire.getPassword();
String         shareToken;

WebSessionInfo info = new WebSessionInfo();
info.setSessionId (ospSessionId);

shareToken = ospSessionWebService.shareSessionWithIntermediary (info,
                                                                intermediaireLogin);

info = ospSessionWebService.decodeSessionInfo (shareToken,
                                               intermediaireLogin,
                                               intermediairePassword);

String delegationSessionId = info.getSessionId();

ospBasketWebService.validateByDelegation (delegationSessionId);
```