MOVING WebServices Read-Only

*Spécifications*

Ce document synthétise les spécifications logicielles relatives aux WebServices en lecture qui devront être exposées par le BackOffice Moving-BO

Benjamin Verney – Thomas Badin

06/05/2014

MOVING WebServices Read-Only

Spécifications

Table des matières

[Introduction 2](#_Toc387740262)

[Contrôle d’accès 3](#_Toc387740263)

[Méthode authenticate 3](#_Toc387740264)

[Méthode getAccessToken 3](#_Toc387740265)

[Accès aux éléments 4](#_Toc387740266)

[Méthode getListAllParcours 4](#_Toc387740267)

[Cas standard 4](#_Toc387740268)

[Cas Limites 4](#_Toc387740269)

[Cas d’erreurs 4](#_Toc387740270)

[Méthode getParcoursArchitectureById 5](#_Toc387740271)

[Cas Standard 5](#_Toc387740272)

[Cas Limites 6](#_Toc387740273)

[Cas d’erreurs 7](#_Toc387740274)

[Méthode getParcoursById 7](#_Toc387740275)

[Cas standard 7](#_Toc387740276)

[Cas Limites 7](#_Toc387740277)

[Cas d’erreurs 7](#_Toc387740278)

[Méthode getSousParcoursById 8](#_Toc387740279)

[Cas standard 8](#_Toc387740280)

[Cas Limites 8](#_Toc387740281)

[Cas d’erreurs 9](#_Toc387740282)

[Méthode getTransitionById 9](#_Toc387740283)

[Méthode getSceneById 9](#_Toc387740284)

[Cas standard 9](#_Toc387740285)

[Cas d’erreurs 9](#_Toc387740286)

[Méthode getMediaById 9](#_Toc387740287)

[Méthode getArtefactById 9](#_Toc387740288)

[Méthode getPointdInteretByTag 9](#_Toc387740289)

[Description des services (WSDL) 9](#_Toc387740290)

# Introduction

Les web services permettent d’exposer des objets métiers vers l’extérieur. Les principaux objets métiers sont les parcours et sous parcours, les scènes, et les éléments (médias ou artefact) liés aux scènes.

Les WebServices seront utilisé à la fois par le Front Office Web et les applications mobiles :



Ce document ne traite que des WebServices permettant la consultation. Si des WebServices de modification sont nécessaire (ce qui a été évoqué lors de la réunion du 28 Avril 2014 : [Wiki](https://tuleap.cervin.org/wiki/index.php?pagename=Compte%20rendu%2028%20avril%202014&group_id=105) ), ils feront l’objet d’un autre document et la partie sera traitée dans un second temps.

Les WebServices seront en [SOAP](http://fr.wikipedia.org/wiki/SOAP), protocole le plus répandu actuellement. Le Framework PHP utilisé pour le back office ([Zend Framework 2](http://framework.zend.com)), permet directement d’exposer des services (cf [Zend/Soap/Server](http://framework.zend.com/manual/2.0/en/modules/zend.soap.server.html) ). C’est donc ce que nous utiliserons pour créer les WebServices.

Dans l’ensemble de ce document, nous allons spécifier la réponse du WebService par un exemple. Ce sera moins formel qu’une définition des réponses au format XSD, mais plus simple à comprendre.

# Contrôle d’accès

A faire (Ben).

## Méthode authenticate

## Méthode getAccessToken

# Accès aux éléments

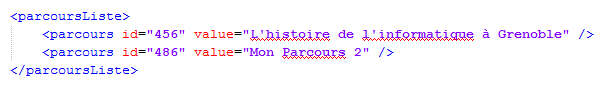
Cette partie regroupe les fonctions d’accès aux différents éléments de Movin. Nous appliquons la convention suivante : quand nous requêtons un élément (fonction getXXXById), nous retournons le détail de l’élément, et la liste non détaillé (juste Id et nom affichable), des sous éléments. Nous rappelons que la hiérarchie des éléments est la suivante :

## Méthode getListAllParcours

Méthode permettant de retourner tout les parcours « Public » du BackOffice. La méthode ne prend pas de paramètre.

### Cas standard

Il y a au moins un parcours « Public » dans le BO. Le XML retourné sera de la forme :



Note : La méthode ne prend pas d’argument, car un Back Office est associé à un seul Front. Dans le cas d’un fonctionnement multi-tenant (ce qui n’est pas prévus à l’heure actuelle), cette méthode devrait être supprimée, et remplacée par une méthode getListParcoursByClientId par exemple.

### Cas Limites

Il n’y a aucun parcours en visibilité « Public » dans le BO. La fonction renvoie une liste vide.

### Cas d’erreurs

Il n’y a pas de cas d’erreur.

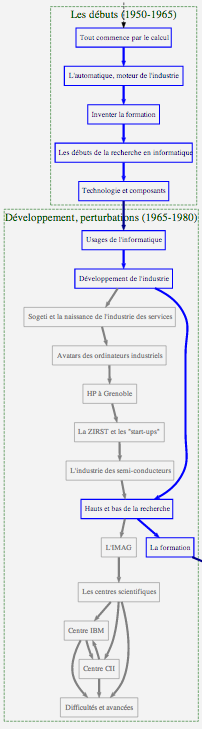
## Méthode getParcoursArchitectureById

Méthode permettant de récupérer l’architecture d’un parcours. L’architecture d’un parcours correspond à sa décomposition en sous-parcours et scène, et les relations entre les éléments. La méthode prend en paramètre l’identifiant du parcours.

### Cas Standard

Le parcours demandé existe dans la base de données, il est « Public » et possède au moins un sous parcours qui lui-même contient au moins une scène.

Prenons par exemple le parcours suivant :



Le XML reçu sera de la forme :

Manque la semantique des transistions

On ne renverra que les identifiants et noms, cette fonction permettra à afficher une vision globale du parcours, sans rentrer dans les détails.

### Cas Limites

Le parcours ne contient aucun sous parcours ou un sous parcours ne contient aucune scène :

* Cela ne devrait pas arriver, un parcours ne contenant pas de sous parcours ou un sous parcours ne contenant pas de scène est probablement du au fait que le parcours est en construction, il n’aurait donc pas du être passé en « Public »
* Il convient donc de logger le problème et de renvoyer un message d’erreur à l’utilisateur. (Parcours incomplet).

### Cas d’erreurs

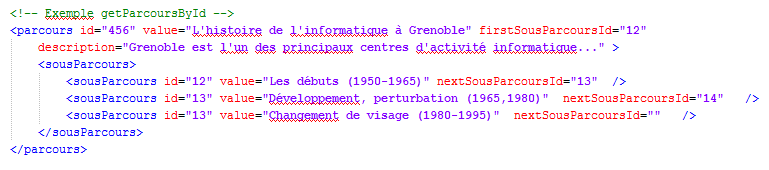
L’id spécifié n’existe pas, ou n’est pas en visibilité public : retourner un message d’erreur à l’utilisateur (Not Found).

## Méthode getParcoursById

Méthode permettant de récupérer les informations d’un parcours. On retournera toutes les informations du parcours, ainsi que la liste de ses sous-parcours. On ne détaillera pas les informations sur les sous-parcours, on se contentera de retourner l’id, le nom et les liens entre les sous-parcours.

### Cas standard

Le parcours existe, il est en visibilité « Public » et contient au moins un sous parcours. Dans ce cas le XML retourné sera de la forme :



### Cas Limites

Le parcours ne contient aucun sous parcours :

* Cela ne devrait pas arriver, un parcours ne contenant pas de sous parcours est probablement du au fait que le parcours est en construction, il n’aurait donc pas du être passé en « Public »
* Il convient donc de logger le problème et de renvoyer un message d’erreur à l’utilisateur. (Parcours incomplet).

### Cas d’erreurs

L’id spécifié n’existe pas, ou n’est pas en visibilité public : retourner un message d’erreur à l’utilisateur (Not Found).

## Méthode getSousParcoursById

Méthode permettant de récupérer les informations d’un sous-parcours. On retournera toutes les informations du sous-parcours, ainsi que la liste de ses scènes. On ne détaillera pas les informations sur les scènes, on se contentera de retourner l’id, le nom et les liens entre les scènes.

### Cas standard

Le sous-parcours existe, le parcours auquel il appartient est en visibilité « Public » et contient au moins une scène. Dans ce cas le XML retourné sera de la forme :

Manque la sémantique transistion

### Cas Limites

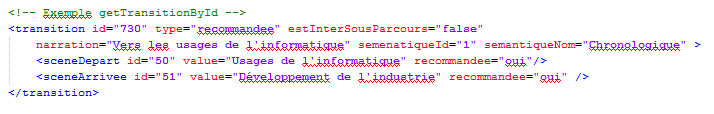
Le sous-parcours ne contient aucune scène:

* Cela ne devrait pas arriver un sous-parcours ne contenant pas de scène est probablement dû au fait que le parcours est en construction, il n’aurait donc pas du être passé en « Public »
* Il convient donc de logger le problème et de renvoyer un message d’erreur à l’utilisateur. (Parcours incomplet).

### Cas d’erreurs

L’id spécifié n’existe pas, l’id du sous parcours ne correspond pas à un sous parcours dont le parent (le parcours) est public : retourner un message d’erreur à l’utilisateur (Not Found).

## Méthode getTransitionById



## Méthode getSceneById

Méthode permettant de récupérer les informations d’une scène. On retournera toutes les informations de la scène, ainsi que la liste de des médias et des artefacts (en visibilité public) lié à cette scène. On ne détaillera pas les informations sur les médias et les artefacts, seulement les ids, les noms et les types (id et nom).

### Cas standard

La scène existe et le parcours auquel elle appartient est en visibilité « Public ». Dans ce cas le XML retourné sera de la forme :

Manque les transition

### Cas d’erreurs

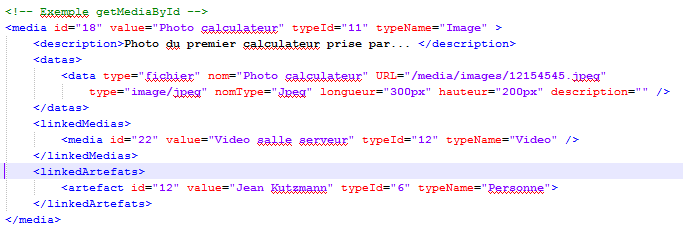
L’id spécifié n’existe pas, l’id de la scène ne correspond pas à une scène dont le parcours est public : retourner un message d’erreur à l’utilisateur (Not Found).

## Méthode getMediaById

Méthode permettant de récupérer les informations d’un média. On retournera toutes les informations du média, ainsi que la liste de des médias et des artefacts (en visibilité public) auquel il est lié. On ne détaillera pas les informations sur les médias et les artefacts liés, seulement les ids, les noms et les types (id et nom).

### Cas standard

Le média existe, il est en visibilité « Public », et il contient au moins une donnée. Dans ce cas le XML retourné sera de la forme :



### Cas Limites

Le média ne contient aucune donnée :

* Cela ne devrait pas arriver, un media contient normalement 1 donnée (il peut en contenir plusieurs, mais ce ne sera pas courant). Le média est donc probablement est en construction, il n’aurait donc pas du être passé en « Public »
* Il convient donc de logger le problème et de renvoyer un message d’erreur à l’utilisateur. (Media incomplet).

### Cas d’erreurs

L’id spécifié n’existe pas, ou le média n’est pas public : retourner un message d’erreur à l’utilisateur (Not Found).

## Méthode getArtefactById

Méthode permettant de récupérer les informations d’un artefact. On retournera toutes les informations de l’artefact, ainsi que la liste de des médias et des artefacts (en visibilité public) auquel il est lié. Le relation entre artefact ayant un sens on précisera si l’artefact en question est l’origine ou la destination, ainsi que sa sémantique (id et nom). On ne détaillera pas les informations sur les médias et les artefacts liés, seulement les ids, les noms et les types (id et nom).

### Cas standard

L’artefact existe et il est en visibilité « Public ». Un artefact ne contient pas forcement de données, une description peut très bien suffire. Dans ce cas le XML retourné sera de la forme :

## D:\utilisateurs\tbadin\Pictures\Capture.PNG

### Cas d’erreurs

L’id spécifié n’existe pas, ou l’artefact n’est pas public : retourner un message d’erreur à l’utilisateur (Not Found).

## Méthode getPointdInteretByTag

Cette méthode très simple prend un paramètre un ID de tag (une string), un type de tag (NFC/QRCode/iBeacon), et renvois le point d’intérêt (scène ou sous-parcours) correspondant.

### Cas standard

Le couple ID / type de tag existe, le XML retourné sera de la forme :



### Cas d’erreurs

Le couple ID / type de tag n’existe pas : retourner un message d’erreur à l’utilisateur (Not Found).

## Méthode getPointdInteretListByGPSRange

Cette méthode prend un paramètre une cordonnée GPS (x et y) et une distance d. La méthode retourne tous les points d’intérêt dont la visibilité est public et qui ont une intersection avec le cercle de centre (x,y) et de rayon d. (Note : il peut y avoir plusieurs points d’intérêts superposé).

**Note** : n’ayant jamais développé d’application utilisant un GPS, cette méthode n’est qu’une « intuition » du fonctionnement de l’application, et peut être inutile et a redéfinir.

### Cas standard

Il y a au moins un point d’intérêt qui correspond, le XML retourné sera de la forme :



### Cas limite

Il n’y pas aucun point d’intérêt dans la zone ( de centre (x,y) et de rayon d). La liste renvoyée sera une liste vide. En principe l’application devra refaire une requête lorsque l’utilisateur s’approchera des limites de la zone de rayon d.

### Cas d’erreurs

Les coordonnées passées en paramètre ne sont pas des coordonnées GPS valide. Renvoyer une erreur à l’utilisateur.

# Description des services (WSDL)