HR Access
a Sopra company

HR Framework

OpenHR

Guide du développeur

HRa Suite 9

International

F OPENH 0900 3 GD XXX

Troisième édition : Mai 2014

Avertissement

Le présent document peut contenir des informations ou des références concernant certains produits, logiciels ou services HR Access non annoncés dans ce pays. Cela ne signifie pas que Sopra HR Software ait l'intention de les y annoncer. Sopra HR Software peut détenir des brevets ou des demandes de brevet couvrant les produits mentionnés dans le présent document. La remise de ce document ne vous donne aucun droit de licence sur ces brevets ou demandes de brevet.
Les logiciels tierce partie inclus dans HR Access ne peuvent être utilisés séparément de HR Access.

Marques

HRa Suite et HR Access sont des marques déposées de Sopra HR Software. Toute utilisation, reproduction ou représentation nécessite l'accord express et préalable de Sopra HR Software.
Les autres noms utilisés pour désigner des sociétés, des produits ou des services sont des marques ayant leur titulaire respectif.

Mise à jour de la documentation

Le contenu de la documentation HR Access est régulièrement mis à jour.
Ces mises à jour sont disponibles sur le site du Support clients : www.support.hraccess.com.

Remarques du lecteur

Vos commentaires et suggestions nous permettent d'améliorer la qualité de nos documentations. Ils jouent un rôle important lors de leur mise à jour. N'hésitez pas à en faire part à la hot-line HR Access.

HR Access
a Sopra company
www.hraccess.com
Le Triangle de l'Arche
8, cours du Triangle
92937 Paris La Défense Cedex

Table des matières

A propos de cette documentation

Cette documentation constitue le Guide du développeur OpenHR. Elle décrit le fonctionnement et l'utilisation de l'API OpenHR, qui est une API Java livrée avec HR Access. Cette API permet de créer des applications tierces appelées "applications clientes OpenHR" (ou applications clientes).

Ce Guide s'adresse aux développeurs Java mais est également compréhensible par un néophyte du progiciel HR Access et de ses concepts. C'est pourquoi, lorsque c'est possible, ce Guide établit un parallèle entre des concepts typiquement HR Access et des concepts issus du monde Java.

CHAPITRE 1

Introduction à OpenHR

A propos de ce chapitre

Ce chapitre propose une présentation de l'API OpenHR.

L'**API OpenHR** est une API Java livrée avec HR Access qui permet de créer des applications tierces appelées "applications clientes OpenHR" (ou applications clientes).

Présentation de l'API OpenHR

Accès aux données

OpenHR offre différents services dont le principal est de permettre l'accès en lecture/écriture et en temps réel aux données gérées par le serveur HR Access tout en réutilisant la logique fonctionnelle existante de contrôle des données (implémentée sous la forme de programmes COBOL).

Schématiquement, ce service fournit une API d'accès aux données du serveur HR Access, accédées à travers des services implémentés en COBOL et représentées sous forme d'objets Java.

Cette API a été conçue spécifiquement pour une utilisation en mode TP (temps réel). C'est pourquoi le volume des données manipulées doit rester raisonnable. Dès lors que le volume de données est trop important pour assurer des performances décentes, il faut envisager une autre solution basée sur l'utilisation du mode batch par exemple.

OpenHR fournit un modèle objet intuitif et cohérent avec les concepts HR Access (dossier, information, occurrence, rubrique, etc.). Un développeur qui possède des connaissances sur le progiciel HR Access retrouvera donc rapidement ses marques.

L'API permet de réutiliser le contrôle d'accès aux données (appelé confidentialité) défini sous forme de rôles via l'application Design Center.

Enfin, OpenHR est une API générique pilotée par le modèle de données. De la même manière que l'API JDBC ne connaît pas a priori la structure des tables de base de données accédées, l'API OpenHR permet de manipuler n'importe quel type de dossier dès lors qu'elle dispose du modèle de données auquel se conforme ce dossier. Ce modèle de données correspond au modèle défini via l'application Design Center.

Documentation javadoc

OpenHR dispose d'une API publique documentée sous forme de javadoc. Il est vivement recommandé de se référer à celle-ci au fur et à mesure des explications pour disposer du maximum d'informations lors de la lecture du présent Guide. Certaines des méthodes de la javadoc ne sont pas documentées ici, soit parce qu'elles sont dépréciées, soit parce que leur utilisation est trop avancée.

Dans tous les cas, il est conseillé de se reporter régulièrement à la javadoc conjointement à la lecture de ce Guide du développeur.

Exemples fournis

Les exemples fournis dans cette documentation ont été écrits pour un niveau de compilation 1.4.2. L'API étant compilée en Java 1.4, les notations de type générique qui apparaissent dans la documentation sont là à titre informatif ; à ce jour, l'API ne supporte pas l'utilisation des "generics".

De plus, les exemples de code Java fournis dans cette documentation se veulent concis. Par conséquent, si un exemple nécessite une référence à un objet IHRSession et que la manière d'obtenir un tel objet a déjà été présentée plus haut dans la documentation, ce code ne sera pas redondé. Il sera simplement évoqué de manière elliptique à l'aide d'une méthode getSession() qui est censée retourner la session OpenHR. Ce procédé permet de fournir des exemples de code compacts qui se focalisent sur le sujet du chapitre courant.

Les exemples ne sont pas utilisables tels quels : ils ne peuvent être copiés / collés dans un IDE et exécutés.

CHAPITRE 2

Mise en œuvre rapide de l'API et architecture

A propos de ce chapitre

Ce chapitre propose un exemple détaillé permettant de mettre en œuvre de façon rapide l'API OpenHR.

De plus, il précise la place d'OpenHR dans l'architecture HR Access et présente le serveur OpenHR.

Exemple de mise en œuvre de l'API OpenHR

L'exemple qui suit montre comment mettre en œuvre l'API OpenHR pour :

- Connecter une session OpenHR
- Connecter un utilisateur
- Charger un dossier de salarié à partir de sa clé fonctionnelle (réglementation, matricule)
- Modifier la date de naissance du salarié et mettre à jour le dossier sur le serveur HR Access à l'aide d'un des rôles de l'utilisateur

La gestion des erreurs a été volontairement ignorée afin de ne pas alourdir l'exemple.

```
package com.hraccess.openhr.samples;


import org.apache.commons.configuration.PropertiesConfiguration;


import com.hraccess.openhr.HRApplication;
import com.hraccess.openhr.HRSessionFactory;
import com.hraccess.openhr.IHRSession;
import com.hraccess.openhr.IHRUser;
import com.hraccess.openhr.beans.HRDataSourceParameters;
import com.hraccess.openhr.dossier.HRDossier;
import com.hraccess.openhr.dossier.HRDossierCollection;
import com.hraccess.openhr.dossier.HRDossierCollectionParameters;
import com.hraccess.openhr.dossier.HRDossierFactory;
import com.hraccess.openhr.dossier.HRKey;
import com.hraccess.openhr.dossier.HROccur;
```

```
public class QuickStartSample {
   public static void main(String[] args) throws Exception {
   // Configuring logging system from given Log4J property configuration file
   HRApplication.configureLogs("C:\\openhr\\conf\\log4j.properties");

   // Creating from given OpenHR configuration file and connecting session
   to HR Access server
   IHRSession session = HRSessionFactory.getFactory().createSession(
               new
   PropertiesConfiguration("C:\\openhr\\conf\\openhr.properties"));

   IHRUser user = null;

   try {
         // Connecting user with given login ID and password
         user = session.connectUser("HRAUSER", "SECRET");

         // Creating configuration to handle HR Access employee dossiers
         HRDossierCollectionParameters parameters = new
   HRDossierCollectionParameters();
         parameters.setType(HRDossierCollectionParameters.TYPE_NORMAL);
         parameters.setProcessName("FS001"); // Using process FS001 to read
   and update dossiers
         parameters.setDataStructureName("ZY"); // Dossiers are based on
   data structure ZY

         // Reading data sections ZY00 (ID) and ZY10 (Birth)
         parameters.addDataSection(new
   HRDataSourceParameters.DataSection("00"));
         parameters.addDataSection(new
   HRDataSourceParameters.DataSection("10"));

         // Instantiating a new dossier collection with given role,
   conversation and configuration
         HRDossierCollection dossierCollection = new
   HRDossierCollection(parameters,
                     user.getMainConversation(),
                     user.getRole("EMPLOYEE(123456)"),
                     new
   HRDossierFactory(HRDossierFactory.TYPE_DOSSIER));
```

```
        // Loading an employee dossier from given functional key (policy
    group, employee ID)
        HRDossier employeeDossier = dossierCollection.loadDossier(new
    HRKey("HRA","123456"));


        // Retrieving the unique occurrence of data section ZY10 (Birth)
        HROccur birthOccurrence =
    employeeDossier.getDataSectionByName("10").getOccur();


        // Updating the employee's birth date (Item ZY10 DATNAI)
        birthOccurrence.setDate(“DATNAI”,
    java.sql.Date.valueOf(“1970-06-18”));


        // Committing the changes to the HR Access server
        employeeDossier.commit();
    } finally {
        if ((user != null) && user.isConnected()) {
            // Disconnecting user
            user.disconnect();
        }
        if ((session != null) && session.isConnected()) {
            // Disconnecting OpenHR session
            session.disconnect();
        }
    }
    }
}
```

Pour exécuter cet exemple, vous devez :

- Créer un projet sous votre IDE Java
- Importer le JAR de la librairie de l'API OpenHR dans votre projet ainsi que ceux des dépendances (cf. librairies livrées avec l'API)
- Créer un package com.hraccess.openhr.samples
- Y créer une classe de nom QuickStartSample et y coller le code source ci-dessus
- Créer un répertoire C:\openhr\conf et y coller le contenu des deux fichiers ci-dessous
- Adapter le contenu du fichier openhr.properties (adresse IP, nom de machine)
- Créer le répertoire C:\openhr\work
- Modifier le sample afin d'utiliser une clé fonctionnelle de dossier de salarié valide
- Exécuter le sample

Fichier log4j.properties

```
log4j.rootCategory=DEBUG, A1

# A1 is set to be a ConsoleAppender (writes to system console).
log4j.appender.A1=org.apache.log4j.ConsoleAppender

# A1 uses PatternLayout.
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
log4j.appender.A1.layout.ConversionPattern=%d %p - %m%n
```

Fichier openhr.properties

```
session.language[1]=F
session.process_list=FS001
session.work_directory=C:\openhr\work

openhr_server.server=10.11.12.13

normal_message_sender.security=disabled
normal_message_sender.port=8800

sensitive_message_sender.security=disabled
sensitive_message_sender.port=8800

privilegied_message_sender.security=disabled
privilegied_message_sender.port=8800
```

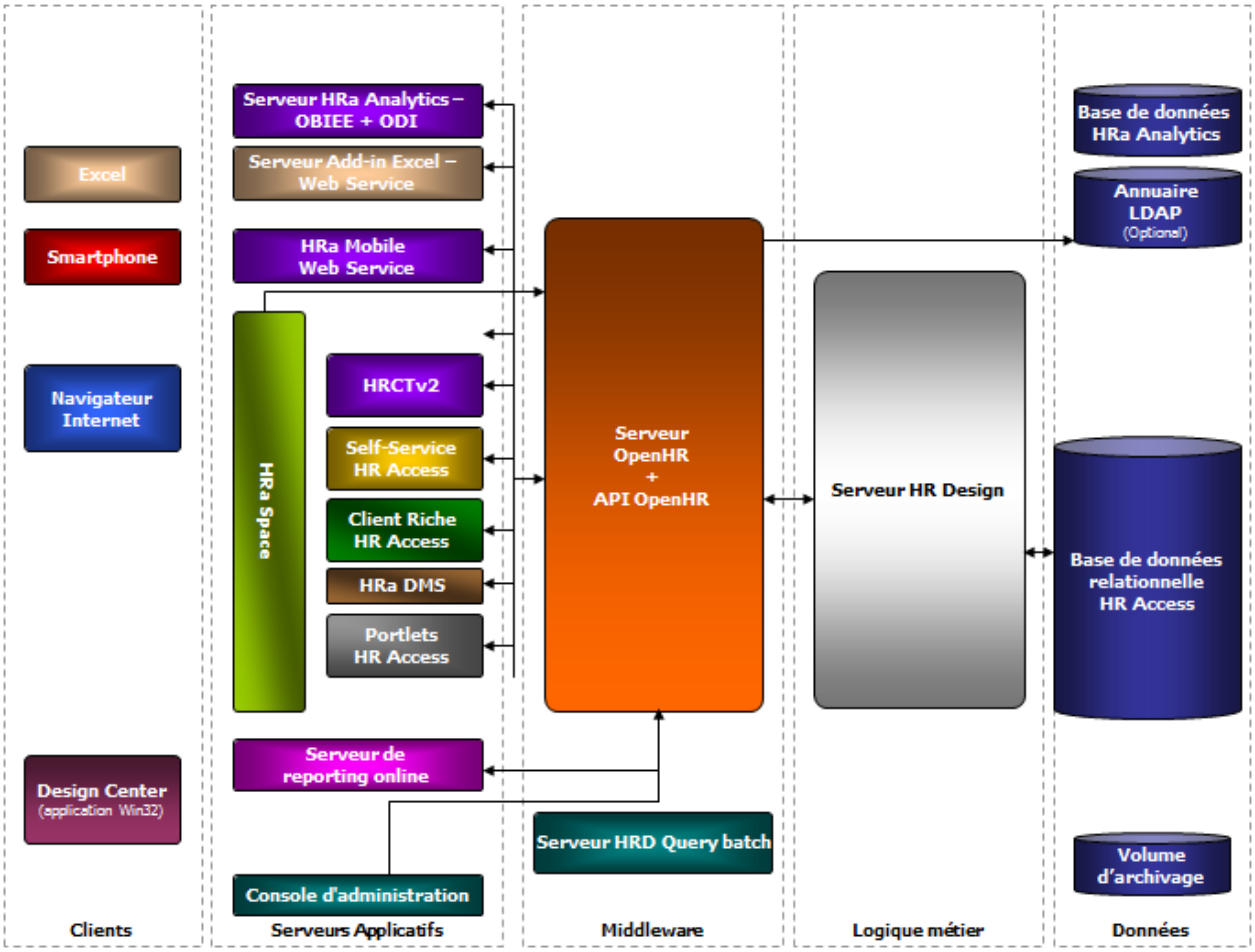
Cet exemple montre qu'il est plutôt simple d'utiliser l'API OpenHR. En relativement peu de code, il est possible de lire et mettre à jour des dossiers sur le serveur HR Access. La sémantique des classes de l'API est suffisamment parlante pour qu'un expert HR Access puisse comprendre ce que fait ce sample.

Nous allons maintenant expliquer en détail en quoi consiste l'API OpenHR et comment la mettre en œuvre.

[1] *Déprécié en 7.30.50 ou supérieure, utiliser session.languages à la place (voir le paragraphe "Propriétés de configuration").*

Architecture

Schéma de l'architecture HR Access



Le schéma ci-dessus représente l'architecture HR Access. On remarque que l'API OpenHR est un élément essentiel de celle-ci car c'est sur elle que sont bâties toutes les applications Java pour accéder aux services et aux données du serveur HR Access.

En utilisant l'API, un développeur peut créer une nouvelle application pour accéder aux dossiers HR Access en réutilisant la confidentialité, c'est-à-dire l'ensemble des règles de sécurité appliquées aux utilisateurs lors de l'accès aux données, définie par HR Access.

Serveur OpenHR

L'API OpenHR communique avec un serveur nommé serveur OpenHR. Cette communication s'effectue de manière synchrone à l'aide de sockets. Le protocole de communication entre client (l'application bâtie au-dessus de l'API OpenHR) et serveur OpenHR est propriétaire de HR Access : il est constitué d'un ensemble de messages au format texte.

Le serveur OpenHR agit comme une porte d'entrée vers le monde HR Access : toute communication de l'API avec le serveur HR Access passe obligatoirement par lui. Pour information, le serveur OpenHR est une application Java stand-alone, c'est-à-dire non hébergée dans un conteneur JEE, chargée de démarrer et d'arrêter les programmes COBOL du serveur HR Access. La manière dont ceux-ci sont déclenchés dépend de la plate-forme d'exécution du serveur HR Access mais il existe deux types de fonctionnement :

- Le serveur OpenHR est hébergé sur la même machine que le serveur HR Access : le déclenchement des programmes COBOL résulte de l'exécution d'une ligne de commande directement sur la machine. C'est le cas des plates-formes Windows et Unix. — *Win* *Unix*
- Le serveur OpenHR est hébergé sur une autre machine que le serveur HR Access : le déclenchement des programmes COBOL s'effectue par utilisation d'une transaction via TCP/IP. C'est le cas des plates-formes MVS. — *MVS*

Pour plus d'informations sur le fonctionnement du serveur OpenHR, reportez-vous au chapitre afférent dans le *Guide technique tous systèmes HR Design*.

L'API OpenHR agit donc comme un middleware pour se connecter au serveur HR Access.

CHAPITRE 3

Les packages de l'API

A propos de ce chapitre

Ce chapitre présente les différents packages que comprend l'API OpenHR.
Vous y trouverez un tableau les récapitulant.

Tableau récapitulatif des packages de l'API

Le tableau suivant donne un aperçu du rôle de chaque package de l'API.
Les package principaux de l'API sont le package com.hraccess.openhr qui définit les classes de base de l'API et com.hraccess.openhr.dossier qui fournit la fonctionnalité de gestion de dossiers.

Package	Rôle
com.hraccess.datasource	Fournit les classes de base pour représenter un nœud de données, une source de données
com.hraccess.dbnav	Fournit les classes de base pour représenter un nœud de données, une source de données nagivationnelle
com.hraccess.openhr	Fournit les classes représentant les concepts fondamentaux de l'API : session, utilisateur, rôle, modèle de données (dictionnaire, structure de données, informations, rubriques…)
com.hraccess.openhr.beans	Fournit des classes permettant de lire des données techniques et applicatives
com.hraccess.openhr.blob	Fournit les classes requises pour la manipulation des données de type BLOB (Binary Large OBject)
com.hraccess.openhr.configuration	Fournit une classe listant les propriétés de configuration de la session OpenHR
com.hraccess.openhr.dossier	Fournit les classes permettant de manipuler en lecture/écriture des dossiers HR Access

Package	Rôle
com.hraccess.openhr.dossier.helper	Fournit des classes (fortement typées) permettant de manipuler un dossier de travail (de structure de données ZO)
com.hraccess.openhr.entity	Fournit les classes permettant de lire les objets déployés (chartes graphiques, HRa Scope, etc.)
com.hraccess.openhr.event	Fournit les classes relatives à la gestion d'événements
com.hraccess.openhr.exception	Fournit les classes d'exception levées par l'API
com.hraccess.openhr.helper	Fournit des classes utilitaires permettant notamment de manipuler la réglementation pour créer les listes de codes réglementaires
com.hraccess.openhr.job	Fournit des classes pour manipuler un dossier de travail (de structure de données ZO)
com.hraccess.openhr.job.query	Fournit des classes pour manipuler un dossier de requête (dossier de phase NPQ)
com.hraccess.openhr.msg	Fournit les classes représentant les messages de type requête / réponse échangés avec le serveur OpenHR
com.hraccess.openhr.query	Fournit les classes pour soumettre de manière synchrone / asynchrone une requête ou une présélection de population
com.hraccess.openhr.security	Fournit les classes pour personnaliser la connexion des utilisateurs
com.hraccess.openhr.topology	Fournit les classes pour récupérer la description de l'objet Topologie système.

CHAPITRE 4

Classes principales

A propos de ce chapitre

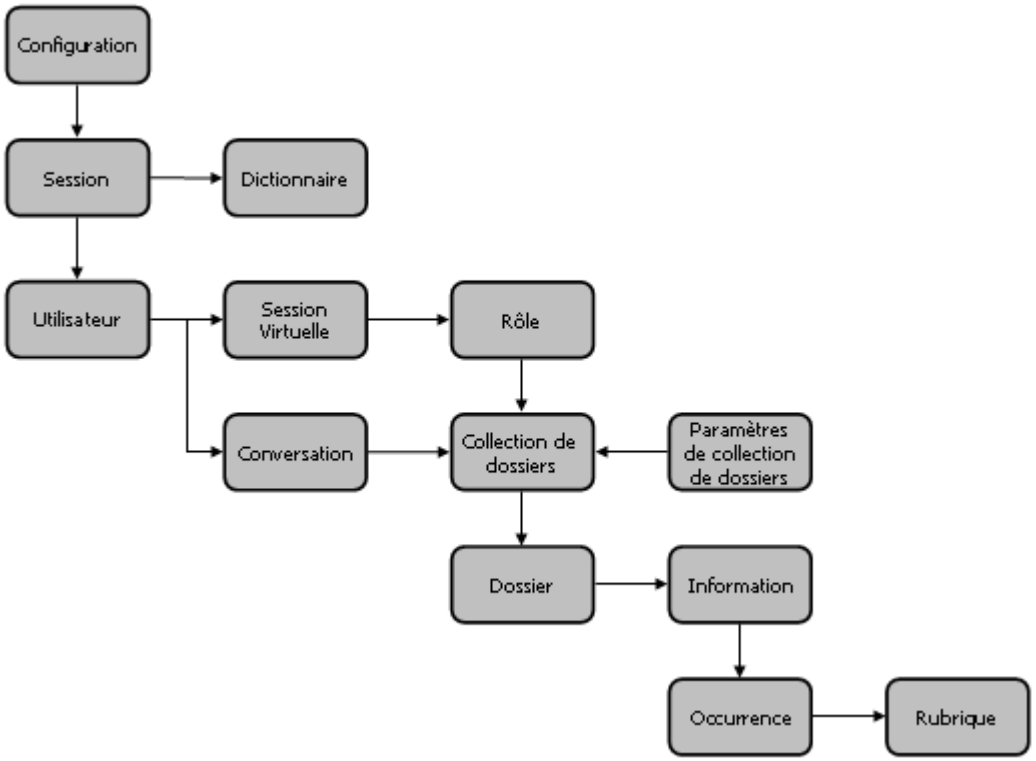
Ce chapitre passe en revue le rôle des principaux objets de l'API OpenHR. Ils sont présentés dans un ordre logique qui correspond à l'ordre dans lequel les objets doivent être créés.

En particulier, il donne des détails sur :

- La session
- Le dictionnaire
- La session virtuelle
- L'utilisateur
- Le rôle
- La conversation
- L'utilisateur de session
- La gestion de dossiers
- L'accès bas niveau aux données

Schéma global des liens inter-objets

Le schéma ci-dessous illustre la relation entre les principaux objets de l'API. Une flèche dirigée d'une classe A vers une classe B indique que la première est une dépendance de la seconde.



Ainsi, pour pouvoir manipuler le dictionnaire (qui contient le modèle des données), il faut disposer d'une session et d'une configuration de session.

La session

La **session OpenHR** (ou tout simplement "session" dans la suite de ce chapitre) représente une conversation de travail avec le serveur HR Access.

La session est ouverte à l'issue d'une opération d'ouverture de session et fermée à l'issue d'une opération de fermeture de session. On utilisera aussi les termes de connexion ou de déconnexion de session par la suite de manière interchangeable avec ceux d'ouverture / fermeture de session.

Lorsque l'on développe une application cliente OpenHR, on ouvre généralement la session au démarrage de l'application cliente et on la ferme à l'arrêt de l'application cliente. L'ouverture d'une session est potentiellement coûteuse en performances (en raison de la construction du dictionnaire - voir la section qui lui est consacrée, plus loin dans ce chapitre). C'est pourquoi on n'utilise en général qu'une seule session par application cliente. Il est techniquement possible d'utiliser plusieurs sessions OpenHR par application cliente mais cela ne présente en général aucun avantage.

La création d'une session est primordiale car c'est grâce à elle que l'on peut :

- Emettre des messages de requête (d'appel de service) au serveur OpenHR (et à travers lui, au serveur HR Access)
- Récupérer une réponse décrivant le résultat de cette invocation.

Une session est représentée par une instance de l'interface com.hraccess.openhr.IHRSession. Il n'est pas possible d'instancier soi-même une session, il faut passer par une classe de fabrique.

Pour créer une session il faut préalablement décrire sa configuration.

Configuration

L'API OpenHR utilise l'API Commons Configuration pour gérer la configuration d'une session. Cela offre une grande souplesse dans la définition de la configuration : celle-ci peut être :

- Définie de manière statique sous forme :
  - d'un fichier de propriétés
  - d'un fichier XML
- Ou créée dynamiquement par programmation

Une configuration de session OpenHR est représentée par une instance de l'interface org.apache.commons.configuration.Configuration. La classe Configuration peut être assimilée à une Map car elle référence des clés auxquelles sont associées des valeurs. La clé représente le nom d'un paramètre de configuration tandis que la valeur associée représente la valeur de ce paramètre. L'interface Configuration fournit des méthodes pour lire et modifier les propriétés de configuration.

La manière la plus simple de récupérer une configuration de session OpenHR consiste à charger un fichier de propriétés contenant des paires clé / valeur. Traditionnellement, ce fichier se nomme "openhr.properties". Ce nom sera donc utilisé dans la suite de cette documentation pour désigner le fichier de configuration du client OpenHR.

L'exemple suivant montre le contenu d'un tel fichier de configuration.

```
session.language[2]=U
session.process_list=
session.work_directory=C:/temp/openhr

openhr_server.server=10.11.12.13

normal_message_sender.security=disabled
normal_message_sender.port=8800

sensitive_message_sender.security=disabled
sensitive_message_sender.port=8800

privilegied_message_sender.security=disabled
privilegied_message_sender.port=8800
```

Le détail de chaque propriété de ce fichier n'est pas fourni à ce stade de la documentation. Cependant, la liste des noms de paramètres (clés) de configuration disponibles sera détaillée de manière exhaustive plus loin.

Pour charger une telle configuration, il faut écrire le code suivant :

```
// Retrieving the File containing the configuration
File file = new File("C:\\openhr\\conf\\openhr.properties");

// Loading the configuration from the given properties file
Configuration configuration = new PropertiesConfiguration(file);
```

L'API Commons Configuration permet aussi de créer complètement la configuration de manière programmatique comme le montre l'exemple suivant. Dans ce cas, la configuration obtenue est identique à celle chargée précédemment à partir du fichier de propriétés.

```
// Creating a new configuration
Configuration configuration = new PropertiesConfiguration();

// Populating the configuration
configuration.setProperty("session.language"[2], "U");
configuration.setProperty("session.process_list", "");
configuration.setProperty("session.work_directory", "C:/temp/openhr");

configuration.setProperty("openhr_server.server", "10.11.12.13");
```

[2] *Déprécié en 7.30.50 ou supérieure, utiliser session.languages à la place (voir le paragraphe "Propriétés de configuration").*

```
configuration.setProperty("normal_message_sender.security", "disabled");
configuration.setProperty("normal_message_sender.port", "8800");

configuration.setProperty("sensitive_message_sender.security",
   "disabled");
configuration.setProperty("sensitive_message_sender.port", "8800");

configuration.setProperty("privilegied_message_sender.security",
   "disabled");
configuration.setProperty("privilegied_message_sender.port", "8800");
```

Les possibilités de configuration sont nombreuses et si les deux exemples illustrés ici ne correspondent pas à vos attentes, il existe certainement une classe de l'API Commons Configuration qui couvre votre besoin. Pour plus d'informations, reportez-vous à la documentation du projet Commons Configuration.

Configuration de la log

L'API OpenHR utilise une couche d'abstraction afin de logger ses messages à la manière de l'API Commons Logging. Par défaut, c'est l'API Jakarta Log4J qui sert de système de log mais il est possible de le changer pour utiliser l'API Commons Logging ou pour logger vers la sortie standard.

Pour sélectionner l'implémentation du système de log, il faut utiliser la classe com.hraccess.openhr.HRApplication de la manière suivante. La classe HRLoggingSystem énumère sous forme de constantes les différentes implémentations de système de log supportées par l'API OpenHR.

```
// Configuring logging system to use Log4J
HRApplication.setLoggingSystem(HRLoggingSystem.LOG4J);

// Configuring logging system to use Commons Logging
HRApplication.setLoggingSystem(HRLoggingSystem.COMMONS_LOGGING);

// Configuring logging system to use standard output
HRApplication.setLoggingSystem(HRLoggingSystem.STANDARD);
```

Une fois que le système de log est sélectionné, il faut le configurer. Pour Log4J, cela consiste à charger un fichier de propriétés généralement nommé log4j.properties. L'exemple suivant illustre comment configurer le système de log sous-jacent à l'aide d'un fichier de configuration.

```
// Configuring logging system to use Log4J
HRApplication.setLoggingSystem(HRLoggingSystem.LOG4J);

// Configuring Log4J from given configuration file
```

```
HRApplication.configureLogs("C:\\openhr\\conf\\log4j.properties");
```

Où le contenu du fichier log4j.properties est le suivant :

```
log4j.rootCategory=DEBUG, A1

# A1 is set to be a ConsoleAppender (writes to system console).
log4j.appender.A1=org.apache.log4j.ConsoleAppender

# A1 uses PatternLayout.
log4j.appender.A1.layout=org.apache.log4j.PatternLayout
log4j.appender.A1.layout.ConversionPattern=%p %c{2} %x - %m%n
```

Les possibilités de configuration sont nombreuses. Pour plus d'informations sur la manière de configurer les logs, reportez-vous à la documentation du projet Jakarta Log4J.

Connexion

Il n'est pas possible d'instancier directement une IHRSession. C'est pourquoi il faut utiliser la méthode de fabrique createSession(Configuration) de la classe com.hraccess.openhr.HRSessionFactory.

```
// Retrieving the configuration to create a new session
Configuration configuration = getConfiguration();

// Creating and connecting a new session from given configuration
IHRSession session =
    HRSessionFactory.getFactory().createSession(configuration);
```

L'appel de la méthode HRSessionFactory.createSession(Configuration) :

- Crée une nouvelle instance de IHRSession
- Lui transmet la configuration donnée pour lui permettre de s'initialiser
- Connecte la session OpenHR au serveur HR Access.

La session récupérée en retour de méthode est donc déjà connectée et prête à l'emploi.

Déconnexion

Pour déconnecter une session, il faut appeler la méthode IHRSession.disconnect().

```
// Retrieving an existing session
IHRSession session = getSession();

// Disconnecting the session
session.disconnect();
```

Une fois déconnectée, une session ne peut plus être reconnectée. Il faut alors créer une nouvelle instance de IHRSession via la classe HRSessionFactory.

Notification des événements

Il est possible d'écouter les événements levés par une IHRSession. Il faut alors implémenter l'interface com.hraccess.openhr.event.SessionChangeListener et s'enregistrer en tant qu'auditeur auprès d'une session connectée. Les événements levés par une IHRSession sont représentés par la classe com.hraccess.openhr.event.SessionChangeEvent.

Etant donné qu'une session ne peut être reconnectée et que sa récupération et sa connexion sont prises en charge par la méthode HRSessionFactory.createSession(Configuration), le seul événement qu'il est possible d'intercepter correspond à la déconnexion de la session.

Pour ne plus être notifié des événements d'une session, il faut utiliser la méthode IHRSession.removeSessionChangeListener(SessionChangeListener).

```
// Retrieving an existing session
IHRSession session = getSession();

SessionChangeListener listener = new SessionChangeListener() {
    public void sessionStateChanged(SessionChangeEvent event) {
    IHRSession session = (IHRSession) event.getSource();

    if (!session.isConnected()) {
            // Process the session's disconnection event (not detailed here)
    }
    }
};

session.addSessionChangeListener(listener);

// Disconnecting the session
session.disconnect();

session.removeSessionChangeListener(listener);
```

Propriétés de configuration

La configuration d'une session est composée d'un ensemble de paires clé / valeur appelées propriétés de configuration.

Les propriétés d'une configuration ont un nom de la forme suivante :

```
Nom (complet) de propriété = préfixe de configuration + “.” + nom (local)
   de la propriété
```

Le préfixe associé à chaque propriété permet de les catégoriser et de rendre le fichier de configuration plus lisible. Ainsi, l'aspect de la configuration auquel se rattache la propriété devient évident.

Dans l'exemple ci-dessous, le fichier utilise cinq préfixes de configuration : "session", "openhr_server", "normal_message_sender", "sensitive_message_sender" et "privilegied_message_sender".

```
session.language[3]=U

session.process_list=

session.work_directory=C:/temp/openhr


openhr_server.server=10.11.12.13


normal_message_sender.security=disabled

normal_message_sender.port=8800


sensitive_message_sender.security=disabled

sensitive_message_sender.port=8800


privilegied_message_sender.security=disabled

privilegied_message_sender.port=8800
```

La liste des préfixes qu'il est possible d'utiliser est une liste ouverte : la plupart des préfixes sont fixés par l'API mais il existe un cas pour lequel l'installateur peut utiliser un préfixe de son choix (ce cas sera détaillé plus loin).

La classe com.hraccess.openhr.configuration.ConfigurationKeys centralise les préfixes de configuration et les noms locaux des propriétés sous forme de constantes :

- Les constantes de la forme PREFIX_XXX listent les préfixes de configuration
- Les constantes de la forme PROPERTY_XXX listent les noms (locaux) de propriété de configuration

[3] *Déprécié en 7.30.50 ou supérieure, utiliser session.languages à la place (voir le tableau suivant).*

Pour former un nom complet de propriété, il faut associer un préfixe à un nom (local) de propriété. Cependant, toutes les combinaisons ne sont pas possibles : certaines propriétés ne sont pertinentes que pour un préfixe donné. Le tableau suivant liste les noms complets de propriété valides et leur signification.

Nom de propriété	Rôle
session.work_directory	Propriété obligatoire. Définit le chemin du répertoire de travail de la session OpenHR. Les fichiers de travail de la session y seront stockés. Exemple : "C:\openhr\work"
session.language Déprécié en 7.30.50 ou supérieure, utiliser session.languages à la place.	Propriété obligatoire. Définit la langue de travail de la session OpenHR (celle-ci n'est donc pas internationalisée). Sert entre autres à récupérer les libellés des objets constituant le dictionnaire (structures de données, informations, etc.). Les valeurs valides correspondent aux codes langue définis par HR Access. Exemple : "U" (Anglais), "F" (Français)
session.languages Version 7.30.50 ou supérieure	Propriété obligatoire. Définit les langues de travail de la session OpenHR. Sert entre autres à récupérer les libellés des objets constituant le dictionnaire (structures de données, informations, etc.). Les valeurs valides correspondent aux codes langue définis par HR Access. Les valeurs doivent être séparées par des virgules (sans espaces). La première langue spécifiée est considérée comme la langue par défaut. Exemple : "U" (Anglais), "F" (Français), "F,U" (Français et Anglais)

Nom de propriété	Rôle
session.application_id	Propriété facultative. Identifie l'application logique à laquelle est rattachée la session OpenHR. Le concept d'application logique est à rapprocher de celui de cluster. Si la propriété est renseignée, alors le serveur HR Access ouvre la session et la rattache au cluster de nom donné ; autrement, un cluster virtuel (dont l'identifiant est généré de manière unique par le serveur HR Access) est ouvert et la session ouverte y est rattachée. Dans ce dernier cas, la session est assurée d'être isolée dans son propre cluster.
session.user_cache_max_size	Propriété facultative. Définit le nombre maximal d'utilisateurs que la session OpenHR est capable de stocker en cache. Le cache est de type LRU (Least Recently Used) si bien que si le cache est saturé d'objets utilisateur, les entrées les plus anciennes du cache sont évincées au fur et à mesure de son remplissage afin de conserver une taille fixe. Valeur par défaut : 50.
session.max_retry_to_wait_dispatcher	Propriété facultative. Au démarrage, la session contacte le serveur OpenHR (également appelé historiquement "dispatcher" d'où le nom de la propriété) afin de connaître son état. Les deux états possibles sont "en cours de démarrage" ou "démarré". Si l'état est "en cours de démarrage", la session peut attendre un temps donné que le serveur OpenHR ait fini de démarrer, et cela, plusieurs fois de suite, afin de ne pas entraver le démarrage du client OpenHR. Cette propriété indique le nombre maximal de fois que la session attendra que le serveur ait démarré avant de lever une erreur. Valeur par défaut : 0

Nom de propriété	Rôle
session.retry_interval_to_wait_dispatcher	Propriété facultative. Indique (en secondes) le temps que le client OpenHR accorde au serveur OpenHR pour passer de l'état « en cours de démarrage » à l'état « démarré ». cf propriété « session.max_retry_to_wait_dispatcher ». Valeur par défaut : 60 secondes.
session.process_list	Propriété obligatoire. Enumère sous forme d'une liste les noms de processus (séparés par des virgules) qui seront utilisés par la session OpenHR pour manipuler les dossiers HR Access. Cette liste sert à construire le dictionnaire rattaché à la session OpenHR. Les processus listés ici doivent être de qualification "Gestion de dossiers", de type conversationnel ou mixte, déclarés au niveau de la plate-forme physique et compilés.
session.mass_update_limit	Propriété facultative. Définit le seuil (en nombre de dossiers) au-delà duquel une mise à jour de dossiers HR Access de type « répercussion collective » bascule du mode “temps réel” en mode “asynchrone”. En mode temps réel, les mises à jour sur dossiers sont appliquées immédiatement aux données en table. En mode asynchrone, les données sont stockées dans une table technique du serveur HR Access et il faut exécuter une chaîne batch afin de les appliquer aux données en table.
session.user_cache_scavenger_enabled	Propriété facultative. Indique si le thread d'épuration du cache des utilisateurs doit être exécuté au démarrage de la session. Ce thread permet de vider le cache des entrées correspondant à des utilisateurs dont la connexion a été fermée. Cf. propriété "session.user_cache_scavenger_period" Valeurs valides : true, false Valeur par défaut : true

Nom de propriété	Rôle
session.user_cache_scavenger_period	Propriété facultative. Indique (en secondes) le temps d'attente entre deux exécutions du thread d'épuration du cache des utilisateurs. Cf propriété « session.user_cache_scavenger_enabled ». Valeur par défaut : 1h (3 600 sec).
session.preload_role_template_cache	Propriété facultative. Indique si le cache de modèles de rôle interne à la session doit être pré-chargé au démarrage. Ce paramètre permet d'optimiser la récupération des informations décrivant les modèles de rôle. Si le pré-chargement est désactivé, les modèles de rôle sont alors récupérés au fur et à mesure de l'exécution en fonction des besoins (lazy loading). Cf. propriété "session.cached_role_template_life_time" Valeurs valides : true, false Valeur par défaut : false
session.cached_role_template_life_time	Propriété facultative. Indique (en secondes) le temps maximal de résidence en cache d'un modèle de rôle. Passé ce délai, l'entrée de cache est considérée comme invalide et sera rechargée à partir du serveur HR Access. Cf. propriété "session.preload_role_template_cache". Valeur par défaut : 0 sec (pas de mise en cache)
session.cache_directory	Propriété facultative. Définit le chemin du répertoire de cache de la session OpenHR. Par défaut, le répertoire de cache correspond au répertoire de travail de la session.

Nom de propriété	Rôle
session.default_role_precision	Propriété facultative. Lorsqu'un utilisateur se connecte, le serveur HR Access résout ses rôles. Or un rôle possède de nombreuses propriétés dont certaines sont calculées à l'aide d'un traitement spécifique COBOL du programme BCR. Le calcul de toutes ces propriétés peut se révéler coûteux. C'est pourquoi par défaut, les rôles d'un utilisateur sont résolus avec un niveau de précision faible. Si l'une des propriétés du rôle est demandée et n'a pas encore été calculée, une requête au serveur est émise pour calculer celle-ci dynamiquement. La propriété "session.default_role_precision" permet d'indiquer au serveur HR Access le niveau de précision par défaut qu'il doit utiliser lors du calcul des rôles d'un utilisateur connecté. Dans certaines conditions, il est possible d'optimiser les performances en demandant immédiatement à la connexion de l'utilisateur le calcul de toutes les propriétés de rôles. Valeurs valides : 1, 2, 3, 4 ou "max" Valeur par défaut : 1

Nom de propriété	Rôle
session.login_module_hint	Propriété facultative. La session permet de connecter des utilisateurs. Il est possible de personnaliser le processus de connexion afin de connecter des utilisateurs décrits de manière spécifique (Single Sign-On, connexion auprès d'un annuaire LDAP, etc.). A chaque type de description d'utilisateur spécifique est associée une logique de traitement de connexion implémentée sous la forme d'un Login Module et hébergée par le serveur OpenHR. Lorsque plusieurs implémentations de Login Module sont disponibles au niveau du serveur OpenHR, le client OpenHR peut, lors de la demande de connexion d'utilisateur, en "sélectionner" une (indirectement) en fournissant, en même temps que la description d'utilisateur, une indication (hint en anglais) sur la manière dont cette description doit être traitée. Le paramètre "session.login_module_hint" permet de spécifier l'indication qui est envoyée, lors des demandes de connexion d'utilisateur, au serveur OpenHR **lorsque le client OpenHR n'en fournit aucune explicitement**. Valeur par défaut : null
session.dictionary_name	Propriété facultative. Permet de modifier le nom sous lequel le dictionnaire local sera sérialisé dans le répertoire de cache de la session. Par défaut, le dictionnaire est sérialisé dans un fichier (*.dic) dont le nom correspond au nom de la plate-forme HR Access cible. Exemple : PPCLIUNO.dic

Nom de propriété	Rôle
session.dictionary_refresh_packet_size	Définit la taille des paquets d'objets lus lors de la phase de rafraîchissement du dictionnaire. Les objets sont des structures de données, des informations, des rubriques, des types de dossier ou des liens. Si trop d'objets dans le dictionnaire doivent être rapatriés depuis le serveur HR Access, il est nécessaire de lire celles-ci par paquets afin de contourner certaines limitations de taille mémoire du serveur HR Access. Valeur par défaut : 250

Nom de propriété	Rôle
session.dictionary_keep_unused_entities	Propriété facultative. Dans certaines circonstances, la liste des processus rattachés à la session OpenHR ne peut être connue qu'une fois la session démarrée. C'est par exemple le cas si cette liste est calculée à partir de ressources locales référençant des processus (scripts de post-production, scripts de jauge, etc.). En l'absence de ce paramétrage, une reconstruction complète (et donc pénalisante) du dictionnaire est nécessaire à chaque redémarrage de l'application : lors du démarrage, la liste des processus étant inconnue, la session est ouverte avec une liste vide et donc un dictionnaire vide. Puis la liste des processus est calculée à partir des ressources locales et reportée au niveau du dictionnaire qui est alors rafraîchi dynamiquement. Cette opération rapatrie la description des objets référencés par cette liste de processus. Au prochain démarrage, le dictionnaire local qui comporte des objets se retrouve vidé de tous ses objets pour la raison donnée plus haut et la reconstruction complète du dictionnaire lors de la seconde phase a lieu à chaque redémarrage. Pour éviter un tel problème, ce paramètre permet de conserver dans le dictionnaire local les objets qui s'y trouvent mais ne sont plus utilisés par la liste des processus du dictionnaire. Ces objets pourront alors être réutilisés si les processus en question sont rajoutés au dictionnaire. Valeurs valides : true, false Valeur par défaut : false
session.dictionary_refresh_on_startup	Propriété facultative. Indique si le rafraîchissement du dictionnaire au démarrage de la session OpenHR doit être forcé. Ce paramètre permet de reconstruire le dictionnaire, que celui-ci soit déphasé ou non. Valeurs valides : true, false Valeur par défaut : false

Nom de propriété	Rôle
session.check_process_main_datastructure	Propriété facultative. L'API émet au serveur OpenHR certains messages qui portent sur une structure de données comme ceux de lecture / mise à jour de dossiers par exemple. Ces messages sont traités par un processus HR Access qui doit être configuré au niveau de la session (cf. propriété session.process_list). La propriété "session.check_process_main_datastructure" permet d'activer un contrôle qui vérifie que la structure de données sur laquelle porte le message émis correspond bien à la structure de données principale du processus utilisé pour traiter ce message. Valeurs valides : true, false Valeur par défaut : true
openhr_server.server	Propriété obligatoire. Indique le nom d'hôte ou l'adresse IP de la machine sur laquelle est hébergé le serveur OpenHR.
openhr_server.block_size	Propriété facultative. Indique la taille des blocs de message (en nombre de caractères, pas d'octets) échangés entre le client et le serveur OpenHR. Valeur par défaut : 8000
openhr_server.encoding	Propriété facultative. Indique la page de code (jeu de caractères) utilisée afin de convertir les caractères des messages en octets. Cette page de code doit correspondre à celle utilisée par le serveur OpenHR lors de la conversion inverse (octets vers caractères). Valeur par défaut : le code page par défaut de la JVM d'exécution du client OpenHR.
openhr_server.timeout	Propriété facultative. Indique le délai (en secondes) imparti au serveur OpenHR pour traiter un message de requête avant que le client OpenHR ne lève une erreur. Toute valeur négative ou nulle est interprétée comme un délai infini (pas de timeout). Valeur par défaut : 0 sec (pas de timeout).

Les propriétés suivantes peuvent être utilisées avec les trois préfixes :

- "normal_message_sender"
- "sensitive_message_sender"
- "privilegied_message_sender"

Pour simplifier l'explication, on utilisera un préfixe symbolique "abc" pour représenter au choix l'un de ces trois préfixes.

Nom de propriété	Rôle
abc.security	Propriété obligatoire. Indique le niveau de sécurité appliqué au transport des messages vers le serveur OpenHR. Il n'existe que trois valeurs valides : "disabled", "SSL" et "SSL2". La valeur "disabled" indique que le transport des messages n'est pas sécurisé. La valeur "SSL" indique une sécurisation de type 1 (authentification du serveur et chiffrage des données) du transport des messages. La valeur "SSL2" indique une sécurisation de type 2 (authentification mutuelle du client et du serveur et chiffrage des données) du transport des messages. Le caractère obligatoire ou facultatif des propriétés qui suivent dépend de la valeur donnée à cette propriété. Valeurs valides : disabled, SSL, SSL2
abc.use_configuration_with_prefix	Propriété facultative. L'utilité de cette propriété est expliquée plus loin. L'utilisation de cette propriété est possible quelle que soit la valeur de la propriété "security".
abc.port	Propriété obligatoire. Définit le numéro de port sur lequel communiquer avec le serveur OpenHR.
abc.ca_certificate_filename	Propriété obligatoire si "security" est à "SSL" ou "SSL2". Définit le nom du fichier keystore CA (Certification Authority) permettant d'authentifier le serveur OpenHR.
abc.ca_certificate_password	Propriété facultative si "security" est à "SSL" ou "SSL2", interdite autrement. Définit le mot de passe utilisé afin de vérifier l'intégrité du keystore CA permettant d'authentifier le serveur OpenHR.
abc.ca_key_management_algorithm	Propriété facultative si "security" est à "SSL" ou "SSL2", interdite autrement. Définit le nom de l'algorithme de gestion des clés à utiliser pour lire le fichier keystore CA. Valeur par défaut : l'algorithme par défaut configuré au niveau de la JVM d'exécution du client OpenHR.

Nom de propriété	Rôle
abc.certificate_filename	Propriété obligatoire si "security" est à "SSL2". Définit le nom du fichier keystore contenant le certificat utilisé pour authentifier le client OpenHR auprès du serveur.
abc.certificate_password	Propriété obligatoire si "security" est à "SSL2". Définit le mot de passe à utiliser afin d'authentifier le client OpenHR à l'aide de son certificat.
abc.key_management_algorithm	Propriété facultative si "security" est à "SSL2", interdite autrement. Définit le nom de l'algorithme de gestion des clés à utiliser pour lire le fichier keystore. Valeur par défaut : l'algorithme par défaut configuré au niveau de la JVM d'exécution du client OpenHR.

Configuration de base

La configuration suivante correspond à la configuration minimale d'un client OpenHR.

```
session.language = U                                                    (1)
session.process_list = FS001,FS00B,AS906
        (2)
                                                                        (3)
openhr_server.server = myserver.mydomain.com
   (4)
                                                                        (5)
normal_message_sender.security = disabled
        (6)
normal_message_sender.port = 8000
   (7)
                                                                        (8)
sensitive_message_sender.security = disabled
   (9)
sensitive_message_sender.port = 8000
   (10)
                                                                        (11)
privilegied_message_sender.security = disabled
        (12)
privilegied_message_sender.port = 8000
   (13)
```

Détail du contenu du fichier

- La ligne (1) indique la langue (U = Anglais) dans laquelle la session OpenHR doit être connectée. Cette langue sert entre autres à récupérer le libellé des objets (les structures de données par exemple) qui constituent le dictionnaire de la session. En 7.30.50 ou supérieure, la propriété "session.language" est dépréciée, il faut utiliser "session.languages" à la place (voir le paragraphe "Propriétés de configuration").
- La ligne (2) énumère, sous forme d'une liste séparée par des virgules, les noms des processus HR Access qui seront invoqués par la session. Cette liste peut être vide. Elle sert à déterminer la liste des structures de données, informations, rubriques, liens et types de dossier du dictionnaire.
- La ligne (4) identifie la machine qui héberge le serveur OpenHR cible. La valeur de la propriété peut représenter un nom de machine ou une adresse IP.
- Les lignes (6) et (7) définissent le paramétrage du message sender chargé d'émettre les messages de nature "normal" vers le serveur OpenHR.
  - La propriété "security" alimentée à "disabled" indique que le transport des messages n'est pas sécurisé : la communication s'effectue en clair via deux sockets.
  - La propriété "port" indique le numéro de port sur lequel le serveur OpenHR distant réceptionne les messages de nature "normal".
- Les lignes (9) et (10) sont similaires aux lignes (6) et (7) sauf que le paramétrage s'applique au message sender chargé d'émettre les messages de nature "sensitive" vers le serveur OpenHR.
- Les lignes (12) et (13) sont similaires aux lignes (6) et (7) sauf que le paramétrage s'applique au message sender chargé d'émettre les messages de nature "privilegied" vers le serveur OpenHR.

Dans cet exemple, les ports définis en lignes (7), (10) et (13) sont identiques mais ce n'est pas obligatoire. En effet, le serveur OpenHR peut être configuré de sorte à recevoir les trois natures de messages sur un même port d'écoute ou sur des ports différents. Pour plus de détails sur la manière de configurer le serveur OpenHR, reportez-vous au chapitre afférent dans le *Guide technique tous systèmes HR Design*.

La propriété abc.use_configuration_with_prefix

L'exemple précédent correspond à la configuration minimale d'un client OpenHR. Cependant, elle présente un inconvénient : si l'on souhaite par exemple modifier le numéro de port du serveur OpenHR distant, il faut faire trois modifications dans le fichier (en lignes 7, 10 et 13). Cela n'est pas bien long dans le cas ci-dessus mais si chacun des messages senders est configuré en mode "SSL2", il faut alors renseigner au minimum 4 propriétés pour chacun d'eux. Cela fait alors 12 propriétés à modifier en tout. Pour éviter les redondances dans le fichier de configuration, il est possible de définir des propriétés de configuration à l'aide d'un préfixe non réservé (c'est-à-dire libre ou non imposé par la configuration de l'API OpenHR) puis de faire référence à ces propriétés pour les réutiliser. C'est le seul cas de figure où des propriétés de configuration peuvent être qualifiées à l'aide d'un préfixe libre.

La propriété "abc.use_configuration_with_prefix" est disponible pour les trois préfixes "normal_message_sender", "sensitive_message_sender" et "privilegied_message_sender". Lorsqu'elle est valorisée à une valeur "xxx", le paramétrage du message sender concerné est alors récupéré en utilisant ce préfixe de surcharge.

L'exemple précédent peut alors être réécrit de la manière suivante :

```
session.language[4] = U
session.process_list = FS001,FS00B,AS906

openhr_server.server = myserver.mydomain.com

shared_configuration.port = 8000

normal_message_sender.security = disabled
normal_message_sender.use_configuration_with_prefix =
    shared_configuration

sensitive_message_sender.security = disabled
sensitive_message_sender.use_configuration_with_prefix =
    shared_configuration

privilegied_message_sender.security = disabled
privilegied_message_sender.use_configuration_with_prefix =
    shared_configuration
```

Lorsque l'on surcharge le préfixe de configuration (abc) d'un message sender, les propriétés avec ce préfixe (à l'exception des propriétés "abc.security" et "abc.use_configuration_with_prefix") ne sont plus utilisées et peuvent donc être supprimés de la configuration. Dans l'exemple ci-dessus, le numéro de port n'est plus défini qu'à un seul endroit.

[4] *Déprécié en 7.30.50 ou supérieure, utiliser session.languages à la place (voir le paragraphe "Propriétés de configuration".*

Le dictionnaire

Introduction

Cette section présente en détail la manière dont le modèle de données HR Access est défini au niveau de l'API OpenHR. Sa lecture n'est pas indispensable car il est possible de manipuler des dossiers HR Access sans nécessairement connaître en détail comment le modèle de données des dossiers est décrit par l'API. Si vous êtes dans cette situation, vous pouvez directement passer à la section suivante.

La lecture de cette section peut cependant se révéler utile si vous souhaitez exploiter la description du modèle de données HR Access afin, par exemple, de générer du code à partir du modèle de données (Model Driven Architecture). Un tel exemple d'application serait la génération d'un service d'accès aux données (DAO) HR Access basé sur l'API JDBC.

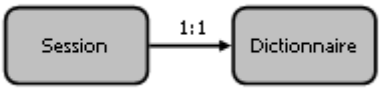
Définition

Le **dictionnaire** est le référentiel local qui contient la description du modèle de données manipulé. Il est créé lors de la connexion de la session et peut être remis à jour dynamiquement alors que celle-ci est connectée (opération de "rafraîchissement" du dictionnaire).

La session possède un unique dictionnaire que l'on peut récupérer, lire et modifier mais l'API permet également de créer des dictionnaires annexes (comme cela sera illustré plus loin dans la documentation).

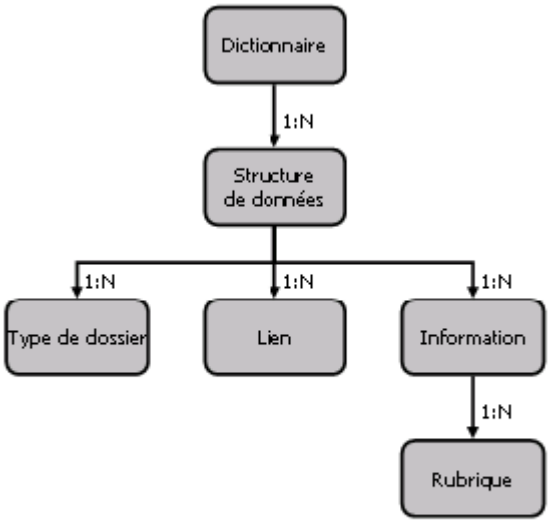
Un dictionnaire est représenté par une instance de la classe com.hraccess.openhr.IHRDictionary.

La session OpenHR possède un dictionnaire créé lors de la connexion :



Structure

Le dictionnaire se présente sous une forme arborescente. Les nœuds de cet arbre représentent le dictionnaire, les structures de données, les types de dossier, les liens, les informations et les rubriques :



HR Access

Le dictionnaire peut être représenté de manière arborescente où chaque nœud représente un objet Java de l'API :

- Le nœud racine correspond au dictionnaire. Le dictionnaire possède des nœuds enfants.
- Ces nœuds enfants représentent les structures de données. Une structure de données possède trois types de nœud enfant :
  - les types de dossiers (également appelés répertoires dans le cas d'une structure de données réglementaire)
  - les liens
  - les informations

Les types de dossiers et les liens sont des feuilles de l'arbre. Une information possède des nœuds enfants qui représentent les rubriques de l'information.

Génération du dictionnaire

Le dictionnaire est généré à partir de la liste des processus de la session OpenHR :

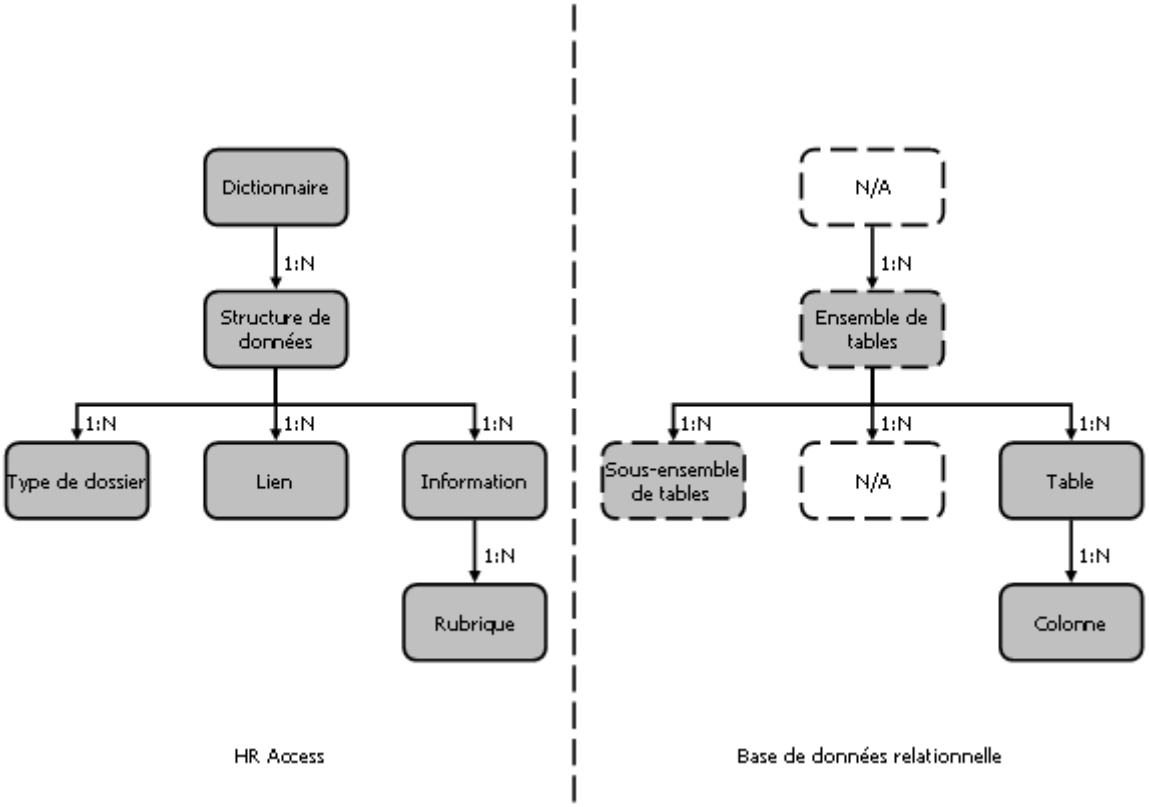
- Les structures de données du dictionnaire correspondent aux structures de données sur lesquelles sont définis les processus. Si la liste des processus de la session OpenHR (propriété "session.process_list") est alimentée à "FS001" (Informations sur le personnel), alors le dictionnaire contiendra la structure de données "ZY" (correspondant à la structure de données principale du processus FS001) mais aussi les structures de données secondaires référencées par le processus, à savoir ZA (Postes), ZC (Emplois), ZD (Réglementation RH) et ZE (Unités organisationnelles).
- Les informations du dictionnaire correspondent à l'union des informations paramétrées au niveau de tous les processus. Si la liste des processus de la session OpenHR (propriété "session.process_list") est alimentée à "FS001, FS00B", alors les informations de la structure de données "ZY" contenues dans le dictionnaire seront celles calculées par la formule : (Informations référencées uniquement par le processus FS001) + (Informations référencées uniquement par le processus FS00B) + (Informations communes aux processus FS001 et FS00B).
- Les rubriques du dictionnaire correspondent aux rubriques définies dans le modèle de données HR Access à l'aide de Design Center. Dès qu'une information apparaît dans le dictionnaire, toutes ses rubriques sont également présentes.
- Idem pour les types de dossiers. Dès qu'une structure de données apparaît dans le dictionnaire, tous ses types de dossiers sont également présents.

Idem pour les liens. Dès qu'une structure de données apparaît dans le dictionnaire, tous ses liens sont également présents.

Nœuds

Les nœuds de cette structure arborescente peuvent partiellement être mis en correspondance avec des concepts relatifs aux bases de données relationnelles.

Le modèle de données (à gauche) est exploité afin de générer les DDL des tables HR Access. Certains des concepts du dictionnaire trouvent une correspondance avec un concept de la base de données.



Le concept de **rubrique** correspond à la notion de colonne dans la base de données relationnelle : c'est d'ailleurs la définition de la rubrique qui sert à générer le fragment de DDL pour créer la colonne.

Le concept d'**information** correspond à la notion de table dans la base de données relationnelle : c'est d'ailleurs la définition de l'information (et de ses rubriques) qui sert à générer la DDL pour créer la table.

La notion de **structure de données** n'a pas de correspondance exacte dans la base de données relationnelle. Cependant, comme une structure de données regroupe un ensemble d'informations, elle peut être représentée par un ensemble de tables relationnelles présentant une "affinité fonctionnelle".

La notion de **type de dossier** n'a pas de correspondance exacte dans la base de données relationnelle. Cependant, comme un type de dossier regroupe un sous-ensemble d'informations d'une structure de données, il peut être représenté par un sous-ensemble des tables relationnelles représentant la structure de données.

La notion de **dictionnaire** n'a pas de correspondance dans la base de données relationnelle.

Construction du dictionnaire

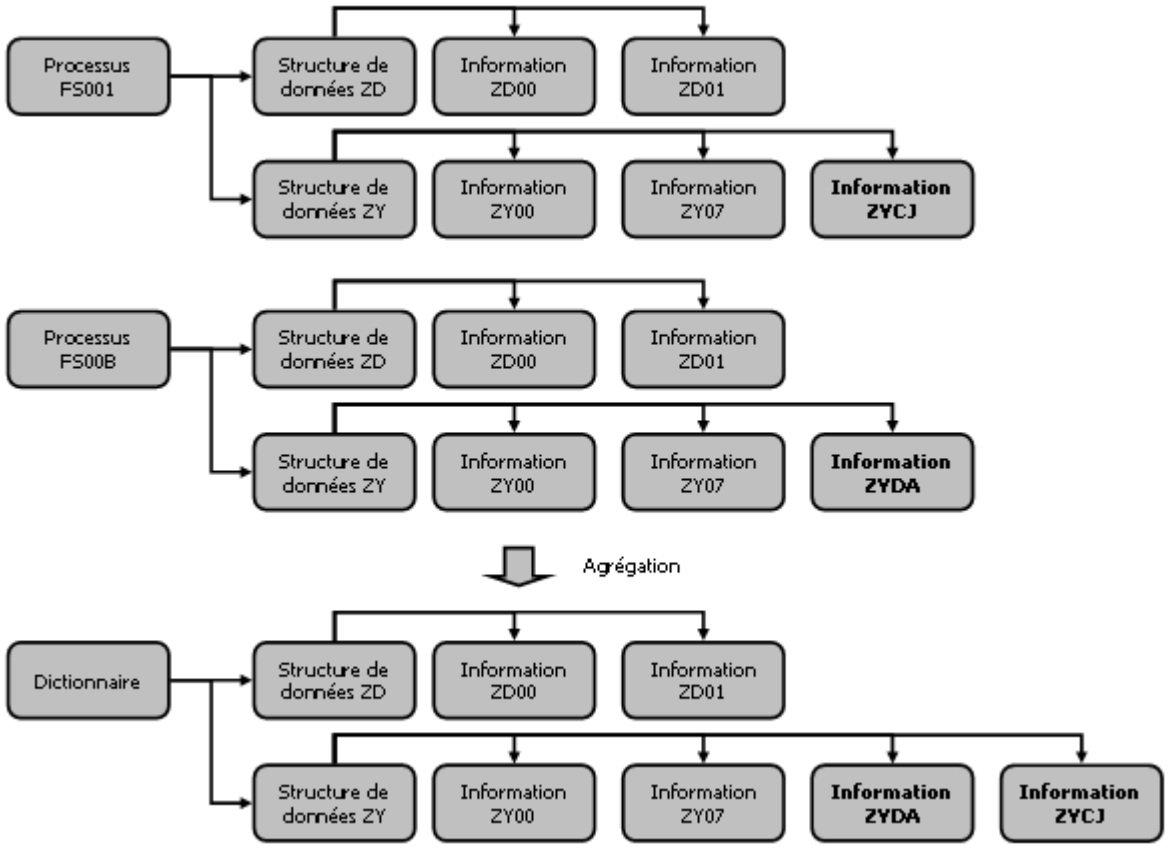
Lors de la connexion de la session OpenHR, l'API OpenHR envoie au serveur HR Access la liste des processus de travail (définie par la propriété "session.process_list" du fichier openhr.properties).

Le serveur HR Access :

- Vérifie que ces processus sont bien définis au niveau de la plate-forme physique et sont bien de type "GD" (Gestion de dossiers)
- Puis il récupère, pour chaque processus, la liste des structures de données et la liste des informations (rattachées explicitement) qu'il référence.
- Ensuite il fusionne la liste des structures de données de chaque processus en une liste unique afin de supprimer les doublons et fait de même pour les informations.

La résolution des types de dossiers et liens qui doivent être présents dans le dictionnaire est plus simple : dès lors qu'une structure de données est présente dans le dictionnaire, tous ses liens et types de dossiers sont également présents. De même, dès lors qu'une information est présente dans le dictionnaire, toutes ses rubriques le sont aussi.

Le processus de résolution des objets constituant le dictionnaire fusionne les objets référencés par chacun des processus du dictionnaire :



Dans l'exemple ci-dessus, le dictionnaire est construit à partir de deux processus "FS001" (Informations sur le personnel) et "FS00B" (Activités & Absentéisme). La définition de chaque processus permet de déterminer une liste de structures de données et, pour chacune d'elles, des informations. Le dictionnaire final résulte de la fusion de ces différentes listes de structures de données et d'informations.

Après calcul des objets référencés par la liste de processus, le serveur HR Access retourne cette liste au client OpenHR.

La session OpenHR récupère cette liste de structures de données, d'informations, de liens et de types de dossiers puis émet au serveur HR Access des demandes pour récupérer la description de chacun de ces objets. Ces descriptions sont récupérées soit depuis les tables DI** soit depuis les tables GE**. Les tables DI** contiennent la description des objets telles qu'ils sont visibles via Design Center alors que les tables GE** contiennent la description des objets telles qu'ils étaient en table DI** au moment de la compilation du processus. Ainsi, lors de la compilation d'un processus, la description des informations est recopiée en table GE**. Cela permet de continuer à modifier des objets sous Design Center sans impacter les processus déjà compilés avec l'ancienne version de l'objet.

Lorsque toutes les descriptions d'objet sont récupérées par la session OpenHR, celle-ci construit le dictionnaire en instanciant les nœuds de l'arbre et en les reliant entre eux. Au final, le dictionnaire est disponible sous forme d'un graphe d'objets Java.

Après construction, le dictionnaire est sauvegardé localement dans un fichier situé dans le répertoire de cache de la session OpenHR (qui est par défaut le répertoire de travail). Le nom de ce fichier est de la forme "<platform-name>.dic" où <platform-name> désigne le nom de la plate-forme physique HR Access, à moins que le nom ait été surchargé par la propriété "session.dictionary_name" du fichier openhr.properties.

Lors du prochain démarrage, le dictionnaire local caché est rechargé. Pour s'assurer que les objets du dictionnaire local sont toujours en phase avec leur contrepartie serveur, on utilise un système basé sur des horodatages représentant l'horodatage de dernière modification de l'objet. Si les horodatages local et serveur d'un objet sont égaux, la version locale est à jour ; autrement, cela signifie qu'elle a été modifiée sur le serveur depuis que le dictionnaire local a été créé. En cas de déphasage, le dictionnaire est reconstruit de manière différentielle : les objets locaux à jour sont réutilisés et seuls les objets déphasés sont récupérés du serveur. Cela permet d'optimiser l'opération de rafraîchissement du dictionnaire.

Récupération du dictionnaire

Pour récupérer le dictionnaire rattaché à la session, il suffit d'appeler la méthode IHRSession.getDictionary() de la manière suivante :

```
// Retrieving an existing session
IHRSession session = getSession();

// Retrieving the dictionary
IHRDictionary dictionary = session.getDictionary();
```

Le dictionnaire ainsi récupéré représente la racine d'un arbre que l'on peut exploiter.

Exploitation du dictionnaire

Propriétés

Le dictionnaire possède des propriétés qui reflètent en partie ce qui a été configuré dans le fichier openhr.properties. Le tableau suivant présente certaines des méthodes permettant de lire ces propriétés.

Méthode	Rôle
long getCompilationTimestamp()	Retourne sous forme d'un nombre de millisecondes l'horodatage auquel le dictionnaire a été créé (compilé).
char getLanguage() Déprécié en 7.30.50 et supérieure, utiliser getDefaultLanguage() à la place	Retourne sous forme d'un caractère le code langue par défaut des libellés du dictionnaire. Ce code langue correspond au paramètre "session.language"[5] ou à la première langue du paramètre "session.languages" du fichier openhr.properties.
char getDefaultLanguage()[6]	Retourne sous forme d'un caractère le code langue par défaut des libellés du dictionnaire. Ce code langue correspond à la première langue du paramètre "session.languages" (ou au paramètre "session.language"[5]) du fichier openhr.properties.
Set<Character> getLanguages()[6]	Retourne sous forme d'un Set<Character> les codes langues des libellés du dictionnaire. Ces langues correspondent au paramètre "session.languages" du fichier openhr.properties.
int getDataStructureCount()	Retourne le nombre de structures de données référencées par le dictionnaire.
Set<String> getProcesses()	Retourne sous forme d'un Set<String> le nom des processus HR Access utilisés pour construire le dictionnaire. Ce Set contient le nom des processus paramétré au niveau de la propriété "session.process_list" du fichier openhr.properties.
int getRefreshPacketSize()	Retourne sous forme d'entier le nombre d'objets lus à chaque chargement d'objets depuis le serveur HR Access. Ce nombre correspond à la valeur de la propriété "session.dictionary_refresh_packet_size" du fichier openhr.properties. Valeur par défaut : 250

[5] *Déprécié en 7.30.50 ou supérieure, utiliser session.languages à la place (voir le paragraphe "Propriétés de configuration".*
[6] *7.30.50 ou supérieure*

Méthode	Rôle
boolean isKeepUnusedEntities()	Indique si les objets devenus inutiles suite à un rafraîchissement de dictionnaire doivent être conservés dans celui-ci pour une éventuelle réutilisation ultérieure ou être supprimés. Cette propriété correspond à la valeur de la propriété "session. dictionary_keep_unused_entities" du fichier openhr.properties.

```
// Retrieving the dictionary from an existing session
IHRDictionary dictionary = getSession().getDictionary();


// Dumping some of the dictionary's properties
System.out.println("The dictionary was created on " + new
    Date(dictionary.getCompilationTimestamp()));
System.out.println("The dictionary's default language is <" +
    dictionary.getLanguage()[7] + ">");
System.out.println("The dictionary's processes are <" +
    dictionary.getProcesses() + ">");
System.out.println("The dictionary's refresh packet size is " +
    dictionary.getRefreshPacketSize());
System.out.println("Keeping dictionary's unused entities ? " +
    dictionary.isKeepUnusedEntities());
```

Récupération des structures de données

Une structure de données est représentée par l'interface com.hraccess.openhr.IHRDataStructure. Pour récupérer les structures de données d'un dictionnaire, il existe deux méthodes : IHRDictionary.getDataStructures() et IHRDictionary.getDataStructureMap().

La méthode IHRDictionary.getDataStructures() retourne une List<IHRDataStructure>.

```
// Retrieving the dictionary from an existing session
IHRDictionary dictionary = getSession().getDictionary();


// Retrieving the dictionary's data structures as a List<IHRDataStructure>
List dataStructures = dictionary.getDataStructures();


for(Iterator it=dataStructures.iterator(); it.hasNext();) {
    IHRDataStructure dataStructure = (IHRDataStructure) it.next();
    System.out.println("Found data structure <" + dataStructure.getName()
    + "> " + dataStructure.getLabel());
```

[7] *Déprécié en 7.30.50 et supérieure, utiliser getDefaultLanguage() à la place. Voir également getLanguages().*

```
}
```

La méthode IHRDictionary.getDataStructureMap(), elle, retourne une Map<String, IHRDataStructure> référençant les structures de données par nom.

```
// Retrieving the dictionary from an existing session
IHRDictionary dictionary = getSession().getDictionary();


// Retrieving the dictionary's data structures as a Map<String,
   IHRDataStructure>
Map dataStructures = dictionary.getDataStructureMap();


for(Iterator it=dataStructures.keySet().iterator(); it.hasNext();) {
   String dataStructureName = (String) it.next();
   IHRDataStructure dataStructure = (IHRDataStructure)
   dataStructures.get(dataStructureName);
   System.out.println("Found data structure <" + dataStructureName + "> "
   + dataStructure.getLabel()));
}
```

Récupération d'une structure de données par son nom

La méthode IHRDictionary.getDataStructureByName(String) permet de récupérer une structure de données à partir de son nom (code).

```
// Retrieving the dictionary from an existing session
IHRDictionary dictionary = getSession().getDictionary();


// Retrieving a named data structure
IHRDataStructure dataStructure =
   dictionary.getDataStructureByName("ZY");


if (dataStructure == null) {
   System.err.println("Unable to find data structure <ZY>");
} else {
   System.out.println("Found data structure <ZY> " +
   dataStructure.getLabel());
}
```

Tester l'existence d'une structure de données

La méthode IHRDictionary.hasDataStructure(String) permet de tester si la structure de données de nom donné existe.

```
// Retrieving the dictionary from an existing session
IHRDictionary dictionary = getSession().getDictionary();


// Testing the presence of a named data structure
System.out.println("Data structure <ZY> exists ? " +
    dictionary.hasDataStructure("ZY"));
```

Rafraîchissement du dictionnaire

L'opération dite de "**rafraîchissement**" du dictionnaire consiste à vérifier si celui-ci est en phase avec la version serveur et à le remettre à jour si ce n'est pas le cas.

Le dictionnaire est considéré comme déphasé si l'un de ses objets (une structure de données, une information, une rubrique, un lien, un type de dossier ou un processus) a été modifié côté serveur depuis la création ou le dernier rafraîchissement du dictionnaire.

Le dictionnaire étant utilisé afin de formater les messages de lecture / mise à jour de données au serveur HR Access, un dictionnaire déphasé conduit de manière certaine à des erreurs système lors de tentatives d'accès aux dossiers HR Access. Un dictionnaire à jour est un pré-requis à toute lecture ou modification de données.

Le rafraîchissement fonctionne de manière différentielle : si un objet a été modifié côté serveur HR Access et que celui-ci doit être rafraîchi dans le dictionnaire, alors seul cet objet sera chargé depuis le serveur. Cela permet de minimiser les échanges réseau avec le serveur et d'optimiser le temps de reconstruction du dictionnaire. Pour pouvoir implémenter une telle stratégie, on mémorise, pour chaque objet du dictionnaire, son horodatage de dernière modification côté serveur. La demande de rafraîchissement consiste alors à comparer (pour tous les objets) l'horodatage local d'un objet avec sa contrepartie serveur ; si les horodatages diffèrent, alors l'objet doit être rechargé.

Le rafraîchissement de dictionnaire peut être demandé à l'aide des deux méthodes refresh() et refresh(boolean) de l'interface IHRDictionary.

Méthode	Rôle
boolean refresh()	Déclenche un rafraîchissement du dictionnaire et indique en retour si au moins un des objets du dictionnaire a été modifié. Si la méthode retourne false, alors le dictionnaire était déjà à jour ; autrement, le dictionnaire était désynchronisé avec le serveur. Appeler cette méthode équivaut à appeler la méthode refresh(boolean) avec false en paramètre (cf. ci-dessous).
boolean refresh(boolean)	Déclenche un rafraîchissement du dictionnaire et indique en retour si au moins un des objets du dictionnaire a été modifié. Le paramètre de type booléen permet de forcer le rafraîchissement du dictionnaire même si celui-ci est à jour.

```
// Retrieving the dictionary from an existing session
IHRDictionary dictionary = getSession().getDictionary();

// Refreshing the dictionary (the two method calls are equivalent)
System.out.println("Refreshing dictionary ... Dictionary updated ? " +
    dictionary.refresh());
System.out.println("Refreshing dictionary ... Dictionary updated ? " +
    dictionary.refresh(false));

// Forcing a dictionary refresh
System.out.println("Refreshing dictionary (forced) ... Dictionary updated ? "
    + dictionary.refresh(true));
```

Modification du dictionnaire

Un dictionnaire peut être lu mais également modifié par le client OpenHR. Parmi les propriétés modifiables, on trouve :

- La liste des processus du dictionnaire
- La langue ou les langues[8] du dictionnaire
- La taille des paquets d'objets lus lors d'un rafraîchissement
- La stratégie de gestion des objets devenus inutiles suite à un rafraîchissement du dictionnaire

[8] *7.30.50 ou supérieure*

Lorsque les propriétés "liste des processus" et "langue"/"langues"[8] sont modifiées, le dictionnaire existant devient invalide. Par exemple, le fait de changer la langue ou les langues[8] du dictionnaire invalide tous les libellés existants pour chacun des objets du dictionnaire. Le dictionnaire est alors marqué "désynchronisé" et doit être remis à jour, c'est-à-dire rafraîchi. Le rafraîchissement peut être demandé de deux manières :

- **Explicite** : consiste à explicitement reconstruire le dictionnaire en effectuant une demande de rafraîchissement à l'aide d'une des méthodes refresh() (eager loading).
- **Implicite** : consiste à laisser le dictionnaire se rafraîchir automatiquement à la volée au prochain appel d'une des méthodes de l'interface IHRDictionary (lazy loading). Le dictionnaire mémorise un témoin indiquant si sa configuration a changé depuis sa création (ou son dernier rafraîchissement). Ce témoin est vérifié lors des appels de méthode du dictionnaire afin de rafraîchir à la volée celui-ci s'il n'est pas à jour.

Modification des processus du dictionnaire

Pour modifier la liste des processus d'un dictionnaire, il faut appeler l'une des méthodes suivantes.

Méthode	Rôle
void addProcess(String)	Ajoute le processus de nom donné à la liste des processus du dictionnaire. L'appel de cette méthode invalide le dictionnaire, ce qui produira son rafraîchissement implicite lors de l'appel de l'une de ses méthodes.
void addProcesses(Collection<String>)	Ajoute les processus de noms donnés à la liste des processus du dictionnaire. L'appel de cette méthode invalide le dictionnaire, ce qui produira son rafraîchissement implicite lors de l'appel de l'une de ses méthodes.
void set Processes(Collection<String>)	Remplace les processus du dictionnaire par ceux de noms donnés. L'appel de cette méthode invalide le dictionnaire, ce qui produira son rafraîchissement implicite lors de l'appel de l'une de ses méthodes.

```
// Retrieving the dictionary from an existing session
IHRDictionary dictionary = getSession().getDictionary();

// Adding one process to the dictionary's process list
dictionary.addProcess("AS100");

// Adding some processes to the dictionary's process list
dictionary.addProcesses(Arrays.asList(new String[] { "AS800", "AG0GD"
    }));

// Setting the dictionary's process list
```

```
dictionary.setProcesses(Arrays.asList(new String[] { “AS100”, “AS800”,
    “AG0GD”, “AD9TA” }));

// Refreshing the dictionary
dictionary.refresh();
```

Modification de la langue ou des langues du dictionnaire

Version antérieure à la 7.30.50

La langue du dictionnaire permet de déterminer le libellé des objets. Un dictionnaire n'est donc pas internationalisé. Pour modifier la langue du dictionnaire, il faut appeler la méthode setLanguage(char), ce qui l'invalide.

```
// Retrieving the dictionary from an existing session
IHRDictionary dictionary = getSession().getDictionary();

// Modifying the dictionary's language
dictionary.setLanguage(“F”);

// Refreshing the dictionary
dictionary.refresh();
```

Version 7.30.50 ou supérieure

Le dictionnaire est désormais internationalisé (voir paramètre "session.languages" du fichier "openhr.properties"). Pour modifier les langues du dictionnaire, il faut appeler les méthodes setLanguage(char) ou addLanguage(char) ou setLanguages(Set<Character>), ce qui l'invalide.

```
// Retrieving the dictionary from an existing session
IHRDictionary dictionary = getSession().getDictionary();

// Modifying the dictionary's languages
dictionary.setLanguages(“F,U”);

// Refreshing the dictionary
dictionary.refresh();
```

Modification de la stratégie de gestion des objets devenus inutiles

Lorsque le dictionnaire est rafraîchi, les objets locaux qui sont devenus obsolètes sont, par défaut, supprimés du dictionnaire. C'est le cas si une structure de données a été supprimée du dictionnaire car le processus qui la référençait a été supprimé de la liste des processus : la structure de données de travail de ce processus se retrouve obsolète dans le dictionnaire (à moins qu'elle ne soit référencée par un autre processus dont c'est la structure de données principale).

Il est possible d'indiquer au dictionnaire comment il doit gérer les objets devenus obsolètes suite à un rafraîchissement, grâce à la propriété keepUnusedEntities. Par défaut, celle-ci est valorisée à false.

Valeur de keepUnusedEntities	Signification
false (valeur par défaut)	Les objets devenus obsolètes sont supprimés du dictionnaire local. Si d'aventure, l'objet devait réapparaître dans le dictionnaire (à la suite d'une modification du dictionnaire et d'un rafraîchissement), sa description devrait être rechargée depuis le serveur HR Access.
true	Les objets devenus obsolètes sont conservés dans le dictionnaire local. Cependant, elles n'apparaîtront pas au client OpenHR lors d'une exploration du dictionnaire : tout se passera comme si l'objet n'existait pas. Si d'aventure, l'objet devait réapparaître dans le dictionnaire (à la suite d'une modification du dictionnaire et d'un rafraîchissement), sa description locale cachée serait réutilisée et comparée à la valeur du serveur HR Access.

L'utilisation de ce paramètre correspond à une utilisation avancée du dictionnaire. Il est peu probable que vous en ayez besoin.

La modification de cette propriété n'invalide pas le dictionnaire.

Modification de la taille des paquets de chargement

Lors de la construction, les objets du dictionnaire sont chargés par paquets (de 250 par défaut). Le fait de charger les objets par paquets permet d'améliorer la consommation mémoire lors de la construction du dictionnaire. C'est particulièrement flagrant lorsque celui-ci comporte beaucoup de processus, donc d'objets. Il est possible de modifier la taille des paquets d'objets lus à l'aide de la méthode setRefreshPacketSize(int).

```
// Retrieving the dictionary from an existing session
IHRDictionary dictionary = getSession().getDictionary();

// Logging the initial value
System.out.println("The initial refresh packet size is " +
    dictionary.getRefreshPacketSize());

// Updating the refreshPacketSize property
```

```
dictionary.setRefreshPacketSize(100);

// Logging the new value
System.out.println("The new refresh packet size is " +
    dictionary.getRefreshPacketSize());
```

La modification de cette propriété n'invalide pas le dictionnaire.

Tester si le dictionnaire doit être rafraîchi

La méthode IHRDictionary.isUpToDate() permet de déterminer si la configuration du dictionnaire a été modifiée depuis sa construction ou son dernier rafraîchissement. La méthode retourne :

- True juste après la construction du dictionnaire
- False lors de l'utilisation des méthodes addProcess(String), addProcesses(Collection<String>), setProcesses(Collection<String>), setLanguage(char), addLanguage(char)[9] et setLanguages(Set<Character>)[9]

Attention à ne pas confondre la sémantique de la méthode isUpToDate() avec celle des méthodes refresh() :

- La méthode isUpToDate() indique si la configuration du dictionnaire a été modifiée localement, ce qui oblige à le rafraîchir pour qu'il soit valide.
- Les méthodes refresh() permettent de vérifier que le dictionnaire est bien synchronisé avec le modèle de données coté serveur, cette vérification ne porte donc pas sur la configuration du dictionnaire.

Valeur de isUpToDate()	Signification
true	La configuration du dictionnaire n'a pas été modifiée depuis sa construction (ou son dernier rafraîchissement), le dictionnaire est donc à jour et (théoriquement) en phase avec la description des objets côté serveur.
false	La configuration du dictionnaire a été modifiée depuis sa construction (ou son dernier rafraîchissement), le dictionnaire n'est donc plus à jour et doit être rafraîchi.

Les structures de données

Une structure de données est représentée par l'interface com.hraccess.openhr.IHRDataStructure. Les structures de données peuvent être récupérées depuis l'interface IHRDictionary (cf. méthodes getDataStructures() … citées plus haut).

Dans l'arbre représentant le dictionnaire, la structure de données est un nœud dont les nœuds enfants sont les informations, les types de dossier et les liens.

[9] *7.30.50 ou supérieure*

Propriétés

Une structure de données possède plusieurs propriétés accessibles via les méthodes suivantes.

Méthode	Rôle
int getDataSectionCount()	Retourne le nombre d'informations référencées par cette structure de données.
IHRDictionary getDictionary()	Retourne le dictionnaire auquel est rattachée cette structure de données.
int getDossierTypeCount()	Retourne le nombre de types de dossier référencés par cette structure de données.
String getDossierTypeItem()	Retourne le nom de la rubrique de l'information 00 désignant le type du dossier avec cette structure de données. Exemple : retourne "TYPDOS" dans le cas de la structure de données ZY (Dossiers du personnel).
List<IHRItem> getKeyItems()	Retourne, sous forme d'une List<IHRItem>, les rubriques qui sont les arguments de tri de l'information 00 de cette structure de données. Exemple : retourne une liste avec deux instances de IHRItem représentant les rubriques ZY00 SOCCLE (Réglementation) et ZY00 MATCLE (Matricule) dans le cas de la structure de données ZY (Dossiers du personnel).
String getLabel() Dépréciée en 7.30.50 ou supérieure, utiliser getDefaultLabel() à la place	Retourne le libellé de cette structure de données. La langue du libellé correspond à la langue par défaut de la session OpenHR.
String getDefaultLabel() 7.30.50 ou supérieure	Retourne le libellé par défaut de cette structure de données. La langue du libellé correspond à la langue par défaut de la session OpenHR.
String getLabel(char) 7.30.50 ou supérieure	Retourne le libellé de cette structure de données pour une langue donnée.
Map<Character, String> getLabels() 7.30.50 ou supérieure	Retourne, sous forme d'une Map<Character, String> les libellés par langue de cette structure de données.
long getLastUpdateTimestamp()	Retourne sous forme d'un nombre de millisecondes l'horodatage de dernière modification côté serveur de la structure de données. Cet horodatage permet de détecter les objets ayant évolué côté serveur.

Méthode	Rôle
int getLinkCount()	Retourne le nombre de liens référencés par cette structure de données.
String getName()	Retourne le nom (identifiant) de cette structure de données. Exemple : retourne "ZY" dans le cas de la structure de données "Dossiers du personnel".
IHRDataStructure getRuleSystemDataStructure()	Retourne la structure de données réglementaire associée à cette structure de données. Celle-ci peut ne pas exister, auquel cas la méthode retourne null. Exemple : retourne "ZD" dans le cas de la structure de données ZY (Dossiers du personnel).
String getRuleSystemItem()	Retourne le nom de la rubrique de l'information 00 désignant la réglementation associée au dossier avec cette structure de données. Exemple : retourne "SOCCLE" (réglementation) dans le cas de la structure de données ZY (Dossiers du personnel).
short getType()	Retourne sous forme d'un short le type de la structure de données. Les différentes valeurs que peut retourner cette méthode sont énumérées sous forme de constantes de nom TYPE_XXX au niveau de l'interface IHRDataStructure. Les types de structure de données possible sont "mono-type", "multi-type", "réglementaire" et "opération".
boolean isRuleSystemDataStructure()	Indique si cette structure de données est de type "réglementaire". Exemples : retourne true pour la structure de données ZD (Réglementation RH) et false pour la structure de données ZY (Dossiers du personnel).

```
// Retrieving the dictionary from an existing session
IHRDictionary dictionary = getSession().getDictionary();
```

```
// Retrieving the data structure with name “ZY” (representing an employee dossier)
IHRDataStructure dataStructure =
    dictionary.getDataStructureByName(“ZY”);
```

```
// Dumping data structure's properties
System.out.println("Name is " + dataStructure.getName());
System.out.println("Label is " + dataStructure.getLabel());
System.out.println("Data structure type is " + dataStructure.getType());
System.out.println("Data structure was last modified on " + new
   Date(dataStructure.getLastUpdateTimestamp()));
System.out.println("Key items are " + dataStructure.getKeyItems());
System.out.println("Item for dossier type is " +
   dataStructure.getDossierTypeItem());
System.out.println("Rule system item is " +
   dataStructure.getRuleSystemItem());

if (dataStructure.getRuleSystemDataStructure() == null) {
   System.out.println("Data structure has no defined rule system data
   structure");
} else {
   System.out.println("Data structure is mapped to rule system data
   structure "
         + dataStructure.getRuleSystemDataStructure().getName());
}

System.out.println("Rule system data structure ? " +
   dataStructure.isRuleSystemDataStructure());
```

Récupération des informations

A partir d'une instance de IHRDataStructure, il est possible de connaître les informations qui y sont rattachées. Une information est représentée par une instance de com.hraccess.openhr.IHRDataSection.

Attention ! Pour qu'une information soit présente dans le dictionnaire il faut que celle-ci soit rattachée explicitement à l'un des processus HR Access configurés au niveau du fichier openhr.properties. Si le rattachement n'est pas explicite, l'information ne peut être manipulée par l'API OpenHR et sa description ne sera pas rapatriée pour construire le dictionnaire.

Pour récupérer les informations rattachées à une structure de données, il existe deux méthodes : getDataSections() et getDataSectionMap().

```
// Retrieving the data structure from an existing session and dictionary
IHRDataStructure dataStructure =
   getSession().getDictionary().getDataStructureByName("ZY");
```

```
// Retrieving the data structure's data sections as a List<IHRDataSection>
List dataSections = dataStructure.getDataSections();

for(Iterator it=dataSections.iterator(); it.hasNext();) {
   IHRDataSection dataSection = (IHRDataSection) it.next();
   System.out.println("Found data section <" + dataSection.getName() + "> " + dataSection.getLabel());
}

// Retrieving the data structure from an existing session and dictionary
IHRDataStructure dataStructure =
   getSession().getDictionary().getDataStructureByName("ZY");

// Retrieving the data structure's data sections as a Map<String, IHRDataSection>
Map dataSections = dataStructure.getDataSectionMap();

for(Iterator it=dataSections.keySet().iterator(); it.hasNext();) {
   String dataSectionName = (String) it.next();
   IHRDataSection dataSection = (IHRDataSection) dataSections.get(dataSectionName);
   System.out.println("Found data section <" + dataSectionName + "> " + dataSection.getLabel()));
}
```

Récupération d'une information par son nom

La méthode IHRDataStructure.getDataSectionByName(String) permet de récupérer une information à partir de son nom.

```
// Retrieving the data structure from an existing session and dictionary
IHRDataStructure dataStructure =
   getSession().getDictionary().getDataStructureByName("ZY");

// Retrieving a named data section
IHRDataSection dataSection = dataStructure.getDataSectionByName("00");

if (dataSection == null) {
   System.err.println("Unable to find data section <ZY00>");
} else {
```

```
        System.out.println("Found data section <ZY00> " +
        dataSection.getLabel());
    }
```

Tester l'existence d'une information

La méthode IHRDataStructure.hasDataSection(String) permet de tester si l'information de nom donné existe.

```
    // Retrieving the data structure from an existing session and dictionary
    IHRDataStructure dataStructure =
        getSession().getDictionary().getDataStructureByName("ZY");


    // Testing the presence of a named data section
    System.out.println("Data section <ZY00> exists ? " +
        dataStructure.hasDataSection("00"));
```

Récupération des types de dossier

A partir d'une instance de IHRDataStructure, il est possible de connaître les types de dossier qui lui sont rattachés. Un type de dossier est représenté par une instance de com.hraccess.openhr.IHRDossierType.

Il n'y a aucune opération particulière à faire pour qu'un type de dossier soit présent dans un dictionnaire. A partir du moment où une structure de données apparaît dans le dictionnaire, tous les types de dossiers qui s'y rattachent sont automatiquement ajoutés.

Pour récupérer les types de dossier rattachés à une structure de données, il existe deux méthodes : getDossierTypes() et getDossierTypeMap().

```
    // Retrieving the data structure from an existing session and dictionary
    IHRDataStructure dataStructure =
        getSession().getDictionary().getDataStructureByName("ZY");


    // Retrieving the data structure's dossier types as a List<IHRDossierType>
    List dossierTypes = dataStructure.getDossierTypes();


    for(Iterator it= dossierTypes.iterator(); it.hasNext();) {
        IHRDossierType dossierType = (IHRDossierType) it.next();
        System.out.println("Found dossier type <" + dossierType.getName() +
        "> " + dossierType.getLabel());
    }


    // Retrieving the data structure from an existing session and dictionary
    IHRDataStructure dataStructure =
        getSession().getDictionary().getDataStructureByName("ZY");
```

```
// Retrieving the data structure's dossier types as a Map<String,
   IHRDossierType>
Map dossierTypes = dataStructure.getDossierTypeMap();


for(Iterator it=dossierTypes.keySet().iterator(); it.hasNext();) {
   String dossierTypeName = (String) it.next();
   IHRDossierType dossierType = (IHRDossierType)
   dossierTypes.get(dossierTypeName);
   System.out.println("Found dossier type <" + dossierTypeName + "> " +
   dossierType.getLabel()));
}
```

Récupération d'un type de dossier par son nom

La méthode IHRDataStructure.getDossierTypeByName(String) permet de récupérer un type de dossier à partir de son nom.

```
// Retrieving the data structure from an existing session and dictionary
IHRDataStructure dataStructure =
   getSession().getDictionary().getDataStructureByName("ZY");


// Retrieving a named dossier type
IHRDossierType dossierType =
   dataStructure.getDossierTypeByName("SAL");


if (dossierType == null) {
   System.err.println("Unable to find dossier type <ZYSAL>");
} else {
   System.out.println("Found dossier type < ZYSAL > " +
   dossierType.getLabel());
}
```

Tester l'existence d'un type de dossier

La méthode IHRDataStructure.hasDossierType(String) permet de tester si le type de dossier de nom donné existe.

```
// Retrieving the data structure from an existing session and dictionary
IHRDataStructure dataStructure =
   getSession().getDictionary().getDataStructureByName("ZY");


// Testing the presence of a named dossier type
System.out.println("Dossier type <ZYSAL> exists ? " +
   dataStructure.hasDossierType("SAL"));
```

Récupération des liens

A partir d'une instance de IHRDataStructure, il est possible de connaître les liens qui lui sont rattachés. Un lien est représenté par une instance de com.hraccess.openhr.IHRLink.

Il n'y a aucune opération particulière à faire pour qu'un lien soit présent dans un dictionnaire. A partir du moment où une structure de données apparaît dans le dictionnaire, tous les liens qui s'y rattachent sont automatiquement ajoutés.

Pour récupérer les liens rattachés à une structure de données, il existe deux méthodes : getLinks() et getLinkMap().

```
// Retrieving the data structure from an existing session and dictionary
IHRDataStructure dataStructure =
   getSession().getDictionary().getDataStructureByName("ZY");


// Retrieving the data structure's links as a List<IHRLink>
List links = dataStructure.getLinks();


for(Iterator it= links.iterator(); it.hasNext();) {
   IHRLink link = (IHRLink) it.next();
   System.out.println("Found link <" + link.getName() + "> " +
   link.getLabel());
}
```

```
// Retrieving the data structure from an existing session and dictionary
IHRDataStructure dataStructure =
   getSession().getDictionary().getDataStructureByName("ZY");


// Retrieving the data structure's links as a Map<String, IHRLink>
Map links = dataStructure.getLinkMap();


for(Iterator it=links.keySet().iterator(); it.hasNext();) {
   String linkName = (String) it.next();
   IHRLink link = (IHRLink) links.get(linkName);
   System.out.println("Found link <" + linkName + "> " + link.getLabel()));
}
```

Récupération d'un lien par son nom

La méthode IHRDataStructure.getLinkByName(String) permet de récupérer un lien à partir de son nom.

```
// Retrieving the data structure from an existing session and dictionary
IHRDataStructure dataStructure =
   getSession().getDictionary().getDataStructureByName("ZY");


// Retrieving a named link
IHRLink link = dataStructure.getLinkByName("10");


if (link == null) {
   System.err.println("Unable to find link <ZY10>");
} else {
   System.out.println("Found link <ZY10> " + link.getLabel());
}
```

Tester l'existence d'un lien

La méthode IHRDataStructure.hasLink(String) permet de tester si le lien de nom donné existe.

```
// Retrieving the data structure from an existing session and dictionary
IHRDataStructure dataStructure =
   getSession().getDictionary().getDataStructureByName("ZY");


// Testing the presence of a named link
System.out.println("Link <ZY10> exists ? " + dataStructure.hasLink("10"));
```

Les informations

Une information est représentée par l'interface com.hraccess.openhr.IHRDataSection. Les informations peuvent être récupérées depuis l'interface IHRDataStructure (cf. méthodes getDataSections() ... citées plus haut).

Dans l'arbre représentant le dictionnaire, l'information est un nœud dont les nœuds enfants sont les rubriques de l'information.

Propriétés

Une information possède plusieurs propriétés accessibles via les méthodes suivantes.

Méthode	Rôle
List<IHRItem> getAllItems()	Retourne sous forme d'une liste de IHRItems toutes les rubriques composant cette information. Cela inclut toutes les rubriques définies via Design Center (élémentaires, groupes, virtuelles, redéfinitions …) dans l'ordre où elles apparaissent dans Design Center.
IHRDataStructure getDataStructure()	Retourne la structure de données à laquelle est rattachée l'information.
IHRDictionary getDictionary()	Retourne le dictionnaire auquel est rattachée cette information (par transitivité).
List<IHRItem> getElementaryItems()	Retourne sous forme d'une liste de IHRItems toutes les rubriques élémentaires composant cette information. Une rubrique élémentaire est une rubrique qui est associée à une (et une seule) colonne de base de données. Cela exclut donc les rubriques groupes et virtuelles ainsi que les redéfinitions de rubrique.
String getFullName()	Retourne le nom complet (identifiant) identifiant cette information. Exemple : retourne "ZY00" pour l'information "00" de la structure de données "Dossiers du personnel".
IHRItem getItemByRole(short)	Retourne la première rubrique de l'information de rôle donné. Le paramètre de type short désigne un rôle de rubrique. Les valeurs valides pour ce paramètre sont énumérées sous forme de constantes de nom ROLE_XXX au niveau de l'interface IHRItem. Les rôles valides sont "BLOB", "devise", "date d'effet", "date de fin", "aucun" et "date de début".
Iterator<IHRItem> getItemIterator()	Retourne un Iterator<IHRItem> sur les rubriques de l'information.

Méthode	Rôle
List<IHRItem> getItems(int)	Retourne les rubriques de l'information de niveau donné. Le niveau d'une rubrique dépend de sa profondeur dans la branche située sous le nœud information. Si aucune rubrique groupe n'existe dans l'information, alors toutes les rubriques sont au même niveau (1). Si une rubrique groupe existe et chapeaute deux rubriques élémentaires, alors la rubrique groupe a un niveau 1 et les rubriques élémentaires un niveau 2. Le niveau d'une rubrique correspond au nombre de liens qui séparent cette rubrique de son nœud parent de type information dans le dictionnaire.
Set<String> getItemsWithBlobRole()	Retourne un Set<String> contenant le nom des rubriques possédant un rôle BLOB.
int getKeyCount()	Retourne le nombre de rubriques de type clé (argument de tri) de l'information.
List<IHRItem> getKeyItems()	Retourne sous forme d'une List de IHRItems les rubriques clés (arguments de tri) de cette information, s'il y en a.
String getLabel() Dépréciée en 7.30.50 ou supérieure, utiliser getDefaultLabel() à la place	Retourne le libellé de cette information. La langue du libellé correspond à la langue par défaut de la session OpenHR.
String getDefaultLabel() 7.30.50 ou supérieure	Retourne le libellé par défaut de cette information. La langue du libellé correspond à la langue par défaut de la session OpenHR.
String getLabel(char language) 7.30.50 ou supérieure	Retourne le libellé de cette information pour une langue donnée.
Map<Character, String> getLabels() 7.30.50 ou supérieure	Retourne, sous la forme d'une Map<Character, String>, les libellés par langue de cette information.
long getLastUpdateTimestamp()	Retourne sous forme d'un nombre de millisecondes l'horodatage de dernière modification côté serveur de l'information. Cet horodatage permet de détecter les objets ayant évolué côté serveur.
String getName()	Retourne le nom (identifiant) de cette information. Exemple : retourne "10" dans le cas de l'information "Naissance" de la structure de données "Dossiers du personnel".

Méthode	Rôle
short getOrder()	Retourne sous forme d'entier le numéro d'ordre de l'information. Ce numéro définit l'ordre dans lequel les informations sont mises à jour sur le serveur HR Access : une information avec un numéro d'ordre 1 sera mise à jour avant une information avec un numéro d'ordre 2, etc.
short getReadBufferLength()	Retourne sous forme d'entier la longueur du buffer contenant les données (en lecture) d'une occurrence d'information sérialisée sous forme texte. La longueur de ce buffer dépend de la définition des rubriques de l'information. Méthode avancée.
int getRedefinedItemCount()	Retourne sous forme d'entier le nombre de rubriques redéfinies de cette information.
List<IHRItem> getRedefinedItems()	Retourne sous forme d'une List de IHRItems les rubriques redéfinies de cette information.
String getSegmentName()	Retourne le nom du segment qui stocke en base relationnelle les données de cette information. En général, le nom de l'information et le nom du segment sont identiques, ce qui signifie que l'information est stockée dans une table relationnelle avec son nom (exemple : "ZYAG"). Il est possible (pour des raisons de performance) de stocker les données de plusieurs informations "unique fixe" dans une même table relationnelle (exemple : l'information unique fixe ZY05 a pour segment ZY00, les données de ZY05 sont donc stockées dans la table de nom ZY00). Exemple de nom de segment : "00".
String getTableName()	Retourne le nom de la table relationnelle stockant les données de cette information. Voir getSegmentName() Exemple de nom de table : "ZY00".
short getType()	Retourne sous forme d'entier le type de cette information. Les différentes valeurs que peut retourner cette méthode sont énumérées sous forme de constantes de nom TYPE_XXX au niveau de l'interface IHRItem. Les types d'information valides sont : "répétitive historique", "unique historique", "répétitive fixe", "unique fixe", "information gestionnaire", "information paramètre" et "information virtuelle".

Méthode	Rôle
short getUpdateBufferLength()	Retourne sous forme d'entier la longueur du buffer contenant les données (en écriture) d'une occurrence d'information sérialisée sous forme texte. La longueur de ce buffer dépend de la définition des rubriques de l'information. Méthode avancée.
int getVirtualItemCount()	Retourne le nombre de rubriques virtuelles de cette information.
int getWritableItemCount()	Retourne le nombre de rubriques de cette information qui peuvent être modifiées. Une rubrique peut être modifiée si elle n'est pas virtuelle.
boolean hasItemWithRole(short)	Indique si l'information possède une rubrique de rôle donné. Le paramètre de type short désigne un rôle de rubrique. Les valeurs valides pour ce paramètre sont énumérées sous forme de constantes de nom ROLE_XXX au niveau de l'interface IHRItem. Les rôles valides sont "BLOB", "devise", "date d'effet", "date de fin", "aucun" et "date de début".
boolean hasTechnicalItems()	Indique si cette information possède des rubriques techniques. Les rubriques techniques de nom "TIMJIF" et "NOMBRE" désignent respectivement l'horodatage de dernière modification de l'information et le nombre d'occurrences de l'information. Ces deux données sont lues à partir de la table technique xxTD12 où xx désigne le nom de la structure de données de l'information. Seules les informations de type "unique fixe", "unique historique", "répétitive fixe" et "répétitive historique" ont des rubriques techniques.
boolean is00()	Indique si cette information est l'information de nom "00".
boolean isDatedRepetitive()	Indique si cette information est de type "répétitive historique".
boolean isDatedUnique()	Indique si cette information est de type "unique historique".
boolean isFixedRepetitive()	Indique si cette information est de type "répétitive fixe".
boolean isFixedUnique()	Indique si cette information est de type "unique fixe".

Méthode	Rôle
boolean isManagedByLanguage()	Indique si cette information est gérée par langue. Exemple d'information gérée par langue : "ZD01" représentant les libellés d'un code réglementaire.
boolean isManagementDossier()	Indique si cette information est de type "information gestionnaire".
boolean isMultiple()	Indique si cette information est "multiple". Seules les informations de type "unique fixe", "unique historique", "répétitive fixe" et "répétitive historique" sont "multiples".
boolean isParameter()	Indique si cette information est de type "information paramètre".
boolean isReadOnly()	Indique si cette information est en lecture seule, c'est-à-dire qu'elle ne peut être modifiée. Une information peut se retrouver en lecture seule si elle est virtuelle.
boolean isReal()	Indique si cette information est réelle. Seules les informations de type "unique fixe", "unique historique", "répétitive fixe" et "répétitive historique" sont "réelles".
boolean isVirtual()	Indique si cette information est de type "information virtuelle".

```
// Retrieving the data section from an existing session, dictionary and data
   structure
IHRDataSection dataSection =
   getSession().getDictionary().getDataStructureByName(“ZY”)

   .getDataSectionByName(“AG”);

System.out.println(“Data section name is ” + dataSection.getName());

// Dumping the data section's items
for(Iterator it=dataSection.getAllItems().iterator(); it.hasNext();) {
   IHRItem item = (IHRItem) it.next();
   System.out.println(“Found item <” + item.getName() + “> ” +
   item.getLabel());
}

// Dumping the data section's elementary items
```

```
for(Iterator it=dataSection.getElementaryItems().iterator(); it.hasNext();) {
    IHRItem item = (IHRItem) it.next();
    System.out.println("Found elementary item <" + item.getName() + "> "
    + item.getLabel());
}

System.out.println("Data section full name is " +
    dataSection.getFullName());
System.out.println("Data section label is " + dataSection.getLabel());
System.out.println("Data section has " + dataSection.getKeyCount() + " key
    item(s)");

for(Iterator it=dataSection.getKeyItems().iterator(); it.hasNext();) {
    IHRItem item = (IHRItem) it.next();
    System.out.println("Found key item <" + item.getName() + "> " +
    item.getLabel());
}

System.out.println("Data section was last updated on " + new
    Date(dataSection.getLastUpdateTimestamp()));
System.out.println("Data section update order is " +
    dataSection.getOrder());
System.out.println("Data section segment name is " +
    dataSection.getSegmentName());
System.out.println("Data section type is " + dataSection.getType());
```

Récupération des rubriques

Pour récupérer les rubriques d'une information, il existe plusieurs méthodes (sur l'interface IHRDataSection) : getAllItems(), getElementaryItems(), getKeyItems(), getItemIterator(), getItems() et getRedefinedItems() selon le type des rubriques que vous souhaitez récupérer.

La méthode getAllItems() retourne toutes les rubriques de l'information. Les autres méthodes permettent de retourner un sous-ensemble de ces rubriques en les filtrant sur un critère donné (leur caractère élémentaire, le fait qu'elle soit de type clé -argument de tri -, etc.).

Méthode	Rôle
List<IHRItem> getAllItems()	Retourne sous forme d'une liste de IHRItems toutes les rubriques composant cette information. Cela inclut toutes les rubriques définies via Design Center (élémentaires, groupes, virtuelles, redéfinitions …).
List<IHRItem> getElementaryItems()	Retourne sous forme d'une liste de IHRItems toutes les rubriques élémentaires composant cette information. Une rubrique élémentaire est une rubrique qui est associée à une (et une seule) colonne de base de données. Cela exclut donc les rubriques groupes et virtuelles ainsi que les redéfinitions de rubrique.
List<IHRItem> getKeyItems()	Retourne sous forme d'une List de IHRItems les rubriques clés (arguments de tri) de cette information, s'il y en a.
Iterator<IHRItem> getItemIterator()	Retourne un Iterator<IHRItem> sur les rubriques de l'information.
List<IHRItem> getItems()	Retourne sous forme d'une liste de IHRItems les rubriques composant cette information.
List<IHRItem> getRedefinedItems()	Retourne sous forme d'une List de IHRItems les rubriques redéfinies de cette information.

```
// Retrieving the data section from an existing session, dictionary and data
   structure
IHRDataSection dataSection =
   getSession().getDictionary().getDataStructureByName("ZY")

   .getDataSectionByName("AG");

System.out.println("Data section name is " + dataSection.getName());

// Dumping the data section's items
for(Iterator it=dataSection.getAllItems().iterator(); it.hasNext();) {
```

```
    IHRItem item = (IHRItem) it.next();
    System.out.println("Found item <" + item.getName() + "> " +
    item.getLabel());
}

// Dumping the data section's elementary items
for(Iterator it=dataSection.getElementaryItems().iterator(); it.hasNext();) {
    IHRItem item = (IHRItem) it.next();
    System.out.println("Found elementary item <" + item.getName() + "> "
    + item.getLabel());
}
```

Récupération d'une rubrique par son nom

La méthode IHRDataSection.getItemByName(String) permet de récupérer une rubrique à partir de son nom.

```
// Retrieving the data section from an existing session, dictionary and data
   structure
IHRDataSection dataSection =
   getSession().getDictionary().getDataStructureByName("ZY")

   .getDataSectionByName("AG");

// Retrieving the named item
IHRItem item = dataSection.getItemByName("MOTIFA");

if (item == null) {
   System.err.println("Unable to find item <ZYAG MOTIFA>");
} else {
   System.out.println("Found item <ZYAG MOTIFA> " + item.getLabel());
}
```

Tester l'existence d'une rubrique

La méthode IHRDataSection.hasItem(String) permet de tester si la rubrique de nom donné existe.

```
// Retrieving the data section from an existing session, dictionary and data
   structure
IHRDataSection dataSection =
   getSession().getDictionary().getDataStructureByName("ZY")
```

```
    .getDataSectionByName("AG");

// Testing the presence of a named item
System.out.println("Item <ZYAG MOTIFA> exists ? " +
    dataSection.hasItem("MOTIFA"));
```

Les rubriques

Une rubrique est représentée par l'interface com.hraccess.openhr.IHRItem. Les rubriques peuvent être récupérées depuis l'interface IHRDataSection (cf. méthodes getAllItems() … citées plus haut).

Dans l'arbre représentant le dictionnaire, la rubrique est un nœud feuille.

Les types de rubriques

Il convient d'introduire le vocabulaire relatif aux différents types de rubriques car Design Center permet de créer des rubriques réelles, virtuelles, de type redéfinition, groupes, filles, élémentaires, de type clé, etc.

Précisons qu'une rubrique peut cumuler certaines de ses propriétés car toutes ne sont pas mutuellement exclusives.

Rubrique clé

Une rubrique est dite "clé" (ou argument de tri) quand elle participe à l'identifiant de l'information.

Une information peut avoir zéro, une ou plusieurs rubriques clés. Le n-uplet formé par les rubriques clés d'une information identifie de manière unique une occurrence et sert à générer une contrainte d'unicité au niveau de la table relationnelle associée à l'information.

Exemple : l'information "ZY00" identifiant un dossier de salarié possède deux rubriques clés : SOCCLE (Réglementation) et MATCLE (Matricule).

Rubrique virtuelle

Une rubrique est dite "virtuelle" quand sa valeur est issue d'un calcul. La rubrique n'est donc associée à aucune colonne en table relationnelle puisque sa valeur est calculée à l'aide d'un traitement spécifique.

L'exemple type de rubrique virtuelle est illustré par les rubriques AGEAJO et AGEMJO de l'information ZY10 (Naissance) : elles permettent (suite à un calcul par traitement spécifique basé sur la date de naissance) de connaître l'âge du salarié à la date du jour.

Rubrique groupe

Une rubrique est dite "groupe" lorsqu'elle est la mère de plusieurs rubriques. Design Center permet de créer des rubriques groupe de la même manière que le langage COBOL. En valorisant la rubrique groupe, on valorise indirectement ses rubriques filles.

Exemple : la rubrique IDENTI (Identifiant) de l'information ZY00 (Identifiant de salarié) est une rubrique groupe, mère des deux rubriques SOCCLE (Réglementation) et MATCLE (Matricule).

Rubrique fille

Une rubrique est dite "fille" lorsqu'elle possède une rubrique mère (de type groupe).

Exemple : la rubrique IDENTI (Identifiant) de l'information ZY00 (Identifiant de salarié) est une rubrique groupe, mère des deux rubriques (filles) SOCCLE (Réglementation) et MATCLE (Matricule).

Rubrique élémentaire

Une rubrique est dite "élémentaire" quand elle n'est ni virtuelle, ni groupe, ni redéfinie, ni technique. Une telle rubrique est alors forcément associée à une colonne en table relationnelle.

Exemple : dans l'information ZY00 (Identifiant de salarié), les rubriques SOCCLE (Réglementation) et MATCLE (Société) sont élémentaires mais pas la rubrique mère, IDENTI (Identifiant).

Rubrique de type redéfinition

Une rubrique est qualifiée de "redéfinition" quand elle redécoupe la zone mémoire associée à une rubrique. Une même rubrique peut ainsi être redéfinie plusieurs fois. La technique de redéfinition est semblable à celle du langage COBOL : une même zone mémoire peut être vue selon différents masques correspondant aux redéfinitions de la zone mémoire associée à une rubrique.

Exemple : l'information ZY0F (Adresses d'un salarié) redéfinit la rubrique ZONADA (de format alphanumérique de longueur 40) selon plusieurs masques, en fonction du pays du salarié (Pays-Bas, France, Allemagne, Italie, etc.).

Rubrique redéfinie

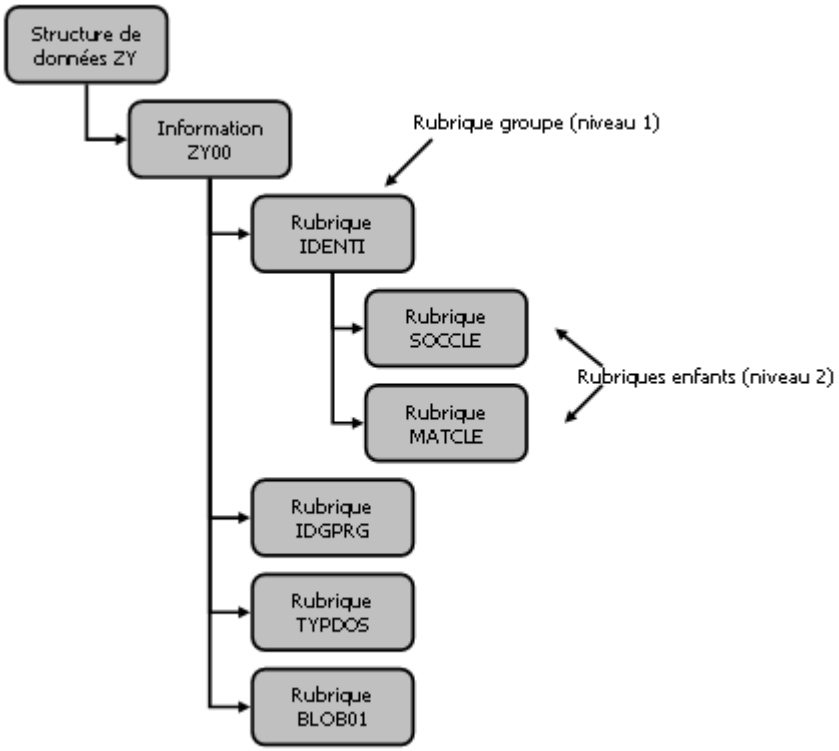
Une rubrique est dite "redéfinie" lorsqu'il existe au moins une rubrique (de type redéfinition) qui la redéfinit.

Exemple : la rubrique ZONADA de l'information ZY0F (Adresses d'un salarié) est redéfinie par les rubriques ZONANL (Pays-Bas), ZONAFR (France), etc.

L'arborescence des rubriques

Le fait que l'on puisse définir des rubriques groupe permet d'agencer les rubriques de manière arborescente. Ainsi, les rubriques ne sont pas forcément les feuilles de l'arbre du dictionnaire comme cela a été dit plus haut.

L'exemple suivant montre l'arbre des rubriques pour l'information ZY00 (Identifiant d'un salarié). Les rubriques peuvent elles-mêmes s'agencer de manière hiérarchique (rubrique groupe) :



Le niveau d'une rubrique - dont il sera question plus loin - indique le niveau de profondeur de la rubrique dans l'arbre ayant pour racine le nœud information.

Méthodes

Une rubrique possède de nombreuses propriétés qui découlent du paramétrage effectué via Design Center. Le tableau suivant liste les principales méthodes de l'interface IHRItem.

Méthode	Rôle
List<IHRItem> getAllRedefinitions()	Retourne une List de IHRItems représentant les rubriques qui redéfinissent cette rubrique ainsi que leurs rubriques descendantes (filles, petites-filles, etc.). Exemple : l'appel de cette méthode pour la rubrique ZY0F CDPOST retourne une liste contenant les rubriques (de ZY0F) : POSTNL, CODPNL, CDPCNU, CDPCAL, FIPCNL, POSTDE, etc.
IHRDataSection getDataSection()	Retourne l'information à laquelle cette rubrique est rattachée.
IHRDataStructure getDataStructure()	Retourne la structure de données à laquelle cette rubrique est rattachée (par transitivité).
short getDecimal()	Retourne sous forme d'entier le nombre de décimales associé à cette rubrique. L'utilisation de cette méthode n'est pertinente que si la rubrique en question est de type numérique ; dans le cas contraire, la méthode retourne 0.
IHRDictionary getDictionary()	Retourne le dictionnaire auquel cette rubrique est rattachée (par transitivité).
String getFullName()	Retourne le nom complet de cette rubrique de la forme <data-structure><data-section><item>. Exemple : "ZY00SOCCLE".
IHRItem[] getGroupChildren()	Retourne sous forme d'un tableau de IHRItems les rubriques filles (directes) de cette rubrique. Exemple : retourne les rubriques CODPFR et UITPFR pour la rubrique ZY0F POSTFR.
IHRItem getGroupParent()	Retourne la rubrique mère de cette rubrique. L'utilisation de cette méthode n'est pertinente que si la rubrique en question est fille d'une rubrique groupe.
String getLabel() Dépréciée en 7.30.50 ou supérieure, utiliser getDefaultLabel() à la place	Retourne le libellé de la rubrique. La langue du libellé correspond à la langue par défaut de la session OpenHR.

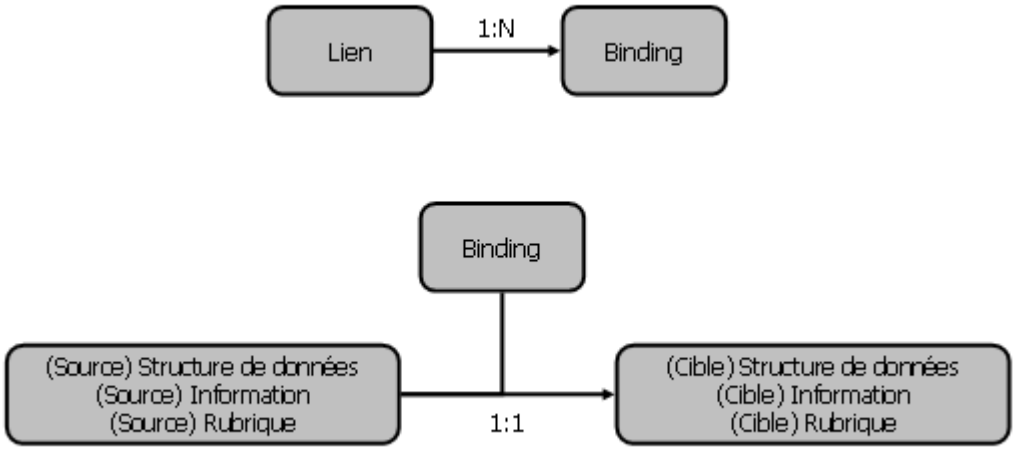
Méthode	Rôle
String getDefaultLabel() 7.30.50 ou supérieure	Retourne le libellé par défaut de la rubrique. La langue du libellé correspond à la langue par défaut de la session OpenHR.
String getLabel(char language) 7.30.50 ou supérieure	Retourne le libellé de la rubrique pour une langue donnée.
Map<Character, String> getLabels() 7.30.50 ou supérieure	Retourne, sous la forme d'une Map<Character, String>, les libellés par langue de la rubrique.
short getLevel()	Retourne sous forme d'entier le niveau de la rubrique. Le niveau d'une rubrique représente la profondeur à laquelle celle-ci est située dans l'arbre ayant pour racine l'information mère de la rubrique. Exemples : retourne 1 pour ZY00 IDENTI, 2 pour ZY00 SOCCLE et ZY00 MATCLE.
String getName()	Retourne le nom (identifiant) de cette rubrique. Exemple : retourne "MATCLE" pour la rubrique ZY00 MATCLE.
IHRItem getRedefinedItem()	Retourne la rubrique redéfinie par cette rubrique, s'il y a lieu. Exemple : retourne "CDPOST" pour la rubrique ZY0F POSTFR.
IHRItem[] getRedefinitions()	Retourne sous forme d'un tableau de IHRItems les rubriques qui redéfinissent cette rubrique mais pas leurs rubriques descendantes (filles, petites-filles, etc.). Exemple : l'appel de cette méthode pour la rubrique ZY0F CDPOST retourne une liste contenant les rubriques (de ZY0F) : POSTNL, POSTDE, POSTFR, POSTES, POSTIT, POSTBE, POSTCH, POSTUK, POSTPO, GPUSCP, GPCACP. En revanche, cette liste ne comporte pas les rubriques filles de POSTNL, POSTDE, etc.
short getRole()	Retourne sous forme d'entier le rôle de cette rubrique. Les différentes valeurs que peut retourner cette méthode sont énumérées sous forme de constantes de nom ROLE_XXX au niveau de l'interface IHRItem. Les rôles valides sont "BLOB", "devise", "date d'effet", "date de fin", "aucun" et "date de début".

Méthode	Rôle
short getSize()	Retourne la longueur maximale autorisée pour cette rubrique. Cette longueur correspond au nombre de caractères maximal d'une rubrique de type texte. Dans le cas d'une rubrique numérique, cette longueur correspond à la somme des longueurs des parties entière et décimale. Par conséquent, getDecimal() retourne le nombre de décimales de la rubrique alors que getSize() - getDecimal() retourne la longueur de la partie entière de la rubrique.
short getType()	Retourne sous forme d'entier le type de cette rubrique. Les différentes valeurs que peut retourner cette méthode sont énumérées sous forme de constantes de nom TYPE__XXX au niveau de l'interface IHRItem.
IHRItem.Type getTypeAsObject()	Retourne sous forme d'une instance de IHRItem.Type le type de cette rubrique. Cette méthode est comparable à getType() sauf qu'elle utilise une énumération de type sûr ("safe type enumeration") comme type de retour.
boolean hasBlobRole()	Indique si cette rubrique possède le rôle "BLOB".
boolean hasCurrencyRole()	Indique si cette rubrique possède le rôle "devise".
boolean hasRedefinition()	Indique si cette rubrique est redéfinie par une (ou plusieurs) rubrique(s).
boolean isComputed()	Indique si cette rubrique est virtuelle (calculée).
boolean isDate()	Indique si cette rubrique est de type date. Retourne true pour les rubriques définies dans Design Center avec les formats suivants : "date", "horodatage", "date quantifiée", "année" et "mois".
boolean isGroup()	Indique si cette rubrique est de type groupe, c'est-à-dire qu'elle est la mère d'autres rubriques.
boolean isGroupChild()	Indique si cette rubrique est une rubrique fille, c'est-à-dire située sous une rubrique groupe.
boolean isKey()	Indique si cette rubrique est une rubrique clé (argument de tri).
boolean isNullValueAllowed()	Indique si cette rubrique autorise les valeurs à null.

Méthode	Rôle
boolean isNumber()	Indique si la valeur de cette rubrique est de type numérique. Retourne true pour les rubriques définies dans Design Center avec les formats suivants : "numérique entier binaire", "numérique condensé", "numérique display non signé", "numérique display signé", "montant financier avec devise", "montant financier sans devise" et "durée".
boolean isReadOnly()	Indique si cette rubrique est en lecture seule. Retourne true si la rubrique est virtuelle (calculée).
boolean isRedefinition()	Indique si cette rubrique est une redéfinition de rubrique.
boolean isRequired()	Indique si cette rubrique est obligatoire.
boolean isString()	Indique si la valeur de cette rubrique est de type texte. Retourne true pour les rubriques définies dans Design Center avec les formats suivants : "Alphanumérique sans conversion des minuscules", "Alphanumérique avec conversion des minuscules" et "Alphanumérique de longueur variable".
boolean isSystem()	Indique si cette rubrique est l'une des deux rubriques système (techniques). Les rubriques techniques de nom "TIMJIF" et "NOMBRE" désignent respectivement l'horodatage de dernière modification et le nombre d'occurrences d'une information. Ces deux données sont lues à partir de la table technique xxTD12 où xx désigne le nom de la structure de données de l'information. Les rubriques techniques ne sont pas de vraies rubriques paramétrées via Design Center.
boolean isUpperCase()	Indique si cette rubrique a pour format Design Center "Alphanumérique avec conversion des minuscules", c'est-à-dire que la rubrique n'accepte pour valeur que des chaînes de caractères en majuscules.
boolean isVirtual()	Indique si cette rubrique est virtuelle (calculée).

Les liens

Un lien est représenté par l'interface com.hraccess.openhr.IHRLink. Un lien est constitué de 1 à N correspondances. Chaque correspondance met en rapport une rubrique (source) d'une information (source) d'une structure de données (source) avec une rubrique (cible) d'une information (cible) d'une structure de données (cible) :



Le concept de lien peut être comparé au mécanisme de clé étrangère utilisé dans les bases de données relationnelles.

Un **lien** permet de définir une contrainte entre le contenu de colonnes (ou ensemble de colonnes) de deux tables relationnelles T1 et T2. Cette contrainte assure que la colonne C1 (ou l'ensemble de colonnes) de la table T1 ne peut prendre une valeur que si celle-ci existe dans la colonne C2 (ou l'ensemble de colonnes) de la table T2.

Avec un lien, on définit une contrainte qui assure que la valeur d'une rubrique R1 (ou d'un ensemble de rubriques) d'une information I1 ne peut prendre une valeur que si celle-ci existe pour la rubrique R2 (ou un ensemble de rubriques) d'une information I2. Autrement dit, la rubrique R2 définit un ensemble de valeurs distinctes que peut prendre la rubrique R1.

Le mécanisme de lien est une extrapolation du mécanisme de correspondance avec un répertoire réglementaire. Alors que ce dernier ne permet de choisir comme valeur de rubrique que l'une des valeurs définies dans le répertoire réglementaire, le mécanisme de lien est plus générique et permet d'établir une correspondance vers une structure de données qui n'est pas nécessairement réglementaire et vers une information qui n'est pas nécessairement l'information "00" identifiant un dossier.

A un lien sont associées une ou plusieurs correspondances (désignées par le terme "Binding" sur le schéma ci-dessus).

Une **correspondance** désigne une contrainte entre une rubrique R1 d'une information I1 (d'une structure de données S1) et la rubrique R2 d'une information I2 (d'une structure de données S2).

Attention cependant car rien n'oblige les structures de données S1 et S2 à être identiques. De plus, il est possible de définir, pour un même lien, des correspondances vers des informations différentes. Le concept de correspondance est représenté par l'interface com.hraccess.openhr.IHRLink.Binding.

Méthodes d'un lien

Le tableau suivant liste les méthodes utiles de l'interface IHRLink.

Méthode	Rôle
int getBindingCount()	Retourne le nombre de correspondances associées à ce lien.
List<IHRLink.Binding> getBindings()	Retourne sous forme d'une liste de IHRLink.Bindings les correspondances de ce lien.
String getLabel() Dépréciée en 7.30.50 ou supérieure, utiliser getDefaultLabel() à la place	Retourne le libellé de ce lien. La langue du libellé correspond à la langue par défaut de la session OpenHR.
String getDefaultLabel() 7.30.50 ou supérieure	Retourne le libellé par défaut de ce lien. La langue du libellé correspond à la langue par défaut de la session OpenHR.
String getLabel(char language) 7.30.50 ou supérieure	Retourne le libellé de ce lien pour une langue donnée.
Map<Character, String> getLabels() 7.30.50 ou supérieure	Retourne, sous la forme d'une Map<Character, String>, les libellés par langue de ce lien.
long getLastUpdateTimestamp()	Retourne sous forme d'un nombre de millisecondes l'horodatage de dernière modification côté serveur du lien. Cet horodatage permet de détecter les objets ayant évolué côté serveur.
String getName()	Retourne le nom (identifiant) de ce lien.
IHRDataSection getSourceDataSection()	Retourne l'information source de ce lien. L'appel de cette méthode n'est pertinent que si toutes les correspondances du lien portent sur la même information source, c'est-à-dire qu'il n'existe qu'une seule information source. Dans le cas contraire, l'appel de cette méthode lève une exception.
int getSourceDataSectionCount()	Retourne le nombre d'informations sources distinctes définies pour les correspondances de ce lien.

Méthode	Rôle
String getSourceDataSectionName()	Retourne le nom (identifiant) de l'information source de ce lien. L'appel de cette méthode n'est pertinent que si toutes les correspondances du lien portent sur la même information source, c'est-à-dire qu'il n'existe qu'une seule information source. Dans le cas contraire, l'appel de cette méthode lève une exception.
Set<String> getSourceDataSectionNames()	Retourne un Set<String> contenant le nom (identifiant) des informations source distinctes définies pour les correspondances de ce lien.
Set<IHRDataSection> getSourceDataSections()	Retourne un Set<IHRDataSection> contenant les informations sources distinctes définies pour les correspondances de ce lien.
IHRDataStructure getSourceDataStructure()	Retourne la structure de données source de ce lien.
String getSourceDataStructureName()	Retourne le nom (identifiant) de la structure de données source de ce lien.
IHRDataSection getTargetDataSection()	Retourne l'information cible de ce lien. L'appel de cette méthode n'est pertinent que si toutes les correspondances du lien portent sur la même information cible c'est-à-dire qu'il n'existe qu'une seule information cible. Dans le cas contraire, l'appel de cette méthode lève une exception.
int getTargetDataSectionCount()	Retourne le nombre d'informations cibles distinctes définies pour les correspondances de ce lien.
String getTargetDataSectionName()	Retourne le nom (identifiant) de l'information cible de ce lien. L'appel de cette méthode n'est pertinent que si toutes les correspondances du lien portent sur la même information cible c'est-à-dire qu'il n'existe qu'une seule information cible. Dans le cas contraire, l'appel de cette méthode lève une exception.
Set<String> getTargetDataSectionNames()	Retourne un Set<String> contenant le nom (identifiant) des informations cibles distinctes définies pour les correspondances de ce lien.
Set<IHRDataSection> getTargetDataSections()	Retourne un Set<IHRDataSection> contenant les informations cibles distinctes définies pour les correspondances de ce lien.
IHRDataStructure getTargetDataStructure()	Retourne la structure de données cible de ce lien.

Méthode	Rôle
String getTargetDataStructureName()	Retourne le nom (identifiant) de la structure de données cible de ce lien.
IHRDossierType getTargetDirectory()	Retourne le type de dossier cible de ce lien, s'il y a lieu.
String getTargetDirectoryName()	Retourne le nom du type de dossier cible de ce lien, s'il y a lieu.
boolean isLinkToSelf()	Indique si ce lien est un lien d'un dossier vers lui-même.

Méthodes d'une correspondance

Le tableau suivant liste les méthodes utiles de l'interface IHRLink.Binding.

Méthode	Rôle
IHRDataSection getSourceDataSection()	Retourne l'information source de cette correspondance.
String getSourceDataSectionName()	Retourne le nom de l'information source de cette correspondance.
IHRItem getSourceItem()	Retourne la rubrique source de cette correspondance.
String getSourceItemName()	Retourne le nom de la rubrique source de cette correspondance.
IHRDataSection getTargetDataSection()	Retourne l'information cible de cette correspondance.
String getTargetDataSectionName()	Retourne le nom de l'information cible de cette correspondance.
IHRItem getTargetItem()	Retourne la rubrique cible de cette correspondance.
String getTargetItemName()	Retourne le nom de la rubrique cible de cette correspondance.
String getValue()	Retourne la valeur définie pour cette correspondance.
int getValueDecimalCount()	Retourne le nombre de décimales de la valeur définie pour cette correspondance.
int getValueLength()	Retourne la longueur de la valeur définie pour cette correspondance.
boolean isConstantValue()	Indique si la correspondance porte sur une valeur constante.

Les types de dossier

Un type de dossier est représenté par l'interface com.hraccess.openhr.IHRDossierType.

Une structure de données permet de représenter un concept métier, par exemple, un salarié ou une règle de gestion. Ce concept peut être plus ou moins abstrait et être décliné sous des formes présentant des différences d'un point de vue fonctionnel. Par exemple, le concept de salarié représenté par une structure de données peut être abstrait par le concept de "personne". Celui-ci peut alors être décliné en différents concepts concrets tels qu'un salarié, un utilisateur, un postulant à une offre d'emploi, un retraité, etc. Parmi toutes les informations définies pour une structure de données, toutes ne sont pas valides pour tous les concepts. Par exemple, une information nommée "lettre de candidature" serait utile pour représenter un postulant à une offre d'emploi mais pas un retraité.

Le **type d'un dossier** permet de définir, parmi toutes les informations existantes pour une structure de données, lesquelles sont pertinentes pour le concept représenté.

Ainsi à chaque concept concret représenté par une structure de données, on associe un type de dossier qui définit la liste des informations autorisées pour ce concept.

Le concept de type de dossier est appelé "répertoire" (réglementaire) dans le cas particulier d'une structure de données réglementaire.

Méthodes

Le tableau suivant liste les méthodes utiles de l'interface IHRDossierType.

Méthode	Rôle
short getCodeFormat()	Retourne sous forme d'entier le format de l'identifiant (code) d'un dossier avec ce type. Les différentes valeurs que peut retourner cette méthode sont énumérées sous forme de constantes de nom FORMAT__XXX au niveau de l'interface IHRDossierType. Les formats de code valides sont : "alphabétique", "alphanumérique" et "numérique non signé".
short getCodeLength()	Retourne la longueur de l'identifiant (code) du dossier avec ce type.
Set<String> getDataSectionNames()	Retourne un Set<String> contenant les noms (identifiants) des informations autorisées pour ce type de dossier.
List<IHRDataSection> getDataSections()	Retourne une List<IHRDataSection> contenant les informations autorisées pour ce type de dossier.
IHRDataStructure getDataStructure()	Retourne la structure de données sur laquelle porte ce type de dossier.
String getDomainNumber()	Retourne sous forme de String le numéro du domaine réglementaire auquel le dossier (avec ce type) appartient. L'appel de cette méthode n'est pertinent que si le type de dossier représente un répertoire réglementaire, c'est-à-dire que la structure de données concernée est de type réglementaire.
String getLabel() Dépréciée en 7.30.50 ou supérieure, utiliser getDefaultLabel() à la place	Retourne le libellé du type de dossier. La langue du libellé correspond à la langue par défaut de la session OpenHR.
String getDefaultLabel() 7.30.50 ou supérieure	Retourne le libellé par défaut du type de dossier. La langue du libellé correspond à la langue par défaut de la session OpenHR.
String getLabel(char language) 7.30.50 ou supérieure	Retourne le libellé du type de dossier pour une langue donnée.
Map<Character, String> getLabels() 7.30.50 ou supérieure	Retourne, sous la forme d'une Map<Character, String>, les libellés par langue du type de dossier.

Méthode	Rôle
long getLastUpdateTimestamp()	Retourne sous forme d'un nombre de millisecondes l'horodatage de dernière modification côté serveur du type de dossier. Cet horodatage permet de détecter les objets ayant évolué côté serveur.
String getName()	Retourne le nom (identifiant) de ce type de dossier.
boolean isDirectory()	Indique si ce type de dossier est en réalité un répertoire réglementaire. Retourne true si et seulement si la structure de données concernée est de type réglementaire.

Les devises

Le dictionnaire donne également accès aux devises paramétrées sur HR Access.

Le référentiel des devises est le répertoire réglementaire ZD U4Y (Devise), pas la table FM d'identifiant "03" (Devise) configurée dans Design Center ! Cette dernière n'est pas complète car elle ne définit pas le nombre de décimales associées à chaque devise.

Une devise est représentée par la classe com.hraccess.openhr.HRCurrency. Les méthodes utiles de cette classe sont les suivantes :

Méthode	Rôle
String getName()	Retourne le code identifiant la devise sous forme d'une chaîne de caractères. Ce code correspond au code du dossier réglementaire du répertoire ZD U4Y définissant la devise. Il s'agit d'une chaîne de 3 caractères. Exemple : "USD" (Dollar américain), "EUR" (Euro), etc.
String getLabel()	Retourne le libellé de la devise dans la langue de la session OpenHR. Ce libellé correspond au libellé saisi dans le dossier réglementaire définissant la devise.
int getDecimal()	Retourne le nombre de décimales associées aux valeurs numériques saisies dans cette devise. Le nombre de décimales varie d'une devise à une autre. Par exemple, il n'existait pas de sous-multiple à l'ancienne lire italienne si bien que son nombre de décimales associé est à zéro. A contrario, une devise qui possède des sous-multiples comme l'euro peut posséder des décimales pour représenter des centimes.

L'interface IHRDictionary fournit les méthodes suivantes pour récupérer les devises à partir du dictionnaire.

Méthode	Rôle
Map<String, HRCurrency> getCurrencies()	Retourne sous forme d'une Map<String, HRCurrency> les devises référencées par leur nom (identifiant).
HRCurrency getCurrency(String)	Retourne sous forme de HRCurrency la devise de nom (identifiant) donné.
String[] getCurrencyNames()	Retourne sous forme d'un tableau de chaînes de caractères le nom (identifiant) des devises du dictionnaire.

```
// Retrieving the dictionary from an existing session
IHRDictionary dictionary = getSession().getDictionary();

// Dumping the currencies
Map currencies = dictionary.getCurrencies();

for(Iterator it=currencies.keySet().iterator(); it.hasNext(); ) {
    String currencyName = (String) it.next();
    HRCurrency currency = (HRCurrency) currencies.get(currencyName);

    System.out.println("Currency <" + currencyName + "> (" +
    currency.getLabel()
     + ") has " + currency.getDecimal() + " decimal(s)");
}

// Retrieving a named currency
HRCurrency dollarCurrency = dictionary.getCurrency("USD");
System.out.println("Currency <USD> (" + currency.getLabel() + ") has " +
    currency.getDecimal() + " decimal(s)");

// Dumping the currency names
String[] currencyNames = dictionary.getCurrencyNames();

for(int i=0; i<currencyNames.length; i++) {
    String currencyName = currencyNames[i];
    System.out.println("Found currency <" + currencyName + ">");
}
```

Pour information, les devises ne sont pas chargées dans le dictionnaire au moment de sa construction mais dynamiquement à l'appel d'une des méthodes citées plus haut. Les devises ne sont pas sérialisées dans le dictionnaire OpenHR.

Création d'un dictionnaire personnalisé

Une session OpenHR possède un dictionnaire construit lors de la connexion de la session. L'API OpenHR permet également de créer un (ou plusieurs) dictionnaire(s) personnalisé(s) qui possède(nt) les mêmes propriétés que le dictionnaire de la session. Chaque dictionnaire est alors distinct des autres et peut être manipulé indépendamment.

Cette possibilité peut être utilisée afin d'explorer le modèle de données HR Access.

Pour créer un dictionnaire personnalisé, il suffit d'appeler la méthode de fabrique IHRSession.newDictionary(). Le dictionnaire ainsi retourné est vide, il ne référence aucun processus HR Access. Il faut alors référencer manuellement les processus et rafraîchir le dictionnaire.

```
// Retrieving an existing session
IHRSession session = getSession();

// Creating a new dictionary
IHRDictionary dictionary = session.newDictionary();

// Referencing some HR Access processus
dictionary.addProcess("AS100");
dictionary.addProcess("FS001");
dictionary.addProcess("AG0GD");

// Refreshing the dictionary
dictionary.refresh();
```

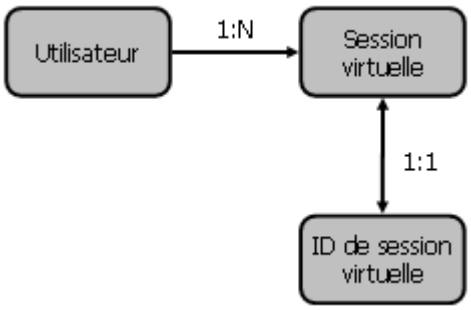
La session virtuelle

Définition

Une **session virtuelle** représente la connexion d'un utilisateur au serveur HR Access. Elle est créée sur le serveur à l'issue d'une opération de connexion comprenant :

- Une étape d'authentification de l'utilisateur (généralement à l'aide d'un mot de passe).
- Une étape de résolution de ses droits sur le système (en termes de rôle).

Un utilisateur peut ouvrir simultanément plusieurs connexions (sessions virtuelles) au serveur HR Access. Chacune est identifiée par un identifiant de session virtuelle unique :



Identifiant de session virtuelle

Un identifiant généré (par le serveur OpenHR) lors de la connexion et appelé "Identifiant de session virtuelle" identifie de manière unique une session virtuelle (donc une connexion d'utilisateur au serveur HR Access). Cet identifiant est composé de 64 caractères alphanumériques choisis parmi [a-z], [A-Z] et [0-9]. Il est véhiculé dans l'en-tête de toutes les requêtes envoyées au serveur HR Access au titre de l'utilisateur. Cet identifiant sert de token de connexion et permet d'attester que l'utilisateur à l'origine d'une requête s'est bien préalablement connecté au serveur.

Nombre de sessions virtuelles ouvertes simultanément

Le nombre de sessions virtuelles qu'un même utilisateur peut ouvrir sur le serveur HR Access est théoriquement illimité (il est seulement limité par les ressources physiques telles que la mémoire du serveur HR Access, la capacité de la base de données, etc.). Lorsqu'un utilisateur est connecté simultanément plusieurs fois au serveur HR Access, chaque session virtuelle possède son propre identifiant et est complètement indépendante des autres sessions virtuelles de l'utilisateur. Que deux sessions virtuelles appartiennent à un même utilisateur ou pas ne fait pas de différence du point de vue du serveur HR Access.

Récupération d'une session virtuelle

Lorsqu'une session virtuelle est ouverte sur le serveur HR Access, il est possible de la récupérer à partir de son identifiant. En effet, la session virtuelle est établie avec le serveur HR Access, pas avec l'application HR Access (HRa Space, par exemple) qui a permis de l'obtenir. Ce mécanisme permet ainsi à l'utilisateur de se connecter une fois au serveur HR Access via une application A puis de se connecter à d'autres applications B et C ayant pour cible le même serveur HR Access (B et C doivent être adaptées pour autoriser la récupération de la connexion de l'utilisateur).

Cette fonctionnalité de réutilisation de la connexion d'un utilisateur sert à implémenter un Single Sign-On natif basé sur la récupération d'une session virtuelle existante à partir de son identifiant.

Contrairement à ce qui était en vigueur en version 5, où le Single Sign-On d'un utilisateur impliquait d'ouvrir autant de connexions au serveur qu'il existait d'applications accédées, ici, c'est bien la même connexion d'utilisateur qui est partagée par les applications visitées. En conséquence, si l'utilisateur ferme sa session virtuelle depuis l'application C, l'accès aux applications A et B ne sera plus possible.

Ouverture et fermeture d'une session virtuelle

Une session virtuelle peut être fermée mais ne peut pas être ré-ouverte. Si l'utilisateur a besoin de se reconnecter au serveur HR Access, alors il doit ouvrir une autre session virtuelle.

Concrètement, l'ouverture d'une session virtuelle se traduit par la création d'un enregistrement dans la table technique MX10 (Sessions virtuelles). La clé primaire de cet enregistrement est l'identifiant de session virtuelle. L'enregistrement en table MX10 permet de mémoriser la description de l'utilisateur connecté. Cette description est composée de plusieurs propriétés telles que :

- L'identifiant de l'utilisateur
- Son nom de présentation
- Sa langue
- L'horodatage auquel la session virtuelle a été ouverte
- Etc.

Pour information, l'enregistrement MX10 d'un utilisateur est couplé à zéro, un ou plusieurs enregistrement(s) en table MX20 (Rôles) avec la même clé primaire. Ceux-ci représentent les rôles de l'utilisateur qui ont été calculés lors de la phase de connexion (cela sera détaillé plus loin).

C'est le fait de stocker la description de l'utilisateur et ses rôles au niveau de la base de données (et pas dans la mémoire de la JVM de l'application qui connecte l'utilisateur) qui permet à d'autres applications de récupérer cette description et ses rôles et par là même, la connexion de l'utilisateur.

Au niveau de l'API OpenHR, le concept de session virtuelle n'est pas visible directement sous forme de classe ou d'interface. Cependant sa compréhension est essentielle si vous souhaitez personnaliser la connexion des utilisateurs.

Attention à ne pas confondre les notions de "session" et de "session virtuelle" dans la suite de cette documentation.

Table MX10

La table MX10 contient les sessions virtuelles des utilisateurs connectés. Sa description est donnée ci-dessous.

Colonne	Format	Signification
VSESID	CHARACTER(64)	Identifiant de session virtuelle.
STATAL	CHARACTER(1)	Non documenté.
USERID	CHARACTER(25)	Identifiant d'utilisateur. A ne pas confondre avec l'identifiant de connexion (cf. plus bas).
LABEL	CHARACTER(100)	Nom de présentation de l'utilisateur.
CDLANG	CHARACTER(1)	Code langue de l'utilisateur.
PGPUSE	INTEGER(4)	Code PGP (cf. plus bas) de l'utilisateur.
TSTART	TIMESTAMP	Horodatage de connexion de l'utilisateur. Permet de calculer le temps de vie de la session virtuelle.
TSACTI	TIMESTAMP	Horodatage de dernière activité connue de l'utilisateur. Lorsqu'un message est reçu au titre d'une session virtuelle donnée, cet horodatage est remis à jour avec l'heure système. Il permet de détecter les sessions virtuelles "fantômes" dont l'utilisateur n'a plus été actif depuis un certain temps.
INTERN	CHARACTER(1)	Témoin indiquant si l'utilisateur a sa description stockée en interne (dans la table UC10). Permet au serveur HR Access de récupérer les rôles attribués explicitement à l'utilisateur depuis la table UC15.
USRDOS	INTEGER(4)	Numéro du dossier (de salarié) associé à l'utilisateur connecté.
USRSTD	CHARACTER(2)	Structure de données du dossier (de salarié) associé à l'utilisateur connecté.
USRATT	VARCHAR(3000)	Buffer libre permettant de stocker des informations sur l'utilisateur. Ce buffer peut être utilisé par les traitements spécifiques COBOL.

L'utilisateur

Un utilisateur est représenté par l'interface com.hraccess.openhr.IHRUser.

Identification

Un utilisateur HR Access possède deux identifiants qu'il convient de bien distinguer :

- Le premier identifiant, nommé "identifiant de connexion" ("login ID" en anglais), permet à l'utilisateur de se connecter à HR Access. Il est typiquement saisi sur le formulaire de connexion de HRa Space. Il s'agit d'une chaîne alphanumérique qui peut faire jusqu'à 254 caractères de long. La longueur de l'identifiant autorise l'utilisation d'adresses électroniques pour connecter un utilisateur. Cet identifiant étant visible de l'utilisateur final, il doit être significatif pour lui, c'est-à-dire être porteur d'une sémantique intuitive.
- Le second identifiant, nommé "identifiant d'utilisateur" ("user ID" en anglais), est une chaîne alphanumérique qui peut faire jusqu'à 25 caractères de long. Il s'agit d'un identifiant technique qui n'est pas visible de l'utilisateur final et qui est attribué par le serveur HR Access (au moyen d'un traitement spécifique) à l'utilisateur qui se connecte.

Différences entre ces deux identifiants

Un identifiant de connexion est typiquement construit en utilisant des données propres à l'utilisateur final comme ses nom et prénom ou son matricule, ce qui facilite sa mémorisation. Or ces données peuvent évoluer dans le temps en raison, par exemple, d'un changement d'état-civil du salarié ou d'un changement de politique d'immatriculation des salariés au sein de l'entreprise. L'identifiant de connexion étant visible de l'utilisateur final, il doit refléter ces changements. Si les actions d'un utilisateur sont consignées (loggées) avec, comme clé, l'identifiant de connexion de l'utilisateur, lorsqu'un tel changement survient, l'utilisateur perd son historique de mise à jour (à moins de répercuter la modification de clé sur celui-ci, ce qui peut se révéler lourd).

Pour éviter cela, toutes les actions utilisateur qui doivent être consignées le sont à l'aide d'un identifiant technique pérenne : l'identifiant d'utilisateur. Ce dernier ne change pas quand l'état-civil du salarié évolue car il ne porte pas de sémantique, il s'agit d'une clé purement technique comparable à la clé primaire d'une base de données relationnelle.

Lors de la connexion, l'utilisateur saisit son identifiant de connexion. HR Access calcule alors un identifiant d'utilisateur et l'attribue à l'utilisateur. C'est cet identifiant qui est utilisé par la suite par le serveur HR Access : l'identifiant de connexion n'est plus utilisé et n'est plus disponible une fois l'étape de connexion passée.

Connexion

En standard, les utilisateurs sont définis via Design Center et stockés dans la table technique UC10. Il est possible de connecter des utilisateurs décrits de manière externe (dans un annuaire LDAP, par exemple) mais pour simplifier les explications à ce stade, le cas standard va être envisagé : l'utilisateur est défini via Design Center et stocké en table UC10. Il s'authentifie à l'aide d'un mot de passe (stocké en colonne UC10 CDPASS).

Pour connecter un utilisateur au serveur HR Access, il faut appeler l'une des méthodes connectUser() déclarées au niveau de l'interface IHRSession (la session agit comme une fabrique d'utilisateurs).

Pour connecter un utilisateur à l'aide de son identifiant de connexion et de son mot de passe, il faut exécuter le code suivant :

```
// Retrieving an existing session
IHRSession session = getSession();


String loginId = “HRUSER”;
String password = “SECRET”;


try {
    // Connecting a HR Access user from a given login ID and password
    IHRUser user = session.connectUser(loginId, password);


    System.out.println(“User <” + user.getUserId() + “> is connected”);
} catch (UserConnectionException e) {
    // An error occurred during the user connection (event handling not
    detailed here)
} catch (AuthenticationException e) {
    // The authentication failed (event handling not detailed here)
}
```

La méthode IHRSession.connectUser(String, String) prend en premier paramètre l'identifiant de connexion de l'utilisateur et en second paramètre son mot de passe de connexion. L'appel de cette méthode peut retourner de trois manières différentes :

- La connexion réussit et la méthode retourne une instance de IHRUser représentant l'utilisateur connecté. Pour rappel, cet utilisateur est associé à une session virtuelle côté serveur HR Access.
- La connexion échoue car l'authentification a été rejetée, la méthode lève une AuthenticationException.
- L'authentification réussit mais le reste de la connexion (la résolution des rôles, par exemple) échoue ; la méthode lève alors une UserConnectionException.

Il est possible, lors de la demande de connexion, de modifier le mot de passe de l'utilisateur. Pour cela, il faut appeler la méthode IHRSession.connectUser(String, String, String). Le troisième paramètre représente le nouveau mot de passe à prendre en compte. Il va sans dire que pour que le changement de mot de passe soit effectif, il faut que le mot de passe de connexion soit correct.

```
// Retrieving an existing session
IHRSession session = getSession();


String loginId = “HRUSER”;
String password = “SECRET”;
String newPassword = “TERCES”;


try {
    // Connecting a HR Access user from a given login ID and password and changing its password
    IHRUser user = session.connectUser(loginId, password, newPassword);


    System.out.println(“User <” + user.getUserId() + “> is connected and password has been changed”);
} catch (UserConnectionException e) {
    // An error occurred during the user connection (event handling not detailed here)
} catch (AuthenticationException e) {
    // The authentication failed (event handling not detailed here)
}
```

L'interface IHRSession compte trois autres méthodes de nom connectUser() qui seront détaillées plus loin. Sachez simplement que les deux méthodes détaillées ci-dessus sont des méthodes qui s'appuient sur ces trois méthodes et simplifient leur utilisation.

Déconnexion

Pour déconnecter un utilisateur, il faut appeler la méthode IHRUser.disconnect().

```
// Retrieving a connected user
IHRUser user = getUser();


try {
    // Saving the user ID (won't be available any more after the user disconnection)
    String userId = user.getUserId();
```

```
    // Disconnecting user
    user.disconnect();

    System.out.println("User <" + user.getUserId() + "> has been
    disconnected");
} catch (UserConnectionException e) {
    // An error occurred during the user disconnection (event handling not
    detailed here)
}
```

Si l'appel de la méthode IHRUser.disconnect() réussit, la session virtuelle associée à l'utilisateur est fermée côté serveur (et son enregistrement en table MX10 supprimé). En cas d'erreur lors de la déconnexion, la méthode lève une UserConnectionException.

Il existe une autre manière de déconnecter un utilisateur qui consiste à appeler la méthode IHRSession.disconnectUser(String). Le paramètre de cette méthode de type String correspond à l'identifiant de session virtuelle associée à l'utilisateur à déconnecter (récupérable par la méthode IHRUser.getVirtualSessionId()).

```
// Retrieving an existing session and a connected user
IHRSession session = getSession();
IHRUser user = getUser();

try {
    // Saving the user ID (won't be available any more after the user
    disconnection)
    String userId = user.getUserId();

    // Disconnecting user by its virtual session ID
    session.disconnectUser(user.getVirtualSessionId());

    System.out.println("User <" + user.getUserId() + "> has been
    disconnected");
} catch (UserConnectionException e) {
    // An error occurred during the user disconnection (event handling not
    detailed here)
}
```

Différence entre ces deux méthodes

La première méthode nécessite de disposer de la référence au IHRUser à déconnecter. Or, pour avoir cette référence, il faut soit que la session ait servi à connecter l'utilisateur (auquel cas cette référence est présente dans le cache des utilisateurs de la session et accessible tout de suite), soit la créer dynamiquement en récupérant la connexion de l'utilisateur à partir de son identifiant de session virtuelle (cette fonctionnalité est détaillée juste après), ce qui peut se révéler relativement lourd, d'autant que la connexion récupérée est ensuite immédiatement fermée. La méthode IHRSession.disconnectUser(String) permet de déconnecter un utilisateur sans disposer d'une référence à un IHRUser, elle évite donc de récupérer la connexion d'utilisateur.

Récupération d'une connexion

L'API permet de récupérer la connexion d'un utilisateur à l'aide de l'identifiant de session virtuelle qui lui est associé.

Dans quels cas utiliser cette fonctionnalité

Il existe deux cas de figure :

- Pour connecter l'utilisateur à une application à l'aide d'un mécanisme de Single Sign-On. L'identifiant de session virtuelle permet de récupérer une instance de IHRUser représentant un utilisateur connecté.
- Pour déployer une application en cluster sans affinité de session. Si une application Web bâtie sur l'API OpenHR est déployée en mode cluster sans affinité de session, chaque requête Web émise par le navigateur peut être dirigée vers un nœud différent du cluster. Dans ce cas, l'application Web doit être "stateless" (c'est-à-dire ne mémoriser côté serveur Web aucune information de session) et être capable de récupérer la connexion de l'utilisateur à partir de la requête http. La méthode IHRSession.retrieveUser(String) permet de récupérer la connexion d'un utilisateur sans la stocker sous forme d'attribut de session http.

L'exemple suivant montre comment récupérer la connexion d'un utilisateur.

```
// Retrieving an existing session
IHRSession session = getSession();


// Retrieving the virtual session ID (from a web cookie for instance) (not detailed here)
String virtualSessionId = getVirtualSessionId();


// Retrieving the user connection from a given virtual session ID
IHRUser user = session.retrieveUser(virtualSessionId);


System.out.println("User <" + user.getUserId() + "> has been (re)connected");
```

Lors de l'appel de la méthode IHRSession.retrieveUser(String), l'API effectue une requête au serveur HR Access et lui demande la description d'utilisateur ainsi que les rôles associés à la session virtuelle d'identifiant donné (stockés respectivement en table MX10 et MX20). Ces données permettent alors de créer en mémoire une nouvelle instance de IHRUser.

Avec cette fonctionnalité, il est donc possible d'avoir plusieurs applications clientes OpenHR distinctes sur lesquelles un même utilisateur est connecté. C'est précisément ce qui se passe dans le cas de HRa Space : l'utilisateur connecté au portail est connecté au titre d'une session virtuelle qui est partagée par les différentes applications Web hébergées par HRa Space.

Notification

Il est possible d'écouter les événements levés par un IHRUser. Il faut alors implémenter l'interface com.hraccess.openhr.event.UserChangeListener et s'enregistrer en tant qu'auditeur auprès d'un utilisateur connecté. Les événements levés par un IHRUser sont représentés par la classe com.hraccess.openhr.event.UserChangeEvent.

Etant donné qu'un utilisateur ne peut être reconnecté et que sa récupération et sa connexion sont prises en charge par les méthodes HRSession.connectUser(), le seul événement qu'il est possible d'intercepter correspond à la déconnexion de l'utilisateur.

Pour ne plus être notifié des événements d'une session, il faut utiliser la méthode IHRSession.removeSessionChangeListener(SessionChangeListener).

Enfin, la méthode IHRUser.clearChangeListeners() permet de supprimer tous les listeners enregistrés au niveau d'un utilisateur.

```
// Retrieving a connected user
IHRUser user = getUser();

UserChangeListener listener = new UserChangeListener() {
    public void userStateChanged(UserChangeEvent event) {
    IHRUser user = (IHRUser) event.getSource();

    if (!user.isConnected()) {
            // Process the user's disconnection event (not detailed here)
    }
    }
};

user.addUserChangeListener(listener);

// Disconnecting the user
user.disconnect() ;
```

```
user.removeUserChangeListener(listener);
user.clearUserChangeListeners();
```

Propriétés

L'interface IHRUser déclare plusieurs méthodes listées dans le tableau ci-dessous. La liste qui suit n'est pas exhaustive, les méthodes manquantes seront présentées en temps utile lorsque seront présentés les concepts de rôle et de conversation.

Méthode	Rôle
IHRUser.Description getDescription()	Retourne sous forme d'une instance de IHRUser.Description la description de l'utilisateur connecté. Pour information, cette description correspond à la description persistée en base de données sous forme d'un enregistrement de table MX10 et résolue lors de la connexion de l'utilisateur.
String getLabel()	Retourne le nom de présentation (libellé) de l'utilisateur connecté.
char getLanguage()	Retourne sous forme d'un caractère le code langue associé à l'utilisateur connecté. La liste des valeurs valides retournées par cette méthode est issue de la table FM de code "04" (Langues). Exemple : "F" (Français), "U" (Anglais), "D" (Allemand), etc.
IHRSession getSession()	Retourne la session à laquelle l'utilisateur est lié. Cette session est celle qui a permis de connecter l'utilisateur ou de récupérer sa connexion.
String getUserId()	Retourne l'identifiant (d'utilisateur) associé à l'utilisateur connecté. Cette chaîne de caractères identifie de manière unique l'utilisateur et peut contenir jusqu'à 25 caractères. Attention de ne pas confondre "identifiant de connexion" et "identifiant d'utilisateur".

Méthode	Rôle
String getVirtualSessionId()	Retourne l'identifiant de session virtuelle associée à l'utilisateur connecté.
List<Authentication.Warning> getWarnings()	Retourne sous forme d'une liste de Authentication.Warnings les éventuelles alertes levées lors de la connexion de l'utilisateur. Ces alertes servent à notifier l'utilisateur d'un problème latent comme l'expiration prochaine d'un mot de passe.
boolean isConnected()	Indique si l'utilisateur est connecté. La méthode retourne true tant que la méthode IHRUser.disconnect() (ou IHRSession.disconnectUser(String)) n'a pas été appelée.
boolean isValid()	Indique si l'utilisateur est encore valide. Cette méthode est utile afin de déterminer si une instance de IHRUser connectée au niveau d'une session OpenHR est encore associée côté serveur HR Access à une session virtuelle ouverte. En effet, en raison du partage possible de la connexion d'un utilisateur entre plusieurs applications, il est possible qu'une application déconnecte l'utilisateur (avec sa session virtuelle), invalidant ainsi les autres instances de IHRUser pointant vers cette session virtuelle. L'appel de cette méthode déclenche une requête au serveur afin de déterminer si l'enregistrement de table MX10 associé à la session virtuelle existe encore (ce que ne fait pas la méthode isConnected()).

Description

La description d'un utilisateur est représentée par l'interface com.hraccess.openhr.IHRUser.Description. La description de l'utilisateur est résolue lors de sa connexion. En standard, elle est récupérée depuis la table technique UC10 qui définit les utilisateurs. La classe IHRUser.Description est un bean représentant la description de l'utilisateur telle qu'elle est persistée côté serveur HR Access en table MX10.

Le tableau suivant liste les méthodes utiles de cette classe.

Méthode	Rôle
Timestamp getConnectionTimestamp()	Retourne l'horodatage de connexion de l'utilisateur, c'est-à-dire le moment où la session virtuelle a été créée. Correspond à la colonne MX10 TSTART.
String getLabel()	Voir méthode IHRUser.getLabel(). Correspond à la colonne MX10 LABEL.
char getLanguage()	Voir méthode IHRUser.getLanguage(). Correspond à la colonne MX10 CDLANG.
String getPgp()	Retourne le "code PGP" associé à l'utilisateur. PGP est l'abréviation de "Partition Policy Group" et désigne une technique de partitionnement des données applicatives en base de données qui permet à une partition de ne pas voir les données situées dans une autre partition. Lorsqu'un utilisateur est associé à un code PGP, il ne peut voir que les dossiers situés dans cette partition. Le partitionnement de la base de données permet donc d'appliquer un filtrage aux données visibles de l'utilisateur qui vient s'ajouter à la confidentialité classique définie par un rôle. Le code PGP est une chaîne de 9 caractères. L'utilisation d'un tel partitionnement n'a de sens que lorsque HR Access héberge des données de sociétés différentes au sein de la même base de données, ce qui est typiquement le cas d'un hébergement en infogérance. Correspond à la colonne MX10 PGPUSE.

Méthode	Rôle
String getUserDossierDataStructure()	Retourne le code de la structure de données du dossier associé à l'utilisateur connecté. Lors de la connexion, il est possible d'associer (à l'aide d'un traitement spécifique) à l'utilisateur connecté un dossier HR Access. Cela permet, par exemple, d'associer à un utilisateur connecté son dossier de salarié (de structure de données ZY). Correspond à la colonne MX10 USRSTD.
int getUserDossierNudoss()	Retourne le numéro du dossier associé à l'utilisateur connecté. Lors de la connexion, il est possible d'associer (à l'aide d'un traitement spécifique) à l'utilisateur connecté un dossier HR Access. Cela permet, par exemple, d'associer à un utilisateur connecté son dossier de salarié (de structure de données ZY). Le numéro de dossier désigne la clé primaire de l'enregistrement en table xx00 représentant ce dossier (où xx désigne le code de la structure de données du dossier). Correspond à la colonne MX10 USRDOS.

Le rôle

Définition

Le concept de rôle est représenté par l'interface com.hraccess.openhr.IHRRole.

Lors de la connexion, l'utilisateur se voir attribuer zéro, un ou plusieurs rôle(s) qui sont résolus par le serveur HR Access à l'aide d'une logique personnalisable (sous forme de traitement spécifique COBOL). Toute la sécurité HR Access repose sur l'utilisation de rôles ; ainsi, l'accès à un dossier doit s'effectuer en précisant le rôle au titre duquel l'accès est réalisé. Si la sécurité du rôle permet l'opération demandée sur le dossier (lecture, mise à jour, suppression, etc.), alors l'opération est autorisée. Le contrôle de cette sécurité est assuré par le serveur HR Access, c'est pourquoi toutes les requêtes émises au serveur HR Access traitant de données applicatives doivent véhiculer le rôle au titre duquel l'accès est réalisé.

Les rôles sont décrits (paramétrés) via Design Center puis déployés afin de pouvoir être utilisés par HR Access.

Le concept de rôle n'est pas détaillé ici car ce n'est pas l'objet de cette documentation. Pour plus d'informations, reportez-vous au *Guide de* gestion de *la sécurité* et au *Compagnon* de Design Center.

Identification

Les rôles sont identifiés de manière unique par leur "forme canonique". Il s'agit d'une chaîne de caractères calculée de la manière suivante :

```
<Forme canonique> = <Modèle de rôle> + « ( » + <Paramètre de rôle> + « ) »
```

Le modèle de rôle est une chaîne alphanumérique de 20 caractères au maximum qui ne peut pas être vide.

Le paramètre de rôle est une chaîne alphanumérique de 64 caractères au maximum qui peut prendre la valeur vide.

Exemples de forme canonique :

- EMPLOYEE(123456) désigne le rôle "EMPLOYEE" de matricule 123456
- MANAGER(ORG_UNIT_1) désigne le rôle "MANAGER" pour l'unité organisationnelle ORG_UNIT_1
- DRH() désigne le rôle "DRH" sans paramètre de rôle

Récupération des rôles

L'interface IHRUser déclare plusieurs méthodes pour manipuler les rôles.

Méthode	Rôle
IHRRole getRole(String)	Retourne le rôle de l'utilisateur connecté de forme canonique donnée. Si la forme canonique n'est pas valide ou désigne un rôle que l'utilisateur ne possède pas, retourne null.
int getRoleCount()	Retourne le nombre de rôles que l'utilisateur connecté possède.
Map<String, IHRRole> getRoleMap()	Retourne une Map<String, IHRRole> contenant les rôles de l'utilisateur connecté référencés par forme canonique.
List<IHRRole> getRoles()	Retourne une List<IHRRole> contenant les rôles de l'utilisateur connecté.
boolean hasRole()	Indique si l'utilisateur possède au moins un rôle.
boolean hasRole(IHRRole)	Indique si l'utilisateur possède le rôle donné.
boolean hasRole(String)	Indique si l'utilisateur possède le rôle de forme canonique donnée.

Méthodes

Le tableau suivant liste les méthodes utiles de l'interface IHRRole. Cette liste n'est pas exhaustive car un IHRRole donne accès à de nombreuses informations paramétrées via Design Center dont la connaissance n'est pas nécessaire pour manipuler des dossiers HR Access. Seules les informations utiles sont présentées ici. Le reste sera détaillé plus loin.

Méthode	Rôle
AccessLevel getAccessLevel(String, String)	Retourne le niveau d'accès défini pour ce rôle et pour l'information de structure de données (premier paramètre) et de code donné (second paramètre). Le niveau d'accès est défini sous forme d'une instance de AccessLevel. Cette classe énumère les différents niveaux d'accès possibles : "non défini", "interdit", "lecture seule", "mise à jour seule" et "création / suppression". Il indique les droits d'accès du rôle par rapport à une information ou un dossier.
String getCanonicalForm()	Retourne la forme canonique du rôle.
Category getCategory()	Retourne la catégorie à laquelle appartient ce rôle. Les catégories de rôle sont énumérées sous forme de constantes au niveau de la classe IHRRole.Category. Les catégories de rôle valides sont "collaborateur", "manager" et "Expert RH.
String getLabel()	Retourne le libellé du rôle. Ce libellé est calculé à l'aide d'un traitement spécifique.
String getLocalization()	Retourne la localisation rattachée au rôle. Pour plus d'informations sur le concept de localisation, reportez-vous aux documentations décrivant les rôles (*Guide de gestion de la sécurité* et *Compagnon* de Design Center).
String getParameter()	Retourne la valeur du paramètre de ce rôle.
int getPgp()	Retourne le code PGP rattaché à ce rôle. Ce code PGP désigne l'identifiant de partition de base de données HR Access dont les dossiers sont visibles du rôle.

Méthode	Rôle
String getCustomerId()	Dans le cas d'une plate-forme multi-clients, retourne l'identifiant client associé au rôle.
boolean isSuperUser()	Dans le cas d'une plate-forme multi-clients, indique si ce rôle correspond à un "super user", pour lequel le filtrage automatique des données par client ne s'applique pas.
Timestamp getPublishingTimestamp()	Retourne l'horodatage de déploiement du rôle.
String getTemplate()	Retourne l'identifiant de modèle de rôle.
String getTemplateLabel()	Retourne le libellé du modèle de rôle dans la langue par défaut (celle de la session OpenHR).
String getTemplateLabel(char)	Retourne le libellé du modèle de rôle dans la langue de code donné.

Table MX20

La table MX20 contient la description des rôles associés à une session virtuelle. La structure de cette table est donnée ci-dessous.

Colonne	Format	Signification
VSESID	CHARACTER(64)	Identifiant de session virtuelle auquel est rattaché le rôle.
ROLMOD	CHARACTER(20)	Modèle de rôle.
ROLVAL	CHARACTER(64)	Valeur d'instanciation du rôle.
FIMPLI	CHARACTER(1)	Indique si le rôle a été attribué implicitement (par traitement) ou explicitement (manuellement).
TYDELE	CHARACTER(1)	Type de délégation associé au rôle (s'il y a lieu). Les valeurs que peut prendre cette colonne sont énumérées sous forme de constantes au niveau de la classe IHRRole.Delegation (safe type enumeration). Les types de délégation valides sont "aucune", "rôle donné à un tiers", "rôle partagé avec un tiers" et "rôle reçu d'un tiers".
ROWNER	CHARACTER(25)	Identifiant de l'utilisateur qui a délégué son rôle (s'il y a lieu).
STATAL	CHARACTER(1)	Non documenté.
CACTOR	CHARACTER(5)	Catégorie à laquelle appartient ce rôle. Les valeurs que peut prendre cette colonne sont énumérées sous forme de constantes au niveau de la classe IHRRole.Category (safe type enumeration). Les catégories de rôle valides sont "collaborateur", "manager" et "Expert RH".
MISSIO	CHARACTER(8)	Mission dont dérive le rôle (s'il y a lieu).
NNSLMX	INTEGER(4)	Nombre maximal de dossiers que le rôle autorise à sélectionner.
STRUCT	CHARACTER(8)	Structure dont dérive le rôle (s'il y a lieu).
PGPROL	INTEGER(4)	Code PGP associé au rôle.
ROLDOS	INTEGER(4)	Numéro du dossier associé au rôle (s'il y a lieu).
IDLOCA	CHARACTER(2)	Localisation associée au rôle.
ROLATT	VARCHAR(900)	Buffer utilisé afin de stocker de manière banalisée les attributs relatifs au rôle.
ZONBCR	VARCHAR(2600)	Non documenté.
METHOD	CHARACTER(8)	Non documenté.

La conversation

Introduction

Une conversation est représentée par l'interface com.hraccess.openhr.IHRConversation.

Précisons que l'utilisation d'une conversation ne se justifie que si vous manipulez des informations d'un type spécial, appelées "informations paramètres".

Informations paramètres

Une **information paramètre** permet de passer des paramètres (et leurs valeurs) au serveur HR Access afin que celui-ci les persiste dans un contexte côté serveur (i.e. une table relationnelle) en vue d'une utilisation ultérieure.

Si vous n'avez pas fonctionnellement besoin d'informations paramètres, alors le concept de conversation vous est inutile. Cependant, l'API devant supporter tous les cas de figure possibles, elle impose l'utilisation d'une conversation lors de l'accès aux données, même si cela n'est pas justifié par l'utilisation d'informations paramètres.

Pour résumer, l'utilisation d'une conversation est obligatoire pour des raisons de signature d'API mais cela ne se justifie vraiment que lors de l'utilisation d'informations d'un type spécial, les "informations paramètres".

Conversation : identification et cycle de vie

La **conversation** représente une session de dialogue entre un utilisateur connecté et le serveur HR Access.

La conversation est identifiée par un numéro (dit "numéro de conversation") sur quatre positions dont la numérotation commence à "0000" puis est généré de manière incrémentale par le serveur HR Access. Le nombre maximal de conversations qu'un utilisateur peut donc ouvrir est de 9999.

Le cycle de vie d'une conversation est délimité par les événements d'ouverture puis de fermeture. Lors de la demande de création (ou d'ouverture) d'une conversation, c'est le serveur HR Access (qui mémorise la liste des conversations ouvertes d'un utilisateur en table MX40 - Conversations) qui attribue le numéro de conversation afin d'éviter les collisions lors de la numérotation. Lors de la fermeture de la conversation, celle-ci est supprimée de la liste des conversations ouvertes de l'utilisateur. Comme avec une connexion d'utilisateur, lorsqu'une conversation est fermée, il n'est pas possible de la rouvrir, il faut en ouvrir une autre.

Conversation 0000

La conversation de numéro "0000" est particulière car son ouverture a implicitement lieu lors de la connexion d'un utilisateur ou de la récupération de sa connexion. Cette conversation spéciale est appelée "conversation principale". Pour les raisons exposées ci-dessus relatives à l'utilisation des informations paramètres, si vous n'utilisez pas ce type particulier d'informations, alors l'utilisation de la conversation principale suffit et il n'est pas nécessaire de créer d'autres conversations.

Une autre particularité de cette conversation est qu'elle ne peut être fermée (autrement l'utilisateur ne pourrait plus dialoguer avec le serveur HR Access). Sa fermeture survient lors de la déconnexion de l'utilisateur.

Autres conversations

Les conversations dont le numéro est supérieur à "0000" sont des conversations créées explicitement par le développeur (par l'appel d'une méthode qui sera exposée ci-dessous) et sont appelées "conversations secondaires". Celles-ci peuvent être ouvertes et fermées à volonté.

La conversation est propre à l'utilisateur, c'est pourquoi les méthodes permettant de la créer (ou récupérer) sont situées au niveau de l'interface IHRUser (cf .ci-dessous). Les conversations principales de deux utilisateurs A et B connectés à HR Access sont donc indépendantes.

Le tableau suivant liste les méthodes de l'interface IHRUser relatives à la gestion des conversations.

Méthode	Rôle
IHRConversation createConversation()	Crée et retourne une nouvelle conversation (secondaire) pour cet utilisateur. L'appel de cette méthode se traduit par une requête au serveur HR Access, lequel attribue un nouveau numéro de conversation pour l'utilisateur (et le mémorise en table MX40).
IHRConversation getMainConversation()	Retourne la conversation principale de cet utilisateur. L'appel de cette méthode ne se traduit pas par une requête au serveur HR Access car la conversation principale est dynamiquement créée par le serveur au moment de la connexion de l'utilisateur.
IHRConversation retrieveConversation(String)	Récupère et retourne la conversation de numéro donné pour cet utilisateur. Cette méthode permet de récupérer une conversation (secondaire) existante. Elle est similaire à la méthode IHRSession.retrieveUser(String) qui permet de récupérer une connexion d'utilisateur à partir de son identifiant de session virtuelle.

Méthodes

L'interface IHRConversation offre les méthodes suivantes.

Méthode	Rôle
void close()	Ferme la conversation côté serveur HR Access. L'appel de cette méthode se traduit par une requête au serveur HR Access pour fermer la conversation et supprimer sa référence des conversations de l'utilisateur connecté (située en table MX40).
String getConversationNumber()	Retourne le numéro (identifiant) de cette conversation.
IHRUser getUser()	Retourne l'utilisateur auquel cette conversation est liée.
boolean isClosed()	Indique si la conversation est fermée.

L'interface déclare une méthode send(HRUserMessage, IHRRole) qui sera détaillée plus loin. Sachez simplement qu'un utilisateur ne peut envoyer de requête (en son nom) au serveur HR Access qu'en utilisant une conversation.

Notification

Il est possible d'écouter les événements levés par une IHRConversation. Il faut alors implémenter l'interface com.hraccess.openhr.event.ConversationChangeListener et s'enregistrer en tant qu'auditeur auprès d'une conversation. Les événements levés par une IHRConversation sont représentés par la classe com.hraccess.openhr.event.ConversationChangeEvent.

Etant donné qu'une conversation ne peut être rouverte et que son ouverture est prise en charge par la méthode IHRUser.createConversation(), le seul événement qu'il est possible d'intercepter correspond à la fermeture de la conversation.

Pour ne plus être notifié des événements d'une conversation, il faut utiliser la méthode IHRConversation.removeConversationChangeListener(ConversationChangeListener).

```
// Creating a new (secondary) conversation from an existing user
IHRConversation conversation = getUser().createConversation();

ConversationChangeListener listener = new ConversationChangeListener() {
    public void conversationStateChanged(ConversationChangeEvent event) {
    IHRConversation conversation = (IHRConversation) event.getSource();

    if (conversation.isClosed()) {
        // Process the conversation's disconnection event (not detailed here)
    }
```

```
    }
};

conversation.addConversationChangeListener(listener);

// Closing the conversation
conversation.close();

conversation.removeConversationChangeListener(listener);
```

Envoi simultané de messages

La documentation présentera plus loin comment émettre des requêtes au serveur HR Access au titre d'un utilisateur connecté et de l'un de ses rôles. Cela s'effectue à l'aide de la méthode IHRConversation.sendMessage(HRUserMessage, IHRRole).

La conversation est capable de gérer l'envoi simultané de requêtes au serveur HR Access sans que cela ne produise de collisions au niveau du serveur HR Access. La limite du nombre de requêtes qu'une conversation est capable d'émettre simultanément vers le serveur HR Access est fixée à 256. Cette valeur n'est pas modifiable. Dans la précédente version, cette limite était fixée à 8 requêtes simultanées.

L'utilisateur de session

Définition

L'utilisateur de session est représenté par l'interface com.hraccess.openhr.IHRSessionUser (sous-type de IHRUser).

La fonctionnalité dite "utilisateur de session" est une facilité offerte par l'API OpenHR pour accéder aux dossiers HR Access **sans confidentialité**. Ceci en fait donc une fonctionnalité critique du point de vue de la sécurité. C'est pourquoi elle bénéficie d'une sécurisation spécifique qui sera détaillée plus loin.

L'utilisateur de session est un utilisateur fictif mis à disposition par l'API pour accéder au serveur HR Access sans confidentialité. Le côté fictif de cet utilisateur est dû au fait qu'il s'agit d'un utilisateur "en mémoire" : il n'est pas réellement connecté au serveur HR Access car il ne possède pas de session virtuelle (en table MX10) ni de rôle réel (en table MX20). Son obtention est instantanée car elle ne requiert pas de requête au serveur HR Access : tout se passe au niveau de la JVM. Corollaire : l'utilisateur n'étant pas réellement connecté, il n'est pas possible de le déconnecter.

Comme l'utilisateur de session ne possède pas de session virtuelle, il ne peut pas ouvrir de conversation secondaire vers le serveur HR Access. Il ne possède en fait qu'une conversation principale, fictive elle aussi, car associée à aucun enregistrement en table MX40.

Rôle "fictif"

L'API impose, lorsque l'on souhaite accéder aux données applicatives (dossiers HR Access) de fournir une conversation (pour le contexte d'informations paramètres) et un rôle (pour définir la confidentialité). Or l'utilisateur de session ne possède pas de vrai rôle qui lui donnerait tous les droits. A la place, on lui attribue un rôle fictif "en mémoire" qui permet de court-circuiter les contrôles de confidentialité lors de l'accès aux données. Ce rôle fictif est connu "en dur" du serveur HR Access qui désactive les contrôles de confidentialité lorsqu'il le détecte. Il est repérable par son modèle de rôle constitué de 20 blancs et par son paramètre de rôle constitué de 64 blancs.

Limitations de l'utilisateur de session

Une des limitations de l'utilisateur de session est qu'il ne peut être utilisé là où un rôle réel est nécessaire. C'est le cas par exemple lors de la soumission d'une demande de travail (dossier de structure de données ZO). Dans ce cas-là, le rôle de l'utilisateur qui soumet la demande y est mémorisé et sert à calculer la confidentialité à appliquer aux données. Il en est de même lors de la soumission d'un Processus Guidé.

Enfin, l'utilisateur de session ne peut manipuler d'information paramètre en raison du caractère fictif de sa conversation principale.

Avertissement

Cette fonctionnalité est critique car elle donne un accès non contrôlé à la base de données HR Access.

C'est pourquoi l'API permet de configurer la sécurité des communications entre client et serveur OpenHR afin d'authentifier les clients OpenHR qui utilisent cette fonctionnalité (à l'aide d'un SSL bidirectionnel).

Il est par conséquent **impératif** de sécuriser le serveur OpenHR sur un environnement critique (comme la production) afin de fermer cette porte d'entrée vers la base HR Access ! Si cette sécurité n'est pas mise en place, les autres mesures de sécurité sont parfaitement inutiles.

Récupération

Pour récupérer l'utilisateur de session, il faut appeler l'une des méthodes IHRSession.getSessionUser().

Méthode	Rôle
IHRSessionUser getSessionUser()	Retourne l'utilisateur de session rattaché à cette session OpenHR en le créant dynamiquement si nécessaire. Deux appels consécutifs à la méthode retournent la même instance de IHRSessionUser (singleton).
IHRSessionUser getSessionUser(IHRUser)	Retourne un utilisateur de session agissant au nom de l'utilisateur donné. Deux appels consécutifs à la méthode retournent des instances différentes de IHRSessionUser. Cette méthode permet d'effectuer un accès à la base de données au titre d'un utilisateur connecté. Cela permet notamment de récupérer certains libellés dans la langue de cet utilisateur et de tenir compte du code PGP de l'utilisateur pour lequel les données sont lues.

```
// Retrieving an existing session and user
IHRSession session = getSession();
IHRUser user = getUser();

// Retrieving the session user
IHRSessionUser sessionUser = session.getSessionUser();

// Retrieving the session user on behalf of a given connected user
IHRSessionUser sessionUser2 = session.getSessionUser(user);
```

Un IHRSessionUser étant un IHRUser, une fois un utilisateur de session récupéré, il est possible d'accéder aux dossiers HR Access de la même manière qu'avec un véritable utilisateur. Le code qui manipule des dossiers peut donc être écrit de manière générique et le choix de l'utilisation d'un utilisateur de session ou d'un véritable utilisateur peut être considéré comme du paramétrage pouvant être reporté à plus tard.

Méthodes

L'interface IHRSessionUser déclare une méthode supplémentaire par rapport à l'interface IHRUser.

Méthode	Rôle
IHRRole getRole()	Retourne le rôle fictif associé à l'utilisateur de session. Ce rôle est caractérisé par un modèle de rôle et une valeur d'instanciation "à blanc".

```
// Retrieving an existing session and user
IHRSession session = getSession();

// Retrieving the session user
IHRSessionUser sessionUser = session.getSessionUser();

// Retrieving the session user's unique (fake) role
IHRRole role = sessionUser.getRole()
```

La gestion de dossiers

Présentation

La fonctionnalité de gestion de dossiers HR Access est la principale fonctionnalité de l'API OpenHR. Il existe d'autres fonctionnalités utiles dont la liste est présentée en début de cette documentation. Cependant, la gestion de dossiers constitue la grande majorité des cas d'usage de l'API.

Commençons par aborder la philosophie de cette fonctionnalité. Le but est de donner au développeur un aperçu de la manière dont celle-ci fonctionne et de lui permettre de faire des parallèles avec des techniques / frameworks issus du monde Java.

La fonctionnalité de gestion de dossiers HR Access permet de manipuler des dossiers sous forme d'objets. Le modèle de données des dossiers est dynamique : l'API permet de manipuler n'importe quel type de dossier car le modèle représente un paramétrage de l'API.

L'accès aux données des dossiers requiert l'utilisation de deux objets qui sont la conversation et le rôle.

Conversation et rôle

La conversation permet de préciser les (éventuelles) informations paramètres qui doivent être prises en compte (paramètres contextuels persistés côté serveur HR Access).

Le rôle, lui, définit la confidentialité appliquée lors de l'accès aux données.

Il est intéressant de noter que pour manipuler des dossiers HR Access, on n'utilise pas la notion d'utilisateur mais les notions de conversation et de rôle qui sont rattachées à l'utilisateur. Précisons également que cette conversation et ce rôle peuvent être récupérés depuis un véritable utilisateur ou depuis un utilisateur de session !

Accès aux dossiers

Avant d'accéder à un dossier, il est nécessaire de décrire sa structure et de préciser notamment les informations du dossier que l'on souhaite manipuler.

La lecture des dossiers ne s'effectue pas directement depuis la base de données comme ce serait le cas avec une API de bas niveau telle que JDBC. A la place, la lecture des tables relationnelles est assurée par un programme COBOL qui se charge de :

- Rechercher les dossiers par leur numéro (la clé primaire en table)
- Lire les enregistrements
- Faire des jointures entre tables
- Retourner ces données au client OpenHR à l'origine de la requête de lecture.

Ce programme COBOL est déclenché par le serveur OpenHR, qui est lui-même déclenché lorsqu'il reçoit une requête de lecture émise par le client OpenHR. Les données des dossiers retournées au client OpenHR sont sérialisées sous forme de caractère (buffer). La lecture et l'interprétation de ce (buffer) par le client OpenHR l'oblige à connaître la structure des données lue (laquelle est disponible grâce au dictionnaire OpenHR).

Mise à jour des dossiers

Il en va de même pour la mise à jour des dossiers : les données sont mises à jour par un programme COBOL, pas directement par l'API.

Lorsque l'on utilise un dossier, on manipule sa représentation objet qui s'apparente à un arbre constitué de nœuds représentant les informations, les occurrences (d'information) et les rubriques.

Lorsque l'on modifie le dossier, l'API mémorise ces modifications et les utilise afin de générer les requêtes de mise à jour (différentielle) de données au serveur HR Access. Cette manière de fonctionner est comparable à ce que fait Hibernate lorsque l'on manipule des beans mappés à une table relationnelle. Le fait de modifier l'une des propriétés d'un bean est mémorisé. Lors de la sauvegarde des modifications, Hibernate va générer un ordre SQL de mise à jour (UPDATE) pour persister la donnée modifiée en table. Ici, le fonctionnement est identique sauf que la mise à jour est assurée par un programme COBOL.

Données techniques et applicatives

La base HR Access contient deux types de données :

- Les données dites "techniques"
- Les données dites "applicatives"

Les données applicatives sont celles dont la structure est décrite via l'application Design Center (Objets Modèles de données) en termes de structure de données, informations, rubriques, liens et types de dossier.

Pour savoir si une donnée est technique ou applicative, il suffit de considérer le nom de la table qui la contient : si celui-ci commence par la lettre X, Y ou Z, alors la donnée est de type "applicative" ; autrement, elle est de type "technique".

Par exemple, les tables de nom ZY10, ZD00 sont des tables applicatives, tandis que les tables UC10 et MX10 sont des tables techniques. La fonctionnalité de gestion de dossiers ne permet d'accéder qu'aux données applicatives, celles pour lesquelles il est possible de compiler un processus HR Access qui effectuera les lectures de dossiers en table.

Pour lire des données techniques, il faut utiliser une autre technique appelée "extraction de données" (présentée plus loin dans la documentation).

Processus HR Access

La gestion de dossiers repose sur l'utilisation d'un processus HR Access, c'est-à-dire d'un ensemble de programmes COBOL compilés sur le serveur HR Access et chargés de différentes tâches : application de la confidentialité, lecture / mise à jour de la base de données, contrôles sur données, calculs à la volée sur les données lues, etc. Ce processus peut être personnalisé à l'aide de traitements spécifiques si bien que la logique métier d'accès aux données peut être factorisée au niveau du processus COBOL et être utilisée à la fois par l'API OpenHR et un traitement du serveur HR Access. L'API n'embarque aucune logique fonctionnelle, elle se limite à invoquer cette logique fonctionnelle implémentée sous forme de programmes COBOL.

Collection de dossiers

Les dossiers HR Access sont récupérés / créés à l'aide d'un objet central appelé la "collection de dossiers" et qui agit comme une véritable fabrique de dossiers. Elle peut être comparée à un DAO (Data Access Object) qui centralise la logique de matérialisation et de persistance des dossiers.

Les sections qui suivent sont abordées selon un ordre logique : l'ordre dans lequel les objets et leurs dépendances doivent être créés, ou de l'opération la plus simple à la plus complexe.

Mode opératoire

Le mode opératoire ci-dessous indique comment mettre en œuvre la fonctionnalité de gestion de dossiers HR Access (attendu que la session OpenHR et l'utilisateur ont déjà été connectés).

1. Créer une configuration de collection de dossiers.

2. Paramétrer la configuration de collection de dossiers afin de définir le modèle des dossiers à manipuler.

3. Instancier une collection de dossiers en fournissant (entre autres) :

- La configuration de la collection de dossiers
- Une conversation
- Un rôle

1. Charger et récupérer, à l'aide de la collection de dossiers, un dossier à l'aide de sa clé primaire (son numéro de dossier) ou d'une sélection dynamique (requête SQL de sélection).

2. Modifier le dossier (Créer / supprimer des occurrences d'informations, modifier le contenu de rubriques).

3. Soumettre les modifications au serveur HR Access.

Configuration

Ce paragraphe explique comment configurer programmatiquement le modèle de données des dossiers que l'on souhaite manipuler. Cette configuration servira de paramétrage à la collection de dossiers et lui permettra de générer les requêtes de lecture / mise à jour de dossiers.

Il faut voir la collection de dossiers comme un objet DAO (Data Access Object) dont le modèle des objets gérés n'est pas statique (codé en dur) mais dynamique et fourni par paramétrage.

La configuration d'une collection de dossiers est représentée par la classe com.hraccess.openhr.dossier.HRDossierCollectionParameters. Cette classe fournit plusieurs constructeurs dont certains sont dépréciés. Il est fortement conseillé, pour la lisibilité du code, d'utiliser le constructeur par défaut (sans argument).

Méthodes

Le tableau suivant liste les méthodes utiles de cette classe. Ci-dessous, les références au type DataSection désignent en réalité la classe HRDataSourceParameters.DataSection. La classe HRDataSourceParameters est un vestige de l'ancienne API OpenHR qui utilisait des sources de données pour manipuler les dossiers.

Méthode	Rôle
void addDataSection(DataSection)	Ajoute le paramétrage de l'information donnée à cette configuration. Cette information peut être de type "réelle", "paramètre" ou "virtuelle".
void addDataSections(Collection<DataSection>)	Ajoute le paramétrage des informations données à cette configuration. Ces informations peuvent être de type "réelle", "paramètre" ou "virtuelle".
AccessMode getAccessMode()	Retourne le mode de lecture des dossiers sous forme de AccessMode. Les différentes valeurs valides que peut retourner cette méthode sont énumérées sous forme de constantes au niveau de la classe AccessMode ("safe type enumeration"). Les modes de lecture valides sont : "différé" et "immédiat". La différence entre ces différents modes sera expliquée plus loin.
String getActivity()	Retourne le nom de l'activité au titre de laquelle les dossiers sont lus / mis à jour. Cette propriété est fournie au serveur HR Access lors des lectures / mises à jour de dossiers et peut être exploitée dans les traitements spécifiques COBOL pour écrire des conditionnements par exemple. Cette activité doit être une activité valide définie au niveau du rôle utilisé pour accéder aux dossiers.

Méthode	Rôle
String getAttribute(String)	Retourne la valeur de l'attribut de nom donné. Les attributs sont des informations transmises au serveur HR Access sous forme de paires clé / valeur lors de l'envoi de requêtes. La clé (ou nom d'attribut) identifie l'attribut et correspond à une chaîne de 8 caractères au plus. La valeur de l'attribut est une chaîne de 8 caractères au plus. L'API OpenHR supporte nativement trois attributs correspondant aux notions d'activité, de code action contextuel et de type de dossier. Les noms de ces trois attributs sont énumérés sous forme de constantes de nom ATTRIBUTE_XXX au niveau de la classe HRDataSourceParameters. Ainsi l'appel de la méthode getActivity() est équivalent à l'appel getAttribute(HRDataSourceParameters.ATTRIBUTE_ACTIVITY)
Map<String, String> getAttributes()	Retourne les attributs définis pour cette configuration sous forme d'une Map<String, String>. La clé de cette Map correspond au nom de l'attribut et la valeur, à la valeur de l'attribut.
String getContextualActionCode()	Retourne le nom du code action contextuel au titre duquel les dossiers sont lus / mis à jour. Cette propriété est fournie au serveur HR Access lors des lectures / mises à jour de dossiers et peut être exploitée dans les traitements spécifiques COBOL pour écrire des conditionnements par exemple.
List<DataSection> getDataSections()	Retourne la liste des informations paramétrées au niveau de cette configuration.
String getDataStructureName()	Retourne le nom (code) identifiant la structure de données des dossiers gérés. Une configuration permet de paramétrer une collection de dossiers, laquelle ne peut manipuler que des dossiers de la même structure de données.

Méthode	Rôle
String getDossierType()	Retourne le type des dossiers paramétrés pour cette configuration. Cette propriété est fournie au serveur HR Access lors des lectures / mises à jour de dossiers et peut être exploitée dans les traitements spécifiques COBOL pour écrire des conditionnements par exemple.
int getPacketSize()	Retourne la taille des paquets utilisée pour lire les données des dossiers depuis le serveur HR Access. Par défaut, cette taille est à 99. La taille doit être comprise dans l'intervalle [1-99].
String getProcessName()	Retourne le nom (code) identifiant le processus HR Access invoqué pour traiter les lectures / mises à jour de dossiers. La structure de données principale de ce processus (configurée via Design Center) doit correspondre à la structure de données des dossiers définie pour cette configuration (cf. méthode getDataStructureName()).

Méthode	Rôle
String getType()	Retourne le type de la collection de dossiers que l'on souhaite créer. Les différentes valeurs valides que peut retourner cette méthode sont énumérées sous forme de constantes de nom TYPE_XXX au niveau de la classe HRDossierCollectionParameters. Les types valides sont : "normal" et "saisie de masse". Le type "normal" permet de créer une collection de dossiers pour laquelle il est nécessaire de lire les données du dossier, les modifier localement puis soumettre les modifications (différentielles) au serveur HR Access. Le type "saisie de masse" permet de créer une collection de dossiers pour laquelle il n'est pas nécessaire de lire les dossiers au préalable : les mises à jour peuvent être envoyées au serveur HR Access "à l'aveugle" en fournissant l'identifiant du dossier et le "mouvement", c'est-à-dire la modification apportée au dossier (création ou modification d'occurrence, la suppression n'étant pas supportée).
UpdateMode getUpdateMode()	Retourne le mode de mise à jour des dossiers. Les différentes valeurs valides que peut retourner cette méthode sont énumérées sous forme de constantes au niveau de la classe UpdateMode (safe type enumeration). Les modes de mise à jour valides sont : "normal", "sans réponse" et "simulation". La différence entre ces différents modes est expliquée plus loin.
boolean isIgnoreSeriousWarnings()	Indique si, lors de la mise à jour des dossiers, les erreurs de poids 3 et 4 de type "avertissement avec confirmation" doivent être ignorées. Le mode de gestion des erreurs est détaillé plus loin.

Méthode	Rôle
void setAccessMode(AccessMode)	Définit le mode de lecture des dossiers à partir du AccessMode donné. Les différentes valeurs acceptées par cette méthode sont énumérées sous forme de constantes au niveau de la classe AccessMode ("safe type enumeration"). Les modes de lecture valides sont : "différé" et "immédiat". La différence entre ces différents modes est expliquée plus loin.
void setActivity(String)	Définit le nom de l'activité au titre de laquelle les dossiers sont lus / mis à jour. Cette propriété est fournie au serveur HR Access lors des lectures / mises à jour de dossiers et peut être exploitée dans les traitements spécifiques COBOL pour écrire des conditionnements, par exemple. Cette activité doit être une activité valide définie au niveau du rôle utilisé pour accéder aux dossiers.
void setAttribute(String, String)	Positionne l'attribut de nom (premier paramètre) et de valeur donnée (second paramètre). Les attributs sont des informations transmises au serveur HR Access sous forme de paires clé / valeur lors de l'envoi de requêtes. La clé (ou nom d'attribut) identifie l'attribut et correspond à une chaîne de 8 caractères au plus. La valeur de l'attribut est une chaîne de 8 caractères au plus. L'API OpenHR supporte nativement trois attributs correspondant aux notions d'activité, de code action contextuel et de type de dossier. Les noms de ces trois attributs sont énumérés sous forme de constantes de nom ATTRIBUTE_XXX au niveau de la classe HRDataSourceParameters. Ainsi, l'appel de la méthode setActivity("ABC") est équivalent à l'appel setAttribute(HRDataSourceParameters.ATTRIBUTE_ACTIVITY, ("ABC")

Méthode	Rôle
void setAttributes(Map<String, String>)	Définit les attributs définis pour cette configuration sous forme d'une Map<String, String>. La clé de cette Map correspond au nom de l'attribut et la valeur, à la valeur de l'attribut.
void setContextualActionCode(String)	Définit le nom du code action contextuel au titre duquel les dossiers sont lus / mis à jour. Cette propriété est fournie au serveur HR Access lors des lectures / mises à jour de dossiers et peut être exploitée dans les traitements spécifiques COBOL pour écrire des conditionnements, par exemple.
void setDataSections(List<DataSection>)	Définit les informations des dossiers à manipuler. Ces informations peuvent être de type "réelle", "paramètre" ou "virtuelle".
void setDataStructureName(String)	Définit le nom (code) identifiant la structure de données des dossiers gérés. Une configuration permet de paramétrer une collection de dossiers, laquelle ne peut manipuler que des dossiers de la même structure de données.
void setDossierType(String)	Définit le type des dossiers paramétrés pour cette configuration. Cette propriété est fournie au serveur HR Access lors des lectures / mises à jour de dossiers et peut être exploitée dans les traitements spécifiques COBOL pour écrire des conditionnements, par exemple.
void setIgnoreSeriousWarnings(boolean)	Définit si, lors de la mise à jour des dossiers, les erreurs de poids 3 et 4 de type "avertissement avec confirmation" doivent être ignorées. Le mode de gestion des erreurs est détaillé plus loin.
void setPacketSize(int)	Définit la taille des paquets utilisés pour lire les données des dossiers depuis le serveur HR Access. Par défaut, cette taille est à 99. La taille doit être comprise dans l'intervalle [1-99].

Méthode	Rôle
void setProcessName(String)	Définit le nom (code) identifiant le processus HR Access invoqué pour traiter les lectures / mises à jour de dossiers. La structure de données principale de ce processus (configurée via Design Center) doit correspondre à la structure de données des dossiers définie pour cette configuration (cf. méthode getDataStructureName()).
void setType(String)	Définit le type de la collection de dossiers que l'on souhaite créer. Les différentes valeurs acceptées par cette méthode sont énumérées sous forme de constantes de nom TYPE_XXX au niveau de la classe HRDossierCollectionParameters. Les types valides sont : "normal" et "saisie de masse". Le type "normal" permet de créer une collection de dossiers pour laquelle il est nécessaire de lire les données du dossier, les modifier localement puis soumettre les modifications (différentielles) au serveur HR Access. Le type "saisie de masse" permet de créer une collection de dossiers pour laquelle il n'est pas nécessaire de lire les dossiers au préalable : les mises à jour peuvent être envoyées au serveur HR Access "à l'aveugle" en fournissant l'identifiant du dossier et le "mouvement", c'est-à-dire la modification apportée au dossier (création, modification d'occurrence mais pas suppression).
void setUpdateMode(UpdateMode)	Définit le mode de mise à jour des dossiers. Les différentes valeurs acceptées par cette méthode sont énumérées sous forme de constantes au niveau de la classe UpdateMode ("safe type enumeration"). Les modes de mise à jour valides sont : "normal", "sans réponse" et "simulation". La différence entre ces différents modes est expliquée plus loin.

Illustration

L'exemple suivant illustre comment créer une configuration de collection de dossiers afin de lire plusieurs informations d'un dossier de salarié.

Pré-requis : le processus HR Access utilisé ici doit appartenir à la liste des processus paramétrés au niveau de la session OpenHR (cf. propriété session.process_list du fichier openhr.properties). Le processus doit être de type "Gestion de dossiers", être déclaré au niveau de la plate-forme physique et compilé. Les informations paramétrées doivent être rattachées explicitement au niveau du processus.

```
// Creating a new configuration (to create a dossier collection)
HRDossierCollectionParameters parameters = new
   HRDossierCollectionParameters();


// We'll handle some dossiers the “normal” way (not in the “group entry” mode)
   (mandatory)
parameters.setType(HRDossierCollectionParameters.TYPE_NORMAL);


// The process “FS001” (Personal informations) will be used to read and update
   the dossiers (mandatory)
parameters.setProcessName(“FS001”);


// Setting the data structure of the dossiers to handle to “ZY” (Employee)
   (mandatory)
parameters.setDataStructureName(“ZY”);


// The data sections “00” (ID), “10” (Birth) and “AG” (Vacations) (mandatory)
parameters.addDataSection(new HRDataSourceParameters.DataSection(“00”));
parameters.addDataSection(new HRDataSourceParameters.DataSection(“10”));
parameters.addDataSection(new HRDataSourceParameters.DataSection(“AG”));


// Dossier's data will be lazily fetched from the HR Access server (optional)
parameters.setAccessMode(AccessMode.DELAYED);


// The update mode is set to “normal”: updates will actually update the
   HR Access data base (optional)
parameters.setUpdateMode(UpdateMode.NORMAL);


// Setting the activity on behalf of which we're reading the dossiers
   (optional)
parameters.setActivity(“PAYROLL”);
```

```
// Setting the contextual action code on behalf of which we're reading the
   dossiers (optional)
parameters.setContextualActionCode(“CTX1234”);


// Ignore serious warnings when updating dossiers (optional)
parameters.setIgnoreSeriousWarnings(true);


// Changing the packet size so as to fetch data by group of 50 dossiers /
   occurrences (optional)
parameters.setPacketSize(50);
```

L'exemple ci-dessus crée une instance de HRDossierCollectionParameters de type "normal" (cf. getType()) afin de lire les informations "00" (Identification), "10" (Naissance) et "AG" (Absences de gestion) de dossiers de salariés (structure de données "ZY" (Dossiers du personnel)). Le processus utilisé pour effectuer les accès en base de données est "FS001" (Informations personnelles). La lecture des dossiers s'effectue en mode différé, c'est-à-dire à la demande, et leur mise à jour a lieu en mode "normal", c'est-à-dire que la base de données est mise à jour lors de la soumission des modifications d'un dossier au serveur HR Access. L'accès aux dossiers (lecture / mise à jour) s'effectue au titre de l'activité de nom "PAYROLL" et du code action contextuel "CTX1234". Les erreurs de poids 3 et 4 (avertissements avec demande de confirmation) sont ignorées. Enfin, les données des dossiers sont lues par paquet de 50 dossiers ou occurrences depuis le serveur HR Access.

L'exemple montre que toutes les propriétés n'ont pas besoin d'être positionnées car elles prennent une valeur par défaut qui convient dans la plupart des cas. Les données qu'il faut obligatoirement fournir sont les suivantes :

- La structure de données des dossiers accédés
- Les informations de cette structure de données auxquelles on souhaite accéder
- Le processus chargé d'accéder à la base de données

Si l'on ne conserve que ce qui est obligatoire, l'exemple ci-dessus se réduit au minimum suivant :

```
// Creating a new configuration (to create a dossier collection)
HRDossierCollectionParameters parameters = new
   HRDossierCollectionParameters();
parameters.setType(HRDossierCollectionParameters.TYPE_NORMAL);
parameters.setProcessName(“FS001”);
parameters.setDataStructureName(“ZY”);
parameters.addDataSection(new HRDataSourceParameters.DataSection(“00”));
parameters.addDataSection(new HRDataSourceParameters.DataSection(“10”));
parameters.addDataSection(new HRDataSourceParameters.DataSection(“AG”));
```

Paramétrage fin

L'exemple précédent permet de manipuler les informations Identification (ZY00), Naissance (ZY10) et Absences de gestion (ZYAG) de dossiers de salariés. L'API permet également de paramétrer finement la manière dont chaque information est lue.

La classe HRDataSourceParameters.DataSection représente le paramétrage utilisé pour lire une information. Cette classe déclare les méthodes listées dans le tableau ci-dessous. Pour plus de clarté, toutes les méthodes relatives à la gestion des BLOBs ne sont pas listées ici, elles seront présentées en temps utile.

Méthode	Rôle
void addExternal(ExternalDesc)	Ajoute la "rubrique externe" de description donnée au paramétrage d'information. Voir plus loin ce qu'est une "rubrique externe".
void addExternals(Collection<ExternalDesc>)	Ajoute les "rubriques externes" de descriptions données au paramétrage d'information. Voir plus loin ce qu'est une "rubrique externe".
void addItemName(String)	Ajoute le nom de rubrique donné à la liste des rubriques de l'information qui doivent être lues. Cette méthode permet de limiter la lecture d'une information à certaines rubriques. Le nom de cette rubrique doit correspondre à une rubrique définie au niveau de l'information.
void addItemNames(Collection<String>)	Ajoute les noms de rubriques donnés à la liste des rubriques de l'information qui doivent être lues. Cette méthode permet de limiter la lecture d'une information à certaines rubriques. Les noms de ces rubriques doivent correspondre à des rubriques définies au niveau de l'information.
String getDataSectionName()	Retourne le nom (code) identifiant l'information paramétrée.
List<ExternalDesc> getExternals()	Retourne sous forme d'une liste<ExternalDesc> la description des "rubriques externes" paramétrées pour cette information.
List<String> getItemNames()	Retourne la liste des noms de rubriques auxquelles la lecture d'information a été restreinte.

Méthode	Rôle
int getPacketSize()	Retourne le nombre d'occurrences chargées lors des lectures depuis le serveur HR Access. Par défaut, cette taille est de 99 (occurrences). La valeur est située dans l'intervalle [1-99].
String getProcedureCode()	Retourne le nom (code) du traitement spécifique COBOL qui est configuré pour personnaliser la lecture / mise à jour de l'information. Voir setProcedure(String, String).
String getProcedureGroup()	Retourne le nom (code) du groupe de traitement spécifique COBOL qui est configuré pour personnaliser la lecture / mise à jour de l'information. Voir setProcedure(String, String).
String getSqlFilter()	Retourne le filtre SQL utilisé afin de filtrer la lecture des occurrences de l'information. Cette propriété permet de restreindre la lecture de l'information à certaines occurrences répondant à la condition exprimée par le filtre. Le filtre SQL est une chaîne de caractères utilisée comme condition dans la clause WHERE de l'ordre SQL de sélection des occurrences de l'information d'un dossier. Exemple : « TYPADD="1" », « MOTIFA="RTT" ».
boolean isForceModification()	Indique si une occurrence d'information doit être considérée comme modifiée quelles que soient les valeurs initiale et nouvelle d'une rubrique. Cette propriété permet de forcer la modification d'une rubrique même si celle-ci prend pour nouvelle valeur sa valeur actuelle.

Méthode	Rôle
boolean isReverseOrder()	Indique si les occurrences de l'information doivent être lues en ordre inverse (de l'ordre naturel défini par les arguments de tri de l'information). Cette propriété permet de récupérer les dernières occurrences d'une information en premier. Cela peut se révéler utile dans le cas de consultation d'une information de type "historique".
void setDataSectionName(String)	Définit le nom (code) identifiant l'information paramétrée. Cette information peut être de type "réelle", "paramètre" ou "virtuelle".
void setExternals(Collection<ExternalDesc>)	Définit les "rubriques externes" de descriptions données du paramétrage d'information. Voir plus bas ce qu'est une "rubrique externe".
void setForceModification(boolean)	Indique si une occurrence d'information doit être considérée comme modifiée quelles que soient les valeurs initiale et nouvelle d'une rubrique. Cette propriété permet de forcer la modification d'une rubrique même si celle-ci prend pour nouvelle valeur sa valeur actuelle.
void setItemNames(Collection<String>)	Définit la liste des noms de rubriques auxquelles la lecture d'information a été restreinte.
void setPacketSize(int)	Définit le nombre d'occurrences chargées lors des lectures depuis le serveur HR Access. Par défaut, cette taille est de 99 (occurrences). La valeur doit être située dans l'intervalle [1-99].
void setProcedure(String, String)	Définit le groupe (premier paramètre) et le nom (second paramètre) du traitement spécifique COBOL qui est configuré pour personnaliser la lecture / mise à jour de l'information.
void setProcedureCode(String)	Définit le nom (code) du traitement spécifique COBOL qui est configuré pour personnaliser la lecture / mise à jour de l'information.

Méthode	Rôle
void setProcedureGroup(String)	Définit le groupe du traitement spécifique COBOL qui est configuré pour personnaliser la lecture / mise à jour de l'information.
void setReverseOrder(boolean)	Définit si les occurrences de l'information doivent être lues en ordre inverse (de l'ordre naturel défini par les arguments de tri de l'information). Cette propriété permet de récupérer les dernières occurrences d'une information en premier. Cela peut se révéler utile dans le cas de consultation d'une information de type "historique".
void setSqlFilter(String)	Définit le filtre SQL utilisé afin de filtrer la lecture des occurrences de l'information. Cette propriété permet de restreindre la lecture de l'information à certaines occurrences répondant à la condition exprimée par le filtre. Le filtre SQL est une chaîne de caractères utilisée comme condition dans la clause WHERE de l'ordre SQL de sélection des occurrences de l'information d'un dossier. Exemple : « TYPADD="1" », « MOTIFA="RTT" ».

```
// Creating a new configuration (to create a dossier collection)
HRDossierCollectionParameters parameters = new
   HRDossierCollectionParameters();
parameters.setType(HRDossierCollectionParameters.TYPE_NORMAL);
parameters.setProcessName(“FS001”);
parameters.setDataStructureName(“ZY”);
parameters.addDataSection(new HRDataSourceParameters.DataSection(“00”));
parameters.addDataSection(new HRDataSourceParameters.DataSection(“10”));
```

```
// Creating configuration for data section ZYAG
HRDataSourceParameters.DataSection dataSectionZYAG =
   HRDataSourceParameters.DataSection(“AG”));
```

```
// Adding an external to retrieve the long label of rule system code
   referenced by item MOTIFA (optional)
```

```
dataSectionZYAG.addExternal(new ExternalDesc(“MOTIFA_L”, “MOTIFA”,
    “01”, “LIBLON”, null, null);

// Restricting items to a given subset (optional)
dataSectionZYAG.addItemName(“DATDEB”); // start date
dataSectionZYAG.addItemName(“DATFIN”); // end date
dataSectionZYAG.addItemName(“MOTIFA”); // absence code

// Every update upon items has to be processed has an effective update
    (optional)
dataSectionZYAG.setForceModification(true);

// Using a custom procedure to customize the way data section ZYAG is read
    and updated (optional)
dataSectionZYAG.setProcedure(“AS”, “123456”);

// Reading occurrences in reverse order: last occurrences are read first
    (optional)
dataSectionZYAG.setReverseOrder(true);

// Reads upon the HR Access database will return up to 50 occurrences of
    this data section (optional)
dataSectionZYAG.setPacketSize(50);

// Finally keep occurrences whose absence code has a given value (optional)
dataSectionZYAG.setSqlFilter(“MOTIFA=<QB>RTT<QE>”);

// Register the data section for reading
parameters.addDataSection(dataSectionZYAG);
```

L'exemple ci-dessus permet de configurer une collection de dossiers de salariés de façon à :

- Récupérer sous forme d'une rubrique externe de nom "MOTIFA_L" le libellé long du code réglementaire référencé par la rubrique "MOTIFA" (Motif d'absence). La langue de ce libellé correspond à la langue de l'utilisateur qui va effectuer la lecture des dossiers. Le libellé est récupéré à partir de la rubrique "LIBLON" (Libellé long) de l'information "01" (Libellés) du dossier réglementaire désigné par la rubrique MOTIFA. La structure de données "ZY" (Salariés) étant associée à la structure de données réglementaire "ZD", le libellé long est donc issu de la rubrique ZD01 LIBLON.
- Lire uniquement les valeurs des rubriques DATDEB (date de début), DATFIN (date de fin) et MOTIFA (motif d'absence).

- Gérer toute modification de valeur de rubrique comme une modification réelle (même si la nouvelle valeur de la rubrique est identique à la valeur actuelle).
- Invoquer le traitement spécifique de nom "123456" du groupe "AS" afin de personnaliser la lecture / mise à jour de l'information.
- Lire les occurrences dans l'ordre inverse, de la dernière à la première.
- Ne conserver que les occurrences de ZYAG dont la rubrique "MOTIFA" a pour valeur "RTT".

Avec un tel paramétrage, l'information aura 4 rubriques lisibles correspondant aux 3 rubriques de restriction (DATDEB, DATFIN et MOTIFA) et à la rubrique externe configurée (MOTIFA_L). Seules les rubriques correspondant à une donnée en table ZYAG sont modifiables (DATDEB, DATFIN et MOTIFA).

Rubrique externe

Une rubrique externe permet de récupérer une donnée située dans une table autre (externe) que la table de l'information que l'on désire lire. Pour ce faire, la rubrique externe est "calculée" à l'aide d'une jointure entre tables relationnelles. La jointure en question peut résulter d'un contrôle de correspondance au niveau d'une rubrique, auquel cas la rubrique externe sera une rubrique du dossier réglementaire désigné par la rubrique d'origine. Mais la jointure peut aussi, de manière plus générale, résulter d'un lien défini au niveau de la structure de données.

Le mécanisme de rubrique externe est pratique pour récupérer le libellé de codes réglementaires.

```
...


// Creating configuration for data section ZYAG

HRDataSourceParameters.DataSection dataSectionZYAG =
   HRDataSourceParameters.DataSection(“AG”));


// Adding an external to retrieve the long label of rule system code referenced
   by item MOTIFA (optional)

dataSectionZYAG.addExternal(new ExternalDesc(“MOTIFA_L”, “MOTIFA”, “01”,
   “LIBLON”, null, null);


...
```

Prenons l'exemple de l'information ZYAG (Absences d'un salarié). Celle-ci définit une rubrique MOTIFA (Motif d'absence) en correspondance avec un dossier réglementaire du répertoire ZD DSJ (Motifs d'absences et de présences). La rubrique ZYAG MOTIFA ne peut prendre pour valeur que l'une des valeurs de rubrique ZD00 CDCODE des dossiers du répertoire ZD DSJ. La rubrique "pointe" donc vers un dossier de ZD DSJ. Lorsqu'on lit l'information ZYAG en lui configurant une rubrique externe comme dans l'exemple ci-dessus, on indique au serveur HR Access qu'il doit rechercher dans l'information ZD01 (Libellés) du dossier réglementaire (pointé par la rubrique MOTIFA) la valeur de la rubrique LIBLON (Libellé long) et rajouter cette valeur sous le nom "MOTIFA_L".

Il est aussi possible d'utiliser la définition d'un lien pour créer une rubrique externe.

Filtre SQL

Le filtre SQL est un paramétrage qui permet de restreindre les occurrences d'informations à lire. Chaque information peut être configurée avec son propre filtre SQL. Il s'agit d'une chaîne de caractères exprimant une condition SQL qui sera utilisée dans la clause WHERE de l'ordre SELECT de récupération des occurrences en table. Ainsi, un filtre SQL alimenté à « MOTIFA="1" » et appliqué à l'information ZYAG (Absences d'un salarié) génèrera (schématiquement) un ordre SQL du type : « SELECT * FROM ZYAG WHERE MOTIFA="1" ». Il est possible d'utiliser, dans le filtre SQL, toutes les rubriques de l'information correspondant à des colonnes de table. Cela n'inclut pas les rubriques de type redéfinition, groupe, virtuelle. Idem pour les mots-clés SQL de la base de données sous-jacente.

Portabilité du filtre SQL

Le filtre étant écrit dans une syntaxe SQL, il se pose le problème de sa portabilité. En effet, HR Access peut tourner sur plusieurs moteurs de base de données dont Oracle, SQL Server, DB2. Le filtre peut ainsi fonctionner sur une base de données mais pas sur toutes. Afin d'améliorer la portabilité du SQL utilisé dans les filtres, HR Access supporte un système de mot-clé propriétaire qui permet de référencer des fonctions ou des propriétés dépendantes de la base de données sous-jacente.

Le tableau suivant liste ces mots-clés.

Mot-clé	Rôle
<QB>	Littéralement "Quote Begin". Mot-clé utilisé afin de représenter un séparateur portable marquant le début d'un littéral. Sera résolu en double ou simple quote selon la base de données cible.
<QE>	Littéralement "Quote End". Mot-clé utilisé afin de représenter un séparateur portable marquant la fin d'un littéral. Sera résolu en double ou simple quote selon la base de données cible.
<DAYDATE>	Mot-clé utilisé pour représenter la date système. Sera résolu en SYSDATE (Oracle), CURRENT DATE (DB2), etc.
<DAYTIME>	Mot-clé utilisé pour représenter l'horodatage système. Sera résolu en SYSDATE (Oracle), CURRENT TIME (DB2), etc.
<MINDATE>	Mot-clé utilisé pour représenter la date spéciale indiquant une date (de début) non renseignée (01/01/0001). Attention ! SQL Server ne pouvant gérer de date inférieure au 01/01/0001, cette date spéciale sera transformée en 01/01/1753. Ce n'est pas le cas avec les autres bases de données.
<MINTIME>	Mot-clé utilisé pour représenter l'horodatage spécial indiquant un horodatage (de début) non renseigné (01/01/0001 00:00:00). Attention ! SQL Server ne pouvant gérer de date inférieure au 01/01/0001, cette date spéciale sera transformée en 01/01/1753 00:00:00. Ce n'est pas le cas avec les autres bases de données.
<MAXDATE>	Mot-clé utilisé pour représenter la date spéciale indiquant une date (de fin) non renseignée (31/12/2999).
<MINTIME>	Mot-clé utilisé pour représenter l'horodatage spécial indiquant un horodatage (de fin) non renseigné (31/12/2999 23:59:59).

Mot-clé	Rôle
<USERLANG>	Mot-clé utilisé pour représenter le code langue de l'utilisateur effectuant la lecture des dossiers. Sera remplacé par "F" pour un utilisateur français, "U" pour un utilisateur anglais, etc.

Exemples de filtres SQL utilisant des mots-clés.

```
MOTIFA=<QB>RTT<QE>
```

Appliqué à l'information ZYAG (Absences de gestion), il permet de ne lire que les absences d'un salarié correspondant à des congés RTT.

```
(DATDEB <= <DAYDATE>) AND (<DAYDATE> <= DATFIN)
```

Appliqué à l'information ZYAG (Absences de gestion), il permet de ne lire que les absences en cours (en vigueur) d'un salarié.

Modes d'accès

Le mode d'accès (sous-entendu en lecture) permet de configurer la manière dont les données des dossiers sont récupérées depuis le serveur HR Access.

Un dossier contient des informations et ces informations sont constituées d'occurrences. Lorsqu'OpenHR lit un dossier, afin de pouvoir construire sa représentation object, il faut théoriquement lire toutes les occurrences de toutes les informations de ce dossier. En raison de limitations introduites par les programmes COBOL, il n'est pas possible de lire, pour un même dossier, plus de 99 occurrences d'une information à la fois. Par conséquent, un dossier dont l'une des informations contient 150 occurrences demande donc deux requêtes au serveur pour charger celles-ci (attendu que les données sont chargées avec une taille de paquet de 99) : la première pour charger les occurrences de 1 à 99, la seconde pour les occurrences de rang 100 à 150. Parfois, la récupération de toutes ces données peut ne pas être utile car seules les 10 premières occurrences sont utiles. Le chargement de la totalité des données d'un dossier est donc coûteux en performance.

OpenHR permet de définir comment les données des dossiers (les occurrences) doivent être chargées. Il existe deux manières de traiter le problème :

- Le chargement préemptif : lorsqu'un dossier est chargé, toutes les occurrences de toutes ses informations sont chargées à l'initialisation. Cela entraîne donc potentiellement plusieurs requêtes au serveur HR Access afin de charger les occurrences par paquets de 99 (taille des paquets en standard).
- Le chargement différé : le dossier est initialisé avec le minimum de données en mémoire et les données manquantes sont récupérées dynamiquement par émission d'une requête au serveur lorsque cela est nécessaire.

En fonction du contexte dans lequel survient le chargement du dossier et du traitement appliqué aux données, l'une ou l'autre stratégie se révèle plus adaptée. Si toutes les données du dossier sont nécessaires pour effectuer un traitement, alors les données doivent être chargées de manière préemptive afin de minimiser les échanges réseau. Si au contraire, seule une partie des occurrences est nécessaire pour effectuer un traitement, alors le chargement différé peut se révéler plus avantageux.

Chacune de ces stratégies de lecture de données est représentée par une instance de la classe AccessMode qui est une énumération de type "sûre" ("safe type enumeration"). La constante "DELAYED" représente le mode de chargement différé alors que la constante "ONE_SHOT" représente le mode de chargement préemptif.

Par défaut, c'est le mode préemptif qui est positionné car c'est celui qui se révèle le plus souvent optimal.

Modes de mise à jour

Il est possible de configurer la manière dont la mise à jour des données s'effectue sur le serveur HR Access. C'est la classe UpdateMode, une énumération de type "sûre" ("safe type enumeration") qui représente le concept de mode de mise à jour des données.

Cette classe définit, sous forme de constantes, 5 modes de mises à jour. Cependant, dans le cas qui nous intéresse, seuls 3 sont pertinents : les modes représentés par les constantes "NORMAL", "NO_REPLY" et "SIMULATION".

Le mode de mise à jour des données par défaut est positionné à "NORMAL".

Mode de mise à jour	Signification
UpdateMode.NORMAL	La soumission des modifications d'un dossier au serveur HR Access modifie les données en table et retourne les éventuelles erreurs levées lors de l'opération à l'API OpenHR afin de lui permettre de synchroniser la représentation object du dossier avec sa version en base de données. Ce mode de fonctionnement est le mode de base, d'où son nom.
UpdateMode.NO_REPLY	La soumission des modifications d'un dossier au serveur HR Access modifie les données en table mais ne retourne pas à l'API OpenHR les éventuelles erreurs levées lors de l'opération. Cela ne lui permet pas de synchroniser la représentation object du dossier avec sa version en base de données. Ce mode de mise à jour est optimal lorsqu'il s'agit d'effectuer des modifications unitaires sur des dossiers et que ces modifications ne peuvent échouer. La contrepartie de ce mode dégradé du mode normal est qu'il n'est pas possible de soumettre deux fois de suite des modifications sur un même dossier : la version objet du dossier n'ayant pas été resynchronisée avec la version serveur après la première soumission, la seconde mise à jour sera rejetée en raison d'un conflit (virtuel) de mise à jour concurrente (gestion des mises à jour concurrentes basées sur l'utilisation de timestamps (optimistic locking)).
UpdateMode.SIMULATION	Ce mode de mise à jour est similaire au mode "NORMAL" sauf que la transaction vers la base de données n'est pas validée. Elle est annulée juste avant la mise à jour des données. Ce mode spécial permet de simuler une mise à jour de dossiers sans modifier la base de données.

L'API OpenHR permet de modifier dynamiquement le mode de mise à jour des dossiers, i.e. de basculer du mode "NORMAL" au mode "SIMULATION" et vice versa. Cela sera expliqué plus loin.

La gestion des avertissements avec confirmation

Le serveur HR Access utilise un système d'erreurs dotées d'un poids allant de 1 à 5 pour signaler les erreurs rencontrées lors des mises à jour de dossiers. Le tableau suivant indique le sens de chaque poids d'erreur.

Poids d'erreur	Type d'erreur	Confirmation
1	Anomalie (non mémorisée)	Sans
2	Anomalie (mémorisée)	
3	Anomalie (non mémorisée)	Avec
4	Anomalie (mémorisée)	
5	Erreur	Sans objet

Plus le poids d'erreur est élevé, plus l'erreur est grave.

Les erreurs de poids 5 désignent des erreurs bloquantes qui empêchent la mise à jour d'un dossier. Par exemple, une erreur bloquante peut survenir si toutes les rubriques obligatoires d'une information n'ont pas été renseignées ou si une rubrique en correspondance avec un répertoire réglementaire se voit assigner une valeur non valide.

Les erreurs de poids 1 à 4 représentent les anomalies détectées lors d'une mise à jour. Si le poids est égal à 1 ou 2, alors l'anomalie est bénigne et la mise à jour du dossier ne sera pas rejetée ; l'anomalie fait office d'avertissement. Si le poids de l'anomalie est égal à 3 ou 4, alors elle est considérée comme suffisamment grave pour demander à l'utilisateur à l'origine de la mise à jour si celle-ci doit être effectuée ou non (d'où le nom d'anomalie avec confirmation).

L'API OpenHR permet de définir la manière dont les anomalies avec confirmation (erreurs de poids 3 et 4) doivent être gérées. Par défaut, celles-ci bloqueront la mise à jour du dossier et l'utilisateur devra confirmer qu'il souhaite que celle-ci ait lieu malgré les anomalies. Cependant, parfois, il n'y a pas d'utilisateur pour confirmer la mise à jour, par exemple dans le cas d'un traitement batch. Dans ce cas-là, il est possible d'ignorer les anomalies avec confirmation : elles seront alors interprétées comme des anomalies sans confirmation.

Pour définir la manière dont les anomalies avec confirmation doivent être gérées, il faut positionner la propriété ignoreSeriousWarnings de la classe HRDossierCollectionParameters.

```
// Creating a new configuration (to create a dossier collection)
HRDossierCollectionParameters parameters = new
   HRDossierCollectionParameters();
parameters.setType(HRDossierCollectionParameters.TYPE_NORMAL);
parameters.setProcessName(“FS001”);
parameters.setDataStructureName(“ZY”);
parameters.addDataSection(new HRDataSourceParameters.DataSection(“00”));
...

// Ignoring the errors with a weight of 3 or 4
parameters.setIgnoreSeriousWarnings(true);
```

Création d'une fabrique de dossiers

La collection de dossiers - que l'on présentera plus loin - sert à matérialiser les dossiers HR Access. Cependant, ce n'est pas elle qui crée (instancie) concrètement les dossiers. Cette tâche est assurée par un objet implémentant l'interface com.hraccess.openhr.dossier.IHRDossierFactory appelé "fabrique de dossiers".

L'interface IHRDossierFactory déclare deux méthodes.

Méthode	Rôle
HRDossier createDossier(AbstractDossierCollection)	Crée et retourne une nouvelle instance de HRDossier (représentant un dossier vide) secondé par la collection de dossiers donnée.
HRDossier createDossier(AbstractDossierCollection, HRDossierId)	Crée et retourne une nouvelle instance de HRDossier (représentant un dossier vide d'identifiant donné) secondé par la collection de dossiers donnée.

L'API OpenHR livre 3 implémentations de l'interface IHRDossierFactory. Cependant, pour l'instant, seule l'implémentation qui nous intéresse sera présentée (les autres le seront ultérieurement).

La classe com.hraccess.openhr.dossier.HRDossierFactory est l'implémentation générique de l'interface IHRDossierFactory. Elle retourne des instances de com.hraccess.openhr.HRDossier qui permettent de manipuler n'importe quelle structure de données. C'est cette classe que l'on utilise la plupart du temps.

Pour instancier la classe HRDossierFactory, il faut indiquer le type des dossiers que l'on souhaite créer à l'aide d'une des constantes HRDossierFactory.TYPE_DOSSIER ou HRDossierFactory.TYPE_DOSSIER_CODE. La première constante représente un dossier de n'importe quel type de structure de données, alors que la seconde désigne des dossiers de type réglementaire.

```
...


// Creating a new dossier factory to create instances of HRDossier
IHRDossierFactory dossierFactory = new
   HRDossierFactory(HRDossierFactory.TYPE_DOSSIER);


...
```

Si la HRDossierFactory est créée avec le type "DOSSIER", alors la fabrique retourne des instances de HRDossier. Si la HRDossierFactory est créée avec le type "DOSSIER_CODE", alors la fabrique retourne des instances de HRDossierCode. Cette classe est une sous-classe de HRDossier qui rajoute des méthodes de type getter spécifiques à un dossier réglementaire pour lire le code du dossier (cf. getCode()), son répertoire réglementaire (cf. getDirectory()), etc.

En réalité, il est possible de manipuler des dossiers réglementaires à l'aide d'une fabrique de dossiers dont le type est simplement "DOSSIER". Dans ce cas, les instances retournées sont des HRDossier (pas des HRDossierCodes) et la récupération du code d'un dossier requiert d'écrire le code suivant :

```
// Retrieving a dossier representing a rule system dossier (not detailed here)
```

```
HRDossier dossier = getRuleSystemDossier();


// Reading the dossier's code
String code =
   dossier.getDataSectionByName("00").getOccur().getString("CDCODE")
```

à la place de :

```
// Retrieving a dossier representing a rule system dossier (not detailed here)
HRDossierCode dossier = getRuleSystemDossierCode();


// Reading the dossier's code
String code = dossier.getCode();
```

La classe HRDossierCode n'est donc pas indispensable pour manipuler un dossier réglementaire mais fournit quelques méthodes pratiques pour réduire (un peu) la verbosité du code.

Création d'une collection de dossiers

La collection de dossiers est l'objet par lequel les dossiers sont créés et chargés (matérialisés). C'est pourquoi elle peut être comparée à un objet DAO (Data Access Object).

C'est la classe com.hraccess.openhr.dossier.HRDossierCollection qui représente le concept de collection de dossiers.

Pour instancier une collection de dossiers, il faut fournir au constructeur de la classe :

- La configuration de la collection de dossiers (HRDossierCollectionParameters) pour lui fournir le modèle de données qui sera manipulé.
- Une conversation d'utilisateur (IHRConversation) afin de définir un contexte d'informations paramètres (qu'il y ait des informations de type paramètre configurées ou pas).
- Un rôle (IHRRole) afin de définir la confidentialité qui s'appliquera à la lecture / mise à jour des dossiers.
- Une fabrique de dossiers (IHRDossierFactory) afin de créer les dossiers.

```
// Retrieving the configuration to create a new dossier collection (not
   detailed here)

HRDossierCollectionParameters parameters =
   getDossierCollectionConfiguration();


// Retrieving a connected user (not detailed here)

IHRUser user = getConnectedUser();


// Retrieving a conversation and a role from this user

IHRConversation conversation = user.getMainConversation();

IHRRole role = user.getRole("EMPLOYEE(123456)");


// Creating a factory to create dossiers

IHRDossierFactory factory = new
   HRDossierFactory(HRDossierFactory.TYPE_DOSSIER);


// Creating a new dossier collection with these parameters

HRDossierCollection dossierCollection = new
   HRDossierCollection(parameters, conversation, role, factory);
```

Après l'instanciation, la collection de dossiers est prête à l'emploi.

Identification des dossiers

Les dossiers HR Access sont identifiés à l'aide de deux clés :

- Une clé technique appelée "numéro de dossier" correspondant à la clé primaire du dossier en base de données (et stockée dans la colonne de table de nom NUDOSS). Cette clé est numérique et est attribuée par le serveur HR Access lors de la création du dossier. Pour information, il est possible de personnaliser la stratégie de génération de cette clé à l'aide d'un traitement spécifique COBOL. Cette clé primaire permet de récupérer l'enregistrement représentant un dossier à partir de la table xx00 où xx désigne le nom de la structure de données du dossier. Exemple de numéro de dossier : 1000.
- Une clé fonctionnelle correspondant au n-uplet des valeurs des rubriques clés (ou arguments de tri) de l'information "00" (Identification) utilisée pour identifier un dossier. Cette clé n'est pas générée par le serveur HR Access, c'est le client OpenHR qui décide de la valeur des rubriques composant la clé. Ces rubriques clés servent à définir une contrainte d'unicité au niveau de la base de données. Exemple de clé fonctionnelle : un dossier de salarié (de structure de données ZY) est identifié fonctionnellement par le doublet de rubriques (ZY00 SOCCLE (Réglementation), ZY00 MATCLE (Matricule)). Exemple d'identifiant fonctionnel de dossier de salarié: ("HRA", "123456").

La classe com.hraccess.openhr.HRDossierId représente un identifiant de dossier avec ses clés technique et fonctionnelle. La clé technique est représentée sous forme d'un entier (int) alors que la clé fonctionnelle est représentée par l'interface com.hraccess.openhr.dossier.IHRKey (implémentée par la classe com.hraccess.openhr.dossier.HRKey).

L'exemple ci-dessous illustre comment créer un HRDossierId identifiant un dossier.

```
// Creating a new ID to denote a given HR Access dossier
HRDossierId id = new HRDossierId();


// Setting the number (technical key) mapped to this dossier
id.setNudoss(1000);


// Setting the functional key mapped to this dossier
id.setDossierKey(new HRKey(“HRA”, “123456”));
```

La création d'un HRDossierId ne requiert pas de définir les deux clés, seule l'une de ces deux clés suffit pour pouvoir manipuler le dossier. L'exemple ci-dessus est donc théorique.

Chargement d'un dossier

Pour charger un dossier, il faut appeler l'une des méthodes loadDossier() de la classe HRDossierCollection en lui passant une clé identifiant le dossier : soit la clé technique (le numéro de dossier) soit la clé fonctionnelle.

L'exemple suivant illustre ces deux cas de figure :

```
// Retrieving an existing dossier collection (not detailed here)
HRDossierCollection dossierCollection = getDossierCollection();


// Loading a dossier from its dossier number (technical key)
HRDossier dossier = dossierCollection.loadDossier(1000);


// Loading a dossier from its functional key
HRDossier dossier2 = dossierCollection.loadDossier(new HRKey("HRA",
    "123456"));
```

Si la clé est invalide, c'est-à-dire qu'elle désigne un dossier inexistant, alors ces deux méthodes retournent null.

Il faut savoir que le chargement d'un dossier n'est pas un processus incrémental : à partir d'une collection de dossiers nouvellement créée (donc vide), le chargement d'un dossier va charger les données (du dossier) au niveau de la collection de dossiers.

Une seconde demande de chargement de dossier ne va pas induire le chargement d'un dossier supplémentaire au niveau de la collection mais le remplacement du dossier existant ! Il faut bien garder cela à l'esprit autrement cela peut être source d'incompréhension sur la manière dont fonctionne une collection de dossiers.

Chargement d'un ou plusieurs dossiers

La classe HRDossierCollection permet de charger plusieurs dossiers à la fois à l'aide des méthodes loadDossiers() (notez le pluriel). Nous allons passer en revue 5 de ces méthodes, il en reste d'autres qui sont dépréciées ou dont l'utilité est discutable. Les méthodes qui sont présentées ici permettent néanmoins de couvrir l'essentiel des besoins.

Chacune de ces méthodes est présentée dans le tableau ci-dessous.

Méthodes	**Rôle**
HRDossierListIterator loadDossiers(IHRKey[])	Charge et retourne sous forme d'un itérateur de dossiers (HRDossierListIterator) les dossiers de clés fonctionnelles données.
HRDossierListIterator loadDossiers(int[])	Charge et retourne sous forme d'un itérateur de dossiers (HRDossierListIterator) les dossiers de clés techniques (numéros de dossier) données.
HRDossierListIterator loadDossiers(String)	Charge et retourne sous forme d'un itérateur de dossiers (HRDossierListIterator) les dossiers sélectionnés par la requête SQL de sélection donné. Cette requête SQL doit être un ordre de sélection de numéros de dossier de la forme "SELECT A.NUDOSS FROM ZY00 A". Il est capital que la requête retourne au moins le NUDOSS des dossiers en première colonne et que les tables soient associées à des alias (A ici). Le fait que d'autres colonnes soient retournées par la requête n'a pas d'incidence sur la sélection des dossiers. La confidentialité du rôle utilisé pour sélectionner les dossiers doit être compatible avec les tables référencées par la requête SQL. Autrement, la sélection ne retournera aucun dossier. Enfin, la requête SQL peut utiliser les mots-clés spécifiques à HR Access pour améliorer sa portabilité (cf. "Le filtre SQL").
HRDossierListIterator loadDossiers(String, int)	Identique à la méthode loadDossiers(String) sauf que le chargement des dossiers retournera au plus le nombre de dossiers indiqué en second paramètre.

Méthodes	Rôle
HRDossierListIterator loadDossiers(String, int, int)	Identique à la méthode loadDossiers(String) sauf que le chargement des dossiers retournera au plus le nombre de dossiers indiqué (troisième paramètre) à partir de l'offset donné (second paramètre). Le second paramètre désigne l'offset à partir duquel les dossiers sélectionnés par la requête SQL doivent être pris en compte. En faisant varier l'offset, il est possible de parcourir les différentes "pages" d'une liste de dossiers.

Le chargement des dossiers peut s'effectuer de manière statique en listant explicitement les clés (techniques ou fonctionnelles) des dossiers à charger, ou de manière dynamique en utilisant une requête SQL de sélection.

```
// Retrieving an existing dossier collection (not detailed here)
HRDossierCollection dossierCollection = getDossierCollection();


// Loading some dossiers from their dossier numbers (technical keys) (static selection)
HRDossierListIterator iterator = dossierCollection.loadDossiers(new int[] { 1000, 2000 });


// Loading some dossiers from their functional keys (static selection)
HRDossierListIterator iterator2 = dossierCollection.loadDossiers(new IHRKey[] {
    new HRKey("HRA", "123456"),
    new HRKey("HRA", "234567")
});


// Loading some dossiers from a SQL statement (dynamic selection of employees sorted by name)
HRDossierListIterator iterator3 = dossierCollection.loadDossiers(
    "SELECT A.NUDOSS, A.NOMUSE FROM ZY00 A ORDER BY A.NOMUSE");


// Loading up to 10 dossiers from a SQL statement (dynamic selection of (up to) 10 employees sorted by name
// in range [0 .. 9])
HRDossierListIterator iterator4 = dossierCollection.loadDossiers(
    "SELECT A.NUDOSS, A.NOMUSE FROM ZY00 A ORDER BY A.NOMUSE", 10);
```

```
// Loading up to 10 dossiers from a SQL statement (dynamic selection of (up
    to) 10 employees sorted by name
// in range [5 .. 14])
HRDossierListIterator iterator4 = dossierCollection.loadDossiers(
    "SELECT A.NUDOSS, A.NOMUSE FROM ZY00 A ORDER BY A.NOMUSE", 5,
    10);
```

La classe HRDossierListIterator (du package com.hraccess.openhr.dossier) représente une implémentation de l'interface java.util.ListIterator pour parcourir une List<HRDossier>. Les différentes méthodes loadDossiers() retournent une instance de HRDossierListIterator qui permet d'itérer sur les dossiers chargés. Cependant cette classe n'apporte pas beaucoup de valeur ajoutée par rapport à l'interface java.util.ListIterator (si ce n'est déclarer des méthodes typées pour éviter de caster explicitement les objets retournés par les méthodes de l'interface ListIterator en HRDossier). C'est pourquoi il est possible qu'un jour cette classe soit dépréciée et remplacée par l'interface java.util.ListIterator.

Récupération des dossiers chargés

La classe HRDossierCollection définit plusieurs méthodes pour récupérer les dossiers chargés.

Les principales méthodes sont données dans le tableau ci-dessous.

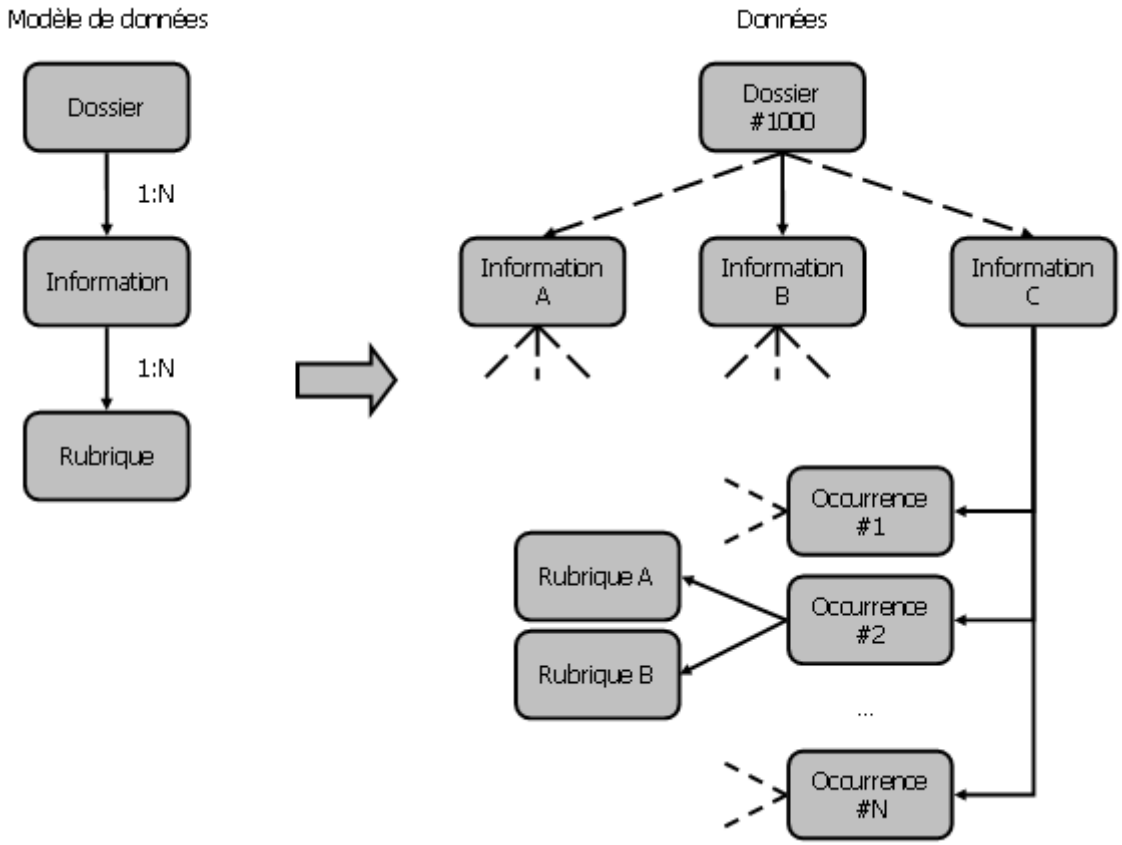
Méthode	Rôle
HRDossier getDossier(IHRKey)	Retourne le dossier de clé fonctionnelle donnée. Si le dossier n'a pas été chargé dans la collection de dossiers, retourne null. Cette méthode n'entraîne pas le chargement du dossier absent, contrairement à la méthode loadDossier(IHRKey).
HRDossier getDossier(int)	Retourne le dossier de clé technique (numéro de dossier) donnée. Si le dossier n'a pas été chargé dans la collection de dossiers, retourne null. Cette méthode n'entraîne pas le chargement du dossier absent contrairement à la méthode loadDossier(int).
HRDossierListIterator getDossiers()	Retourne les dossiers chargés au niveau de la collection de dossiers sous forme d'un HRDossierListIterator.
HRDossierListIterator getDossiers(IHRKey[])	Retourne les dossiers de clés fonctionnelles données. Cette méthode n'entraîne pas le chargement des dossiers absents, contrairement à la méthode loadDossiers(IHRKey[]).
HRDossierListIterator getDossiers(int[])	Retourne les dossiers de clés techniques (numéro de dossier) données. Cette méthode n'entraîne pas le chargement des dossiers absents, contrairement à la méthode loadDossiers(int[]).

```
// Retrieving an existing dossier collection (not detailed here)
HRDossierCollection dossierCollection = getDossierCollection();

// Retrieving a dossier from its technical key (dossier number)
HRDossier dossier = dossierCollection.getDossier(1000);

// Retrieving a dossier from its functional key
HRDossier dossier2 = dossierCollection.getDossier(new HRKey("HRA",
   "123456"));

// Retrieving the collection's dossiers
HRDossierListIterator iterator = dossierCollection.getDossiers();

// Retrieving some dossiers from their technical keys (dossier number)
HRDossierListIterator iterator2 = dossierCollection.getDossiers(new int[] {
   1000, 2000 });

// Retrieving some dossiers from their functional keys
HRDossierListIterator iterator3 = dossierCollection.getDossiers(new
   IHRKey[] {
   new HRKey(“HRA”, “123456”),
   new HRKey(“HRA”, “234567”)
});
```

Structure d'un dossier

Un dossier peut être représenté sous la forme d'un arbre.

Le modèle de données (à gauche) définit la structure des dossiers manipulés (à droite). Le dossier se représente de manière arborescente où les nœuds représentent le dossier, les informations, les occurrences (d'information) et les rubriques :



La racine de l'arbre représente le dossier.

Les nœuds enfants du dossier représentent les informations du dossier : leur nombre dépend du modèle qui a été défini pour les données. En fait, il y a autant de nœuds "information" qu'il a été paramétré d'informations dans le modèle de données, et ce, indépendamment du nombre d'occurrences que ces informations portent.

Les nœuds enfants des nœuds "information" représentent les occurrences de l'information. Selon le type de l'information (unique / fixe ou répétitive / historique), le nombre de nœuds "occurrence" peut être de zéro, un ou plus.

Enfin, les nœuds enfants des nœuds "occurrence" représentent les valeurs associées aux rubriques de l'information. Il y a autant de nœuds "rubrique" qu'il existe de rubriques définies pour l'information.

Dans cet arbre, il n'est pas possible de créer ou de supprimer de nœuds "information" car cela supposerait de modifier dynamiquement le modèle de données, ce qui n'est pas permis. Il est en revanche possible de créer ou supprimer des nœuds "occurrence". Enfin, il n'est pas possible de créer ou de supprimer de nœuds "rubrique" car cela supposerait encore de modifier dynamiquement le modèle de données.

Les actions qu'il est possible d'effectuer sur un dossier portent donc sur les occurrences (création, suppression) et sur les rubriques (modification).

Méthodes du dossier

Le tableau suivant liste les méthodes utiles de la classe HRDossier représentant un dossier HR Access.

Méthode	Rôle
ICommitResult commit()	Soumet au serveur HR Access les modifications en attente du dossier et retourne le résultat de la mise à jour sous forme d'un ICommitResult. Cette méthode sera détaillée plus tard dans la documentation.
void delete()	Demande la suppression du dossier. Cette méthode sera détaillée plus tard dans la documentation.
void deleteAllOccurs()	Supprime toutes les occurrences de toutes les informations portées par ce dossier (sauf pour l'information "00" (Identification). Cette méthode sera détaillée plus tard dans la documentation.
HRDataSect getDataSectionByName(String)	Retourne l'information de nom (code) donné sous forme d'une instance de HRDataSect. Retourne null si le nom donné est invalide.
int getDataSectionCount()	Retourne le nombre d'informations portées par ce dossier.
List<HRDataSect> getDataSections()	Retourne une List<HRDataSect> contenant les informations portées par ce dossier.
String getDataStructureName()	Retourne le nom (code) de la structure de données de ce dossier.
AbstractDossierCollection getDossierCollection()	Retourne la collection de dossiers à laquelle ce dossier est lié.
HRDossierId getDossierId()	Retourne l'identifiant de ce dossier sous forme d'une instance de HRDossierId.
IHRKey getDossierKey()	Retourne la clé fonctionnelle de ce dossier.
int getNudoss()	Retourne la clé technique (numéro de dossier) de ce dossier.

Méthode	Rôle
String getRuleSystem()	Retourne le nom (code) de la réglementation associée à ce dossier. Le nom de la rubrique de l'information "00" (Identification) contenant le code de la réglementation associée au dossier est déterminé à partir de la définition de la structure de données du dossier.
boolean isReal()	Indique si ce dossier est "réel", c'est-à-dire s'il correspond à un dossier persisté en base de données HR Access. Un dossier qui a été créé en mémoire mais dont les modifications n'ont pas encore été soumises au serveur HR Access retourne false.
void setDossierId(HRDossierId)	Définit l'identifiant de ce dossier.

```
// Retrieving an already loaded dossier (not detailed here)
HRDossier dossier = getDossier();

// Dumping some data about the dossier
System.out.println("The dossier's data structure is <" +
    dossier.getDataStructureName() + ">");
System.out.println("The dossier's functional key is " +
    dossier.getDossierKey());
System.out.println("The dossier's technical key is " + dossier.getNudoss());
System.out.println("The dossier's associated rule system is " +
    dossier.getRuleSystem());

// Looping over the dossier's data sections
System.out.println("The dossier has " + dossier.getDataSectionCount() + "
    data section(s)");

for (Iterator iter = dossier.getDataSections().iterator(); iter.hasNext(); ) {
    HRDataSect dataSection = (HRDataSect) iter.next();
    System.out.println("Dossier contains data section <" +
    dataSection.getDataSectionName() + ">");
}
```

```
System.out.println("The dossier is " + (dossier.isReal() ? "persistent" :
   "transient"));
```

Identification d'une occurrence

Les occurrences d'information HR Access sont identifiées à l'aide de deux clés :

- une clé technique appelée "numéro de ligne" identifiant de manière unique une occurrence d'information parmi toutes celles **d'un dossier donné**. Contrairement au "numéro de dossier", cette clé technique ne correspond pas à la clé primaire de l'occurrence en base de données, c'est le doublet (numéro de dossier, numéro de ligne) qui identifie de manière unique une occurrence en table. Le numéro de ligne est stocké dans une colonne technique de table de nom NULIGN ; il est numérique et attribué par le serveur HR Access lors de la création de l'occurrence. Il n'est pas possible de personnaliser la stratégie de génération de cette clé à l'aide d'un traitement spécifique COBOL.
- Une clé fonctionnelle correspondant au n-uplet des valeurs des rubriques clés (ou argument de tri) de l'information. Cette clé n'est pas générée par le serveur HR Access, c'est le client OpenHR qui décide de la valeur des rubriques composant la clé. Ces rubriques clés servent à définir une contrainte d'unicité au niveau de la base de données. Exemple de clé fonctionnelle : un dossier de salarié (de structure de données ZY) est identifié fonctionnellement par le doublet de rubriques (ZY00 SOCCLE (Réglementation), ZY00 MATCLE (Matricule)). Exemple d'identifiant fonctionnel de dossier de salarié: ("HRA", "123456").

La classe com.hraccess.openhr.HROccurId représente un identifiant d'occurrence avec ses clés technique et fonctionnelle. La clé technique est représentée sous forme d'un entier (int) alors que la clé fonctionnelle est représentée par l'interface com.hraccess.openhr.dossier.IHRKey (implémentée par la classe com.hraccess.openhr.dossier.HRKey).

L'exemple ci-dessous illustre comment créer un HROccurId identifiant une occurrence.

```
// Creating a new ID to denote a given occurrence of data section
HROccurId id = new HROccurId();


// Setting the line number (technical key) mapped to this occurrence
id.setNulign(1);


// Setting the functional key mapped to this occurrence
id.setOccurKey(new HRKey("HRA", "123456"));
```

La création d'un HROccurId ne requiert pas de définir les deux clés, seule l'une de ces deux clés suffit pour pouvoir manipuler le dossier. L'exemple ci-dessus est donc théorique.

Les classes HROccurId et HRDossierId présentent des similitudes. C'est dû au fait que pour identifier un dossier, il suffit d'identifier son occurrence d'information "00" (Identification).

Récupération des occurrences d'une information

La classe com.hraccess.openhr.dossier.HRDataSect représente une information. Pour récupérer une instance de HRDataSect, il faut disposer d'une référence à un HRDossier et appeler soit la méthode getDataSectionByName(String) soit la méthode getDataSections().

La classe HRDataSect agit comme une "fabrique d'occurrences" : c'est par elle que les nouvelles occurrences sont créées et que les occurrences existantes sont récupérées ou supprimées. Cela explique pourquoi cette classe fournit de nombreuses méthodes liées à la récupération d'occurrences.

Le tableau suivant détaille les méthodes de lecture des occurrences d'une information de la classe HRDataSect.

Méthode	Rôle
HROccurListIterator getCurrentOccurs()	Retourne les occurrences de cette information en vigueur à la date courante sous forme d'un HROccurListIterator. L'appel de cette méthode retourne potentiellement plusieurs occurrences, d'où le type de retour. L'utilisation de cette méthode n'est pertinente que si l'information est de type unique ou répétitive historique.
HROccurListIterator getCurrentOccurs(Date)	Retourne les occurrences de cette information en vigueur à la date d'effet donnée sous forme d'un HROccurListIterator. L'appel de cette méthode retourne potentiellement plusieurs occurrences, d'où le type de retour. L'utilisation de cette méthode n'est pertinente que si l'information est de type unique ou répétitive historique.
HROccur getOccur()	Retourne la première occurrence de cette information. L'utilisation de cette méthode n'est pertinente que si l'information possède au moins une occurrence, autrement une ArrayIndexOutOfBoundsException sera levée.
HROccur getOccur(int)	Retourne l'occurrence d'information d'index donné. Le paramètre de type entier désigne l'index (zero-based) de l'occurrence recherchée. La valeur doit se situer dans l'intervalle [0 .. getOccurCount()].

Méthode	Rôle
HROccur getOccurByKey(IHRKey)	Retourne l'occurrence d'information de clé fonctionnelle donnée. Cette méthode recherche parmi les occurrences de l'information celle dont les arguments de tri correspondent aux valeurs de rubriques de la clé fonctionnelle donnée.
HROccur getOccurByNulign(int)	Retourne l'occurrence d'information de clé technique (numéro de ligne) donnée.
int getOccurCount()	Retourne le nombre d'occurrences portées par cette information.
HROccurListIterator getOccurs()	Retourne les occurrences de cette information sous forme d'un HROccurListIterator.
HROccurListIterator getOccursInRange(Date, Date)	Retourne les occurrences de cette information en vigueur pendant l'intervalle donné sous forme d'un HROccurListIterator. Les deux dates désignent respectivement les dates de début et de fin d'un intervalle. Les occurrences retournées répondent à la condition : ((date de début occurrence <= date de début intervalle) ET (date de fin intervalle<= date de fin occurrence).
boolean hasOccurs()	Indique si l'information possède au moins une occurrence.

```
// Retrieving an already loaded employee dossier (not detailed here)
HRDossier dossier = getDossier();

// Retrieving data section ZYAG (Absences)
HRDataSect dataSection = dossier.getDataSectionByName("ZYAG");

// Searching for the current occurrences of data section ZYAG
for(Iterator it = dataSection.getCurrentOccurs(); it.hasNext(); ) {
    HROccur occur = (HROccur) it.next();
    System.out.println("The employee is on vacation according to occurrence
    of data section ZYAG from "
```

```
            + occur.getValue("DATDEB") + " to " +
        occur.getValue("DATFIN"));
    }

    // Searching for the current occurrences of data section ZYAG on the 1st of
        january 2010
    for(Iterator it =
        dataSection.getCurrentOccurs(java.sql.Date.valueOf("2010-01-01"));
        it.hasNext(); ) {
        HROccur occur = (HROccur) it.next();
        System.out.println("The employee is on vacation (on 1st of january 2010)
        according to occurrence of "
            + "data section ZYAG from " + occur.getValue("DATDEB") + " to "
        + occur.getValue("DATFIN"));
    }

    // Retrieving the first occurrence of data section ZYAG
    if (dataSection.hasOccurs()) {
        HROccur occur = dataSection.getOccur();
        System.out.println("The employee's first occurrence spans from "
            + occur.getValue("DATDEB") + " to " +
        occur.getValue("DATFIN"));
    } else {
        System.out.println("The employee has no occurrence of data section
        ZYAG");
    }

    // Iterating over the data section's occurrences
    for(Iterator it = dataSection.getOccurs(); it.hasNext(); ) {
        HROccur occur = (HROccur) it.next();
        System.out.println("Found an occurrence of data section ZYAG from "
            + occur.getValue("DATDEB") + " to " +
        occur.getValue("DATFIN"));
    }

    // Searching for the current occurrences of data section ZYAG on the given
        time range
    HROccurListIterator it =
        dataSection.getOccursInRange(java.sql.Date.valueOf("2009-01-01"),
```

```
                    java.sql.Date.valueOf("2009-01-03"));

    while (it.hasNext()) {
        HROccur occur = (HROccur) it.next();
        System.out.println("Found a current occurrence of data section ZYAG
        from "
                + occur.getValue("DATDEB") + " to " +
        occur.getValue("DATFIN"));
    }
```

La classe HROccurListIterator (du package com.hraccess.openhr.dossier) représente une implémentation de l'interface java.util.ListIterator pour parcourir une List<HROccur>. Les différentes méthodes de récupération des occurrences retournent une instance de HROccurListIterator qui permet d'itérer sur les occurrences lues. Cependant, cette classe n'apporte pas beaucoup de valeur ajoutée par rapport à l'interface java.util.ListIterator (si ce n'est déclarer des méthodes typées pour éviter de caster explicitement les objets retournés par les méthodes de l'interface ListIterator en HROccur). C'est pourquoi il est possible qu'un jour cette classe soit dépréciée et remplacée par l'interface java.util.ListIterator.

Lecture d'une occurrence

C'est la classe com.hraccess.openhr.dossier.HROccur qui représente la notion d'occurrence. Une occurrence peut être symbolisée comme une Map dans laquelle les clés sont les noms des rubriques de l'occurrence et les valeurs, les valeurs de ces rubriques.

Pour lire les valeurs des rubriques d'une occurrence, il faut utiliser l'une des méthodes suivantes de la classe HROccur.

Méthode	Rôle
boolean getBoolean(String)	Retourne la valeur de la rubrique de nom donné sous forme de booléen. Le type de la rubrique tel qu'il a été défini dans Design Center doit être compatible avec le type du retour de cette méthode.
Date getDate(String)	Retourne la valeur de la rubrique de nom donné sous forme de Date. Le type de la rubrique tel qu'il a été défini dans Design Center doit être compatible avec le type du retour de cette méthode.
double getDouble(String)	Retourne la valeur de la rubrique de nom donné sous forme de double. Le type de la rubrique tel qu'il a été défini dans Design Center doit être compatible avec le type du retour de cette méthode.
Timestamp getEffectiveDate()	Retourne la date d'effet associée à l'occurrence. Cette date d'effet est la date à partir de laquelle l'occurrence est considérée comme "en vigueur" et correspond à la valeur de la rubrique de format date et de rôle "date de début" ou "date d'effet".
int getInteger(String)	Retourne la valeur de la rubrique de nom donné sous forme d'entier. Le type de la rubrique tel qu'il a été défini dans Design Center doit être compatible avec le type du retour de cette méthode.
long getLong(String)	Retourne la valeur de la rubrique de nom donné sous forme d'entier long. Le type de la rubrique tel qu'il a été défini dans Design Center doit être compatible avec le type du retour de cette méthode.
int getNulign()	Retourne la clé technique (le numéro de ligne) de l'occurrence.

Méthode	Rôle
Object getObject(String)	Retourne la valeur de la rubrique de nom donné sous forme d'Object. Le type réel de la valeur retournée dépend de la manière dont la rubrique a été définie via Design Center. Cela sera détaillé juste après.
HROccurId getOccurId()	Retourne l'identifiant de cette occurrence sous forme de HROccurId.
IHRKey getOccurKey()	Retourne la clé fonctionnelle identifiant cette occurrence.
String getString(String)	Retourne la valeur de la rubrique de nom donné sous forme de String. Le type de la rubrique tel qu'il a été défini dans Design Center doit être compatible avec le type du retour de cette méthode.
Timestamp getTimestamp(String)	Retourne la valeur de la rubrique de nom donné sous forme de Timestamp. Le type de la rubrique tel qu'il a été défini dans Design Center doit être compatible avec le type du retour de cette méthode.
Object getValue(String)	Retourne la valeur de la rubrique de nom donné sous forme d'Object. Cette méthode est analogue à getObject(String).
HRValuesMap getValues()	Retourne les valeurs pour chaque rubrique de cette occurrence sous forme de HRValuesMap.

```
// Retrieving an already loaded occurrence of data section ZYAG (not detailed
   here)
HROccur occur = getOccur();


// Dumping the occurrence's items
Map values = occur.getValues();


for(Iterator it=values.keySet().iterator(); it.hasNext(); ) {
   String itemName = (String) it.next();
   Object value = values.get(itemName);


   System.out.println("Item <" + itemName + "> has value <" + value +
   ">");
}
```

```
// Dumping the value of a named item
System.out.println("Item <MOTIFA> has value <" +
    occur.getValue("MOTIFA") + ">");


// Using strongly-typed methods to get the item values
java.sql.Date startDate = occur.getDate("DATDEB");
String reason = occur.getDate("MOTIFA");
double duration = occur.getDouble("UNITE1");
int startHour = occur.getInteger("HRSDEB");
```

Le tableau donné ci-dessus contient deux types de méthodes :

- Des méthodes fortement typées à utiliser lorsque l'on connaît le type de la rubrique lue. Exemples : getBoolean(String), getDate(String), getDouble(String), getInteger(String), getLong(String), getString(String) et getTimestamp(String).
- Des méthodes faiblement typées à utiliser lorsque l'on souhaite écrire du code générique qui ne connaît pas le type des rubriques manipulées à l'avance. Exemples : getObject(String), getValue(String) et getValues().

Type Java associé au format des rubriques

C'est le format de la rubrique (défini via Design Center) qui détermine quelle méthode fortement typée doit être appelée pour lire sa valeur. La correspondance format de rubrique / méthode de lecture est donnée dans le tableau ci-dessous. La colonne "Format DI60" désigne la valeur prise par la colonne DI60 TYFORM (stockant le format de la rubrique tel qu'elle est définie dans Design Center).

Format Design Center	Format DI60	Méthode typée adaptée
Alphanumérique sans conversion des minuscules	Y	String getString(String)
Montant financier avec devise	F	double getDouble(String)
Montant financier sans devise	G	double getDouble(String)
Date	Q	Date getDate(String)
Numérique condensé	N	int getInteger(String), long getLong(String) ou double getDouble(String) selon le nombre de décimales et la longueur de la partie entière.
Durée	D	int getInteger(String)
Numérique entier binaire	I	int getInteger(String) ou long getLong(String) selon la longueur de la partie entière.
Mois	M	Date getDate(String)
Date quantifiée	R	Date getDate(String)
Numérique display signé	S	int getInteger(String), long getLong(String) ou double getDouble(String) selon le nombre de décimales et la longueur de la partie entière.
Horodatage	H	Timestamp getTimestamp(String)
Numérique display non signé	9	int getInteger(String), long getLong(String) ou double getDouble(String) selon le nombre de décimales et la longueur de la partie entière.
Alphanumérique avec conversion des minuscules	X	String getString(String)
Alphanumérique de longueur variable	V	String getString(String)
Année	A	Date getDate(String)

Le fait de créer (via Design Center) une rubrique de nom XXXXX au format "montant financier avec devise" crée automatiquement, dans l'information, une rubrique de nom XXXXXD correspondant à la devise associée au montant financier.

Les rubriques de format "Durée" sont exprimées comme un nombre de minutes en base de données, ce qui explique pourquoi la méthode utilisée pour lire une durée est getInteger(String).

Les rubriques de format "Mois" et "Année" sont considérées comme des déclinaisons du format "Date", c'est pourquoi il faut utiliser la méthode getDate(String) pour récupérer la valeur de telles rubriques. Dans le cas du mois, la date retournée sera positionnée au premier jour du mois correspondant (exemple : 2007-05-01) et dans le cas de l'année, la date retournée sera positionné au premier Janvier de l'année correspondante (exemple : 2007-01-01).

Modification d'occurrence

Pour modifier la valeur d'une rubrique, il faut appeler l'une des méthodes de type setter au niveau de la classe HROccur.

Méthode	Rôle
void setBoolean(String, boolean)	Définit la valeur de la rubrique de nom donné sous forme de booléen. Le type de la rubrique tel qu'il a été défini dans Design Center doit être compatible avec le type de l'argument de cette méthode.
void setDate(String, Date)	Définit la valeur de la rubrique de nom donné sous forme de Date. Le type de la rubrique tel qu'il a été défini dans Design Center doit être compatible avec le type de l'argument de cette méthode.
void setDouble(String, double)	Définit la valeur de la rubrique de nom donné sous forme de double. Le type de la rubrique tel qu'il a été défini dans Design Center doit être compatible avec le type de l'argument de cette méthode.
void setInteger(String, int)	Définit la valeur de la rubrique de nom donné sous forme d'entier. Le type de la rubrique tel qu'il a été défini dans Design Center doit être compatible avec le type de l'argument de cette méthode.
void setLong(String, long)	Définit la valeur de la rubrique de nom donné sous forme d'entier long. Le type de la rubrique tel qu'il a été défini dans Design Center doit être compatible avec le type de l'argument de cette méthode.

Méthode	Rôle
HROccur setNull()	Supprime la valeur de chaque rubrique et retourne l'occurrence.
HROccur setNull(String)	Supprime la valeur de la rubrique de nom donné et retourne l'occurrence.
void setObject(String, Object)	Définit la valeur de la rubrique de nom donné sous forme d'Object. Le type du paramètre passé dépend de la manière dont la rubrique a été définie via Design Center. Voir plus haut.
void setOccurId(HROccurId)	Définit l'identifiant de cette occurrence sous forme de HROccurId.
void setString(String, String)	Définit la valeur de la rubrique de nom donné sous forme de String. Le type de la rubrique tel qu'il a été défini dans Design Center doit être compatible avec le type de l'argument de cette méthode.
void setTimestamp(String, Timestamp)	Définit la valeur de la rubrique de nom donné sous forme de Timestamp. Le type de la rubrique tel qu'il a été défini dans Design Center doit être compatible avec le type de l'argument de cette méthode.
void setValue(String, Object)	Définit la valeur de la rubrique de nom donné sous forme d'Object. Cette méthode est analogue à setObject(String, Object).
void setValues(HRValuesMap)	Définit les valeurs des rubriques de cette occurrence à partir de la HRValuesMap donnée. Il n'est pas obligatoire de fournir une valeur à chacune des rubriques de l'information. Cette méthode permet principalement d'effectuer plusieurs mises à jour de rubrique en un appel. L'appel de cette méthode est équivalent à setValues(HRValuesMap, false).

Méthode	Rôle
void setValues(HRValuesMap, boolean)	Définit les valeurs des rubriques de cette occurrence à partir de la HRValuesMap donnée. Il n'est pas obligatoire de fournir une valeur à chacune des rubriques de l'information. Cette méthode permet principalement d'effectuer plusieurs mises à jour de rubrique en un appel. Le second paramètre de type booléen indique si la valeur des rubriques doit être supprimée avant de prendre en compte les valeurs fournies par la HRValuesMap. Par conséquent, si le paramètre vaut true et que la HRValuesMap donnée ne définit pas de valeur pour chaque rubrique de l'information, l'appel de cette méthode fait "perdre des données" en réinitialisant les valeurs des rubriques pour lesquelles aucune valeur n'a été fournie.

```
// Retrieving an already loaded occurrence of data section ZYAG (not detailed
   here)
HROccur occur = getOccur();

// Resetting the occurrence's values
occur.setNull();

// Using strongly-typed methods to update the occurrence
occur.setDate("DATDEB", java.sql.Date.valueOf("2008-08-12"));
occur.setString("MOTIFA", "RTT");
occur.setInteger("UNITE1", 10);

// Resetting the value for a named item
occur.setNull("DATDEB");

// Using weakly-typed methods to update the occurrence
occur.setValue("DATDEB", java.sql.Date.valueOf("2008-08-13"));
occur.setValue("MOTIFA", "CPN");
occur.setValue("UNITE1", new Integer(20));

// Updating many items in a single call
```

```
HRValuesMap map = new HRValuesMap();
map.put(“DATDEB”, java.sql.Date.valueOf(“2008-08-14”));
map.put(“MOTIFA”, “RTT”);

occur.setValues(map);
```

Soumission des modifications au serveur HR Access

Toutes les modifications effectuées sur la représentation objet d'un dossier sont mémorisées par l'API et serviront à mettre à jour de manière différentielle les données sur le serveur HR Access.

Toute modification doit donc faire l'objet d'une soumission au serveur HR Access pour mettre à jour les données.

Il existe plusieurs manières de soumettre les modifications en attente.

Soumission des modifications pour un dossier

Pour soumettre les modifications en attente d'un seul dossier, il faut appeler la méthode HRDossier.commit(). La méthode retourne une instance de com.hraccess.openhr.dossier.ICommitResult décrivant les éventuelles erreurs rencontrées lors de la mise à jour des données.

Il faut bien distinguer deux types d'erreurs qui peuvent survenir ici :

- Les erreurs dites "fonctionnelles" : ce sont des erreurs levées par le serveur HR Access car une règle métier n'a pas été respectée. Cette erreur est typiquement levée par un traitement spécifique. Un autre type de règle qui peut ne pas avoir été respectée concerne les contraintes définies au niveau du modèle de données : les contrôles de contenu, de correspondance, de présence au niveau des rubriques mises à jour. Si une occurrence est créée mais que l'une de ses rubriques obligatoires n'est pas alimentée, alors une erreur fonctionnelle sera retournée à l'API. Il est possible de récupérer de ce type d'erreur. La solution consiste à afficher à l'utilisateur (s'il y en a un) le message d'erreur et à l'inviter à corriger les données avant de resoumettre les modifications.
- Les erreurs dites "techniques" : ce sont des erreurs graves levées par le serveur HR Access pour signaler au client OpenHR que sa requête ne peut être satisfaite. Exemples de ce type d'erreur : la définition du modèle de données entre le client OpenHR et le serveur HR Access est désynchronisée, ce qui empêche d'interpréter la requête cliente ; le programme COBOL chargé de l'accès à la base de données n'a pas été bindé ; le temps imparti au serveur pour traiter la requête a été dépassé. Il n'est pas possible de récupérer d'une telle erreur car la cause n'est pas liée à une saisie que l'utilisateur pourrait corriger.

La méthode HRDossier.commit() retourne les erreurs fonctionnelles sous forme de ICommitResult. Ainsi, ce n'est pas parce que la méthode retourne une instance de ICommitResult que la mise à jour des données a réussi. Les erreurs techniques sont, elles, notifiées au client OpenHR par la levée d'une HRDossierCollectionCommitException.

```
// Retrieving an already loaded employee dossier (not detailed here)
HRDossier dossier = getEmployeeDossier();


// Updating the employee's last name (see item ZY07 NOMUSE)
dossier.getDataSectionByName("07").getOccur().setString("NOMUSE",
   "WESTON");


try {
   // Committing the pending changes to the HR Access server
   ICommitResult commitResult = dossier.commit();


   if (commitResult.getErrors().length > 0) {
   // There is at least one functional error
   System.err.println("The dossier update failed for a functional reason");
   } else {
   System.out.println("The dossier update succeeded");
   }
} catch (HRDossierCollectionCommitException e) {
   // A technical error occurred when updating the dossier
   System.err.println("The dossier update failed for a technical reason", e);
}
```

Pour information, la méthode HRDossierCollection.commitDossier(HRDossier) permet également d'arriver au même résultat.

Soumission des modifications de plusieurs dossiers

Pour mettre à jour plusieurs dossiers simultanément sur le serveur HR Access, il faut utiliser la méthode HRDossierCollection.commitDossiers(Iterator). L'itérateur en paramètre est de type Iterator<HRDossier> et permet d'accéder aux dossiers que vous souhaitez mettre à jour sur le serveur HR Access.

```
// Retrieving the dossier collection (not detailed here)
HRDossierCollection dossierCollection = getDossierCollection();


// Retrieving an already loaded employee dossier waiting for commit (not
   detailed here)
HRDossier dossier = getEmployeeDossier();


// Retrieving another already loaded employee dossier waiting for commit (not
   detailed here)
HRDossier dossier2 = getAnotherEmployeeDossier();
```

```
// Creating a List<HRDossier>
List dossiers = new ArrayList();
dossiers.add(dossier);
dossiers.add(dossier2);

try {
    // Committing the pending changes to the HR Access server
    ICommitResult commitResult =
    dossierCollection.commitDossiers(dossiers);

    if (commitResult.getErrors().length > 0) {
    // There is at least one functional error
    System.err.println("The dossiers update failed for a functional reason");
    } else {
    System.out.println("The dossiers update succeeded");
    }
} catch (HRDossierCollectionCommitException e) {
    // A technical error occurred when updating the dossiers
    System.err.println("The dossiers update failed for a technical reason", e);
}
```

La méthode HRDossierCollection.commitAllDossiers() permet aussi de soumettre les modifications de tous les dossiers de la collection de dossiers sans devoir les énumérer sous forme d'Iterator.

```
// Retrieving the dossier collection (not detailed here)
HRDossierCollection dossierCollection = getDossierCollection();

try {
    // Committing the pending changes to the HR Access server
    ICommitResult commitResult = dossierCollection.commitAllDossiers();

    if (commitResult.getErrors().length > 0) {
    // There is at least one functional error
    System.err.println("The dossiers update failed for a functional reason");
    } else {
    System.out.println("The dossiers update succeeded");
    }
```

```
} catch (HRDossierCollectionCommitException e) {
    / / A technical error occurred when updating the dossiers
    System.err.println(“The dossiers update failed for a technical reason”, e);
}
```

La transaction de mise à jour

Lorsque vous soumettez des modifications au serveur HR Access, une requête de mise à jour des dossiers est adressée par l'API au serveur HR Access. La mise à jour de la base de données est réalisée à l'intérieur d'une transaction, c'est-à-dire à l'intérieur d'une unité de travail qui assure que l'ensemble des modifications apportées à la base de données réussissent toutes ensemble ou échouent toutes ensemble.

L'API OpenHR ne permet pas de définir le cycle de vie de cette transaction, c'est-à-dire le moment où celle-ci démarre et où elle est validée (commit) ou annulée (rollback). Cette opération est prise en charge par le programme COBOL (BHR). C'est pourquoi le cycle de vie de la transaction vers la base de données est identique au cycle de vie d'une requête adressée au serveur HR Access : la transaction démarre lors de la réception de la requête de mise à jour des données et est validée (ou annulée) lorsque la requête a été traitée. Cette particularité s'explique par une limitation du serveur COBOL HR Access qui ne peut conserver une transaction ouverte (transaction longue) sur plusieurs requêtes clientes.

Lorsque vous effectuez des mises à jour de dossiers, vous devez donc être conscient de cette limitation car il n'est pas possible d'assurer que la mise à jour soir tout le temps transactionnelle. Ceci est particulièrement vrai dans les cas suivants :

- Le nombre de dossiers à mettre à jour à l'intérieur d'une transaction est supérieur au nombre maximal de dossiers que le serveur HR Access peut traiter en un échange avec le client. Ce nombre étant de 99 dossiers, si un client souhaite mettre à jour 150 dossiers, il devra nécessairement émettre deux requêtes au serveur. La mise à jour des 150 dossiers ne pourra donc s'effectuer au sein de la même transaction. Si la première mise à jour (de 99 dossiers) réussit mais que la seconde (de 51 dossiers) échoue, la base de données se retrouve alors dans un état instable avec une partie des dossiers mis à jour (les 99 premiers).
- La mise à jour transactionnelle porte sur des dossiers de structures de données différentes. Si pour une raison fonctionnelle quelconque, le client OpenHR doit, dans la même transaction, mettre à jour deux dossiers de structures de données différentes, il est obligé d'émettre deux requêtes de mise à jour au serveur HR Access car un processus HR Access ne permet de mettre à jour que des dossiers d'une structure de données donnée. Il est donc possible que la mise à jour de ces deux dossiers laisse la base de données dans un état instable.

Une autre conséquence de l'absence de gestion de la transaction par l'API OpenHR est qu'il n'est pas possible d'inscrire la mise à jour de dossiers HR Access au sein d'une transaction distribuée. Ceci interdit donc d'utiliser l'API OpenHR (à travers un Web service par exemple) pour créer un processus métier distribué et transactionnel.

Pour contourner ces limitations, il est possible de mettre à jour les dossiers en mode batch.

Création d'occurrence

La classe HRDataSect agit comme une "fabrique d'occurrences", c'est pourquoi c'est elle qui définit les méthodes pour créer de nouvelles occurrences d'information.

Le tableau ci-dessous liste les méthodes de la classe HRDataSect pour créer de nouvelles occurrences.

Méthode	Rôle
HROccur createOccur()	Crée une nouvelle occurrence d'information et la retourne sous forme de HROccur. L'appel de cette méthode peut échouer si la confidentialité de l'utilisateur ne lui permet pas de créer de nouvelle occurrence ou si l'information est de type unique fixe et dispose déjà d'une occurrence. Cette méthode ne fonctionne pas pour l'information 00 (Identification) car créer une telle occurrence revient à créer un nouveau dossier. Il existe une méthode dédiée à ce cas-là au niveau de la collection de dossiers qui sera présentée plus tard.
HROccur createOccur(HRKey)	Crée une nouvelle occurrence d'information avec la clé fonctionnelle donnée et la retourne sous forme de HROccur. La clé fonctionnelle (HRKey) donnée permet d'alimenter les rubriques clé (argument de tri) de l'information. L'appel de cette méthode peut échouer si la confidentialité de l'utilisateur ne lui permet pas de créer de nouvelle occurrence. Cette méthode ne fonctionne pas pour l'information 00 (Identification) car créer une telle occurrence revient à créer un nouveau dossier. Il existe une méthode dédiée à ce cas-là au niveau de la collection de dossiers qui sera présentée plus tard.

Méthode	Rôle
HROccur createOccur(HRValuesMap)	Crée une nouvelle occurrence d'information alimentée avec les valeurs de rubriques de la HRValuesMap donnée et la retourne sous forme de HROccur. Le paramètre de type HRValuesMap désigne une Map<String, Object> contenant les valeurs de rubrique par nom de rubrique à prendre en compte pour alimenter l'occurrence. Rien n'oblige la Map à fournir une valeur pour toutes les rubriques de l'information. L'appel de cette méthode peut échouer si la confidentialité de l'utilisateur ne lui permet pas de créer de nouvelle occurrence. Cette méthode ne fonctionne pas pour l'information 00 (Identification) car créer une telle occurrence revient à créer un nouveau dossier. Il existe une méthode dédiée à ce cas-là au niveau de la collection de dossiers qui sera présentée plus tard.

```
// Retrieving an already loaded employee dossier (not detailed here)
HRDossier dossier = getEmployeeDossier();

// Retrieving data section ZYAG (Absences)
HRDataSect dataSection = dossier.getDataSectionByName("AG");

// Creating a new occurrence of data section ZYAG (version 1) and
   populating items
HROccur occurrence = dataSection.createOccur();
occurrence.setDate("DATDEB", java.sql.Date.valueOf("2008-01-01"));
occurrence.setString("MOTIFA", "RTT");
occurrence.setDate("DATFIN", java.sql.Date.valueOf("2008-01-05"));

// Creating a new occurrence of data section ZYAG (version 2) and
   populating items
// Items ZYAG DATDEB and ZYAG MOTIFA are key items (sort arguments)
HROccur occurrence2 = dataSection.createOccur(new
   HRKey(java.sql.Date.valueOf("2008-01-01"), "RTT"));

// Populating mandatory (non-key) items
```

```
occurrence2.setDate(“DATFIN”, java.sql.Date.valueOf(“2008-01-05”));

// Creating a new occurrence of data section ZYAG (version 3) and
   populating items
HRValuesMap map = new HRValuesMap();
map.put(“DATDEB”, java.sql.Date.valueOf(“2008-01-01”));
map.put(“MOTIFA”, “RTT”);
map.put(“DATFIN”, java.sql.Date.valueOf(“2008-01-05”));

HROccur occurrence3 = dataSection.createOccur(map);

// Committing pending changes to the HR Access server (not detailed here)
...
```

Les trois exemples de création d'occurrence ci-dessus sont rigoureusement équivalents.

Suppression d'occurrence(s)

Pour supprimer une occurrence, il faut appeler la méthode HROccur.delete(). L'appel de cette méthode a pour effet de détacher le nœud de l'occurrence de son nœud parent représentant l'information modifiée. Il peut échouer si la confidentialité de l'utilisateur ne lui permet pas de supprimer d'occurrence.

Cette méthode ne fonctionne pas pour l'information 00 (Identification) car supprimer une telle occurrence revient à supprimer un dossier. Il existe une méthode dédiée à ce cas-là au niveau de la collection de dossiers qui sera présentée plus tard.

```
// Retrieving an already loaded employee dossier (not detailed here)
HRDossier dossier = getEmployeeDossier();

// Retrieving data section ZYAG (Absences)
HRDataSect dataSection = dossier.getDataSectionByName(“AG”);

// Retrieving the first occurrence of data section ZYAG
HROccur occurrence = dataSection.getOccur();

try {
   // Deleting the occurrence of data section ZYAG
   occurrence.delete();
   System.out.println(“Deletion of occurrence succeeded”);
} catch (HRDossierCollectionException e) {
```

```
    System.err.println("Deletion of occurrence failed", e);
}

// Committing pending changes to the HR Access server (not detailed here)
...
```

L'API fournit une autre méthode pour supprimer plusieurs occurrences en un seul appel grâce aux méthodes HRDataSect.deleteOccurs() et HRDataSect.deleteOccurs(HROccurListIterator). Leur rôle est détaillé dans le tableau suivant.

Méthode	Rôle
HROccurListIterator deleteOccurs()	Supprime toutes les occurrences de cette information et retourne un HROccurListIterator sur les occurrences supprimées. L'appel de cette méthode peut échouer si la confidentialité de l'utilisateur ne lui permet pas de supprimer d'occurrence. Cette méthode ne fonctionne pas pour l'information 00 (Identification) car supprimer une telle occurrence revient à supprimer un dossier. Il existe une méthode dédiée à ce cas-là au niveau de la collection de dossiers qui sera présentée plus tard.
HROccurListIterator deleteOccurs(HROccurListIterator)	Supprime toutes les occurrences de cette information contenues dans le HROccurListIterator donné et retourne celui-ci. L'appel de cette méthode peut échouer si la confidentialité de l'utilisateur ne lui permet pas de supprimer d'occurrence. Cette méthode ne fonctionne pas pour l'information 00 (Identification) car supprimer une telle occurrence revient à supprimer un dossier. Il existe une méthode dédiée à ce cas-là au niveau de la collection de dossiers qui sera présentée plus tard.

```
// Retrieving an already loaded employee dossier (not detailed here)
HRDossier dossier = getEmployeeDossier();

// Retrieving data section ZYAG (Absences)
HRDataSect dataSection = dossier.getDataSectionByName("AG");

try {
```

```
    // Deleting all the occurrences of data section ZYAG
    dataSection.deleteOccurs();
    System.out.println("Deletion of occurrences succeeded");
} catch (HRDossierCollectionException e) {
    System.err.println("Deletion of occurrences failed", e);
}

// Committing pending changes to the HR Access server (not detailed here)
...
```

Suppression de dossier

Pour supprimer un dossier, il faut appeler la méthode HRDossier.delete(). L'appel de cette méthode a pour effet de détacher le nœud du dossier de son nœud parent représentant la collection de dossiers modifiée. Il peut échouer si la confidentialité de l'utilisateur ne lui permet pas de supprimer de dossier, c'est-à-dire de supprimer d'occurrence sur l'information 00 (Identification).

```
// Retrieving an already loaded employee dossier (not detailed here)
HRDossier dossier = getEmployeeDossier();

try {
    // Deleting the dossier
    dossier.delete();
    System.out.println("Deletion of dossier succeeded");
} catch (HRDossierCollectionException e) {
    System.err.println("Deletion of dossier failed", e);
}

// Committing pending changes to the HR Access server (not detailed here)
...
```

Clonage de dossier

L'API fournit la méthode HRDossierCollection.cloneDossier(HRDossier) afin de cloner un dossier en mémoire. Le paramètre de type HRDossier représente le dossier à cloner. L'appel de cette méthode va créer, au niveau de la collection de dossiers, un nouveau dossier vierge puis va itérer sur le dossier source pour recréer les occurrences de chacune de ses informations. A l'issue de l'opération, la collection de dossiers va donc contenir deux dossiers rigoureusement identiques : le dossier source et le dossier clone.

Attention toutefois car la soumission au serveur HR Access de ce dossier clone tel quel sera rejetée pour cause de violation de contrainte d'unicité. Au minimum, il faut donc modifier l'occurrence de l'information 00 (Identification) du dossier clone afin de s'assurer de l'unicité de sa clé fonctionnelle.

Précisons également que cette méthode de clonage peut être implémentée facilement en dehors de l'API en utilisant les méthodes de création de dossier / occurrence et d'itération sur les informations / occurrences déjà vues. C'est pourquoi chaque opération de création d'occurrence / dossier doit être autorisée par la confidentialité du rôle rattachée à la collection de dossiers.

Cette technique de clonage du dossier n'est pas ce qu'on appelle de la "duplication de dossier". La technique de duplication de dossiers repose sur l'utilisation d'un processus de qualification "DU" (Duplication de dossier) invoqué à travers un processus de type "GD" (Gestion de dossiers). Avec la duplication de dossiers, le travail de duplication intervient côté serveur HR Access alors que dans notre cas, tout s'effectue côté client OpenHR. Voir la section "Duplication de dossier".

```
// Retrieving an already loaded employee dossier (not detailed here)
HRDossier source = getEmployeeDossier();


// Retrieving the dossier collection to which the dossier is attached (not
   detailed here)
HRDossierCollection dossierCollection = getDossierCollection();


try {
    / / Cloning the source employee dossier
    HRDossier clone = dossierCollection.cloneDossier(source);
    System.out.println(“Cloning of dossier succeeded”);
} catch (HRDossierCollectionException e) {
    System.err.println(“Cloning of dossier failed”, e);
}


// Committing pending changes to the HR Access server (not detailed here)
...
```

Création de dossier

On peut créer un dossier de deux manières : avec ou sans utiliser de paramètres. Le cas le plus simple (celui sans paramètre) sera détaillé en premier.

C'est la logique fonctionnelle invoquée lors de la création du dossier qui détermine s'il faut utiliser des paramètres ou pas. Dans le cas d'une création simple de dossier sans logique fonctionnelle poussée, l'utilisation de paramètres est inutile. En revanche, dans un cas plus complexe comme l'embauche d'un salarié, la logique fonctionnelle impose de renseigner des paramètres avant de créer le dossier.

Sans paramètre

Ce paragraphe explique comment créer un dossier HR Access dans un cas simple, c'est-à-dire dans le cas où la logique fonctionnelle de création du dossier ne requiert pas l'utilisation d'informations paramètres. Ce cas plus complexe - qui correspond au cas de l'embauche d'un salarié - nécessite de renseigner, préalablement à la création du dossier, des paramètres d'une manière spécifique. Ce cas sera détaillé plus loin.

Pour l'instant, la création de dossier suppose que l'utilisation d'informations paramètres n'est pas utile.

Pour créer un dossier, il faut appeler l'une des deux méthodes suivantes de la classe HRDossierCollection.

Méthode	Rôle
HRDossier createDossier()	Crée un nouveau dossier au niveau de la collection de dossiers et retourne celui-ci. Le dossier ainsi retourné est complètement vierge. La confidentialité du rôle défini au niveau de la collection de dossiers doit autoriser la création de dossiers.
HRDossier createDossier(HRKey)	Crée un nouveau dossier au niveau de la collection de dossiers en l'initialisant à l'aide de la clé fonctionnelle donnée et retourne celui-ci. La confidentialité du rôle défini au niveau de la collection de dossiers doit autoriser la création de dossiers.

La création d'un dossier induit automatiquement la création de son occurrence d'information 00 (Identification) car cette occurrence représente le dossier. Idem pour la suppression d'un dossier.

```
// Retrieving the dossier collection (not detailed here)
HRDossierCollection dossierCollection = getDossierCollection();

try {
    / / Creating a new blank (employee) dossier
    HRDossier dossier = dossierCollection.createDossier();
```

```
        // Retrieving the occurrene of data section ZY00 (was automatically
        created upon dossier creation)
        HROccur occurrenceZY00 = dossier.getDataSection("00").getOccur();

        // Populating the key items of data section ZY00
        occurrenceZY00.setString("SOCCLE", "HRA");
        occurrenceZY00.setString("MATCLE", "123456");

        // Populating other mandatory data sections (not detailed here)
        ...

        System.out.println("Creation of dossier succeeded");
    } catch (HRDossierCollectionException e) {
        System.err.println("Creation of dossier failed", e);
    }

    // Committing pending changes to the HR Access server (not detailed here)
    ...
```

L'exemple ci-dessus est identique à l'exemple suivant.

```
    // Retrieving the dossier collection (not detailed here)
    HRDossierCollection dossierCollection = getDossierCollection();

    try {
        // Creating a new (employee) dossier with given functional key
        HRDossier dossier = dossierCollection.createDossier(new HRKey("HRA",
        "123456"));

        // Populating other mandatory data sections (not detailed here)
        ...

        System.out.println("Creation of dossier succeeded");
    } catch (HRDossierCollectionException e) {
        System.err.println("Creation of dossier failed", e);
    }

    // Committing pending changes to the HR Access server (not detailed here)
    ...
```

Avec des paramètres

Lorsque la création d'un dossier nécessite de renseigner des paramètres (au moyen d'informations paramètres), il faut préalablement :

- Renseigner les paramètres
- Créer le dossier
- Soumettre les modifications au serveur HR Access

Pour rappel, l'utilisation d'informations de type "paramètre" est une astuce permettant de passer des paramètres (et leurs valeurs) à des traitements spécifiques exécutés au moment de la création du dossier.

Concrètement, les paramètres sont envoyés au serveur lors d'une première soumission (commit). Le serveur HR Access va alors persister **au niveau de** la **conversation utilisée** les valeurs de paramètres reçues.

La seconde soumission au serveur HR Access est la requête de création du dossier. Celle-ci doit s'effectuer au titre de la même conversation afin que le serveur puisse récupérer la valeur des paramètres préalablement persistés. Lors de la création du dossier, les traitements spécifiques sont exécutés avec la valeur des paramètres sous forme de variable COBOL. C'est ce mécanisme en deux temps avec persistance intermédiaire côté serveur des paramètres qui justifie l'existence du concept de conversation : aucune autre spécification ne le justifie. C'est pourquoi, en dehors d'un tel cas d'usage, l'utilisation de la conversation principale de l'utilisateur suffit. Autre conséquence : si un utilisateur souhaite créer simultanément des dossiers de salariés (donc avec des paramètres), il doit utiliser autant de conversations différentes que de threads de création de dossiers. Sans cette précaution, il y aura collision au niveau serveur entres les différentes valeurs de paramètres.

Pour gérer des informations de type paramètre, il faut d'abord déclarer celles-ci au niveau de la configuration de la collection de dossiers (de la même manière que des informations "réelles").

```
// Creating a new configuration (to create a dossier collection)
HRDossierCollectionParameters parameters = new
   HRDossierCollectionParameters();
parameters.setType(HRDossierCollectionParameters.TYPE_NORMAL);
parameters.setProcessName(“FS000”);
parameters.setDataStructureName(“ZY”);
parameters.addDataSection(new HRDataSourceParameters.DataSection(“00”));
parameters.addDataSection(new HRDataSourceParameters.DataSection(“05”));
parameters.addDataSection(new HRDataSourceParameters.DataSection(“06”));
parameters.addDataSection(new HRDataSourceParameters.DataSection(“07”));
parameters.addDataSection(new HRDataSourceParameters.DataSection(“3X”));
   // □Parameter data section
...
```

Les informations réelles sont celles que l'on peut manipuler via la classe HRDossier et qui représentent des dossiers HR Access persistés en base de données. Ces dossiers sont récupérés via les méthodes HRDossierCollection.loadDossier() ou HRDossierCollection.loadDossiers() de chargement.

Dans le cas des informations paramètres, l'API permet de manipuler un dossier virtuel (qui ne représente aucun dossier réellement persisté en base de données) de la même manière qu'un dossier réel. Ce dossier sera nommé "dossier paramètre" dans la suite de cette documentation. Ainsi, pour alimenter des paramètres, il suffit de récupérer l'information de type paramètre sur le dossier paramètre et de manipuler l'unique occurrence permise pour cette information.

La récupération du dossier paramètre s'effectue via la méthode HRDossierCollection.getParameterDossier().

```
// Retrieving an existing dossier collection with some parameter data
   sections (not detailed here)
HRDossier dossierCollection = getDossierCollection();


// Retrieving the virtual “parameter” dossier associated to the dossier
   collection
HRDossier parameterDossier = dossierCollection.getParameterDossier();


// Populating the parameters (items of data section ZY3X)
HROccur occurrence =
   parameterDossier.getDataSectionByName(“3X”).createOccur();
occurrence.setDate(“DTOB01”, java.sql.Date.valueOf(“2009-01-01”));
occurrence.setString(“REGLEM”, “HRA”);
occurrence.setString(“IDCY00”, “HRA”);
occurrence.setString(“MATCLE”, “123456”);
...

// Creating the employee dossier (not detailed here)
...

// Committing the pending changes to the HR Access server
ICommitResult result = dossierCollection.commitAllDossiers();


// Handling the commit result of parameters (not detailed here)
HRDossierCollection.CommitResult parameterCommitResult =
   result.getParameterCommitResult();
...


// Handling the commit result of dossiers (not detailed here)
```

```
HRDossierCollection.CommitResult dossierCommitResult =
    result.getDossierCommitResult();
...
```

L'exemple montre comment alimenter les paramètres requis pour embaucher un salarié (représentés par l'information paramètre ZY3X (Paramètres de sélection & création)). La manipulation du dossier paramètre, de ses informations et rubriques est identique à celle d'un dossier normal.

Après l'alimentation des paramètres, il faut créer localement le dossier de salarié au niveau de la collection de dossiers.

C'est lors de la soumission des modifications au serveur HR Access que le double commit s'effectue ; cela n'est en effet pas visible du développeur qui n'appelle qu'une méthode. Le premier commit envoie au serveur HR Access les modifications relatives aux informations paramètres tandis que le second commit envoie les modifications relatives au dossier de salarié à créer.

C'est ce double commit qui explique pourquoi le retour des méthodes de soumission (cf. HRDossier.commit(), HRDossierCollection.commitDossier(HRDossier), etc.) de type ICommitResult possède deux méthodes de nom getDossierCommitResult() et getParameterCommitResult(). La première méthode retourne le résultat de mise à jour pour les modifications relatives aux dossiers alors que la seconde méthode fait de même mais pour les modifications relatives aux paramètres.

Lorsque vous utilisez des informations paramètres, il faut donc prendre en compte le résultat des deux commits qui ont eu lieu pour détecter les éventuelles erreurs.

Gestion des erreurs de mise à jour

Il a été expliqué que la soumission des modifications d'un (ou plusieurs) dossier(s) au serveur HR Access retournait le résultat de la mise à jour sous forme d'une instance de ICommitResult. Cet objet comporte deux composantes :

- Les éventuelles erreurs relatives à la mise à jour des paramètres sur le serveur HR Access (disponibles uniquement si la collection de dossiers manipule des informations paramètres) récupérables par la méthode ICommitResult.getParameterCommitResult().
- Les éventuelles erreurs relatives à la mise à jour des dossiers sur le serveur HR Access récupérables par la méthode ICommitResult.getDossierCommitResult().

L'objet retourné par ces deux méthodes est de type com.hraccess.openhr.dossier.HRDossierCollection.CommitResult.

Le tableau suivant liste les méthodes utiles de cette classe.

Méthode	Rôle
Vector<HRDossierId> getConfirmationDossierIds()	Retourne la liste des identifiants des dossiers HR Access (sous forme d'un Vector<HRDossierId>) pour lesquels l'utilisateur doit confirmer la mise à jour car celle-ci a levé une erreur de type "avertissement avec confirmation" (erreur de poids 3 ou 4). Voir la section "La gestion des avertissements avec confirmation".
HRResultUserError.Error[] getConfirmations()	Retourne les erreurs de type "avertissement avec confirmation" (erreur de poids 3 ou 4) levées lors de la mise à jour des données. Voir la section "La gestion des avertissements avec confirmation".
Vector<HRDossierId> getCreatedDossierIds()	Retourne la liste des identifiants des dossiers qui viennent d'être créés sur le serveur HR Access.
Vector<HRDossierId> getDeletedDossierIds()	Retourne la liste des identifiants des dossiers qui viennent d'être supprimés sur le serveur HR Access.
HRResultUserError.Error getError(int)	Retourne l'erreur (de poids quelconque) d'index donné. L'index doit être dans l'intervalle [0 .. getErrorCount()].
int getErrorCount()	Retourne le nombre d'erreurs (de poids quelconque) levées lors de la mise à jour des dossiers sur le serveur HR Access.
Vector<HRDossierId> getErrorDossierIds()	Retourne la liste des identifiants des dossiers HR Access (sous forme d'un Vector<HRDossierId>) pour lesquels une erreur (de poids 5) de mise à jour a été levée.
HRResultUserError.Error[] getErrors()	Retourne les erreurs (de poids 5 et éventuellement 3 et 4 (avertissements avec confirmation)) qui ont empêché la mise à jour des dossiers HR Access. Voir la section "La gestion des avertissements avec confirmation".

Méthode	Rôle
HRResultUserError.Error[] getErrors(int)	Retourne les erreurs de poids donné. Le poids doit être dans l'intervalle [1 .. 5]. Voir la section "La gestion des avertissements avec confirmation".
Vector<HRDossierId> getModifiedDossierIds()	Retourne la liste des identifiants des dossiers HR Access (sous forme d'un Vector<HRDossierId>) mis à jour avec succès sur le serveur HR Access.
Vector<HRDossierId> getWarningIds()	Retourne les identifiants des dossiers HR Access mis à jour avec succès pour lesquels un "avertissement sans confirmation" (erreur de poids 1 ou 2) a été levé. Voir la section "La gestion des avertissements avec confirmation".
HRResultUserError.Error[] getWarnings()	Retourne les erreurs (de poids 1 et 2 (avertissements sans confirmation)) qui ont été levées lors de la mise à jour des dossiers HR Access. Voir la section "La gestion des avertissements avec confirmation".
boolean isDossierCreated(HRDossierId)	Indique si le dossier d'identifiant donné a bien été créé sur le serveur HR Access lors de la mise à jour.
boolean isDossierDeleted(HRDossierId)	Indique si le dossier d'identifiant donné a bien été supprimé sur le serveur HR Access lors de la mise à jour.
boolean isDossierModified(HRDossierId)	Indique si le dossier d'identifiant donné a bien été mis à jour sur le serveur HR Access lors de la mise à jour.

Certaines des méthodes de la classe HRDossierCollection.CommitResult peuvent prêter à confusion en raison de l'ambiguïté qui caractérise le terme d'erreur.

Une "**erreur**" désigne une notification retournée par le serveur HR Access à l'issue d'une requête de mise à jour de dossiers.

Cette erreur possède **un poids d'erreur allant de 1 à 5**. En fonction du poids d'erreur, l'erreur se nomme alors :

- "avertissement sans confirmation (de mise à jour)" (poids 1 et 2)
- "avertissement avec confirmation (de mise à jour)" (poids 3 et 4)
- "erreur bloquante" (poids 5)

Il faut donc faire attention à ne pas confondre les notions d'erreur (affectée d'un poids) et d'erreur bloquante qui désigne les erreurs dont le poids est de 5.

Certaines méthodes de la classe HRDossierCollection.CommitResult ont également une sémantique incohérente entre elles : ainsi la méthode getErrorCount() retourne-t-elle le nombre d'erreurs (affectées d'un poids) retournées par le serveur HR Access alors que la méthode getErrors() retourne les erreurs qui ont empêché la mise à jour des dossiers. La sémantique du terme "errors" n'est donc pas la même pour ces deux méthodes.

Ce dernier cas est aggravé par le fait que la méthode getErrors(), au lieu de retourner les erreurs bloquantes (donc uniquement celles de poids 5 comme la sémantique le laisserait penser), retourne également les erreurs de poids 3 et 4 **si celles-ci ne sont pas ignorées**.

En conclusion, Pour bien comprendre ce que fait une méthode de la classe HRDossierCollection.CommitResult, reportez-vous à la javadoc de la méthode correspondante.

La classe HRResultUserError.Error retournée par certaines des méthodes est une classe représentant une erreur de mise à jour affectée d'un poids d'erreur. Cette classe est un bean dont les propriétés publiques sont détaillées ci-dessous.

Propriété	**Signification**
String comment	Commentaire affecté à l'erreur
String dataSectionName	Nom (code) de l'information sur laquelle porte l'erreur.
String dataSectionName2	Nom (code) de l'information #2 sur laquelle porte l'erreur.
String dataStructureName	Nom (code) de la structure de données sur laquelle porte l'erreur.
String dataStructureName2	Nom (code) de la structure de données #2 sur laquelle porte l'erreur.
String errorCode	Code de l'erreur.
String errorDescriptionLineNumber	Numéro de la ligne de description de l'erreur.
String errorLabel	Libellé de l'erreur.
int errorNudoss	Clé technique (numéro du dossier) du dossier à l'origine de l'erreur.
int errorNulign	Clé technique (numéro de ligne) de l'occurrence d'information à l'origine de l'erreur.
String errorZone1	Zone libre 1 utilisée pour afficher des informations supplémentaires sur l'erreur.
String errorZone2	Zone libre 2 utilisée pour afficher des informations supplémentaires sur l'erreur.

Propriété	Signification
String errorZone3	Zone libre 3 utilisée pour afficher des informations supplémentaires sur l'erreur.
String errorZone4	Zone libre 4 utilisée pour afficher des informations supplémentaires sur l'erreur.
String errorZone5	Zone libre 5 utilisée pour afficher des informations supplémentaires sur l'erreur.
String externalLabel	Libellé externe de l'erreur.
String freeZone	Zone libre utilisée pour passer des informations supplémentaires sur l'erreur.
String itemName	Nom de la rubrique sur laquelle porte l'erreur.
String itemName2	Nom de la rubrique #2 sur laquelle porte l'erreur.
int originatorNudoss	Clé technique (numéro de dossier) du dossier à l'origine de l'erreur.
char packageCode	Code "package" associé à l'erreur.
String procedure	Nom du traitement spécifique à l'origine de l'erreur.
String procedureGroup	Nom du groupe de traitements auquel appartient le traitement spécifique à l'origine de l'erreur.
String technicalArea	Zone libre utilisée afin de passer des informations supplémentaires sur l'information.
char typeZone	Identifiant du type d'erreur. Cette propriété peut prendre trois valeurs énumérées sous forme de constantes de nom ERROR_TYPE_XXX au niveau de la classe HRResultUserError.Error. Les types d'erreur valides sont "erreur de contrôle dictionnaire", "erreur utilisateur" (levée par un traitement spécifique) ou "erreur d'un autre type".
int weight	Poids de l'erreur. Prend une valeur dans l'intervalle [1-5]. Les erreurs de poids 1 et 2 sont appelées "avertissement sans confirmation (de mise à jour)". Les erreurs de poids 3 et 4 sont appelées "avertissement avec confirmation (de mise à jour)". Les erreurs de poids 5 sont appelées "erreurs bloquantes". Voir la section "La gestion des avertissements avec confirmation".

```
// Retrieving a dossier collection and committing some pending changes (not
   detailed here)
ICommitResult result = dossierCollection.commitAllDossiers();


// Handling errors (assuming there is no parameter data sections updated
   here)
HRDossierCollection.CommitResult commitResult =
   result.getDossierCommitResult();


// Logging statistics
System.out.println("The commit returned " + commitResult.getErrorCount()
   + " errors");
System.out.println("... including " + commitResult.getWarnings().length
   + " warnings without confirmation (weight 1 & 2)");
System.out.println("... including " + commitResult.getConfirmations().length
   + " warnings with confirmation (weight 3 & 4)");
System.out.println("and including " + commitResult.getErrors().length
   + " errors (weight 5)");


// Looping over the errors
for(int i=0; i<commitResult.getErrorCount(); i++) {
   HRResultUserError.Error error = commitResult.getError(i);
   System.out.println("Found an error whose weight is " + error.weight);


   // Processing error (not detailed here)
   ...
}
```

L'exemple ci-dessus montre comment on peut itérer sur les erreurs de mise à jour retournées par le serveur HR Access et récupérer leur poids d'erreur. Cette manière de gérer les erreurs s'applique indifféremment aux erreurs relatives à la mise à jour des dossiers ou des paramètres.

Réinitialisation de la collection de dossiers

Une fois chargée avec des dossiers et/ou des paramètres, une collection de dossiers peut être réinitialisée à l'aide d'une des méthodes suivantes (de la classe HRDossierCollection).

Méthode	Rôle
void reset()	Réinitialise la collection de dossiers en rechargeant tous les dossiers chargés et en "rechargeant" (fictivement) le dossier paramètre qui lui est associé. Toutes les modifications en attente de soumission (portant sur les paramètres et/ou les dossiers) au serveur HR Access seront perdues. Un appel à cette méthode équivaut à un appel aux méthodes resetParameters() et resetDossiers().
void resetDossiers()	Réinitialise la collection de dossiers en rechargeant tous les dossiers chargés. Toutes les modifications en attente de soumission (portant sur les dossiers, pas les paramètres) au serveur HR Access seront perdues.
void resetParameters()	Réinitialise la collection de dossiers en "rechargeant" (fictivement) le dossier paramètre qui lui est associé. Toutes les modifications en attente de soumission (portant sur les paramètres, pas les dossiers) au serveur HR Access seront perdues.

Changement de mode de mise à jour

Les différents modes de mises à jour que peut utiliser une collection de dossiers ont déjà été présentés dans la section "Modes de mise à jour".

Il est possible de modifier dynamiquement le mode de mise à jour alors que l'on est en train de travailler sur des dossiers. Pour cela, il faut appeler la méthode HRDossierCollection.setUpdateMode(Update) en passant en paramètre l'une des trois constantes Update.NORMAL, Update.NO_REPLY ou Update.SIMULATION.

La bascule entre les modes NORMAL et SIMULATION peut être effectuée sans restriction. Il est possible, à des fins de vérification, de simuler le résultat d'une mise à jour en basculant la collection de dossiers du mode NORMAL au mode SIMULATION puis de repasser en mode NORMAL si la simulation a montré qu'aucune erreur de mise à jour n'avait été levée.

Même s'il est techniquement possible de basculer du mode NORMAL au mode NO_REPLY dynamiquement, le fait est qu'une telle éventualité n'a aucune justification en termes de "use case".

Le mode NO_REPLY est un mode dégradé du mode NORMAL dans lequel les objets représentant les dossiers manipulés ne sont pas resynchronisés avec leur version en base de données après soumission au serveur HR Access. Pour cette raison, une fois qu'un dossier a été modifié en mode NO_REPLY, sa représentation objet se trouve déphasée avec la base de données. En particulier, l'horodatage de dernière modification sur lequel se base le serveur HR Access pour détecter les modifications concurrentes de données (optimistic locking) n'est pas remis à jour. En conséquence, une seconde modification (et soumission) du dossier au serveur lèvera une erreur pour cause de mise à jour concurrente. C'est ce qui explique pourquoi l'utilisation du mode NO_REPLY n'est pas compatible avec les autres modes de mise à jour.

La méthode HRDossierCollection.getUpdateMode() permet de déterminer le mode courant de mise à jour des dossiers.

La gestion des BLOBs

Définition

Jusqu'à présent il a été question de manipuler les données de dossiers HR Access à travers les notions d'information, d'occurrence d'information et de rubrique. L'API permet de manipuler également des données de type BLOB (Binary Large OBject).

Un **BLOB** est une donnée binaire qui peut représenter n'importe quoi : un fichier texte, une image, un fichier son, un document PDF, etc.

Etant donné qu'un BLOB occupe potentiellement beaucoup d'espace de stockage, l'API ne lit pas les BLOBs par défaut : il faut lui dire de le faire explicitement. Comme les dossiers, les BLOBs peuvent être chargés de deux manières : de manière différée (lazy) ou préemptive (eager). En mode différé, le contenu du BLOB est chargé lorsque celui-ci est demandé. En mode préemptif, les BLOBs sont chargés en même temps que les données des dossiers.

Rubriques de rôle BLOB

L'application Design Center permet de configurer des rubriques en leur assignant un "rôle BLOB". Ces rubriques ont alors un format "Alphanumérique avec conversion des minuscules" sur une longueur de 1 (pseudo-format X(1) en COBOL) et la rubrique ne peut prendre que deux valeurs : "0" ou "1" selon qu'il existe ou pas un BLOB associé à la rubrique. Ainsi, il suffit d'interroger la valeur de la rubrique de rôle BLOB pour déterminer si un BLOB lui est associé, ce qui évite d'aller rechercher le BLOB en table pour déterminer sa présence.

HR Access permet de définir autant de rubriques de rôle BLOB que nécessaire sur une information.

Stockage des BLOBs

Physiquement, les BLOBs ne sont pas stockés dans la table de l'information de rattachement : ils sont sauvegardés dans deux tables techniques de nom BX10 et BX20.

La table BX10 contient la description des BLOBs ; c'est pourquoi pour un BLOB donné, il n'existe qu'un et un seul enregistrement en table BX10. La table BX20 contient le contenu du BLOB, c'est-à-dire la donnée elle-même ; c'est pourquoi pour un BLOB, on trouve potentiellement plusieurs enregistrements BX20. Ces enregistrements sont obtenus en gzippant le BLOB, en l'encodant en base 64 et en le découpant en tranches de 3500 caractères.

Pour un BLOB donné, il existe toujours un enregistrement en table BX10 qui décrit le BLOB. En revanche, cela n'est pas vrai concernant la table BX20 car cela dépend de l'endroit où il est stocké. Cela sera expliqué plus loin.

Configuration de la collection de dossiers

Pour pouvoir gérer des rubriques de rôle BLOB, il faut d'abord le définir au niveau de la configuration de la collection de dossiers à l'aide d'une des méthodes suivantes de la classe HRDataSourceParameters.DataSection.

Méthode	Rôle
void addBlob(String)	Configure l'information afin de pouvoir manipuler le BLOB associé à la rubrique (de rôle BLOB) de nom donné. Le BLOB sera lu à la demande, de manière différée (lazy). L'appel de cette méthode équivaut à appeler la méthode addBlob(String, boolean) avec false en second paramètre.
void addBlob(String, boolean)	Configure l'information afin de pouvoir manipuler le BLOB associé à la rubrique (de rôle BLOB) de nom donné. Le second paramètre indique si le BLOB doit être chargé de manière préemptive (eager).
void addBlobs(Map<String, Boolean>)	Configure l'information afin de pouvoir manipuler les BLOBs associés aux rubriques (de rôle BLOB) de noms donnés. La clé de la Map en paramètre représente le nom de la rubrique de rôle BLOB tandis que la valeur associée de type Boolean indique si le BLOB doit être chargé de manière préemptive (eager). Le paramétrage de la Map vient s'ajouter au paramétrage courant de l'information.
Set<String> getBlobItems()	Retourne le nom des rubriques (de rôle BLOB) qui ont été configurées.
boolean isEagerBlob(String)	Indique si la rubrique (de rôle BLOB) de nom donné est configurée pour charger le BLOB de manière préemptive (eager).
void setBlobs(Map<String, Boolean>)	Configure l'information afin de pouvoir manipuler les BLOBs associés aux rubriques (de rôle BLOB) de noms donnés. La clé de la Map en paramètre représente le nom de la rubrique de rôle BLOB tandis que la valeur associée de type Boolean indique si le BLOB doit être chargé de manière préemptive (eager). Le paramétrage de la Map vient se substituer au paramétrage courant de l'information.

L'exemple suivant montre comment configurer une collection de dossiers afin de pouvoir manipuler le BLOB associé à la rubrique de rôle BLOB de nom ZY00 BLOB01.

```
// Creating a configuration to handle the BLOB item ZY00 BLOB01

HRDossierCollectionParameters parameters = new
   HRDossierCollectionParameters();

parameters.setType(HRDossierCollectionParameters.TYPE_NORMAL);

parameters.setProcessName(“FS001”);

parameters.setDataStructureName(“ZY”);


// Configuring the data section ZY00 so as to eagerly fetch the BLOB data

HRDataSourceParameters.DataSection dataSectionZY00 = new
   HRDataSourceParameters.DataSection(“00”);

dataSectionZY00.addBlob(“BLOB01”, true);


parameters.addDataSection(dataSectionZY00);
```

Propriétés d'un BLOB

Le concept de BLOB est représenté par l'interface com.hraccess.openhr.blob.IHRBlob. C'est la classe HROccur qui fait office de "fabrique de BLOBs" et permet de lire, créer et supprimer les BLOBs

Clé d'un BLOB

Un BLOB étant porté par une rubrique d'une occurrence d'une information d'une structure de données d'un dossier, il est identifié de manière unique par une clé composée du quintuplet des valeurs suivantes :

- Le numéro du dossier auquel est rattaché le BLOB
- La structure de données du dossier auquel est rattaché le BLOB
- L'information du dossier auquel est rattaché le BLOB
- La rubrique à laquelle est rattaché le BLOB
- Le numéro de ligne de l'occurrence d'information à laquelle est rattaché le BLOB

La clé d'un BLOB est représentée par la classe IHRBlob.Key. Une clé de BLOB est "immuable", c'est-à-dire non modifiable.

Propriété "storage"

Le BLOB possède une propriété storage, que l'on pourrait traduire par "localisation physique" qui ne peut prendre que deux valeurs : Storage.DATABASE ou Storage.ARCHIVE_VOLUME (la classe Storage est une énumération de type "sûre"). Cette propriété indique où est physiquement sauvegardé le BLOB. Dans la majorité des cas, le BLOB est stocké en base de données (en table BX10 pour la description et en table BX20 pour le contenu BLOB) et la propriété est alimentée à la valeur Storage.DATABASE. Cependant HR Access permet également de **lire** (pas de modifier !) des BLOBs stockés dans un volume d'archivage (par exemple un bulletin de salarié généré par le serveur HRD Query et archivé dans un volume d'archivage). Dans ce cas, la propriété storage est alimentée à Storage.ARCHIVE_VOLUME : la description du BLOB est alors contenue en table BX10 mais le contenu du BLOB est présent dans un volume d'archivage (pas en table BX20).

Un BLOB de type "DATABASE" peut être lu et modifié alors qu'un BLOB de type "ARCHIVE_VOLUME" ne peut être que lu. Autre particularité, si le BLOB est de type "DATABASE", alors il possède une propriété filename qui indique le nom de fichier associé au BLOB (c'est généralement le nom du fichier à partir duquel le BLOB a été chargé). En revanche, si le BLOB est de type "ARCHIVE_VOLUME", le BLOB possède une propriété URL qui désigne l'URL d'accès du BLOB dans le volume d'archivage de la forme "archive://ARCH_VOL_NAME/dir1/dir2/dir3/file.extension". Les propriétés filename et URL d'un BLOB sont mutuellement exclusives puisque la propriété storage ne peut prendre que l'une des deux valeurs autorisées. Tenter de lire la propriété URL sur un BLOB de type "DATABASE" n'a pas de sens et lèvera une exception (et vice versa).

Propriété "thème"

Un BLOB possède une propriété thème qui fournit une information supplémentaire sur le type de la donnée BLOB. Le thème est une chaîne alphanumérique libre de 8 caractères au plus. Exemple de thèmes : "IMAGE", "WORDPROC", etc.

Horodatage de dernière modification

Enfin un BLOB possède un horodatage de dernière modification qui permet au serveur HR Access de détecter les mises à jour concurrentes (optimistic locking). Cet horodatage est alimenté par défaut à "0001-01-01-00.00.00" pour un BLOB ayant été créé par l'API mais n'ayant pas encore été sauvegardé en base de données puis prend une valeur "significative" dès lors que le BLOB a été sauvegardé en base de données.

Le tableau suivant liste les méthodes utiles de l'interface IHRBlob.

Méthode	Rôle
boolean exists()	Indique si le BLOB existe dans la base de données HR Access, c'est-à-dire s'il a déjà été sauvegardé. Pour un BLOB qui vient d'être créé (en mémoire), retourne false. Dès que la création du BLOB a été soumise au serveur HR Access, la méthode retourne true.
InputStream getContent()	Retourne un java.io.InputStream pour lire le contenu du BLOB. Le code client doit s'assurer de fermer proprement l'InputStream après utilisation.
IHRBlob.Description getDescription()	Retourne la description de ce BLOB sous forme de IHRBlob.Description. La description reprend les informations de l'enregistrement BX10 relatif au BLOB.
String getFilename()	Retourne le nom de fichier associé au BLOB. Cette propriété n'est disponible que si le BLOB est de type "DATABASE". Tenter de lire cette propriété pour un BLOB d'un autre type lève une exception.
IHRBlob.Key getKey()	Retourne la clé identifiant le BLOB sous forme de IHRBlob.Key.
Timestamp getLastUpdateTimestamp()	Retourne l'horodatage de dernière modification du BLOB. Cet horodatage permet au serveur HR Access de détecter les modifications concurrentes sur le BLOB (optimistic locking).

Méthode	Rôle
Storage getStorage()	Retourne le "lieu de stockage" du BLOB. Les valeurs valides retournées par cette méthode sont énumérées sous forme de constante au niveau de la classe Storage (safe type enumeration). Les types de stockage valides sont : "base de données" (DATABASE) ou "volume d'archivage" (ARCHIVE_VOLUME). Si le type de stockage est "base de données", alors le BLOB peut être lu et modifié. Si le type de stockage est "volume d'archivage", alors le BLOB ne peut être que lu.
String getTheme()	Retourne le thème associé au BLOB. Le thème est une chaîne alphanumérique libre de 8 caractères au plus.
String getURL()	Retourne l'URL associée au BLOB. Cette URL désigne l'URL de référencement du BLOB dans son volume d'archivage. Cette propriété n'est disponible que si le BLOB est de type "ARCHIVE_VOLUME". Exemple d'URL : "archive://ARCH_VOL_NAME/dir1/dir2/file.ext"
boolean isContentAvailable()	Indique si le contenu du BLOB a été lu depuis le serveur HR Access. Retourne toujours true pour un BLOB chargé de manière préemptive. Pour un BLOB chargé à la volée, retourne true uniquement si le contenu du BLOB a été chargé depuis le serveur HR Access.
void setContent(File)	Définit le contenu du BLOB à partir du fichier donné.
void setContent(InputStream)	Définit le contenu du BLOB à partir de l'InputStream donné.
void setFilename(String)	Définit le nom du fichier associé au BLOB. Cette propriété n'est disponible que si le BLOB est de type "DATABASE". Tenter de lire cette propriété pour un BLOB d'un autre type lève une exception.
void setTheme(String)	Définit le thème associé au BLOB. Le thème est une chaîne alphanumérique libre de 8 caractères au plus.
int writeTo(File)	Copie le contenu du BLOB vers le fichier donné et retourne le nombre d'octets écrits.

Méthode	Rôle
int writeTo(OutputStream)	Copie le contenu du BLOB vers le java.io.OutputStream donné et retourne le nombre d'octets écrits. Le code client doit s'assurer de fermer proprement l'OutputStream après utilisation.

La classe HROccur fournit des méthodes dédiées pour récupérer, créer ou supprimer un BLOB.

Méthode	Rôle
IHRBlob createBlob(String)	Crée et retourne un nouveau BLOB associé à la rubrique (de rôle BLOB) de nom donné. La méthode lève une exception s'il existe déjà un BLOB associé à la rubrique nommée ou si le nom de rubrique est invalide.
bolean deleteBlob(String)	Supprime le BLOB associé à la rubrique de nom donné et indique si l'opération a réussi. La méthode lève une exception s'il n'existe pas de BLOB associé à la rubrique nommée ou si le nom de rubrique est invalide.
IHRBlob getBlob(String)	Retourne le BLOB associé à la rubrique de nom donné.
boolean hasBlob(String)	Indique s'il existe un BLOB associé à la rubrique de nom donné.

Récupération et modification de BLOB

Pour récupérer un BLOB, il faut utiliser la méthode HROccur.getBlob(String).

```
// Retrieving an already loaded occurrence of data section ZY00 (not detailed
   here)
HRoccur occurrenceZY00 = getOccurrenceOfDataSectionZY00();


// Retrieving an existing BLOB mapped to item ZY00 BLOB01
if (occurrenceZY00.hasBlob("BLOB01")) {
    IHRBlob blob = occurrenceZY00.getBlob("BLOB01");


    System.out.println("The BLOB theme is <" + blob.getTheme() + ">");
    System.out.println("The BLOB was last modified on <" +
    blob.getLastUpdateTimestamp() + ">");
    System.out.println("The BLOB storage is <" + blob.getStorage() + ">");


    if (Storage.DATABASE.equals(blob.getStorage())) {
```

```
    // BLOB stored in the HR Access data base -> property filename is
    available
    System.out.println("The BLOB filename is <" + blob.getFilename() +
    ">");

    // The BLOB can be modified since it's stored in the HR Access data base
    blob.setContent(new File("C:\\my-cv.doc"));
    blob.setFilename("cv.doc");
    blob.setTheme("WORDPROC");

    // Committing the pending changes to the HR Access server (not detailed
    here)
    ...
    } else {
    // BLOB stored in an archive volume -> property URL is available
    System.out.println("The BLOB URL is <" + blob.getURL() + ">");
    }
}
```

Création de BLOB

Pour créer un BLOB, il faut utiliser la méthode HROccur.createBlob(String) puis alimenter les propriétés obligatoires du BLOB.

Les propriétés obligatoires sont celles pour lesquelles il existe une méthode de type setter au niveau de l'interface IHRBlob, à savoir : content (le contenu du BLOB), filename (le nom du fichier associé au BLOB) et theme (le thème associé au BLOB).

```
// Retrieving an already loaded occurrence of data section ZY00 (not detailed
   here)
HRoccur occurrenceZY00 = getOccurrenceOfDataSectionZY00();

// Creating a new BLOB mapped to item ZY00 BLOB01
IHRBlob newBlob = occurrenceZY00.createBlob("BLOB01");

// Populating the BLOB's mandatory properties
newBlob.setContent(new File("C:\\cv.doc"));
newBlob.setFilename("cv.doc");
newBlob.setTheme("WORDPROC");

// Committing the pending changes to the HR Access server (not detailed here)
...
```

Si l'on tente de créer un BLOB sans alimenter toutes les propriétés utiles, une exception sera levée lors de la soumission des modifications au serveur HR Access.

Lors du rattachement d'un BLOB à une rubrique de rôle BLOB, la valeur de cette rubrique passe de "0" (Aucun BLOB) à "1" (BLOB présent).

Suppression de BLOB

La suppression d'un BLOB s'effectue en appelant la méthode HROccur.deleteBlob(String).

```
// Retrieving an already loaded occurrence of data section ZY00 (not detailed here)
HRoccur occurrenceZY00 = getOccurrenceOfDataSectionZY00();


// Deleting an existing BLOB mapped to item ZY00 BLOB01
if (occurrenceZY00.hasBlob("BLOB01")) {
    if (occurrenceZY00.deleteBlob("BLOB01")) {
    System.out.println("BLOB deletion succeeded");
    } else {
    System.out.println("BLOB deletion failed");
    }

    // Committing the pending changes to the HR Access server (not detailed here)
    ...
}
```

Lors de la suppression d'un BLOB d'une rubrique de rôle BLOB, la valeur de cette rubrique passe de "1" (BLOB présent) à "0" (Aucun BLOB).

Taille limite de BLOB

L'API limite la taille des BLOBs qu'il est possible de sauvegarder sur le serveur HR Access. Cependant cette limite ne correspond pas à la taille maximale du BLOB que l'on manipule mais à la taille maximale (en caractères) du BLOB une fois gzippé et encodé en base 64, c'est-à-dire la taille du BLOB en base de données. C'est pourquoi il n'est pas possible de prédire si un BLOB de taille donnée dépassera cette limite ou non, cela dépend étroitement de la nature du BLOB : un BLOB au format texte se compresse beaucoup plus qu'une image. De plus, les BLOBs étant encodés en base 64, cette limite est exprimée en nombre de caractères, pas d'octets.

Par défaut, l'API OpenHR impose une taille maximale de 5 Méga caractères (5 x 1024 x 1024 = 5 242 880) pour un BLOB gzippé et encodé en base 64. Il s'agit d'une limite imposée par l'API OpenHR, pas par le serveur HR Access.

Si cette limite devait se révéler trop basse, il est possible de la modifier à l'aide de la propriété système "com.hraccess.openhr.maximum_encoded_blob_size". La valeur de la propriété doit représenter une taille limite exprimée en nombre de caractères.

L'accès bas niveau aux données

L'API de gestion de dossiers permet de manipuler une représentation objet d'un dossier HR Access en faisant un accès de haut niveau aux données en base. Il est possible aussi d'accéder de manière bas niveau aux données de la base HR Access à l'aide d'une technique nommée "extraction de données" comparable à JDBC. La différence est qu'ici, la base de données n'est pas accédée directement à travers un driver JDBC mais est accédée à travers le programme COBOL BNP. Cet accès ne peut se faire qu'en lecture, pas en écriture, à l'aide d'une requête SQL qui sera interprétée par un programme COBOL (BHS) afin de vérifier sa syntaxe et de contrôler que les tables accédées sont bien autorisées pour le rôle utilisé (s'il y a lieu).

Tables accédées

L'extraction de données permet de lire deux types de table :

- Les tables techniques
- Les tables applicatives

Les tables applicatives sont faciles à repérer car leur nom commence par X, Y ou Z (exemple : ZY00, XW00). La structure de ces tables est définie à l'aide de Design Center (Objets Modèles de données) et peut être modifiée en fonction des besoins fonctionnels.

Les tables techniques sont les autres tables de la base de données. Leur structure est fixe, non modifiable et livrée en standard avec le produit.

Confidentialité

Le contrôle d'accès aux données appelé "confidentialité" et mis en œuvre par le serveur HR Access à l'aide de rôles ne porte que sur les tables applicatives, pas les tables techniques. C'est pourquoi les tables techniques peuvent être lues sans utilisateur, c'est-à-dire en disposant juste d'une session OpenHR : aucun contrôle particulier (aux exceptions près) n'est assuré lors de leur lecture. Sur le principe de "qui peut le plus peut le moins", si on peut lire sans utilisateur une table technique, alors un utilisateur peut aussi lire cette table.

Les tables applicatives sont accédées au titre d'un rôle. Ceux-ci sont portés par les utilisateurs. Par conséquent, pour lire une table applicative, il faut disposer d'un utilisateur et d'un rôle. Le tableau suivant résume la situation.

<table>
<tr><th></th><th>Lecture de table technique ?</th><th>Lecture de table applicative ?</th></tr>
<tr><td>Sans utilisateur</td><td rowspan="2">Autorisé
(Pas de rôle requis)</td><td>Interdit
(Absence de rôle)</td></tr>
<tr><td>Avec utilisateur</td><td>Autorisé
(Rôle requis)</td></tr>
</table>

L'API fournit deux classes (HRExtractionSource et HRTechnicalExtractionSource du package com.hraccess.openhr.beans) afin de lire (respectivement) les données applicatives et techniques. Leur utilisation sera détaillée plus loin.

La technique d'extraction de données doit être privilégiée à la gestion des dossiers lorsque les conditions suivantes sont remplies :

- Les données lues en table n'ont pas besoin d'être modifiées.
- Les données lues en table peuvent être récupérées relativement facilement soit parce qu'elles sont stockées dans une table unique, soit parce que différentes tables peuvent être jointes simplement.
- La performance maximale est recherchée.

Cette technique ne peut être utilisée dans les cas suivants :

- Pour lire le contenu de rubriques ou d'informations virtuelles : le programme COBOL invoqué lors de la lecture des données applicatives ne permet pas de calculer les données virtuelles.
- Les données ont besoin d'être remises à jour après lecture.

La grammaire SQL supportée par le serveur HR Access permet d'utiliser le système de mots-clés présenté plus haut afin de maximiser la portabilité du SQL (voir la section "Filtre SQL").

Extraction de données techniques

La classe HRTechnicalExtractionSource du package com.hraccess.openhr.beans permet de lire des données techniques. Le tableau suivant liste les méthodes disponibles sur cette classe.

Méthode	Rôle
int getMaxRowCount()	Retourne le nombre maximal d'enregistrements (de lignes de base de données) que l'extraction de données peut retourner. Par défaut, cette valeur est de 99999. La valeur retournée est située dans l'intervalle [1-99999].
IHRSession getSession()	Retourne la session OpenHR utilisée pour extraire les données de la base HR Access.
String getSQLExtraction()	Retourne la requête SQL utilisée pour extraire les données de la base HR Access. La grammaire SQL de cette requête est celle supportée par le programme COBOL BHS et qui inclut le système de mots-clés SQL (cf. "Le filtre SQL").
boolean isMoreRowsAvailable()	Indique si l'extraction de données n'a retourné qu'une partie des enregistrements en table (et s'il existe d'autres enregistrements disponibles au niveau de la base de données).
void setMaxRowCount(int)	Définit le nombre maximal d'enregistrements (de lignes de base de données) que l'extraction de données peut retourner. Par défaut, cette valeur est de 99999. La valeur retournée est située dans l'intervalle [1-99999].
void setSQLExtraction (String)	Définit la requête SQL utilisée pour extraire les données de la base HR Access. La grammaire SQL de cette requête est celle supportée par le programme COBOL BHS et qui inclut le système de mots-clés SQL (voir la section "Filtre SQL").
TDataNode getDataNode()	Retourne le résultat de l'extraction de données sous forme de TDataNode. La classe TDataNode est comparable à un java.sql.ResultSet (cf. exemple plus bas).
boolean isActive()	Indique si l'extraction de données a été activée, c'est-à-dire si la requête SQL a été exécutée et si les données lues sont disponibles.

Méthode	Rôle
void setActive(boolean)	Active ou désactive l'extraction de données. Lors de l'activation, la requête SQL paramétrée est exécutée par le serveur HR Access et le résultat retourné au client OpenHR.

L'exemple suivant montre comment lister le contenu de la table technique DI60 (contenant la définition des rubriques) à l'aide de la classe HRTechnicalExtractionSource. Reportez-vous à la javadoc de la classe com.hraccess.datasource.TDataNode pour plus d'explications sur le rôle de cette classe.

```
// Retrieving a connected session (not detailed here)
IHRSession session = getSession();

// Extracting some data from technical table DI60 (Item definitions)
HRTechnicalExtractionSource extractionSource = new
    HRTechnicalExtractionSource(session);

// Extracting up to 100 rows (upper bound is 99 999 rows)
extractionSource.setMaxRowCount(100);

// Setting SQL statement to perform
extractionSource.setSQLExtraction("SELECT * FROM DI60");

// Connecting extraction source
extractionSource.setActive(true);

try {
    // Retrieving the result set as a TDataNode
    TDataNode node = extractionSource.getDataNode();

    // Looping over the returned rows
    if (node.first()) {
    int row=0;
    do {
            System.out.println("Dumping record #" + (row+1));

            // Looping over the columns
            for (int i=0; i<node.getColumnCount(); i++) {
                Column column = node.getColumn(i);
```

```
                System.out.println("Column <" + column.getName() + "> =
    <" + node.getValue(i);
            }
            row++;
        } while (node.next());

        System.out.println("Are there any more records available ? " +
        extractionSource.isMoreRowsAvailable());
        }
    } finally {
        if ((extractionSource != null) && extractionSource.isActive()) {
        // Disconnecting extraction source
        extractionSource.setActive(false);
        }
    }
```

Commentons cet exemple.

L'exemple instancie la classe HRTechnicalExtractionSource en lui passant la référence à une IHRSession car c'est par la session que la requête SQL d'extraction de données sera adressée au serveur HR Access.

Le nombre maximal de lignes, c'est-à-dire d'enregistrements, que l'extraction doit retourner est positionné à 100 (par défaut la valeur est de 99 999 lignes).

La requête SQL utilisée extrait toutes les colonnes de la table DI60.

Attention, cet ordre SQL ne sera pas interprété directement par la base de données : il sera d'abord parsé par le programme BHS afin de s'assurer de sa validité en termes de syntaxe. En particulier, le radical des tables défini au niveau de la plate-forme physique HR Access sera inséré dans l'ordre SQL, vous n'avez donc pas à l'écrire vous-même.

De même, si la requête SQL comporte des mots-clés (<QB>, <QE>, <DAYDATE>, etc.) afin de maximiser sa portabilité, ceux-ci seront remplacés dynamiquement dans l'ordre SQL par le programme BHS. Au final, un ordre simple comme "SELECT * FROM DI60" sera transformé en "SELECT * FROM HR.DI60" si "HR" désigne le radical de la table DI60.

Puis l'exemple active l'extraction de données. A ce moment-là, une requête d'extraction part au serveur HR Access via la session OpenHR. La requête est parsée, validée, modifiée et exécutée. Enfin, les données des enregistrements lus sont retournées au client OpenHR, lequel mémorise celles-ci localement.

Pour lire les données extraites, il faut récupérer le nœud de données sous forme de TDataNode. Un TDataNode est comparable à un java.sql.ResultSet : c'est une liste que l'on peut parcourir en avant ou en arrière et qui possède un indice courant. L'indice courant désigne l'enregistrement courant sur lequel est situé le nœud de données. Cet enregistrement semblable à une java.util.Map stocke en clé le nom de la colonne lue et en valeur la valeur de cette colonne. Le type de cette valeur dépend de la définition de cette colonne. Contrairement au java.sql.ResultSet, il faut itérer sur les enregistrements de la manière suivante :

```
TDataNode node = ...;


// Looping over the data node
if (node.first()) {
   do (
   ...
   } while (node.next());
}
```

Il faut donc se positionner sur le premier enregistrement à l'aide de la méthode TDataNode.first() afin de déterminer si au moins un enregistrement a été lu. Puis, il faut traiter celui-ci et continuer tant qu'il y a un enregistrement suivant. L'exemple boucle sur les méta-données du nœud afin de traiter les colonnes lues et afficher leur nom ainsi que la valeur de colonne lue.

Puis on indique si toutes les lignes disponibles au niveau de la base de données ont été lues ou s'il en reste d'autres de disponible.

Enfin, on déconnecte l'extraction de données dans un bloc finally.

Si vous tentez de lire une table applicative à l'aide de la classe HRTechnicalExtractionSource, le programme COBOL BHS détectera que cette table est de type applicative et empêchera l'exécution de cet ordre en levant une erreur.

Extraction de données applicatives

La classe HRExtractionSource du package com.hraccess.openhr.beans permet de lire des données techniques et applicatives. Les méthodes disponibles sur cette classe sont les mêmes que celles de la classe HRTechnicalExtractionSource (cf. plus haut).

L'exemple suivant montre comment lister le contenu de la table applicative ZY00 (contenant les dossiers de salarié) à l'aide de la classe HRExtractionSource. Reportez-vous à la javadoc de la classe com.hraccess.datasource.TDataNode pour plus d'explications sur le rôle de cette classe.

```
// Retrieving a connected user (not detailed here)
IHRUser user = getUser();


// Retrieving a role to read the records (not detailed here)
IHRRole role = getRole();


// Extracting some data from applicative table ZY00 (Employee dossiers)
HRExtractionSource extractionSource = new
    HRExtractionSource(user.getMainConversation(), role);


// Extracting up to 100 rows (upper bound is 99 999 rows)
extractionSource.setMaxRowCount(100);


// Setting SQL statement to perform
extractionSource.setSQLExtraction("SELECT * FROM ZY00 WHERE
    SOCCLE=<QB>HRA<QE>");


// Connecting extraction source
extractionSource.setActive(true);
```

```
try {
   // Retrieving the result set as a TDataNode
   TDataNode node = extractionSource.getDataNode();

   // Looping over the returned rows
   if (node.first()) {
   int row=0;
   do {
         System.out.println(“Dumping record #” + (row+1));

         // Looping over the columns
         for (int i=0; i<node.getColumnCount(); i++) {
               Column column = node.getColumn(i);
               System.out.println(“Column <” + column.getName() + “> = <”
   + node.getValue(i);
         }
         row++;
   } while (node.next());

   System.out.println(“Are there any more records available ? ” +
   extractionSource.isMoreRowsAvailable());
   }
} finally {
   if ((extractionSource != null) && extractionSource.isActive()) {
   // Disconnecting extraction source
   extractionSource.setActive(false);
   }
}
```

Cet exemple est pratiquement identique au précédent. Commentons les différences.

Pour instancier un HRExtractionSource, il faut fournir une conversation et un rôle. La conversation permet de communiquer avec le serveur HR Access au titre de l'utilisateur alors que le rôle définit la confidentialité utilisée pour lire les données.

L'ordre SQL utilisé est « SELECT NUDOSS FROM ZY00 WHERE SOCCLE=<QB>HRA<QE> ». Il utilise les mots-clés <QB> et <QE> afin de représenter les délimiteurs de littéraux de la base de données de manière portable. Le programme BHS transformera cet ordre en « SELECT NUDOSS FROM HR.ZY00 WHERE SOCCLE="HRA" » après avoir inséré le radical de la base de données et remplacé les délimiteurs de littéraux.

L'exploitation du résultat sous forme de TDataNode est identique.

Si vous tentez de lire une table applicative à l'aide de la classe HRExtractionSource alors que le rôle utilisé ne le permet pas, le programme COBOL BHS modifiera l'ordre SQL afin de rendre la requête compatible avec la confidentialité définie. Si la confidentialité consiste à restreindre les dossiers de salarié à une certaine société "ABC", par exemple, alors l'ordre sera modifié par l'utilisation d'une clause WHERE de la manière suivante : « SELECT * FROM HR.ZY00 WHERE SOCCLE="HRA" AND SOCCLE="ABC" ». Si le rôle ne permet pas de lire la structure de données "ZY" ou l'information "ZY00", alors l'ordre sera modifié de la manière suivante : « SELECT * FROM HR.ZY00 WHERE SOCCLE="HRA" AND 1=2 ».

Si vous tentez de lire une table technique à l'aide de la classe HRExtractionSource, il ne se passe rien. En effet, les tables techniques sont lisibles par les utilisateurs, donc par la classe HRExtractionSource.

La classe HRExtractionTemplate

La classe com.hraccess.openhr.beans.HRExtractionTemplate est une classe utilitaire qui facilite l'utilisation des classes HRTechnicalExtractionSource et HRExtractionSource en prenant en charge le cycle de vie de l'objet et éventuellement l'itération sur le résultat retourné. Cette classe s'inspire de la classe JdbcTemplate du framework Spring. Cette classe permet d'extraire des données techniques ou applicatives selon la manière dont elle a été instanciée.

Les méthodes utiles de cette classe sont listées ci-dessous.

Méthode	Rôle
IHRConversation getConversation()	Retourne la conversation sous-jacente utilisée par le HRExtractionTemplate (s'il y a lieu) pour extraire les données.
int getMaxRowCount()	Retourne le nombre maximal d'enregistrements (de lignes de base de données) que l'extraction de données peut retourner. Par défaut, cette valeur est de 99999. La valeur retournée est située dans l'intervalle [1-99999].
IHRRole getRole()	Retourne le rôle sous-jacent utilisé par le HRExtractionTemplate (s'il y a lieu) pour extraire les données.
IHRSession getSession()	Retourne la session sous-jacente utilisée par le HRExtractionTemplate (s'il y a lieu) pour extraire les données.
Object query(String, IDataNodeExtractor)	Exécute l'ordre SQL donné et délègue le traitement des données retournées sous forme de TDataNode à l'instance de IDataNodeExtractor fournie. Enfin retourne le résultat de l'extraction des données sous forme de Object.
void query(String, IRecordCallbackHandler)	Exécute l'ordre SQL donné, itère sur les enregistrements et délègue le traitement de chaque enregistrement (donné sous forme de TDataNode) à l'instance de IRecordCallbackHandler fournie.
List query(String, IRecordMapper)	Exécute l'ordre SQL donné, itère sur les enregistrements et invoque l'instance de IRecordMapper fournie afin de mapper l'enregistrement sous forme d'objet. Enfin retourne une List contenant les enregistrements mappés.

void setMaxRowCount(int)	Définit le nombre maximal d'enregistrements (de lignes de base de données) que l'extraction de données peut retourner. Par défaut, cette valeur est de 99999. La valeur retournée est située dans l'intervalle [1-99999].

L'exemple suivant montre comment utiliser la classe HRExtractionTemplate pour extraire des données techniques.

```
// Retrieving an existing session (not detailed here)
IHRSession session = getSession();

// Creating a HRExtractionTemplate to read technical data
HRExtractionTemplate template = new HRExtractionTemplate(session);
template.setMaxRowCount(100);
Object result = template.query("SELECT * FROM DI60", new
   IDataNodeExtractor() {
   // Processing data node (not detailed here)
   ...
});

System.out.println("Extraction of data returned result <" + result + ">");
```

La manière d'utiliser la classe IDataNodeExtractor sera expliquée plus tard.

Pour extraire des données applicatives, il suffit d'instancier la classe en lui passant une conversation et un rôle.

```
// Retrieving a connected user (not detailed here)
IHRUser user = getUser();

// Retrieving a role to read the records (not detailed here)
IHRRole role = getRole();

// Creating a HRExtractionTemplate to read applicative data
HRExtractionTemplate template = new HRExtractionTemplate(conversation,
   role);
template.setMaxRowCount(100);
Object result = template.query("SELECT * FROM ZY00", new
   IDataNodeExtractor() {
   // Processing data node (not detailed here)
   ...
});
```

```
System.out.println("Extraction of data returned result <" + result + ">");
```

Les classes IDataNodeExtractor, IRecordCallbackHandler et IRecordMapper permettent de simplifier l'extraction de données (techniques et applicatives).

La classe IDataNodeExtractor doit prendre en charge l'itération sur le TDataNode du résultat retourné et retourner un Object représentant le résultat. C'est la technique qui simplifie le moins l'extraction des données car elle ne prend en charge que le cycle de vie de l'objet HRExtractionSource (ou HRTechnicalExtractionSource) sous-jacent.

```
// Retrieving an existing session (not detailed here)
IHRSession session = getSession();


// Creating a HRExtractionTemplate to read technical data
HRExtractionTemplate template = new HRExtractionTemplate(session);
template.setMaxRowCount(100);
List result = (List) template.query("SELECT * FROM DI60", new
   IDataNodeExtractor() {
   public Object extractData(TDataNode node) {
   List records = new ArrayList();


   // Looping over results
   if (node.first()) {
         do {
               // Processing DI60 record (dummy processing)
               records.add(new Object());
         } while (node.next());
   }


   return records;
   }
});


System.out.println("Extraction of data returned result <" + result + ">");
```

La classe IRecordCallbackHandler doit prendre en charge le traitement de chaque enregistrement. Cette technique simplifie beaucoup l'extraction des données car le cycle de vie de l'objet HRExtractionSource (ou HRTechnicalExtractionSource) sous-jacent et l'itération sur le TDataNode représentant le résultat sont pris en charge par la classe HRExtractionTemplate.

```
// Retrieving an existing session (not detailed here)
IHRSession session = getSession();
```

```
// Creating a List to store the extraction result
List records = new ArrayList();

// Creating a HRExtractionTemplate to read technical data
HRExtractionTemplate template = new HRExtractionTemplate(session);
template.setMaxRowCount(100);
template.query("SELECT * FROM DI60", new IRecordCallBackHandler() {
   public void processRecord(TDataNode node) {
   // Processing DI60 record (dummy processing)
   records.add(new Object());
   }
});

System.out.println("Extraction of data returned result <" + records + ">");
```

Enfin, la classe IRecordMapper permet de convertir (mapper) un enregistrement en objet Java. Cette technique simplifie beaucoup l'extraction des données car le cycle de vie de l'objet HRExtractionSource (ou HRTechnicalExtractionSource) sous-jacent et l'itération sur le TDataNode représentant le résultat sont pris en charge par la classe HRExtractionTemplate.

```
// Retrieving an existing session (not detailed here)
IHRSession session = getSession();

// Creating a List to store the mapped records
List beans = new ArrayList();

// Creating a HRExtractionTemplate to read technical data
HRExtractionTemplate template = new HRExtractionTemplate(session);
template.setMaxRowCount(100);
template.query("SELECT * FROM DI60", new IRecordMapper() {
   public Object mapRecord(TDataNode node, int recordNumber) {
   // Converting DI60 record into a Java bean (dummy processing)
   return new Object();
   }
});

System.out.println("Extraction of data returned result <" + beans + ">");
```

La classe HRExtractionTemplate est donc plus simple à utiliser ; cependant, elle ne permet pas d'accéder à la propriété moreRowsAvailable. Si celle-ci se révèle indispensable, il faut alors utiliser les classes de base HRExtractionSource et HRTechnicalExtractionSource.

Les tables techniques sensibles

On trouve parmi les tables techniques livrées avec HR Access certaines tables dont la lecture est restreinte car elles contiennent des données sensibles. Celles-ci peuvent être de trois types :

- Des mots de passe utilisateur. C'est le cas de la table UC10 qui définit en standard les utilisateurs et contient leur mot de passe. HR Access interdit toute extraction de mot de passe (mais pas des autres données) sur cette table. Si un ordre SQL parvient au serveur HR Access en référençant la table UC10, celle-ci sera remplacée par la vue UC11 qui est identique à la table UC10 sauf qu'elle ne contient pas le mot de passe.
- Des identifiants de session virtuelle. On a vu que cet identifiant sert de token au serveur HR Access pour réutiliser la connexion d'un utilisateur connecté. Sa récupération est donc sensible. Les tables techniques référençant l'identifiant de session virtuelle sont MX10 (Sessions virtuelles), MX20 (Rôles utilisateur), MX40 (Conversations utilisateur) et LO10 (Log des événements de connexion / déconnexion). Lorsqu'un ordre SQL portant sur l'une de ces tables parvient au serveur HR Access, une erreur est levée.
- La topologie système. La topologie système décrit l'infrastructure de déploiement de l'environnement HR Access en termes de fonctions (HRa Space, serveur HRD Query, serveur HR Access, etc.) et de liens entre ces fonctions. Celle-ci est sensible car elle contient des mots de passe FTP et JDBC. La topologie système est déployée dans la table EN30. Or cette table contient d'autres données qui ne sont pas sensibles et peuvent être utiles, c'est pourquoi son accès n'est pas totalement bloqué. A la place, le programme BHS modifie l'ordre SQL de lecture de la table EN30 afin d'y ajouter une clause WHERE de filtrage sur le type de l'objet lue : la topologie système étant déployée avec un type "SI", la clause de filtrage rajoutée est de la forme « WHERE TYENTI <> "SI" ».

CHAPITRE 5

Personnalisation et sécurisation de la connexion des utilisateurs

A propos de ce chapitre

Avant de lire ce chapitre, nous vous recommandons vivement de lire le *Guide de gestion de la sécurité*. Celui-ci détaille l'ensemble des composants rentrant en jeu pour connecter un utilisateur et personnaliser sa connexion.

Le présent chapitre explique :

- Comment personnaliser la connexion d'un utilisateur via l'API OpenHR (description, credentials, etc.)
- Comment sécuriser les communications entre l'API OpenHR et le serveur OpenHR

Connexion des utilisateurs

Connexion standard

En standard, l'API OpenHR permet de connecter des utilisateurs définis via Design Center. La description et le mot de passe des utilisateurs sont stockés en table UC10 (Utilisateurs).

Pour connecter un utilisateur il suffit d'indiquer son identifiant de connexion (alias login ID) et son mot de passe. La connexion consiste alors à vérifier que le mot de passe est bien conforme à celui stocké en table UC10 et à récupérer la description de l'utilisateur ainsi que ses rôles.

Utilisateurs externes

L'API OpenHR permet aussi de connecter des utilisateurs définis de manière externe dans un annuaire LDAP ou une base de données relationnelle par exemple.

Pour cela, il faut implémenter un plug-in Java nommé Login Module et déployé au niveau du serveur OpenHR (voir le *Guide de gestion de la sécurité*). C'est le Login Module qui prend en charge les demandes de connexion émises par l'API OpenHR.

Demande de connexion avec ou sans indication

En standard, il existe un Login Module dont l'implémentation exploite la table UC10 comme décrit ci-dessus. Il est possible d'implémenter de nouveaux Login Modules afin de connecter les utilisateurs de manière spécifique. On peut alors soit remplacer l'implémentation standard de Login Module (celle qui exploite la table UC10) par l'implémentation spécifique, soit utiliser cette nouvelle implémentation en plus de l'implémentation standard. Dans ce dernier cas, il faut indiquer au serveur OpenHR, lors de la demande de connexion d'utilisateur, quelle implémentation utiliser. Pour ce faire, on fournit une indication ("hint" en anglais) lors de la demande de connexion. Pour que toutes les demandes de connexion puissent être traitées, l'une des implémentations de Login Module est marquée "implémentation par défaut" et est invoquée quand aucune indication n'est fournie explicitement par l'API OpenHR.

Il y a donc deux différents cas de figure à considérer :

- Demande de connexion d'utilisateur sans indication : c'est le Login Module marqué "par défaut" au niveau du serveur OpenHR qui traitera la demande de connexion.
- Demande de connexion d'utilisateur avec indication : l'indication sert à sélectionner l'implémentation de Login Module parmi celles déployées au niveau du serveur OpenHR et traitera la demande de connexion.

Description d'un utilisateur

Pour connecter un utilisateur, il faut fournir au serveur HR Access sa description. Celle-ci est représentée par la classe com.hraccess.openhr.security.UserDescription. Les méthodes utiles de cette classe sont listées dans le tableau ci-dessous.

Attention à ne pas confondre cette description avec celle représentée par l'interface IHRUser.Description. Cette dernière représente la description d'un utilisateur qui est déjà connecté. La description dont il est question ici correspond à la description d'un utilisateur que l'on souhaite connecter.

Méthode	Rôle
void addCredential(Credential)	Ajoute le credential donné à la liste des credentials de la description d'utilisateur.
void addRole(Role)	Ajoute le rôle donné à la liste des rôles de l'utilisateur.
void addRoles(Collection<Role>)	Ajoute les rôles donnés à la liste des rôles de l'utilisateur.
Set<Credential> getCredentials()	Retourne les credentials rattachés à la description d'utilisateur.
Set<Credential> getCredentials(Class)	Retourne les credentials de classe donnée rattachés à la description d'utilisateur.
String getLabel()	Retourne le nom de présentation (libellé) de la description d'utilisateur.
char getLanguage()	Retourne le code langue de la description d'utilisateur.
String getLoginId()	Retourne l'identifiant de connexion de la description d'utilisateur.
String getPgp()	Retourne le code PGP de la description d'utilisateur.
Set<Role> getRoles()	Retourne les rôles rattachés à la description d'utilisateur.
boolean isInternal()	Indique si la description désigne un utilisateur défini en table UC10. Propriété spécifique à l'implémentation standard de Login Module exploitant la table UC10.
boolean removeCredential(Credential)	Supprime le credential donné de la description d'utilisateur et indique en retour si la suppression a réussi.
boolean removeRole(Role)	Supprime le rôle donné de la description d'utilisateur et indique en retour si la suppression a réussi.
void reset()	Réinitialise la description d'utilisateur.

Méthode	Rôle
void setInternal(boolean)	Définit si la description désigne un utilisateur défini en table UC10. Propriété spécifique à l'implémentation standard de Login Module exploitant la table UC10.
void setLabel(String)	Définit le nom de présentation (libellé) de la description d'utilisateur.
void setLanguage(char)	Définit le code langue de la description d'utilisateur.
void setLoginId(String)	Définit l'identifiant de connexion de la description d'utilisateur.
void setPgp(String)	Définit le code PGP de la description d'utilisateur.

La description est un Java Bean qui est transféré (sérialisé) au serveur OpenHR afin d'être traité par un Login Module.

Il est primordial de comprendre qu'il n'est pas nécessaire de renseigner tous les attributs de la description lorsque l'on souhaite connecter un utilisateur. En effet, l'architecture HR Access est telle que les informations manquantes au niveau de la description peuvent être ajoutées plus tard dans la cinématique de connexion :

- Soit par le Login Module lui-même (données recherchées à partir d'un annuaire LDAP par exemple)
- Soit par un traitement spécifique du programme BCU lors de l'ouverture de session virtuelle de l'utilisateur

Ainsi, lorsque l'on connecte un utilisateur de manière standard en fournissant son identifiant de connexion et son mot de passe (UC10), on ne renseigne qu'une partie de la description. C'est le fait de marquer cette description interne (cf. propriété "internal") qui indique que les informations manquantes (langue, rôles, etc.) doivent être recherchées en table UC10 / UC15.

Credential

Le concept de **credential** désigne une information servant à authentifier un utilisateur, c'est-à-dire de prouver son identité. Un credential peut être un mot de passe, un certificat, un token, etc.

Pour définir un credential, il suffit d'implémenter l'interface com.hraccess.openhr.security.Credential qui (étend java.io.Serializable et) est une interface de marquage (sans méthode). Un credential est donc nécessairement sérialisable car il doit pouvoir être transféré au serveur OpenHR. L'API OpenHR fournit plusieurs implémentations de base de l'interface Credential :

- com.hraccess.openhr.security.PasswordCredential : implémentation standard représentant un credential de type "mot de passe".
- com.hraccess.openhr.security.NewPasswordCredential : implémentation standard représentant un credential de type "nouveau mot de passe".
- com.hraccess.openhr.security.CertificateCredential : implémentation standard représentant un credential de type "certificat".
- com.hraccess.openhr.security.TicketCredential : implémentation standard représentant un credential de type "ticket" (token).

Exemples de création manuelle de description d'utilisateur

Nous avons vu précédemment comment connecter un utilisateur à l'aide de deux méthodes IHRSession.connectUser(). Celles-ci sont des méthodes utilitaires qui permettent de connecter facilement un utilisateur dans le cas standard, en masquant la logique de création de description d'utilisateur.

L'exemple suivant montre comment on peut faire de même en créant manuellement la description de l'utilisateur.

```
// Retrieving an existing session (not detailed here)
IHRSession session = getSession();

// Connecting a UC10 user from a login ID and a password (by using the convenient method)
IHRUser user = session.connectUser("HRUSER", "SECRET");

// Connecting a UC10 user from a login ID and a password (by manually creating the user description)
IHRUser user2 = session.connectUser(new UserDescription("HRUSER", "SECRET"));

// Alternate (longer) version
UserDescription description2 = new UserDescription();
description2.setLoginId("HRUSER");
description2.addCredential(new PasswordCredential("SECRET"));

IHRUser user3 = session.connectUser(description2);
```

L'exemple suivant montre comment connecter l'utilisateur manuellement en changeant son mot de passe lors de la connexion.

```
// Retrieving an existing session (not detailed here)
IHRSession session = getSession();

// Connecting a UC10 user from a login ID, a password and a new password (by using the convenient method)
IHRUser user = session.connectUser("HRUSER", "SECRET", "TERCES");

// Connecting a UC10 user from a login ID, a password and a new password (by manually creating the user description)
IHRUser user2 = session.connectUser(new UserDescription("HRUSER", "SECRET" , "TERCES"));

// Alternate (longer) version
```

```
UserDescription description2 = new UserDescription();
description2.setLoginId(“HRUSER”);
description2.addCredential(new PasswordCredential(“SECRET”));
description2.addCredential(new NewPasswordCredential(“TERCES”));

IHRUser user3 = session.connectUser(description2);
```

Propriétés de la description

La description permet de positionner de nombreuses propriétés et parfois "plus que nécessaire". Si vus voulez connecter un utilisateur standard défini en table UC10, les seules propriétés requises sont l'identifiant de connexion et le mot de passe.

Si vous renseignez le nom de présentation (propriété "label") de la description, celle-ci sera, en standard, sérialisée au serveur OpenHR et au Login Module. Or, le Login Module étant implémenté pour gérer le cas spécial où seuls l'identifiant de connexion et le mot de passe sont renseignés, le nom de présentation ne sera pas pris en compte.

Il est pourtant tout à fait possible de créer une implémentation de Login Module qui ne contrôle pas la description envoyée par l'API OpenHR et prend en compte la description d'utilisateur telle quelle. Le cas extrême correspond à une description d'utilisateur entièrement définie par le client OpenHR et non contrôlée par le Login Module : il est alors possible de connecter des utilisateurs dont la description est complètement fournie par le client OpenHR (cas d'un Single Sign On).

Attribution des rôles à un utilisateur par le Client OpenHR

La classe com.hraccess.openhr.security.Role est un Java Bean représentant un rôle avec ses deux propriétés : modèle de rôle et valeur d'instanciation. Il ne s'agit pas d'un rôle attribué à un utilisateur connecté mais uniquement de l'identifiant (modèle, valeur) d'un rôle que l'on souhaite attribuer à un utilisateur qui se connecte. La classe UserDescription permet de rattacher dynamiquement des rôles à une description d'utilisateur et autorise un client OpenHR à définir lui-même les rôles de l'utilisateur sur le système. En utilisant la classe Role (et en supposant que l'implémentation du Login Module utilisé autorise le client OpenHR à positionner ces rôles), on peut donc complètement externaliser l'attribution des rôles d'un utilisateur.

```
// Retrieving an existing session (not detailed here)
IHRSession session = getSession();

// Connecting a user with some explicit roles
UserDescription description = new UserDescription();
description.setLoginId(“HRUSER”);
description.addRole(new Role(“EMPLOYEE”, “123456”));
description.addRole(new Role(“MANAGER”, “MAIN_ORG_UNIT”));

IHRUser user = session.connectUser(description);
```

Avec cet exemple, l'application cliente OpenHR définit elle-même les rôles de l'utilisateur à connecter.

Implémentations multiples de Login Module

Dans le cas où plusieurs implémentations de Login Module sont hébergées par le serveur OpenHR, l'application cliente OpenHR peut indiquer laquelle elle souhaite utiliser en fournissant une indication lors de la demande de connexion d'utilisateur. C'est grâce à la méthode IHRSession.connectUser(UserDescription, String) que cela est possible.

```
// Retrieving an existing session (not detailed here)
IHRSession session = getSession();


// Creating a description to connect a user
UserDescription description = new UserDescription(“HRUSER”, “SECRET”);


// Connecting the user by providing an hint to select the Login Module to process the description
IHRUser user = session.connectUser(description, “LDAP”);
```

L'exemple ci-dessus montre comment demander la connexion d'un utilisateur dont on fournit la description et une indication ("LDAP" ici) sur la manière de traiter cette description. Reportez-vous au *Guide de gestion de la sécurité* pour savoir, dans le détail, comment cette indication est utilisée par le serveur OpenHR. Cependant du côté client, il suffit de savoir qu'elle permet indiquer une implémentation de Login Module à utiliser.

Il existe une autre manière de fournir cette indication de Login Module qui se base sur le paramétrage du fichier openhr.properties. Cependant cette technique est plus impactante que la précédente car le paramétrage en question s'appliquera à toutes les demandes de connexion **pour lesquelles aucune indication n'est explicitement fournie**.

Voyons cela en détail. L'exemple précédent permet de connecter un utilisateur en passant la valeur "LDAP" comme indication de Login Module. Il faut savoir que lorsqu'on appelle la méthode IHRSession.connectUser(UserDescription) (donc sans passer d'indication), cela revient à passer une indication à null. Celle-ci sera interprétée par le serveur OpenHR de sorte à sélectionner l'implémentation "par défaut" de Login Module. Cependant, cette valeur à null est une valeur par défaut que l'on peut modifier par configuration. Ainsi, en valorisant, dans le fichier openhr.properties, la propriété "session.login_module_hint" à "LDAP", on fait en sorte que l'indication par défaut soit "LDAP" et non null lors des demandes de connexion d'utilisateur **pour lesquelles aucune indication n'est explicitement fournie**. Il faut donc prendre en compte les règles suivantes pour déterminer quelle indication sera envoyée au serveur et au final quel Login Module sera utilisé.

Règles de détermination de l'indication envoyée au serveur

1. Si, lors de la demande de connexion d'un utilisateur, une indication est fournie, alors c'est celle-ci qui est envoyée au serveur OpenHR et détermine quelle est l'implémentation de Login Module utilisée.

   a. Autrement (si aucune indication explicite n'a été fournie), c'est l'indication par défaut configurée par la propriété "session.login_module_hint" qui est utilisée.

   b. Si cette propriété est renseignée dans le fichier openhr.properties, c'est elle qui est envoyée au serveur OpenHR et détermine quelle est l'implémentation de Login Module utilisée.

Autrement (si aucune indication par défaut n'est définie), c'est le Login Module par défaut du serveur OpenHR qui est utilisé.

On voit donc que, grâce à cette propriété, on peut facilement configurer une application cliente OpenHR afin que toutes les demandes de connexion soient traitées par un autre Login Module et cela, sans avoir à modifier le code qui connecte les utilisateurs.

Méthode de connexion de base

Enfin, la classe IHRSession fournit une dernière méthode de connexion des utilisateurs (la plus complète) dont la signature est IHRSession.connectUser(UserDescription, String, Precision).

En fait, toutes les autres méthodes de connexion sont basées sur cette dernière méthode, quitte à lui passer des valeurs par défaut pour les paramètres quand aucune valeur explicite n'est disponible (précision de rôle, indication de Login Module). Le dernier paramètre de type com.hraccess.openhr.security.Precision désigne la précision (ou niveau de détails) avec lequel les rôles de l'utilisateur doivent être calculés lors de la connexion. Lorsque l'on appelle toutes les autres méthodes connectUser() de connexion d'utilisateur de la classe IHRSession, on utilise implicitement une certaine précision (Precision.LEVEL_ONE) afin de calculer le minimum d'informations sur le rôle.

Niveau de précision

Il faut savoir que les propriétés d'un rôle sont calculées par traitement spécifique (de BCR) et peuvent être coûteuses à calculer, c'est pourquoi on utilise une précision faible par défaut. Le rôle est un objet dynamique qui peut charger ses propriétés à la volée lorsque c'est nécessaire. En conséquence, résoudre un rôle avec une précision faible ne signifie pas que l'on aura moins d'informations qu'avec un rôle résolu avec une précision supérieure : la seule différence se situe au moment où les données du rôle sont chargées. Plus la précision est élevée, plus le nombre de propriétés du rôle calculées à la connexion est élevé. Cette méthode de connexion permet par exemple de connecter un utilisateur et de demander immédiatement toutes les données utiles sur ses rôles. Cela peut se révéler nécessaire dans certains cas, afin d'optimiser les performances et éviter les requêtes au serveur HR Access.

L'exemple suivant montre comment connecter un utilisateur et demander la résolution de ses rôles avec le niveau de précision maximal.

```
// Retrieving an existing session (not detailed here)
IHRSession session = getSession();


// Creating a description to connect a user
```

```
UserDescription description = new UserDescription(“HRUSER”, “SECRET”);

// Connecting the user by providing an hint and a precision to resolve the roles
IHRUser user = session.connectUser(description, “LDAP”, Precision.MAX_LEVEL);
```

Le concept de credential étant représenté sous forme d'interface, il est possible d'implémenter son propre credential si les implémentations standard ne couvrent pas un besoin donné. Cette nouvelle classe s'utilise alors comme n'importe quel credential, on peut rattacher ce nouveau credential à la description de l'utilisateur. Sachant que cette description est sérialisée au serveur OpenHR, il est impératif que la classe du credential soit présente dans le class path du serveur OpenHR ; autrement, les demandes de connexion ne pourront aboutir.

Sécurisation des connexions

Introduction

Cette section explique comment sécuriser les communications entre l'API OpenHR et le serveur OpenHR. A ce titre, nous vous recommandons vivement de lire le *Guide de gestion de la sécurité* qui explique comment configurer le serveur OpenHR. Ceci qui représente déjà la moitié du travail nécessaire pour sécuriser les communications.

L'API et le serveur OpenHR communiquent de manière synchrone par échange de messages (de type requête ou réponse)

Communication synchrone

L'API OpenHR et le serveur OpenHR communiquent de manière synchrone à l'aide de sockets par lesquelles transitent des messages au format texte. Ces messages peuvent être de deux types : requête ou réponse. Une requête correspond à une invocation de service du serveur HR Access tandis que la réponse représente le résultat retourné par le serveur HR Access suite au traitement de la requête.

Parmi les différents messages, on trouve (liste non exhaustive) :

- Le message de connexion de session OpenHR
- Le message de connexion d'utilisateur
- Le message d'extraction de données techniques
- Le message d'extraction de données techniques et applicatives
- Le message de lecture de dossier
- Le message de mise à jour de dossier
- Etc.

Sécurité des messages

Du point de vue de la sécurité, toutes ces fonctionnalités ne sont pas identiques. Certains messages ne requièrent pas de sécurité particulière, c'est le cas des messages de récupération du modèle de données (structure de données, informations, rubriques, liens et types de dossiers) car les données accédées ne sont pas sensibles.

Les messages de lecture / mise à jour portant sur des dossiers HR Access (données applicatives) sont déjà plus sensibles (mais pas critiques du point de vue de la sécurité). C'est pourquoi l'accès aux données est régi par la confidentialité de l'utilisateur (définie sous forme de rôle) et contrôlé par le serveur HR Access.

Le message de demande de connexion d'utilisateur est particulier car il véhicule (en standard) un mot de passe. C'est pourquoi il requiert une sécurisation de type SSL : le serveur OpenHR distant est authentifié (afin de ne pas envoyer le mot de passe à un mauvais serveur) et les communications sont chiffrées entre client et serveur pour éviter les interceptions réseau ("spoofing").

Enfin, il existe une dernière catégorie de messages si sensibles que leur utilisation est restreinte à des clients OpenHR parfaitement authentifiés : l'utilisation de ces messages constitue un privilège accordé au client. Ces messages permettent soit un accès total en lecture / mise à jour aux dossiers de la base HR Access (cas de la fonctionnalité de l'utilisateur de session) soit de récupérer des données critiques du point de vue de la sécurité (c'est le cas du message de récupération de la description de la Topologie système qui donne accès à des mots de passe FTP et JDBC). Ce type de message est sécurisé à l'aide d'un SSL bidirectionnel afin d'authentifier mutuellement le client et le serveur OpenHR et de chiffrer les données qui transitent par les sockets : seuls les clients connus peuvent utiliser ces types de messages.

Natures de messages et sécurisation

Si l'on raisonne du point de vue de la sécurité, on parvient à classer ces messages selon trois catégories (ou natures de message) qui sont :

- Les messages "privilèges" : cela inclut les messages émis au titre d'un utilisateur de session et le message de récupération de la Topologie système. Ils doivent être sécurisés par un SSL bidirectionnel.
- Les messages "sensibles" : cela inclut le message de connexion des utilisateurs (car il comporte un mot de passe). Ils doivent être sécurisés par un SSL simple.
- Les messages "normaux" : cela inclut tous les autres messages qui, du point de vue de la sécurité, ne présentent pas un caractère critique. Ils n'ont pas besoin d'être sécurisés.

Sécuriser des communications par SSL est coûteux en termes de CPU. C'est pourquoi idéalement, il faut pouvoir sécuriser indépendamment chaque message selon sa nature afin de ne chiffrer que les messages en ayant besoin et de la manière la plus appropriée. En fait, l'API OpenHR utilise trois sockets pour acheminer au serveur OpenHR les messages en fonctions de leur catégorie. Chaque socket (ou message sender (émetteur de message)) peut être configuré indépendamment des autres.

La section consacrée à la configuration de la session OpenHR a montré que dans le fichier de configuration de la session (traditionnellement nommé openhr.properties), il y avait plusieurs propriétés relatives à l'envoi des messages.

```
session.language[10]=U
session.process_list=
session.work_directory=C:/temp/openhr

openhr_server.server=10.11.12.13

normal_message_sender.security=disabled
normal_message_sender.port=8800

sensitive_message_sender.security=disabled
sensitive_message_sender.port=8800

privilegied_message_sender.security=disabled
privilegied_message_sender.port=8800
```

[10] *Déprécié en 7.30.50 ou supérieure, utiliser session.languages à la place (voir le paragraphe "Propriétés de configuration")*

L'exemple ci-dessus fait référence à des propriétés de préfixe "normal_message_sender", "sensitive_message_sender" et "privilegied_message_sender". Le premier est utilisé pour envoyer les messages normaux, le second les messages sensibles et le dernier les messages privilège.

Le tableau suivant liste les propriétés de configuration de la session OpenHR relatives à l'envoi des messages.

Les propriétés suivantes s'appliquent de manière indifférenciée aux trois préfixes "normal_message_sender", "sensitive_message_sender" et "privilegied_message_sender". Pour plus de clarté, un pseudo-préfixe "abc" est utilisé dans la suite de ce chapitre pour représenter, au choix, l'un de ces trois préfixes. Le caractère obligatoire / optionnel des propriétés dépend de la valeur attribuée à la propriété "security", comme nous allons le voir plus loin.

Nom de propriété	Rôle
abc.security	Propriété obligatoire. Indique le niveau de sécurité appliqué au transport des messages vers le serveur OpenHR. Il n'existe que trois valeurs valides : "disabled", "SSL" et "SSL2". La valeur "disabled" indique que le transport des messages n'est pas sécurisé. La valeur "SSL" indique une sécurisation de type 1 (authentification du serveur et chiffrage des données) du transport des messages. La valeur "SSL2" indique une sécurisation de type 2 (authentification mutuelle du client et du serveur et chiffrage des données) du transport des messages. Le caractère obligatoire ou facultatif des propriétés qui suivent dépend de la valeur donnée à cette propriété. Valeurs valides : "disabled", "SSL", "SSL2".
abc.use_configuration_with_prefix	Propriété facultative. L'utilité de cette propriété est expliquée plus loin. L'utilisation de cette propriété est possible quelle que soit la valeur de la propriété "security".
abc.port	Propriété obligatoire. Définit le numéro de port sur lequel communiquer avec le serveur OpenHR.
abc.ca_certificate_filename	Propriété obligatoire si "security" est à "SSL" ou "SSL2". Définit le nom du fichier keystore CA (Certification Authority) permettant d'authentifier le serveur OpenHR.
abc.ca_certificate_password	Propriété facultative si "security" est à "SSL" ou "SSL2", interdite autrement. Définit le mot de passe utilisé afin de vérifier l'intégrité du keystore CA permettant d'authentifier le serveur OpenHR.

Nom de propriété	Rôle
abc.ca_key_management_algorithm	Propriété facultative si "security" est à "SSL" ou "SSL2", interdite autrement. Définit le nom de l'algorithme de gestion des clés à utiliser pour lire le fichier keystore CA. Valeur par défaut : l'algorithme par défaut configuré au niveau de la JVM d'exécution du client OpenHR.
abc.certificate_filename	Propriété obligatoire si "security" est à "SSL2". Définit le nom du fichier keystore contenant le certificat utilisé pour authentifier le client OpenHR auprès du serveur.
abc.certificate_password	Propriété obligatoire si "security" est à "SSL2". Définit le mot de passe à utiliser afin d'authentifier le client OpenHR à l'aide de son certificat.
abc.key_management_algorithm	Propriété facultative si "security" est à "SSL2", interdite autrement. Définit le nom de l'algorithme de gestion des clés à utiliser pour lire le fichier keystore. Valeur par défaut : l'algorithme par défaut configuré au niveau de la JVM d'exécution du client OpenHR.

Pour le message sender normal, les messages sont théoriquement non sécurisés. La configuration qui lui est associée ressemble à ceci :

```
. . .

normal_message_sender.security=disabled
normal_message_sender.port=8800

...
```

Pour le message sender sensible, les messages sont théoriquement sécurisés par SSL, ce que l'on configure en positionnant la propriété "security" à "SSL". La configuration qui lui est associée ressemble à ceci :

```
. . .
sensitive_message_sender.security=SSL
sensitive_message_sender.port=8801
sensitive_message_sender.ca_certificate_filename=ClientCA.cer
sensitive_message_sender.ca_certificate_password=secret
# sensitive_message_sender.ca_key_management_algorithm=

...
```

Pour le message sender privilegied, les messages sont théoriquement sécurisés par SSL2, ce que l'on configure en positionnant la propriété "security" à "SSL2". La configuration qui lui est associée ressemble à ceci :

```
...

privilegied_message_sender.security=SSL
privilegied_message_sender.port=8802
privilegied_message_sender.ca_certificate_filename=ClientCA.cer
privilegied_message_sender.ca_certificate_password=secret
# privilegied_message_sender.ca_key_management_algorithm=
privilegied_message_sender.certificate_filename=Client.cer
privilegied_message_sender.certificate_password=secret2
# privilegied_message_sender.key_management_algorithm=

...
```

Pour un environnement de développement

Précisons que c'est la configuration du serveur OpenHR qui détermine comment doit être configuré le client OpenHR et non l'inverse. Sur un environnement de développement où les données stockées en base HR Access ne sont pas sensibles, on se contentera d'un fichier de configuration minimaliste comme celui donné ci-dessous.

```
session.language[11]=U
session.process_list=
session.work_directory=C:/temp/openhr

openhr_server.server=10.11.12.13

normal_message_sender.security=disabled
normal_message_sender.port=8800

sensitive_message_sender.security=disabled
sensitive_message_sender.port=8800

privilegied_message_sender.security=disabled
privilegied_message_sender.port=8800
```

[11] *Déprécié en 7.30.50 ou supérieure, utiliser session.languages à la place (voir le paragraphe "Propriétés de configuration")*

Pour un environnement de production

En revanche, sur un environnement de production il est impératif de sécuriser les communications, autrement, un utilisateur malveillant pourrait intercepter les mots de passe de connexion des utilisateurs ou utiliser la fonctionnalité d'utilisateur de session afin de lire et modifier les dossiers HR Access en base de données !

Cette documentation ne détaille pas le protocole SSL ni la manière de générer les certificats client et serveur. Pour cela, reportez-vous au *Guide de gestion de la sécurité*.

La sécurité configurée ici doit s'accompagner de mesures de sécurité au niveau réseau comme l'utilisation d'un pare-feu par exemple. Il faut considérer l'ensemble des mesures de sécurité de l'infrastructure pour déterminer quel est le niveau optimal de sécurité à utiliser et faire le meilleur compromis entre sécurité et performance.

Le fichier openhr.properties donné ci-dessous correspond à une configuration pour un environnement de production.

```
session.language[12]=U
session.process_list=
session.work_directory=C:/temp/openhr

openhr_server.server=10.11.12.13

normal_message_sender.security=disabled
normal_message_sender.port=8800

sensitive_message_sender.security=SSL
sensitive_message_sender.port=8801
sensitive_message_sender.ca_certificate_filename=ClientCA.cer
sensitive_message_sender.ca_certificate_password=secret
# sensitive_message_sender.ca_key_management_algorithm=

privilegied_message_sender.security=SSL
privilegied_message_sender.port=8802
privilegied_message_sender.ca_certificate_filename=ClientCA.cer
privilegied_message_sender.ca_certificate_password=secret
# privilegied_message_sender.ca_key_management_algorithm=
privilegied_message_sender.certificate_filename=Client.cer
privilegied_message_sender.certificate_password=secret2
# privilegied_message_sender.key_management_algorithm=
```

[12] *Déprécié en 7.30.50 ou supérieure, utiliser session.languages à la place (voir le paragraphe "Propriétés de configuration")*

CHAPITRE 6

L'API d'envoi de messages

A propos de ce chapitre

Lorsque l'on manipule des dossiers à l'aide de la fonctionnalité de gestion de dossiers ou que l'on lit des données à l'aide de la technique d'extraction de données, l'API émet des messages au serveur OpenHR via des sockets TCP/IP.

L'API permet aussi d'envoyer soi-même manuellement des messages au serveur sans nécessiter la manipulation d'une classe de haut niveau (comme la classe HRDossier). Cette facilité peut être utilisée afin de développer un client OpenHR optimisé qui manipule les données de manière bas niveau.

Ce chapitre présente :

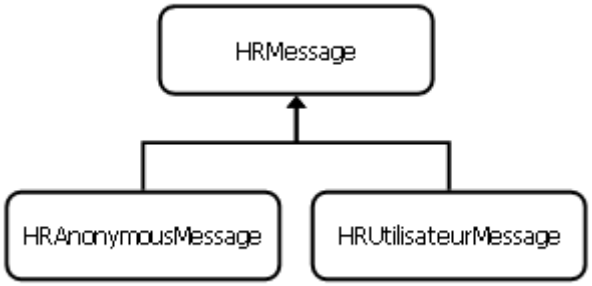
- Les classes de messages disponibles
- La méthode d'envoi de messages au serveur HR Access
- Des exemples d'utilisation

Les classes de message

Le package com.hraccess.openhr.msg fournit l'ensemble des classes Java représentant les messages échangés entre client et serveur OpenHR. La classe de base com.hraccess.openhr.msg.HRMessage représente un message (de requête) émis au serveur OpenHR alors que la classe com.hraccess.openhr.msg.HRResult représente une réponse reçue du serveur OpenHR.

Les classes de message de requête se répartissent en deux catégories : les messages anonymes et les messages utilisateur (la différence est expliquée plus loin). Cette caractéristique se retrouve au niveau de la hiérarchie de classe des messages : la classe HRMessage est sous-classée en deux classes : HRAnonymousMessage et HRUserMessage. Les messages anonymes dérivent de la première alors que les messages utilisateur sous-classent la seconde (cf. schéma ci-dessous).

Les messages se répartissent en deux catégories, les messages anonymes héritant de la classe HRAnonymousMessage et les messages utilisateur héritant de la classe HRUserMessage :



Cette distinction n'existe pas au niveau des classe réponse : toutes les classes de réponse étendent la classe HRResult.

Les classes de requête et de réponse fonctionnent par paire : à une classe Java représentant un message de requête est forcément associée une classe Java représentant la réponse correspondante.

Le tableau suivant liste, de manière non exhaustive, les classes Java de requête et de réponse associées à un service offert par l'API OpenHR. Les classes de requête ont un nom de la forme HRMsgXXX et les classes de réponse, de la forme HRResultXXX.

Service	Classes de message
Connexion d'utilisateur	HRMsgLogin HRResultLogin
Déconnexion d'utilisateur	HRMsgLogout HRResultLogout
Ouverture de session	HRMsgOpenInstance HRResultOpenInstance
Fermeture de session	HRMsgCloseInstance HRResultCloseInstance
Extraction de données techniques	HRMsgExtractTechnicalData HRResultExtractTechnicalData
Extraction de données applicatives	HRMsgExtractData HRResultExtractData
Sélection de population (de dossiers)	HRMsgSelectPopulation HRResultSelectPopulation
Lecture de dossiers	HRMsgGetDossierData HRResultGetGetDossierData
Mise à jour de dossiers	HRMsgUpdateDossier HRResultUpdateDossier

Les messages utilisateur sont des messages qui ne peuvent être émis que par un utilisateur (cela inclut l'utilisateur de session) car le message a besoin d'une connexion utilisateur et d'un rôle pour être traité par le serveur. Cette catégorie inclut en particulier les messages d'accès aux données (lecture / mise à jour de dossiers HR Access), de sélection de dossiers : la sécurité de l'accès aux données est régie par le rôle utilisé pour effectuer l'opération.

Les messages anonymes sont ceux qui sont envoyés au serveur OpenHR sans utilisateur connecté. Cela inclut l'ensemble des messages à caractère technique tels que ceux de connexion de session ou d'utilisateur, de récupération de la description du modèle de données, etc.

Le typage des messages de requête en HRAnonymousMessage ou en HRUserMessage permet de s'assurer qu'un message anonyme ne puisse être envoyé par un utilisateur et qu'un message utilisateur ne puisse être envoyé via la session OpenHR. La signature des méthodes d'envoi de messages (cf. plus bas) est telle qu'il n'est pas possible d'envoyer un message de manière incorrecte.

L'envoi de messages

Cette section explique comment envoyer des messages au serveur HR Access.

Messages anonymes

Les messages anonymes sont ceux dérivant de la classe com.hraccess.openhr.msg.HRAnonymousMessage. Ils ne peuvent être envoyés que par la session OpenHR.

Pour cela, il faut utiliser la méthode IHRSession.sendMessage(HRAnonymousMessage) de la manière suivante.

```
// Retrieving an already connected session (not detailed here)
IHRSession session = getSession();


// Creating a new (anonymous) request message to retrieve the HR Access server's versions
HRMsgGetVersions request = new HRMsgGetVersions();


// Sending message via the session (synchronous task)
HRResultGetVersions result = (HRResultGetVersions) session.sendMessage(request);


// Processing response (not detailed here)
...
```

Cet exemple crée un message de requête (de classe HRMsgGetVersions) permettant d'invoquer le service "GET_VERSIONS". Ce service permet de récupérer la version du serveur HR Access ainsi que la version minimale que doit respecter le client OpenHR pour communiquer avec lui. Le message est envoyé via la méthode IHRSession.sendMessage(HRAnonymousMessage). Cet appel induit une requête au serveur OpenHR et la transmission du message GET_VERSIONS sous forme de message texte. En retour, l'API retourne le résultat sous forme de HRResult (type de retour de la méthode sendMessage(HRAnonymousMessage)). On caste le résultat vers la classe Java correspondant au résultat du service "GET_VERSIONS", c'est-à-dire HRResultGetVersions.

L'émission de messages anonymes au serveur OpenHR est donc simple. En réalité, la partie la plus compliquée du travail consiste à créer le message de requête. Dans l'exemple, cela consiste juste à instancier la classe HRMsgGetVersions mais dans le cas d'une lecture de dossiers (ce message est en fait de type utilisateur mais le raisonnement reste valable), cette tâche est plus compliquée car il faut renseigner de nombreux paramètres : le processus utilisé pour lire les dossiers, la structure des dossiers, les informations, etc. En fait, toutes ces informations correspondent à celles qui sont configurées via la classe HRDossierCollectionParameters lorsque l'on utilise une collection de dossiers. Quelques exemples de création et de paramétrage de message de requête utiles seront fournis plus loin dans la documentation.

Il n'est pas possible de documenter chacun des messages, sachez cependant que la signature des classes de message est bien souvent simple, ce qui rend son utilisation intuitive.

Messages utilisateur

Les messages utilisateur sont ceux dérivant de la classe com.hraccess.openhr.msg.HRUserMessage. Ils ne peuvent être envoyés que par la conversation d'un utilisateur connecté.

Pour cela, il faut utiliser la méthode IHRConversation.send(HRUserMessage, IHRRole). Contrairement à l'envoi de messages anonymes, la méthode impose de spécifier le rôle au titre duquel le message de requête doit être envoyé. Ceci est dû au fait que ces messages portent sur des données sensibles et que la sécurité est contrôlée par rapport au rôle fourni.

L'exemple suivant montre comment émettre un message utilisateur au serveur HR Access.

```
// Retrieving an already connected user (not detailed here)
IHRUser user = getUser();

// Retrieving the user's main conversation to send messages
IHRConversation conversation = user.getMainConversation();

// Retrieving one of the user's roles to send the message
IHRRole role = user.getRole("EMPLOYEE(123456)");
```

```
// Creating a new (user) request message to extract some data from
   employee dossiers
HRMsgExtractData request = new HRMsgExtractData();
request.setFirstRow(0);
request.setMaxRows(100);
request.setSqlStatement("SELECT * FROM ZY00");

// Sending message via the user's conversation (synchronous task)
HRResultExtractData result = (HRResultExtractData)
   conversation.send(request, role);

// Processing response (not detailed here)
...
```

Le code ci-dessus permet de lire 100 enregistrements depuis la table ZY00 (Identifiant de salariés) à l'aide du rôle de forme canonique EMPLOYEE(123456).

L'émission de messages utilisateur au serveur OpenHR est donc simple. Ce qui est plus complexe, c'est de créer et de paramétrer correctement le message de requête.

Exemples d'utilisation

Cette section fournit quelques exemples d'utilisation de l'API d'envoi de messages pour les fonctionnalités les plus utiles.

Extraction de données techniques

La technique d'extraction de données techniques a été présentée dans le paragraphe "Extraction de données techniques" du chapitre "Les Packages de l'API". Ci-dessous, vous trouverez comment arriver au même résultat en utilisant l'API d'envoi de messages.

Le service d'extraction de données techniques (EXTRACT_TECHNICAL_DATA) est associé aux classes HRMsgExtractTechnicalData (requête) et HRResultExtractTechnicalData (réponse). La classe de requête étend la classe HRAnonymousMessage. Le message doit donc être envoyé via la session OpenHR, c'est d'ailleurs pour cela que le message ne permet de lire que des données techniques.

```
// Retrieving an already connected session (not detailed here)
IHRSession session = getSession();

// Creating a new (anonymous) request message to extract some technical
   data from table DI60
HRMsgExtractTechnicalData request = new HRMsgExtractTechnicalData();
request.setFirstRow(0);
```

```
request.setMaxRows(100);
request.setSqlStatement("SELECT * FROM DI60");

// Sending message via the session (synchronous task)
HRResultExtractTechnicalData result = (HRResultExtractTechnicalData)
    session.sendMessage(request);

// Dumping columns
System.out.println("Result has " + result.getColumnCount() + "
    column(s)");

for (Iterator iter=result.getColumns(); iter.hasNext(); ) {
    HRResultExtractTechnicalData.Column column =
    (HRResultExtractTechnicalData.Column) iter.next();

    System.out.println("Column name is <" + column.getName() + ">");
    System.out.println("Column size (length) is <" + column.getSize() +
    ">");
    System.out.println("Column type is <" + column.getType() + ">"); // see
    constants IHRItem.TYPE_XXX
    System.out.println("Column has " + column.getDecimal() + "
    decimal(s)");
}

// Dumping rows
int rowCount = 0;
for (Iterator iter=result.getRows(); iter.hasNext(); rowCount++) {
    Object[] values = (Object[]) iter.next();

    System.out.println("Dumping row #" + rowCount);

    for (int i=0 ; i<values.length; i++) {
    System.out.println("Value (" + rowCount + "," + i + ") = " + values[i]);
    }
}
```

La classe HRResultExtractTechnicalData retourne le résultat de l'extraction sous forme de lignes, chaque ligne étant manipulée sous forme d'un tableau d'Objects. Les données sur les colonnes sont aussi disponibles (le type des colonnes exprimé sous forme de short correspond à l'une des valeurs de nom TYPE_XXX de la classe IHRItem).

Extraction de données applicatives

La technique d'extraction de données applicatives a été présentée dans le paragraphe "Extraction de données applicatives" du chapitre "Les Packages de l'API". Vous trouverez ici comment arriver au même résultat en utilisant l'API d'envoi de messages.

Le service d'extraction de données applicatives (EXTRACT_DATA) est associé aux classes HRMsgExtractData (requête) et HRResultExtractData (réponse). La classe de requête étend la classe HRUserMessage. Le message doit donc être envoyé via la conversation d'un utilisateur connecté, c'est d'ailleurs pour cela que le message permet de lire des données applicatives.

```
// Retrieving an already connected user (not detailed here)
IHRUser user = getUser();

// Retrieving the user's main conversation to send messages
IHRConversation conversation = user.getMainConversation();

// Retrieving one of the user's roles to send the message
IHRRole role = user.getRole("EMPLOYEE(123456)");

// Creating a new (user) request message to extract some data from
   employee dossiers
HRMsgExtractData request = new HRMsgExtractData();
request.setFirstRow(0);
request.setMaxRows(100);
request.setSqlStatement("SELECT * FROM ZY00");

// Sending message via the user's conversation (synchronous task) by using
   the given role
HRResultExtractData result = (HRResultExtractData)
   conversation.send(request, role);

// Dumping columns
System.out.println("Result has " + result.getColumnCount() + "
   column(s)");

for (Iterator iter=result.getColumns(); iter.hasNext(); ) {
   HRResultExtractData.Column column = (HRResultExtractData.Column)
   iter.next();

   System.out.println("Column name is <" + column.getName() + ">");
```

```
        System.out.println("Column size (length) is <" + column.getSize() +
        ">");
        System.out.println("Column type is <" + column.getType() + ">"); // see
        constants IHRItem.TYPE_XXX
        System.out.println("Column has " + column.getDecimal() + "
        decimal(s)");
    }

    // Dumping rows
    int rowCount = 0;
    for (Iterator iter=result.getRows(); iter.hasNext(); rowCount++) {
        Object[] values = (Object[]) iter.next();

        System.out.println("Dumping row #" + rowCount);

        for (int i=0 ; i<values.length; i++) {
        System.out.println("Value (" + rowCount + "," + i + ") = " + values[i]);
        }
    }
```

La classe HRResultExtractData retourne le résultat de l'extraction sous forme de lignes, chaque ligne étant manipulée sous forme d'un tableau d'Objects. Les données sur les colonnes sont aussi disponibles (le type des colonnes exprimé sous forme de short correspond à l'une des valeurs de nom TYPE_XXX de la classe IHRItem).

Sélection de dossiers

Cette fonctionnalité permet de sélectionner dynamiquement des clés techniques (des numéros) de dossiers à partir d'un ordre SQL. Le service invoqué s'appelle SELECT_POPULATION (sous-entendu population de dossiers). Les deux classes Java correspondantes s'appellent HRMsgSelectPopulation et HRResultSelectPopulation.

Pour sélectionner des dossiers, c'est-à-dire des données applicatives, il faut utiliser un rôle : le message est donc de type "utilisateur" et doit être envoyé via une conversation.

L'ordre SQL de sélection utilisé doit être de la forme "SELECT A.NUDOSS FROM xx00 A" car le programme COBOL (BHS) qui interprète l'ordre SQL s'attend à ce que la première colonne sélectionnée soit la colonne NUDOSS. Il faut, de plus, que les tables soient associées à des alias (A ici).

```
    // Retrieving an already connected user (not detailed here)
    IHRUser user = getUser();

    // Retrieving the user's main conversation to send messages
    IHRConversation conversation = user.getMainConversation();
```

```
// Retrieving one of the user's roles to send the message
IHRRole role = user.getRole("EMPLOYEE(123456)");

// Creating a new (user) request message to select some employee dossiers
HRMsgSelectPopulation request = new HRMsgSelectPopulation();
request.setFirstDossier(0);
request.setMaxDossiers(100);
request.setSqlStatement("SELECT A.NUDOSS FROM ZY00 A");
request.setDataStructure("ZY");

// Sending message via the user's conversation (synchronous task) by using
   the given role
HRResultSelectPopulation result = (HRResultSelectPopulation)
   conversation.send(request, role);

// Dumping result
System.out.println("Selection returned " + result.getDossierCount() + "
   dossier(s)");

for (Iterator iter=result.getDossiers(); iter.hasNext(); ) {
   Integer dossierNumber = (Integer) iter.next();

   System.out.println("Selection returned dossier with key <" +
   dossierNumber + ">");
}
```

Lorsque l'on sélectionne des dossiers, il est possible de le faire de plusieurs manières. L'exemple ci-dessus utilise une requête SQL qui peut utiliser les mots-clés SQL supportés par le programme BHS (cf. le paragraphe "Le filtre SQL" du chapitre "Les Packages de l'API"). La sélection va retourner au plus 100 dossiers de salarié (de rang 0-99) parmi ceux que le rôle utilisé permet de sélectionner. Le résultat se présente sous la forme d'une liste d'entiers correspondant aux clés techniques (les numéros) des dossiers sélectionnés.

Il est aussi possible de sélectionner des dossiers au titre d'une population de rôle (cf. onglet "Structure - Périmètre des populations" au niveau d'un modèle de rôle dans Design Center) ou d'une activité. L'utilisation d'une requête SQL, d'une population de rôle et d'une activité doit répondre aux règles suivantes :

- L'utilisation d'une requête SQL n'est pas obligatoire
- Si une requête SQL est fournie, alors il est possible (mais pas obligatoire) d'indiquer une population de rôle ou une activité mais pas les deux à la fois. Ce qui donne 3 cas :
  - Requête SQL seule (cf. exemple ci-dessus)
  - Requête SQL avec population de rôle (cas 2)
  - Requête SQL avec activité (cas 3)
- Si aucune requête SQL n'est fournie, alors l'indication d'une population de rôle est obligatoire (afin de définir les dossiers à sélectionner) et l'utilisation d'une activité est interdite (cas 4).

Les cas 2, 3 et 4 sont illustrés ci-dessous.

Cas 2 : Requête SQL avec population de rôle

La population de dossiers sélectionnée sera celle issue de l'intersection du résultat de sélection de la requête SQL et de la population (de dossiers définie par la population) de rôle d'identifiant donné.

```
// Retrieving the user, conversation and role (not detailed here)
...

// Creating a new (user) request message to select some employee dossiers
HRMsgSelectPopulation request = new HRMsgSelectPopulation();
request.setSqlStatement(“SELECT A.NUDOSS FROM ZY00 A”);
request.setPopulationId(“MY-WORKMATES”);
request.setDataStructure(“ZY”);

// Sending request and processing response (not detailed here)
...
```

Cas 3 : Requête SQL avec activité

La population de dossiers sélectionnée sera celle issue de l'intersection du résultat de sélection de la requête SQL et de l'éventuelle population de restriction associée à l'activité de rôle donnée.

```
// Retrieving the user, conversation and role (not detailed here)
...

// Creating a new (user) request message to select some employee dossiers
HRMsgSelectPopulation request = new HRMsgSelectPopulation();
request.setSqlStatement(“SELECT A.NUDOSS FROM ZY00 A”);
request.setActivity(“PAYROLL”);
request.setDataStructure(“ZY”);

// Sending request and processing response (not detailed here)
...
```

Cas 4 : Population de rôle

La population de dossiers sélectionnée sera celle définie par la population de rôle d'identifiant donné.

```
// Retrieving the user, conversation and role (not detailed here)
...

// Creating a new (user) request message to select some employee dossiers
HRMsgSelectPopulation request = new HRMsgSelectPopulation();
request.setPopulationId(“MY-WORKMATES”);
request.setDataStructure(“ZY”);

// Sending request and processing response (not detailed here)
...
```

Pour information, c'est cette technique de sélection de dossiers qui est mise en œuvre lors des demandes de chargement de dossiers via une collection de dossiers (voir le paragraphe "Chargement d'un ou plusieurs dossiers" dans le chapitre "Les Packages de l'API").

Il est possible de récupérer la population de dossiers sélectionnés sous forme de com.hraccess.openhr.HRPopulation. Cette classe sera notamment utilisée pour la lecture des données de dossiers.

Lecture de dossiers

Le service de lecture de dossiers (de nom GET_DOSSIER_DATA) permet de récupérer les données en table pour une population de dossiers donnée. Les classes Java de message associées sont nommées HRMsgGetDossierData et HRResultGetDossierData. Avant de lire des données depuis le serveur HR Access, il faut définir la structure des dossiers (de la même manière que lorsqu'on crée une configuration de collection de dossiers (voir la section "La gestion de dossiers / Configuration" dans le chapitre "Les Packages de l'API"). Comme les données lues sont de type applicatif, l'envoi du message de requête se fait nécessairement via la conversation d'un utilisateur en spécifiant un rôle.

Voici un exemple de lecture de dossiers basé sur l'utilisation du service GET_DOSSIER_DATA.

```
// Retrieving an already connected user (not detailed here)
IHRUser user = getUser();

// Retrieving the user's main conversation to send messages
IHRConversation conversation = user.getMainConversation();

// Retrieving one of the user's roles to send the message
IHRRole role = user.getRole("EMPLOYEE(123456)");

// Selecting some dossiers (not detailed here)
HRResultSelectPopulation selectionResult = selectPopulation();

// Creating a new (user) request message to read some data from a given
   set of employee dossiers
HRMsgGetDossierData request = new HRMsgGetDossierData();
request.setProcessName("FS00B");
request.setDataType(HRMsgGetDossierData.DATATYPE_REAL);
request.setPopulation(selectionResult.getPopulation());
request.addDataSection(new HRMsgGetDossierData.DataSection("00"));
request.addDataSection(new HRMsgGetDossierData.DataSection("10"));
request.addDataSection(new HRMsgGetDossierData.DataSection("AG"));

// Sending message via the user's conversation (synchronous task) by using
   the given role
HRResultGetDossierData result = (HRResultGetDossierData)
   conversation.send(request, role);

// Dumping result
```

```
System.out.println("Read data for " + result.getDossierCount() + "
    dossier(s)");


for (Iterator iter=result.getDossiers(); iter.hasNext(); ) {
    HRResultGetDossierData.Dossier dossier =
    (HRResultGetDossierData.Dossier) iter.next();


    System.out.println("Processing dossier with key <" +
    dossier.getNudoss() + ">");


    for (Iterator iter2=dossier.getSections(); iter2.hasNext(); ) {
    HRResultGetDossierData.Section dataSection =
    (HRResultGetDossierData. Section) iter2.next();


    System.out.println("Processing data section <" +
    dossier.getDataSectionName() + ">");


    for (int i=0 ; i<dataSection.getRowCount() ; i++) {
            HRResultGetDossierData.Row row = dataSection.getRow(i);


            System.out.println("Processing row <" + i + ">");


            for (int j=0 ; j<row.getValueCount(); j++) {
                    Object value = row.getValue(j);


                    System.out.println("Value " + j + " is <" + value + ">");
            }
    }
    }
}
```

Détaillons cet exemple.

Le code Java récupère un utilisateur connecté (sous forme de IHRUser) puis récupère sa conversation principale et un rôle de forme canonique EMPLOYEE(123456). L'exemple suppose qu'une sélection de population (de dossiers) a été effectuée préalablement et récupère le résultat de cette sélection de population sous forme de HRResultSelectPopulation (voir paragraphe précédent).

Puis on crée une instance de HRMsgGetDossierData afin de lire des dossiers HR Access. On positionne le nom du processus HR Access à utiliser pour effectuer la lecture des données ainsi que le type des données à lire (des dossiers HR Access). En fait, seul le type HRMsgGetDossierData.DATATYPE_REAL est utilisable dans le cas qui nous intéresse.

Pour indiquer la population des dossiers à lire, on utilise la classe com.hraccess.openhr.HRPopulation qui est caractérisée par une structure de données et une liste de clés (techniques ou fonctionnelles) identifiant des dossiers HR Access. Dans cet exemple, l'instance de HRPopulation est récupérée en appelant la méthode getPopulation() sur la classe HRResultSelectPopulation.

Puis on crée des instances de HRMsgGetDossierData.Section représentant les informations que l'on souhaite lire et on les ajoute (via la méthode HRMsgGetDossierData.addDataSection(Section)) au message de requête.

La requête de lecture de dossiers est envoyée via la conversation de l'utilisateur en utilisant son rôle EMPLOYEE(123456) et le résultat est retourné sous forme de HRResultGetDossierData.

Le résultat retourné (de type HRResultGetDossierData) permet d'itérer simplement sur les données lues car celles-ci sont organisées de manière hiérarchique. Les dossiers (de type HRResultGetDossierData.Dossier) donnent accès aux informations (de type HRResultGetDossierData.Section) qui donnent accès aux occurrences (de type HRResultGetDossierData.Row) qui donnent accès aux valeurs des rubriques sous forme d'Object.

Il est donc très simple de créer une requête de lecture de dossiers HR Access.

Comme nous l'avons vu dans la section expliquant la configuration d'une collection de dossiers, il est possible d'indiquer comment chaque information doit être lue par le serveur HR Access. Le paramétrage de lecture de chaque information est représenté par la classe HRMsgGetDossierData.DataSection. Le tableau suivant indique les méthodes utiles de cette classe.

Dans le tableau suivant, les classes Blob et External représentent les classes HRMsgGetDossierData.Blob et HRMsgGetDossierData.External.

Méthode	**Rôle**
void addBlob(Blob)	Ajoute la description de BLOB donnée à la liste des BLOBs de l'information que l'on souhaite lire.
void addBlobs(Collection<Blob>)	Ajoute les descriptions de BLOB données à la liste des BLOBs de l'information que l'on souhaite lire.
void addExternal(External)	Ajoute la description de rubrique externe donnée à la liste des rubriques externes de l'information que l'on souhaite lire.

Méthode	Rôle
void addExternals(Collection<External>)	Ajoute les descriptions de rubrique externe données à la liste des rubriques externes de l'information que l'on souhaite lire.
int getBlobCount()	Retourne le nombre de rubriques de rôle BLOB configurées en lecture pour cette information.
List<BLOB> getBlobs()	Retourne la description des rubriques de rôle BLOB configurées en lecture pour cette information.
String getDataSectionName()	Retourne le nom de l'information à lire.
int getExternalItemCount()	Retourne le nombre de rubriques externes configurées au niveau de l'information.
List<External> getExternals()	Retourne les rubriques externes configurées au niveau de l'information.
int getFrom()	Retourne l'indice de la première occurrence d'information à lire. Cette propriété permet de récupérer les occurrences d'une information par paquet (pagination). Prend la valeur 0 par défaut (première occurrence).
String getProcedureCode()	Retourne l'identifiant (code) du traitement paramétré au niveau de l'information. Ce traitement permet de personnaliser la lecture des occurrences de l'information.
String getProcedureGroup()	Retourne l'identifiant du groupe de traitement paramétré au niveau de l'information. Ce traitement permet de personnaliser la lecture des occurrences de l'information.
int getSize()	Retourne le nombre maximal d'occurrences de l'information que l'on souhaite lire.
String getSQLFilter()	Retourne le filtre SQL utilisé pour filtrer les occurrences de l'information. Voir la section sur le paramétrage du filtre SQL dans le cas d'une collection de dossiers.

Méthode	Rôle
Type getType()	Retourne le mode de lecture des occurrences de l'information. Les différentes valeurs valides que peut retourner cette méthode sont énumérées sous forme de constantes au niveau de la classe HRMsgGetDossierData.Type (safe type enumeration). Les différents modes valides sont : FIRST, LAST et NEXT. Le mode FIRST indique que les occurrences doivent être lues dans l'ordre naturel de l'information tel que défini par ses arguments de tri. Le mode LAST indique que la lecture des occurrences doit s'effectuer en sens inverse du sens naturel (de la dernière à la première). Enfin, le mode NEXT est utilisé afin de récupérer des occurrences qui ne sont ni les premières ni les dernières (cas de la pagination dans les occurrences de l'information).
void setBlobs(Collection<BLOB>)	Définit les rubriques de rôle BLOB de l'information que l'on souhaite lire.
void setExternals(Collection<External>)	Définit les rubriques externes de l'information que l'on souhaite lire.
void setFrom(int)	Positionne l'indice de la première occurrence d'information à lire. Cette propriété permet de récupérer les occurrences d'une information par paquet (pagination). Prend la valeur 0 par défaut (première occurrence).
void setProcedureCode(String)	Positionne l'identifiant du traitement paramétré au niveau de l'information. Ce traitement permet de personnaliser la lecture des occurrences de l'information.
void setProcedureGroup(String)	Positionne l'identifiant du groupe de traitement paramétré au niveau de l'information. Ce traitement permet de personnaliser la lecture des occurrences de l'information.
void setSize(int)	Positionne le nombre maximal d'occurrences de l'information que l'on souhaite lire.
void setSQLFilter(String)	Positionne le filtre SQL utilisé pour filtrer les occurrences de l'information. Voir la section sur le paramétrage du filtre SQL dans le cas d'une collection de dossiers.

Méthode	Rôle
void setType(Type)	Positionne le mode de lecture des occurrences de l'information. Les différentes valeurs valides que peut prendre cette méthode sont énumérées sous forme de constantes au niveau de la classe HRMsgGetDossierData.Type (safe type enumeration). Les différents modes valides sont : FIRST, LAST et NEXT. Le mode FIRST indique que les occurrences doivent être lues dans l'ordre naturel de l'information tel que défini par ses arguments de tri. Le mode LAST indique que la lecture des occurrences doit s'effectuer en sens inverse du sens naturel (de la dernière à la première). Enfin, le mode NEXT est utilisé afin de récupérer des occurrences qui ne sont ni les premières ni les dernières (cas de la pagination dans les occurrences de l'information).

L'utilisation des méthodes de la classe HRMsgGetDossierData.DataSection rappelle ce que l'on a déjà vu au niveau de la configuration d'une collection de dossiers, c'est pourquoi on ne fournira pas d'exemple pour illustrer leur utilisation.

La classe HRMsgGetDossierData.Blob est un wrapper autour d'une java.lang.String qui représente le nom de la rubrique de rôle BLOB que l'on souhaite lire. Son utilisation s'effectue de la manière suivante :

```
// Creating a HRMsgGetDossierData.Blob to read the item ZY00 BLOB01 with a
   BLOB role

HRMsgGetDossierData.Blob blob = new HRMsgGetDossierData.Blob(“BLOB01”);


// Registering BLOB for read

dataSection.addBlob(blob);
```

Attention ! Lorsque l'on configure la requête de lecture pour lire des rubriques de rôle BLOB, la réponse renvoyée par le serveur HR Access ne comporte qu'une partie des données des BLOBs demandés, en l'occurrence la description des BLOBs telle qu'elle est stockée en table BX10.

Le contenu du BLOB (qui est issu de la table BX20 ou du volume d'archivage) n'est, lui, pas disponible. Pour le lire, il faut utiliser un message dédié correspondant au service GET_BLOBS. La description des BLOBs retournée par le serveur HR Access est représentée par l'interface IHRBlob.Description (voir exemple ci-dessous) qui a déjà été présentée dans la section "La gestion des BLOBs" du chapitre "Les packages de l'API". Il s'ensuit que pour lire des données de type BLOB, il faut faire au moins deux requêtes au serveur HR Access : la première pour déterminer si un BLOB est associé à la rubrique de rôle BLOB, la seconde pour récupérer le contenu du BLOB.

```
// Configuring request message to read some dossiers with their BLOB items
   (not detailed here)

...
```

```
// Sending message via the user's conversation (synchronous task) by using
   the given role
HRResultGetDossierData result = (HRResultGetDossierData)
   conversation.send(request, role);


// Dumping result
System.out.println("Read data for " + result.getDossierCount() + "
   dossier(s)");


for (Iterator iter=result.getDossiers(); iter.hasNext(); ) {
   HRResultGetDossierData.Dossier dossier =
   (HRResultGetDossierData.Dossier) iter.next();


   System.out.println("Processing dossier with key <" + dossier.getNudoss()
   + ">");


   for (Iterator iter2=dossier.getSections(); iter2.hasNext(); ) {
   HRResultGetDossierData.Section dataSection = (HRResultGetDossierData.
   Section) iter2.next();


   System.out.println("Processing data section <" +
   dossier.getDataSectionName() + ">");


   for (int i=0 ; i<dataSection.getRowCount() ; i++) {
         HRResultGetDossierData.Row row = dataSection.getRow(i);


         System.out.println("Processing row <" + i + ">");


         for (int j=0 ; j<row.getValueCount(); j++) {
               Object value = row.getValue(j);


               System.out.println("Value " + j + " is <" + value + ">");
         }


         System.out.println("Row has " + row.getBlobDescriptionCount() +
   " BLOB description(s)");


         for (Iterator iter3=row.getBlobDescriptions(); iter3.hasNext(); ) {
```

```
                    IHRBlob.Description description = (IHRBlob.Description)
    iter3.next();

                    / / Processing BLOB descriptions (not detailed here)
                    ...
            }
        }
        }
    }
```

La lecture des rubriques externes (External) s'effectue en utilisant la classe HRMsgGetDossierData.External (cf. exemple ci-dessous). Reportez-vous au paragraphe "Rubrique externe" du chapitre du chapitre "Les packages de l'API" pour connaître la définition de ce type de rubrique.

```
// Creating a HRMsgGetDossierData.External to read an external item
HRMsgGetDossierData.External external = new
   HRMsgGetDossierData.External();
external.setSourceItemName("MOTIFA"); // Setting the name of the source item
   (mapped to a rule directory)
external.setTargetDataSectionName("01");
external.setTargetDataStructureName("ZD");
external.setTargetItemName("LIBLON");


// Registering external for read
dataSectionZYAG.addExternal(external);
```

Le code ci-dessus permet de créer une rubrique externe rattachée à l'information ZYAG (Absences d'un salarié) afin de récupérer le libellé long (Rubrique LIBLON (Libellé long) de l'information ZD01 (Libellés d'un code réglementaire)) associé au motif d'absence défini par la rubrique ZYAG MOTIFA (en correspondance avec le répertoire réglementaire ZD DSJ (Motifs d'absence et de présence)).

L'utilisation de la classe HRMsgGetDossierData.External est proche de ce que l'on a déjà vu dans la section "Configuration de la collection de dossiers" du chapitre "Les packages de l'API".

Les méthodes des classes HRResultGetDossierData, HRResultGetDossierData.Dossier, HRResultGetDossierData.Section et HRResultGetDossierData.Row n'ont pas été détaillées ici car elles se limitent principalement à des méthodes pour lire des propriétés ou pour itérer sur des listes. Leur utilisation est suffisamment intuitive pour ne pas être documentée ici.

Mise à jour de dossiers

Le service de mise à jour de dossiers (de nom UPDATE_DOSSIER) est associé aux deux classes de message HRMsgUpdateDossier et HRResultUpdateDossier. Comme le message de lecture de dossiers, c'est un message utilisateur qui doit être envoyé via la conversation d'un utilisateur en spécifiant un rôle.

Identification de dossier

Pour mettre à jour un dossier, il faut d'abord l'identifier en précisant les informations suivantes :

- La structure de données à mettre à jour
- Le dossier à mettre à jour par sa clé technique ou fonctionnelle
- L'information à mettre à jour
- Le type de mise à jour (création, suppression ou modification)
- L'occurrence à mettre à jour (si suppression ou modification)
- Les valeurs de rubriques à prendre en compte (si création ou modification)

Les dossiers peuvent être identifiés par leur clé technique (le numéro de dossier) ou la clé fonctionnelle. L'identification par clé technique suppose que celle-ci a été préalablement résolue à l'aide d'un message de sélection de population par exemple alors que l'identification par clé fonctionnelle permet de mettre à jour le dossier directement sans devoir transposer celle-ci en une clé technique.

Le service de mise à jour de dossiers permet de mettre à jour plusieurs dossiers pour une même requête. Evidemment, il faut que les dossiers soient de la même structure de données et que les informations manipulées sur chacun de ces dossiers soient manipulables par le processus HR Access utilisé. De plus, il est impératif que la manière d'identifier les dossiers (clé technique ou fonctionnelle) soit la même pour tous les dossiers : il n'est pas possible de mélanger les types de clé.

C'est la classe abstraite com.hraccess.openhr.PrimaryKey qui représente le concept de clé primaire identifiant un dossier. Cette classe possède deux sous-classes IntegerPrimaryKey et CompositePrimaryKey (du package com.hraccess.openhr). La classe IntegerPrimaryKey représente une clé primaire de type entier ; il s'agit du numéro de dossier (la clé technique) alors que la classe CompositePrimaryKey représente une clé composite formée par un n-uplet de valeurs correspondant aux valeurs de rubriques argument de tri ; il s'agit de la clé fonctionnelle.

L'exemple suivant montre comment identifier un dossier par sa clé technique ou fonctionnelle.

```
// Creating a technical key to identify a given HR Access dossier (with
   dossier number 1000)

PrimaryKey technicalKey = new IntegerPrimaryKey(1000);


// Creating a functional key to identify a given HR Access dossier

PrimaryKey functionalKey = new
   CompositePrimaryKey().addValue(“HRA”).addValue(“123456”);
```

Précisons qu'il n'est pas possible de mettre à jour une seule rubrique d'occurrence, la mise à jour porte au minimum sur l'occurrence entière. Cela signifie que lors d'une modification d'occurrence, il faut fournir la valeur de toutes les rubriques de l'information. En conséquence, avant de chercher à mettre à jour une occurrence d'information il faut déjà la lire depuis le serveur HR Access (à l'aide du service GET_DOSSIER_DATA). Le seul cas valide où il est possible d'émettre un message de mise à jour de dossiers sans avoir préalablement lu les données à mettre à jour correspond au cas de création d'occurrence. Dans ce cas en effet, l'utilisateur est obligé de fournir la valeur de toutes les rubriques de l'occurrence à créer.

L'exemple suivant montre comment créer une nouvelle occurrence de nationalité de salarié (information ZY12 (Nationalité)).

```
// Retrieving an already connected user (not detailed here)
IHRUser user = getUser();

// Retrieving the user's main conversation to send messages
IHRConversation conversation = user.getMainConversation();

// Retrieving one of the user's roles to send the message
IHRRole role = user.getRole("EMPLOYEE(123456)");

// Creating a key to identify the dossier to update
PrimaryKey technicalKey = new IntegerPrimaryKey(1000);

// Creating a (user) request message to update an employee dossier
HRMsgUpdateDossier request = new HRMsgUpdateDossier();
request.setProcessName("FS001");
request.setDataStructureName("ZY");
request.setIgnoreSeriousWarnings(true);
request.setPrimaryKeyType(PrimaryKey.INTEGER);
request.setUpdateMode(UpdateMode.NORMAL);

// Creating update request for dossier with given number
HRMsgUpdateDossier.Dossier dossierToUpdate = new
    HRMsgUpdateDossier.Dossier(technicalKey);

// Creating a data section to represent an employee's nationality (ZY12)
HRMsgUpdateDossier.Section dataSectionToUpdate = new
    HRMsgUpdateDossier.Section("12");
dossierToUpdate.addSection(dataSectionToUpdate);
```

```
// Data section ZY12 has 7 items (first item is mandatory, items with index 1-6 are optional)
Object[] values = new Object[7]; // The array must be large enough to contain all the data section's items
values[0] = “FRA”; // Item “Nationality” (mandatory)
// Skipping items with index 1-3 (optional)
values[4] = “1”; // Item “Is it the main nationality ?” (optional)
values[5] = java.sql.Date.valueOf(“2008-01-01”); // Item “Start date” (optional)
values[6] = java.sql.Date.valueOf(“2010-12-31”); // Item “End date” (optional)

// Creating a request to create a new occurrence of data section ZY12
HRMsgUpdateDossier.Row occurrenceToCreate = new HRMsgUpdateDossier.Row(HRMsgUpdateDossier.ROW_INSERT);
occurrenceToCreate.setValues(values);

request.addDossier(dossierToUpdate);

// Sending message via the user's conversation (synchronous task) by using the given role
HRResultUpdateDossier result = (HRResultUpdateDossier) conversation.send(request, role);

// Dumping result
System.out.println(result.getDossierUpdateCount() + “ dossier(s) were updated”);

HRResultUpdateDossier.DossierUpdate[] dossierUpdates = result.getDossierUpdates();

for(int i=0 ; i<dossierUpdates.length ; i++) {
    DossierUpdate dossierUpdate = dossierUpdates[i];

    System.out.println(“Dossier with number ” + dossierUpdate.getNudoss() + “ has been updated”);
    System.out.println(“Dossier is mapped to rule system ” + dossierUpdate.getRuleSystem());
    System.out.println(“Dossier's last update timestamp is ” + dossierUpdate.getLastUpdateTimestamp());
```

```
System.out.println(dossierUpdate.getDataSectionUpdateCount() + " data
section(s) have been updated");


for(Iterator it=dossierUpdate.getDataSectionUpdates(); it.hasNext(); ) {
HRResultUpdateDossier.DataSectionUpdate dataSectionUpdate =
(DataSectionUpdate) it.next();


System.out.println("Data section " +
dataSectionUpdate.getDataSectionName() + " has been updated");
System.out.println("Data section has " +
dataSectionUpdate.getRowUpdateCount() + " row update(s)");


for(Iterator iter=dataSectionUpdate.getRowUpdates(); iter.hasNext(); )
{
        HRResultUpdateDossier.RowUpdate rowUpdate = (RowUpdate)
iter.next();


        System.out.println("The row's new values are " +
rowUpdate.getValues());
        System.out.println("The row's line number is " +
rowUpdate.getNulign());
}
}
}
```

Explications de l'exemple

On souhaite mettre à jour un dossier de salarié (de structure de données ZY) à l'aide du processus FS001 (Informations sur le personnel).

Ce dossier est identifié par son numéro de dossier qui vaut 1000. Celui-ci a été résolu précédemment par une requête de sélection de population par exemple (non présente dans l'exemple). Il est tout à fait possible, ici, d'utiliser la clé fonctionnelle du dossier pour l'identifier.

La mise à jour est configurée afin de ne pas prendre en compte les "erreurs avec confirmation" ; seules les erreurs bloquantes seront reportées au client OpenHR.

La requête de mise à jour est configurée de sorte à identifier les dossiers par leur clé technique (les PrimaryKeys utilisées sont en fait des IntegerPrimaryKeys).

La mise à jour s'effectue en mode "NORMAL" : le serveur HR Access va retourner, dans la réponse, le résultat de mise à jour des dossiers en base de données avec les éventuelles erreurs levées lors de l'opération.

On indique que l'on souhaite mettre à jour le dossier de numéro 1000 en créant une instance de classe HRMsgUpdateDossier.Dossier et en lui passant une IntegerPrimaryKey adéquate.

On indique que l'on souhaite mettre à jour l'information Nationalité (ZY12) du dossier de numéro 1000 en créant une instance de classe HRMsgUpdateDossier.Section et en la rattachant à l'instance de HRMsgUpdateDossier.Dossier (de numéro de dossier 1000).

Puisque l'on cherche à créer une occurrence d'information ZY12, il faut obligatoirement passer au serveur les valeurs pour les rubriques de l'information. En fait, certaines rubriques sont optionnelles, si bien qu'il n'est pas obligatoire de fournir une valeur pour toutes. Cependant, il est primordial que ces valeurs soient passées sous forme de tableau d'Objects et que ce tableau ait une taille suffisante pour contenir la valeur de toutes les rubriques de l'information. Dans notre cas, l'information ZY12 définit 7 rubriques mais on n'en valorise que 4 (NATION (Nationalité), FLPRCI (Témoin de nationalité principale), DTBG00 (Date de début) et DTEN00 (Date de fin)). On crée donc un tableau de taille 7 en plaçant les valeurs à prendre en compte aux indices correspondant aux rubriques valorisées.

Puis la demande de création d'occurrence est rattachée à l'instance de HRMsgUpdateDossier.Section.

Enfin, la demande de modification du dossier (HRMsgUpdateDossier.Dossier) est rattachée au message de requête.

La requête de mise à jour est envoyée au serveur HR Access via la conversation de l'utilisateur et avec le rôle sélectionné. L'API OpenHR retourne le résultat sous forme de HRResultUpdateDossier.

La classe HRResultUpdateDossier retourne le résultat de mise à jour des données sous forme d'une arborescence. A partir du résultat (la racine de l'arbre), on peut récupérer les mises à jour de dossiers sous forme de HRResultUpdateDossier.DossierUpdate. A partir d'une mise à jour de dossier, on peut accéder aux mises à jour sur informations (de ce dossier) sous forme de HRResultUpdateDossier.DataSectionUpdate. Enfin, pour une mise à jour d'information, on peut accéder aux mises à jour sur occurrences sous forme de HRResultUpdateDossier.RowUpdate.

L'exemple montre comment itérer sur le résultat retourné afin de déterminer ce qui a été mis à jour sur le serveur.

Cet exemple est le plus simple qu'il est possible d'écrire car il concerne le cas d'une création d'occurrence dont l'information comporte peu de rubriques. La principale difficulté pour mettre à jour des dossiers de cette manière provient de la manière dont les valeurs des rubriques doivent être passées sous forme de tableau d'Objects : le tableau doit être bien dimensionné et les valeurs de rubriques placées aux bons indices dans celui-ci.

Modification d'une occurrence d'information

Cela se complique si l'on souhaite modifier une occurrence d'information. En effet, la mise à jour portant au niveau le plus fin sur l'occurrence et non sur la rubrique, une modification d'occurrence signifie qu'il faut renseigner toutes les valeurs de rubrique de l'information à mettre à jour.

Cela signifie qu'il faudra d'abord lire l'occurrence d'information à modifier, récupérer la valeur de ses rubriques, modifier ces valeurs puis les retourner au serveur HR Access dans une requête de mise à jour. Si cela n'est pas fait, les rubriques pour lesquelles aucune valeur n'est fournie par le client OpenHR seront écrasées en base de données bien qu'elles possèdent une valeur en table.

Il n'est donc pas possible de mettre à jour une occurrence de dossier sans devoir la lire au préalable.

Dans le cas de modification d'une occurrence, il conviendra de positionner l'horodatage de lecture de l'information depuis le serveur HR Access (afin que celui-ci puisse détecter si une mise à jour concurrente n'est pas survenue dans l'intervalle) à l'aide de la méthode HRMsgUpdateDossier.Section.setTimestamp(long). Il faudra aussi indiquer le numéro de ligne de l'occurrence à modifier via la méthode HRMsgUpdateDossier.Row.setNulign(int).

Dans le cas d'une suppression d'occurrence, il faut préciser l'horodatage de lecture de l'information depuis le serveur HR Access (afin que celui-ci puisse détecter si une mise à jour concurrente n'est pas survenue dans l'intervalle) et le numéro de ligne de l'occurrence à supprimer. Les valeurs de rubrique ne servent à rien dans ce cas-là.

Duplication de dossiers

Le service de duplication de dossiers (DUPLICATE_DOSSIER) permet de demander au serveur HR Access de dupliquer un dossier HR Access identifié par sa clé fonctionnelle. Cette technique ne doit pas être confondue avec le clonage de dossier tel qu'il a été présenté précédemment.

Les classes de message associées à ce service sont HRMsgDuplicateDossier et HRResultDuplicateDossier. Il s'agit d'un message de requête utilisateur (à envoyer à l'aide d'une conversation et d'un rôle) car il porte sur des données applicatives.

Pour dupliquer un dossier, il faut utiliser un processus HR Access de qualification "Gestion de dossiers" qui référence, en tant que processus élémentaire, un processus de qualification "Duplication de dossiers". C'est le cas du processus AS511 (Gestion des emplois) qui référence le processus AS5D1 (Duplication d'emploi).

L'exemple suivant montre comment dupliquer un dossier d'emploi (de structure de données ZC).

```
// Retrieving an already connected user (not detailed here)
IHRUser user = getUser();

// Retrieving the user's main conversation to send messages
IHRConversation conversation = user.getMainConversation();

// Retrieving one of the user's roles to send the message
IHRRole role = user.getRole("EMPLOYEE(123456)");

// Creating a new (user) request message to duplicate a HR Access dossier
HRMsgDuplicateDossier request = new HRMsgDuplicateDossier();
request.setProcessName("AS511");
request.setDataStructureName("ZC");
request.setIgnoreSeriousWarnings(true);
request.setSourceKey(new
    CompositePrimaryKey().addValue("000000000").addValue("JOBID00001
    "));
request.setTargetKey(new
    CompositePrimaryKey().addValue("000000000").addValue("JOBID00002
    "));

// Sending message via the user's conversation (synchronous task) by using
    the given role
HRResultDuplicateDossier result = (HRResultDuplicateDossier)
    conversation.send(request, role);

// Dumping result
System.out.println("The number associated to the duplicated dossier is " +
    result.getNudoss());
```

Pour dupliquer un dossier il faut indiquer :

- La structure de données du dossier à dupliquer
- Le processus (de qualification "gestion de dossiers") utilisé pour la duplication
- La clé fonctionnelle du dossier source (celui à dupliquer)
- La clé fonctionnelle attribuée au dossier cible (le dossier créé par duplication)

En retour d'invocation, le service retourne le numéro du dossier créé.

Notez que ce service ne permet la duplication que d'un seul dossier à la fois.

Habillage de requêtes SQL

Le service d'habillage de requêtes SQL (DRESS_SQL_STATEMENT) permet de modifier un ordre SQL pour le rendre conforme à la confidentialité d'un rôle donné. Cette technique est extrêmement pratique car elle permet au développeur d'accéder à la base de données HR Access via JDBC tout en bénéficiant de la confidentialité gérée par HR Access. Cette technique est utilisée par le serveur HRD Query lorsqu'il a besoin de récupérer les données depuis la base HR Access : les ordres SQL sont "habillés" par le serveur HR Access et utilisés tels quels pour lire la base de données via JDBC.

Les classes de message associées à ce service se nomment HRMsgDressSqlStatement et HRResultDressSqlStatement. Il s'agit d'un message anonyme car il doit pouvoir être exécuté sans requérir la connexion d'un utilisateur.

```
// Retrieving an already connected session (not detailed here)
IHRSession session = getSession();


// Creating a new (anonymous) request message to dress some SQL statements
HRMsgDressSqlStatement request = new HRMsgDressSqlStatement();
request.addStatement("SELECT A.* FROM ZY00 A"); // Data extraction order
request.addStatement("ZY", "SELECT A.NUDOSS FROM ZY00 A"); // Population selection order
request.setActivity("MYACTIV"); // Optional
request.setLanguage("F");
request.setPgp("000000000");
request.setRoleTemplate("EMPLOYEE");
request.setRoleParameter("123456");


// Sending message via the session (synchronous task)
HRResultDressSqlStatement result = (HRResultDressSqlStatement) session.sendMessage(request);


// Processing result
System.out.println(result.getStatementCount() + " statement(s) have been dressed");


for(Iterator it=result.getStatementsAsList(); it.hasNext(); ) {
    String statement = it.next();
    System.out.println("The HR Access server dressed the SQL statement <" + statement + ">");
}
```

Lorsqu'on souhaite habiller une requête SQL, il faut préciser :

- Le modèle de rôle au titre duquel habiller la requête

- Le paramètre de rôle au titre duquel habiller la requête
- Un code langue. Celui-ci est nécessaire si et seulement si la requête utilise le mot-clé <USERLANG> (voir le paragraphe "Le filtre SQL" dans le chapitre "Les packages de l'API")
- Un code PGP qui indique un code partition HR Access
- Une éventuelle activité au titre de laquelle habiller la requête

La classe HRMsgDressSqlStatements permet de traiter deux types de requête :

- Les requêtes d'extraction de données
- Les requêtes de sélection de dossiers.

Les premières sont de la forme « SELECT A.* FROM ZY00 A » et sont utilisées afin de récupérer des données en table. Les secondes sont utilisées afin de sélectionner une population de dossiers et doivent sélectionner le numéro de dossier (NUDOSS) en tant que première colonne, elles sont de la forme « SELECT A.NUDOSS FROM ZY00 A ». Pour ce dernier type de requête, il est impératif de préciser la structure de données des dossiers à sélectionner, c'est pourquoi la méthode addStatement() prend deux paramètres.

L'habillage d'ordre SQL retourne des ordres modifiés compatibles avec la confidentialité HR Access.

Exemple : L'habillage de la requête « SELECT A.* FROM ZY00 A » retournera « SELECT A.* FROM ZY00 A WHERE 0=1 » si le rôle ne permet pas de lire l'information ZY00.

Autres services

L'API OpenHR comporte de nombreux services implémentés sous forme d'une paire de classe de requête / message dans le package com.hraccess.openhr. Tous ne présentent pas un intérêt majeur du point de vue du développeur final. Citons tout de même le service GET_BLOBS qui permet de récupérer des BLOBs stockés soit en base de données HR Access soit dans un volume d'archivage. Concernant la mise à jour des BLOBs, celle-ci est supportée par le service UPDATE_DOSSIER présenté précédemment. Reportez-vous à la javadoc des classes de message concernées pour comprendre comment mettre à jour des BLOBs via l'API OpenHR.

CHAPITRE 7

La topologie système

A propos de ce chapitre

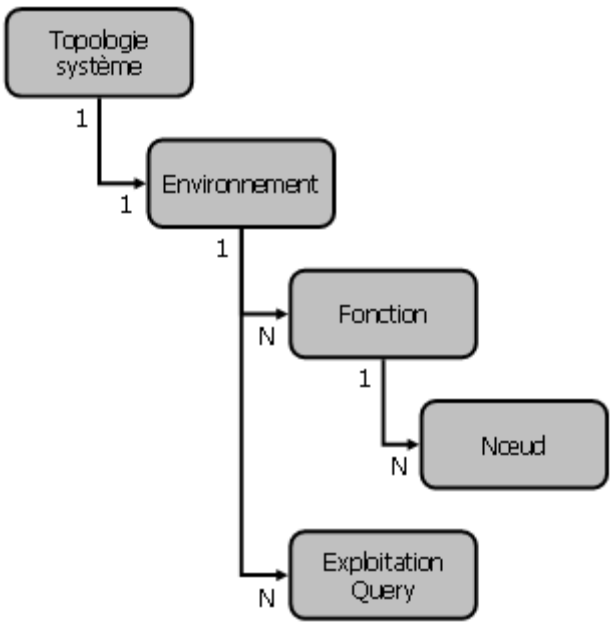
Ce chapitre présente l'objet Topologie système, et détaille sa représentation, ainsi que celle de ses éléments constitutifs, dans OpenHR, en particulier :

- Environnements
- Fonctions
- Nœuds
- Diagramme d'exploitation

Définition

La **Topologie système** (ou Topologie) - que l'on configure via Design Center - représente la cartographie de l'environnement HR Access en termes de fonctions et de nœuds.

La topologie système peut se représenter sous la forme d'un arbre :



Dans cet arbre, la racine représente la Topologie système elle-même.

A une topologie est associé un et un seul nœud représentant l'environnement HR Access. Cet environnement représente l'environnement HR Access sur lequel est configurée la Topologie.

Le nœud environnement possède plusieurs nœuds enfants représentant les fonctions de l'environnement et les diagrammes d'exploitation Query.

Enfin, à une fonction sont associé(s) zéro, un ou plusieurs nœuds.

Représentation de la Topologie système

La Topologie système est représentée par l'interface com.hraccess.openhr.topology.ISystemTopology.

Pour récupérer la topologie, il faut appeler la méthode IHRSession.getSystemTopology().

```
// Retrieving an existing session (not detailed here)
IHRSession session = getSession();

// Retrieving the system topology
ISystemTopology topology = session.getSystemTopology();
```

Attention cependant car comme la Topologie contient des mots de passe (JDBC et FTP), sa récupération depuis le serveur HR Access est sécurisée.

En fait, le service GET_SYSTEM_TOPOLOGY qui permet de la récupérer est de type privilège ; il va donc transiter via un canal dédié. Dans un environnement de production, ce canal de transmission doit être configuré (au niveau du serveur OpenHR), de sorte à authentifier mutuellement le client et le serveur OpenHR à l'aide d'un SSL bi-directionnel. Si cette sécurité n'est pas prise, n'importe qui peut récupérer la Topologie système, récupérer les configurations JDBC et FTP et accéder à la base de données HR Access. Voilà pourquoi l'appel à la méthode IHRSession.getSystemTopology() peut potentiellement échouer (si le client OpenHR travaille sur un environnement où les messages privilèges sont sécurisés par SSL2 et qu'il n'est pas un client authentifié auprès du serveur OpenHR).

La récupération de la Topologie système est une opération lourde car elle implique une phase de déchiffrage RSA.

Les méthodes utiles de cette interface sont listées ci-dessous.

Méthode	Rôle
IEnvironment getEnvironment()	Retourne l'environnement rattaché à cette Topologie système.
boolean isPasswordProtected()	Indique si l'accès à la console d'administration est sécurisé par un mot de passe.
boolean isPasswordValid(String)	Indique si le mot de passe donné correspond au mot de passe de sécurisation de l'accès à la console d'administration.

L'environnement

L'environnement HR Access est représenté par l'interface com.hraccess.openhr.topology.IEnvironment.

Les méthodes utiles de cette interface sont listées ci-dessous.

Méthode	Rôle
String getDatabaseTablePrefix()	Retourne le préfixe utilisé pour qualifier les tables relationnelles HR Access. Cette information est nécessaire si l'on souhaite accéder directement via JDBC à la base de données HR Access afin de préfixer les noms de tables. Par défaut, celui-ci est alimenté à "HR".
IFunction getDefaultFunction(Type)	Retourne la fonction marquée "par défaut" pour le type de fonction donné. Sur l'environnement, pour chaque type de fonction, il n'existe qu'une et une seule fonction marquée "par défaut".
IHRDesignServerFunction getDesignServerFunction()	Retourne la fonction de type "HR Design Server". L'environnement possède obligatoirement une et une seule fonction de ce type.
IFunction getFunction(String)	Retourne la fonction de nom donné. Le nom des fonctions est unique si bien que deux fonctions de type différent ne peuvent porter le même nom.
int getFunctionCount()	Retourne le nombre de fonctions définies pour l'environnement.
Map<String, IFunction> getFunctionMap()	Retourne les fonctions de l'environnement sous forme de Map<String, IFunction> référencées par leur nom.
List<IFunction> getFunctions()	Retourne les fonctions de l'environnement sous forme de List<IFunction>.
List<IFunction> getFunctions(Type)	Retourne les fonctions de l'environnement sous forme de List<IFunction> de type donné.
String getLabel()	Retourne le libellé de l'environnement.
String getLogicalPlatform()	Retourne le nom de la plate-forme logique HR Access associée à l'environnement.

Méthode	Rôle
String getName()	Retourne le nom de l'environnement.
IQueryExploitation getQueryExploitation(String)	Retourne le diagramme d'exploitation de nom donné.
int getQueryExploitationCount()	Retourne le nombre de diagrammes d'exploitation définis pour l'environnement.
Map<String, IQueryExploitation> getQueryExploitationMap()	Retourne les diagrammes d'exploitation sous forme de Map<String, IQueryExploitation> référencés par nom.
List<IQueryExploitation> getQueryExploitations()	Retourne les diagrammes d'exploitation sous forme de List< IQueryExploitation>.
boolean hasFunction(String)	Indique si la fonction de nom donné existe.

La fonction

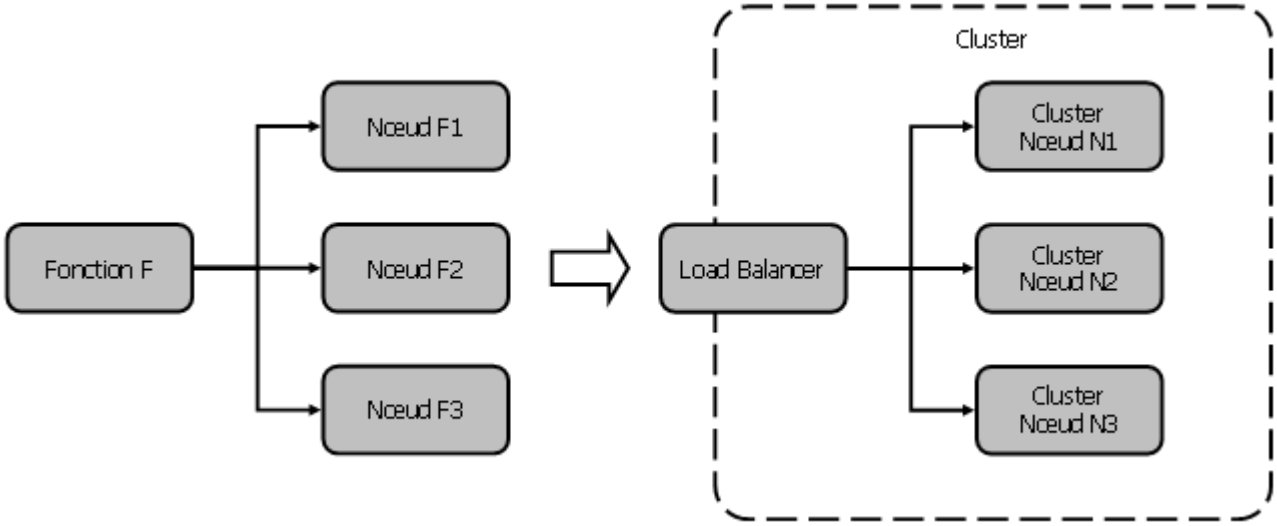
La fonction est représentée par l'interface com.hraccess.openhr.topology.IFunction.

La fonction possède un type représenté par la classe IFunction.Type. Il existe cinq types de fonction possibles :

- Le type "volume d'archivage" représenté par la constante IFunction.Type.ARCHIVE_VOLUME
- Le type "HR Design Server" représenté par la constante IFunction.Type.HR_DESIGN_SERVER
- Le type "HRa Space" représenté par la constante IFunction.Type.HRA_SPACE.
- Le type "OpenHR Server" représenté par la constante IFunction.Type.OPENHR_SERVER
- Et le type "Query Server" représenté par la constante IFunction.Type.QUERY_SERVER

On le voit aux différents types possibles, la fonction représente une brique de l'architecture HR Access. Une fonction doit être considérée comme la partie logique d'une application alors que le nœud (présenté plus loin) est sa partie physique. En d'autres termes, peut-être plus parlants, le fonction représente l'application logique (le cluster) alors que le nœud représente l'instance physique de l'application (le nœud de cluster). Cette distinction explique pourquoi certaines propriétés sont portées par la fonction et d'autres par le nœud : les propriétés utiles pour contacter l'application logique sont portées par la fonction alors que le nœud porte les propriétés spécifiques à une instance de l'application.

La fonction représente la façade visible d'une application pour les autres fonctions de l'environnement. Dans un déploiement en cluster, la fonction représente le répartiteur de charge central alors que les nœuds de fonction représentent les instances de l'application déployées sur les nœuds du cluster :



Seules les fonctions de type "OpenHR Server", "HRa Space" et "Query Server" peuvent être mises en cluster. Pour les fonctions "volume d'archivage" et "HR Design Server", tout se passe comme si la fonction ne possédait qu'un nœud (cluster avec un seul nœud). C'est pourquoi Design Server ne permet pas de créer de nœud pour ce type de fonction et ne fait apparaître dans l'interface que le concept de fonction.

Les méthodes utiles de cette interface sont listées ci-dessous.

Méthode	Rôle
IEnvironment getEnvironment()	Retourne l'environnement auquel cette fonction est rattachée.
String getName()	Retourne le nom de cette fonction. Ce nom est unique, deux fonctions de type différent ne peuvent porter le même nom.
INode getNode(String)	Retourne le nœud de nom donné rattaché à cette fonction.
int getNodeCount()	Retourne le nombre de nœuds rattachés à cette fonction.
Map<String, INode> getNodeMap()	Retourne les nœuds rattachés à cette fonction sous forme de Map<String, INode> et référencés par nom.
List< INode> getNodes()	Retourne les nœuds rattachés à cette fonction sous forme de List<INode>.
Type getType()	Retourne le type de cette fonction. Voir la classe IFunction.Type.
boolean hasNode(String)	Indique si la fonction possède un nœud de nom donné.
boolean isClusterable()	Indique si la fonction peut être déployée en cluster. Seules les fonctions de type "OpenHR Server", "HRa Space" et "Query Server" peuvent l'être.
boolean isClustered()	Indique si la fonction est déployée en cluster.
boolean isDefaultFunction()	Indique si cette fonction est la fonction marquée "par défaut" pour son type.

La fonction volume d'archivage

La classe com.hraccess.openhr.topology.IArchiveVolumeFunction représente une fonction de type "volume d'archivage".

La seule méthode définie sur cette classe (getFtpLink()) permet de récupérer le paramétrage FTP pour accéder au volume d'archivage sous forme de com.hraccess.openhr.topology.IFtpLink.

La fonction HRa Space

La classe com.hraccess.openhr.topology.IHRaSpaceFunction représente une fonction de type "HRa Space".

Les méthodes utiles de cette interface sont listées ci-dessous.

Méthode	Rôle
String getCognosURL()	Retourne l'URL d'accès à l'application Cognos depuis HRa Space.
String getHostName()	Retourne le nom d'hôte de la machine hébergeant HRa Space.
IHttpConnector getHttpConnector()	Retourne le connecteur http permettant de se connecter à HRa Space.
String getScope()	Retourne le HRa Scope associé à la fonction HRa Space.

La fonction HR Design Server

La classe com.hraccess.openhr.topology.IHRDesignServerFunction représente une fonction de type "HR Design Server".

Les méthodes utiles de cette interface sont listées ci-dessous.

Méthode	Rôle
IJdbcConnector getJdbcConnector()	Retourne le connecteur JDBC pour se connecter à la base de données HR Access.
Set<String> getAcceptedTrees()	Retourne le nom des arbres Web acceptant d'autres arbres avec la même localisation dans un HRa Scope.

La fonction OpenHR Server

La classe com.hraccess.openhr.topology.IOpenHRServerFunction représente une fonction de type "OpenHR Server".

Cette interface ne déclare aucune méthode supplémentaire par rapport à IFunction.

La fonction Query Server

La classe com.hraccess.openhr.topology.IQueryServerFunction représente une fonction de type "Query Server".

Les méthodes utiles de cette interface sont listées ci-dessous.

Méthode	Rôle
IFtpLink getFtpLink()	Retourne le paramétrage FTP pour communiquer avec le serveur HRD Query. Cela permet d'émettre, vers le serveur HRD Query, des fichiers de commande pour générer des rapports.
String getHostName()	Retourne le nom d'hôte de la machine hébergeant le serveur HRD Query.
int getRmiPort()	Retourne le numéro de port de la RMI registry du serveur HRD Query.
ITcpConnector getTcpConnector()	Retourne le connecteur TCP utilisé pour invoquer le service de "préselection de population" du serveur HRD Query.
int getTPQueryServicePort()	Retourne le numéro de port du service RMI de "soumission synchrone de rapport" du serveur HRD Query.

Le nœud

Le nœud est représenté par l'interface com.hraccess.openhr.topology.INode.

Les méthodes utiles de cette interface sont listées ci-dessous.

Méthode	Rôle
String getName	Retourne le nom de ce nœud.
IFunction getFunction()	Retourne la fonction à laquelle ce nœud est rattaché.

L'interface INode est sous-classée en 3 sous-interfaces correspondant aux fonctions qui peuvent avoir des nœuds, c'est-à-dire qui peuvent être déployées en cluster.

Le nœud HRa Space

Un nœud HRa Space est représenté par l'interface com.hraccess.openhr.topology.IHRaSpaceNode.

Les méthodes utiles de cette interface sont listées ci-dessous.

Méthode	Rôle
String getHostName()	Retourne le nom d'hôte de la machine hébergeant l'instance d'application HRa Space.
String getConnectionURL()	Retourne l'URL de connexion à l'application HRa Space. Cette URL est typiquement celle utilisée par le répartiteur de charge frontal pour router les requêtes clientes aux différents nœuds du cluster (et pas l'URL utilisée par le navigateur client !).

Le nœud OpenHR Server

Un nœud OpenHR Server est représenté par l'interface com.hraccess.openhr.topology.IOpenHRServerNode.

Les méthodes utiles de cette interface sont listées ci-dessous.

Méthode	Rôle
String getHostName()	Retourne le nom d'hôte de la machine hébergeant l'instance de serveur OpenHR.
int getAdminPort()	Retourne le numéro de port utilisé pour administrer cette instance de serveur OpenHR.

Le nœud Query Server

Un nœud Query Server est représenté par l'interface com.hraccess.openhr.topology.IQueryServerNode.

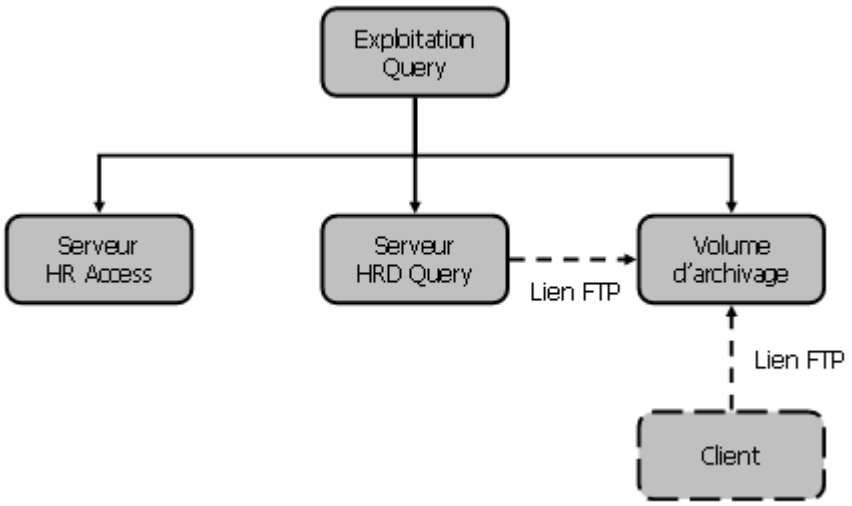
Les méthodes utiles de cette interface sont listées ci-dessous.

Méthode	Rôle
int getAdminPort()	Retourne le numéro de port utilisé pour administrer cette instance de serveur HRD Query.
String getHostName()	Retourne le nom d'hôte de la machine hébergeant l'instance de serveur HRD Query.
String getMailSenderAddress()	Retourne l'adresse mail de l'émetteur des mails envoyés par le serveur HRD Query.
String getSmtpHost()	Retourne le nom d'hôte du serveur SMTP vers lequel seront émis les mails.
int getSmtpPort()	Retourne le numéro de port du service SMTP vers lequel seront émis les mails.
int getTcpPort()	Retourne le numéro de port du service de présélection de population.

Le diagramme d'exploitation

Un diagramme d'exploitation lie une fonction "HR Design Server", une fonction "Query Server" et une fonction "volume d'archivage".

Un diagramme d'exploitation lit un serveur HR Access, un serveur HRD Query et un volume d'archivage. Des liens permettent de récupérer la configuration FTP pour accéder au volume d'archivage depuis le serveur HRD Query ou une application cliente externe :



Le diagramme d'exploitation est utilisé par le serveur HR Access pour déterminer à quel serveur HRD Query il doit adresser les états produits par l'exécution d'une demande de travail (Structure de données ZO (Opérations)) et par le serveur HRD Query pour déterminer dans quel volume d'archivage il doit archiver les états produits. Il fournit également deux liens. Le premier (du serveur HRD Query vers le volume d'archivage) est utilisé pour archiver les rapports produits par le serveur HRD Query. Le second (d'un client vers le volume d'archivage) est utilisé pour récupérer les rapports archivés.

Les méthodes utiles de cette interface sont listées ci-dessous.

Méthode	**Rôle**
IArchiveVolumeFunction getArchiveVolume()	Retourne la fonction de type "volume d'archivage" référencée par ce diagramme d'exploitation.
String getArchiveVolumeName()	Retourne le nom de la fonction de type "volume d'archivage" référencée par ce diagramme d'exploitation.
IFtpLink getArchiveVolumeToClientLink()	Retourne le lien FTP entre le client et le volume d'archivage.
IHRDesignServerFunction getDesignServer()	Retourne la fonction de type "HR Design Server" référencée par ce diagramme d'exploitation.
String getDesignServerName()	Retourne le nom de la fonction de type "HR Design Server" référencée par ce diagramme d'exploitation.
IEnvironment getEnvironment()	Retourne l'environnement HR Access auquel ce diagramme d'exploitation est rattaché.
String getName()	Retourne le nom de ce diagramme d'exploitation.
IQueryServerFunction getQueryServer()	Retourne la fonction de type "Query Server" référencée par ce diagramme d'exploitation.
String getQueryServerName()	Retourne le nom de la fonction de type "Query Server" référencée par ce diagramme d'exploitation.
IFtpLink getQueryServerToArchiveVolumeLink()	Retourne le lien FTP entre le serveur HRD Query et le volume d'archivage.

Index

H

I

J

L

M

N

O

P

R

S

T

U