



Kofax RPA

Guide du développeur

Version : 11.4.0

Date : 2022-11-18

KOFAX

© 2015–2022 Kofax. All rights reserved.

Kofax is a trademark of Kofax, Inc., registered in the U.S. and/or other countries. All other trademarks are the property of their respective owners. No part of this publication may be reproduced, stored, or transmitted in any form without the prior written permission of Kofax.

Table des matières

Préface.....	5
Documentation associée.....	5
Formation.....	6
Obtenir de l'aide pour les produits Kofax.....	7
Chapitre 1 : Guide du programmeur Java.....	8
Java basique.....	8
Premier exemple.....	9
Entrée de robot.....	11
Types d'attribut.....	12
Paramètres d'exécution.....	14
Bibliothèques de robots.....	15
Java avancé.....	17
Répartition de la charge et du basculement.....	17
Executor Logger.....	18
Streaming des données.....	19
SSL.....	22
Exécution en parallèle.....	23
Intégration de répertoire.....	24
Sous le capot.....	25
Fonctionnalités RequestExecutor.....	26
Applications web.....	26
Débogage API.....	27
API Repository.....	28
Dépendances.....	28
Client répertoire.....	28
Déploiement avec le client répertoire.....	31
API Repository REST.....	32
Chapitre 2 : Guide du programmeur .NET.....	40
.NET basique.....	40
Premier exemple.....	40
Entrée de robot.....	42
Types d'attribut.....	43
Paramètres d'exécution.....	46
Bibliothèques de robots.....	47

.NET avancé.....	49
Charger la distribution.....	50
Streaming des données.....	50
SSL.....	54
Intégration de répertoire.....	55
Executor Logger.....	56
Sous le capot.....	57
Fonctionnalités RequestExecutor.....	57
API Repository.....	58
Client répertoire.....	58
Déploiement avec le client répertoire.....	61
API Repository as REST.....	62
Affichage des tâches Management Console à l'aide d'une API.....	62
Chapitre 3 : API Management Console REST.....	65
Tâches.....	65

Préface

Les robots sont exécutés sur RoboServer via une API (Java ou .NET). Vous pouvez utiliser l'API directement dans votre propre application ou indirectement lorsque vous exécutez des robots en utilisant Management Console.

Ce guide se compose de trois parties :

- *Guide du programmeur Java*, qui décrit l'API à utiliser dans les programmes Java.
- *Guide du programmeur .NET*, qui décrit l'API à utiliser dans les applications .NET, y compris les programmes en C#.

La documentation de référence des API Java et .NET est disponible dans votre dossier de documentation hors ligne. La documentation de l'API Java est également disponible sur le site de documentation en ligne. Pour plus d'informations, voir le *Guide d'installation de Kofax RPA*.

Documentation associée

Le jeu de documents de Kofax RPA est disponible ici :¹

<https://docshield.kofax.com/Portal/Products/RPA/11.4.0-vcsft2fhaw/RPA.htm>

Outre ce guide, le jeu de documents comprend les éléments suivants :

Notes de mise à jour de Kofax RPA

Contient des informations de dernière minute et d'autres détails qui ne sont pas disponibles dans l'autre documentation Kofax RPA dont vous disposez.

Spécifications techniques de Kofax RPA

Contient des informations sur les systèmes d'exploitation pris en charge et les autres exigences du système.

Guide d'installation de Kofax RPA

Contient des instructions sur l'installation de Kofax RPA et de ses composants dans un environnement de développement.

¹ vous devez être connecté à Internet pour pouvoir accéder en ligne au jeu complet des documents. Pour y accéder sans connexion Internet, consultez le *Guide d'installation*.

Guide de mise à niveau de Kofax RPA

Contient les instructions de mise à niveau de Kofax RPA et de ses composants vers une version plus récente.

Guide de l'administrateur de Kofax RPA

Décrit les tâches administratives et de gestion dans Kofax RPA.

Aide de Kofax RPA

Explique comment utiliser Kofax RPA. L'aide est également disponible sous la forme d'un fichier PDF appelé *Guide de l'utilisateur Kofax RPA*.

Guide des meilleures pratiques Kofax RPA pour la gestion du cycle de vie du robot

Propose des méthodes et des techniques recommandées pour vous aider à optimiser vos performances et à garantir le succès de l'utilisation de la gestion du cycle de vie du robot dans votre environnement Kofax RPA.

Guide de mise en route de Kofax RPA pour la création de robots

Fournit un tutoriel qui explique l'utilisation de Kofax RPA pour créer un robot.

Kofax RPA - Guide de mise en route avec Document Transformation

Fournit un tutoriel qui explique comment utiliser la fonctionnalité Document Transformation dans un environnement Kofax RPA, y compris l'OCR, l'extraction, le formatage des champs et la validation.

Guide de configuration du Desktop Automation Service de Kofax RPA

Explique comment configurer le service Desktop Automation nécessaire pour utiliser Desktop Automation sur un ordinateur distant.

Documentation sur l'API d'intégration Kofax RPA

Contient des informations sur l'API Java Kofax RPA et l'API .NET Kofax RPA qui fournissent un accès programmatique au produit Kofax RPA. La documentation de l'API Java est disponible depuis la documentation Kofax RPA en ligne et hors ligne, tandis que la documentation de l'API .NET n'est disponible que hors ligne.

i Les API Kofax RPA comprennent de nombreuses références à RoboSuite, le nom d'origine du produit. Le nom RoboSuite est conservé dans les API pour assurer la rétrocompatibilité. Dans le contexte de la documentation des API, le terme RoboSuite est équivalent à Kofax RPA.

Formation

Kofax propose des formations en classe et sur ordinateur pour vous aider à tirer le meilleur parti de votre solution Kofax RPA. Viz le portail de formation Kofax à l'adresse suivante : <https://learn.kofax.com/> pour obtenir des détails sur les formations et les planifications disponibles.

Vous pouvez également consulter le Kofax Intelligent Automation SmartHub à l'adresse suivante : <https://smarthub.kofax.com/> pour explorer d'autres solutions, robots, connecteurs, etc.

Obtenir de l'aide pour les produits Kofax

Le répertoire de la [Kofax Knowledge Base \[Base de connaissances Kofax\]](#) contient des articles qui sont régulièrement mis à jour pour vous tenir informé sur les produits Kofax. Nous vous encourageons à utiliser la base de connaissances pour obtenir des réponses à vos questions sur les produits.

Pour accéder à la Kofax Knowledge Base :

1. Accédez à la page d'accueil du [site web Kofax](#) et sélectionnez **Assistance**.
2. Lorsque la page Assistance s'affiche, sélectionnez **Assistance client > Base de connaissances**.

i La Kofax Knowledge Base est optimisée pour une utilisation avec Google Chrome, Mozilla Firefox ou Microsoft Edge.

La Kofax Knowledge Base propose :

- Puissantes fonctionnalités de recherche pour vous aider à localiser rapidement les informations dont vous avez besoin.
Saisissez vos termes ou votre phrase à rechercher dans le champ **Search [Rechercher]**, puis cliquez sur l'icône Loupe.
- Informations produit, détails de configuration et documentation, notamment les actualités des nouvelles versions.
Faites défiler la page d'accueil de la Kofax Knowledge Base pour localiser une famille de produits. Cliquez ensuite sur le nom d'une famille de produits pour afficher une liste d'articles sélectionnés. Veuillez noter que certaines familles de produits nécessitent un identifiant valide du Kofax Portal [Portail Kofax] pour afficher ces articles sélectionnés.

Depuis la page d'accueil de la base de connaissances, vous pouvez :

- Accédez à la Kofax Community [Communauté Kofax] (pour tous les clients).
Cliquez sur le lien **Community [Communauté]** en haut de la page.
- Accédez au Kofax Customer Portal [Portail client Kofax] (pour les clients éligibles).
Cliquez sur le lien **Support [Assistance]** en haut de la page. Lorsque la Customer & Partner Portals Overview [Présentation des portails client et partenaire] s'affiche, cliquez sur **Log in to the Customer Portal [Se connecter au Portail client]**.
- Accédez au Kofax Partner Portal [Portail partenaire Kofax] (pour les partenaires éligibles).
Cliquez sur le lien **Support** en haut de la page. Lorsque la Customer & Partner Portals Overview s'affiche, cliquez sur **Log in to the Partner Portal [Se connecter au Portail partenaire]**.
- Accédez aux contrats d'assistance Kofax, aux politiques de cycle de vie, aux détails d'exécution électroniques et aux outils en libre-service.
Accédez à la section **General Support [Assistance générale]**, cliquez ensuite sur **Support Details [Détails de l'assistance]**, puis sélectionnez l'onglet approprié.

Chapitre 1

Guide du programmeur Java

Ce chapitre décrit comment exécuter des robots à l'aide de l'API Kofax RPA Java. Ce guide suppose que vous savez comment écrire des robots simples et que vous connaissez le langage de programmation Java.

 La méthode `printStackTrace` est obsolète dans Kofax Kapow version 9.6 et ultérieure.

Vous pouvez trouver des informations sur les classes Java spécifiques dans la section Interface de programmation des applications sur le site de documentation du produit Kofax RPA : <https://docshield.kofax.com/Portal/Products/RPA/11.4.0-vcsft2fhaw/RPA.htm>. Ces informations sont également disponibles dans votre dossier de documentation hors ligne. Pour plus d'informations, voir le *Guide d'installation Kofax RPA*.

Java basique

Les robots exécutés par la Management Console sont exécutés à l'aide de l'API Java, qui vous permet d'envoyer des requêtes à un `RoboServer` qui lui demande d'exécuter un robot particulier. Il s'agit d'une configuration client/serveur classique dans laquelle Management Console agit comme client et `RoboServer` comme serveur.

En utilisant l'API, toute application basée sur Java peut devenir un client pour `RoboServer`. En plus d'exécuter des robots qui stockent des données dans une base de données, vous pouvez également demander aux robots de renvoyer des données directement à l'application cliente. Voici quelques exemples :

- Utilisez plusieurs robots pour effectuer une recherche fédérée, qui regroupe les résultats de plusieurs sources en temps réel.
- Exécutez un robot en réponse à un événement à l'arrière-plan de votre application. Par exemple, exécutez un robot lorsqu'un nouvel utilisateur s'inscrit, pour créer des comptes sur des systèmes web non intégrés directement à l'arrière-plan de votre application.

Ce guide présente les classes principales et explique comment les utiliser pour exécuter des robots. Nous décrirons également comment fournir des entrées aux robots et contrôler leur exécution sur `RoboServer`.

L'API Java est un fichier JAR `/API/robosuite-java-api/lib/robosuite-api.jar` situé dans le dossier d'installation Kofax RPA. Voir « Dossiers importants » dans le *Guide d'installation* pour plus d'informations. Tous les exemples de ce guide se trouvent également dans `/API/robosuite-java-api/examples`. À côté de l'API Java, se trouvent cinq fichiers jar supplémentaires

contenant les dépendances externes de l'API. La plupart des tâches API de base telles que l'exécution de robots peuvent être effectuées sans utiliser aucune de ces bibliothèques tierces, tandis que certaines fonctionnalités avancées nécessitent l'utilisation d'une ou de plusieurs de ces bibliothèques. Les exemples de ce guide indiquent quand de telles bibliothèques sont requises.

- Le fichier JAR de l'API Java nécessite JAXP version 1.5, de sorte que les anciennes implémentations, telles que Xalan-Java 2.7.2, ne fonctionneront pas avec le fichier JAR Java.

Premier exemple

Voici le programme requis pour exécuter le robot nommé `NewsMagazine.robot`, qui se trouve dans le dossier `Tutoriels` du projet par défaut. Le robot écrit ses résultats à l'aide de l'activité d'étape `Valeur de retour`, ce qui facilite la gestion des données de sortie par programmation à l'aide de l'API. D'autres robots (généralement ceux exécutés dans une planification par `Management Console`) stockent leurs données directement dans une base de données à l'aide de l'activité d'étape `Stocker` dans la base de données, auquel cas les données collectées par le robot ne sont pas renvoyées au client API.

Dans l'exemple suivant, le robot `NewsMagazine` est exécuté et les données de sortie sont traitées par programmation.

Exécuter un robot sans entrée :

```
import com.kapowtech.robosuite.api.java.repository.construct.*;
import com.kapowtech.robosuite.api.java.rql.*;
import com.kapowtech.robosuite.api.java.rql.construct.*;

/**
 * Example that shows you how to execute NewsMagazine.robot from tutorial1
 */
public class Tutorial1 {

    public static void main(String[] args) throws ClusterAlreadyDefinedException {

        RoboServer server = new RoboServer("localhost", 50000);
        boolean ssl = false;
        Cluster cluster = new Cluster("MyCluster", new RoboServer[]{ server}, ssl);

        Request.registerCluster(cluster); // you can only register a cluster once per
                                         // application

        try {
            Request request = new Request("Library:/Tutorials/NewsMagazine.robot");
            request.setRobotLibrary(new DefaultRobotLibrary());
            RQLResult result = request.execute("MyCluster");

            for (Object o : result.getOutputObjectsByName("Post")) {
                RQLObject value = (RQLObject) o;
                String title = (String) value.get("title");
                String preview = (String) value.get("preview");
                System.out.println(title + ", " + preview);
            }
        }
    }
}
```

Le tableau suivant répertorie les classes impliquées et leurs responsabilités.

RoboServer	Il s'agit d'un objet de valeur simple qui identifie un RoboServer qui peut exécuter des robots. Chaque RoboServer doit être activé par un Management Console et le KCU doit lui être assigné avant utilisation.
Cluster	Un cluster est un groupe de RoboServers fonctionnant comme une seule unité logique.
Request	Cette classe est utilisée pour construire la requête du robot. Avant de pouvoir exécuter des requêtes, vous devez enregistrer un cluster avec la classe Request.
DefaultRobotLibrary	Une bibliothèque de robot indique à RoboServer où trouver le robot identifié dans la requête. Des exemples ultérieurs explorent les différents types de bibliothèque de robot et quand/comment les utiliser.
RQLResult	Cette classe contient le résultat d'une exécution du robot. Le résultat contient des réponses de valeur, avec les messages de journalisation et du serveur.
RQLObject	Chaque valeur renvoyée par un robot à l'aide de l'activité Valeur de retour est accessible en tant que RQLObject.

Les lignes suivantes indique à l'API que notre RoboServer s'exécute sur le port localhost 50000.

```
RoboServer server = new RoboServer("localhost", 50000);
```

Le programme suivant définit un cluster avec un seul RoboServer. Le cluster est enregistré avec la classe Request, ce qui vous permet d'exécuter des requêtes sur ce cluster. Chaque cluster ne peut être enregistré qu'une seule fois.

Enregistrement d'un cluster :

```
boolean ssl = false;
Cluster cluster = new Cluster("MyCluster", new RoboServer[]{ server}, ssl);
Request.registerCluster(cluster);
```

Le programme suivant crée une requête qui exécute le robot nommé `NewsMagazine.robot` et situé dans `Library:/Tutorials.Library:/`, faisant référence à la bibliothèque de robot configurée pour la requête. Ici est utilisée la `DefaultRobotLibrary`, qui demande à `RoboServer` de rechercher le robot dans le système de fichiers local pour le serveur. Voir [Bibliothèques de robots](#) pour plus d'informations sur l'utilisation des bibliothèques de robots.

```
Request request = new Request("Library:/Tutorials/NewsMagazine.robot");
request.setRobotLibrary(new DefaultRobotLibrary());
```

La ligne suivante exécute le robot sur le cluster nommé `MyCluster` (le cluster précédemment enregistré) et renvoie le résultat une fois l'exécution du robot terminée. Par défaut, `execute` déclenche une exception si le robot génère une exception API.

```
RQLResult result = request.execute("MyCluster")
```

Ici, nous traitons les valeurs extraites. Tout d'abord, nous récupérons toutes les valeurs extraites du type nommé `Post` et les parcourons. Pour chaque `RQLObject`, nous accédons aux attributs du type `Post` et nous affichons le résultat. Les attributs et les mappages sont traités dans une section ultérieure.

```
for (Object o : result.getOutputObjectsByName("Post")) {
    RQLObject value = (RQLObject) o;
    String title = (String) value.get("title");
    String preview = (String) value.get("preview");
    System.out.println(title + ": " + preview);
}
```

Entrée de robot

La plupart des robots exécutés via l'API sont paramétrés via une entrée, telle qu'un mot-clé de recherche ou des identifiants de connexion. L'entrée dans un robot fait partie de la requête envoyée à RoboServer et est fournie à l'aide de la méthode `createInputVariable` sur la requête.

Entrée utilisant RQLObjectBuilder implicite :

```
Request request = new Request("Library:/Input.robot");
request.createInputVariable("userLogin").setAttribute("username", "scott")
.setAttribute("password", "tiger");
```

Dans cet exemple, une requête est créée et `createInputVariable` est utilisé pour créer une variable d'entrée nommée `userLogin`. Ensuite, `setAttribute` est utilisé pour configurer les attributs Nom d'utilisateur et Mot de passe de la variable d'entrée.

L'exemple précédent est une notation abrégée courante, mais peut également être exprimé plus en détail à l'aide de RQLObjectBuilder :

Entrée utilisant RQLObjectBuilder explicite :

```
Request request = new Request("Library:/Input.robot");
RQLObjectBuilder userLogin = request.createInputVariable("userLogin");
userLogin.setAttribute("username", "scott");
userLogin.setAttribute("password", "tiger");
```

Les deux exemples sont identiques. Le premier invoque la méthode en cascade sur le `RQLObjectBuilder` anonyme et est donc plus court.

Lorsque RoboServer reçoit cette requête, les événements suivants se produisent :

- RoboServer charge `Input.robot` (quelle que soit la façon dont est configuré `RobotLibrary` pour la requête).
- RoboServer vérifie que le robot a une variable nommée `userLogin` et que cette variable est marquée comme entrée.
- RoboServer vérifie ensuite que les attributs configurés en utilisant `setAttribute` sont compatibles avec le type de variable `userLogin`. Par conséquent, le type doit avoir des attributs nommés Nom d'utilisateur et Mot de passe et ils doivent tous deux être des attributs textuels (la section suivante décrit le mappage entre l'API et les attributs Design Studio).
- Si toutes les variables d'entrée sont compatibles, RoboServer démarre l'exécution du robot.

Si un robot nécessite plusieurs variables d'entrée, vous devez toutes les créer pour exécuter le robot. Vous n'avez qu'à configurer les attributs requis ; tout attribut non requis que vous ne configurez pas via l'API aura une valeur nulle. Si vous avez un robot qui nécessite une connexion à Facebook et Twitter, vous pouvez définir l'entrée comme suit.

```
Request request = new Request("Library:/Input.robot");
request.createInputVariable("facebook").setAttribute("username", "scott")
.setAttribute("password", "facebook123");
request.createInputVariable("twitter").setAttribute("username", "scott")
.setAttribute("password", "twitter123");
```

Types d'attribut

Lorsque vous définissez un nouveau type dans Design Studio, sélectionnez un type pour chaque attribut. Certains attributs peuvent contenir du texte, comme du texte court, du texte long, un mot de passe, du HTML, du XML, et lorsqu'ils sont utilisés à l'intérieur d'un robot, il peut être nécessaire de stocker du texte dans ces attributs. Si vous stockez du texte dans un attribut XML, ce texte doit être un document XML valide. Cette validation se produit lorsque le type est utilisé dans un robot, mais comme l'API ne sait rien sur ce type, elle ne valide pas les valeurs d'attribut de la même manière. Par conséquent, l'API n'a que huit types d'attributs et Design Studio a 19 types disponibles. Ce tableau montre le mappage entre les types d'attribut Design Studio et l'API.

Mappage API sur Design Studio

Type d'attribut API	Type d'attribut Design Studio
Texte	Texte court, texte long, mot de passe, HTML, XML, propriétés, langue, pays, devise, clé de recherche
Entier	Entier
Booléen	Booléen
Nombre	Nombre
Caractère	Caractère
Date	Date
Session	Session
Binaire	Binaire, Image, PDF

Les types d'attributs API sont ensuite mappés sur Java de la manière suivante.

Types Java pour les attributs

Type d'attribut API	Classe Java
Texte	java.lang.String
Entier	java.lang.Long
Booléen	java.lang.Boolean
Nombre	java.lang.Double
Caractère	java.lang.Character
Date	java.util.Date
Session	com.kapowtech.robosuite.api.construct.Session
Binaire	com.kapowtech.robosuite.api.construct.Binary

La méthode `setAttribute` de `RQ1ObjectBuilder` est surchargée, vous n'avez donc pas besoin de spécifier explicitement le type d'attribut lors de la configuration d'un attribut via l'API, tant que la classe Java correcte est utilisée comme argument. Voici un exemple qui montre comment définir les attributs d'un objet avec tous les types d'attributs Design Studio possibles.

Utilisation recommandée de setAttribute :

```
Request request = new Request("Library:/AllTypes.robot");
RQLObjectBuilder inputBuilder = request.createInputVariable("AllTypes");
inputBuilder.setAttribute("anInt", new Long(42L));
inputBuilder.setAttribute("aNumber", new Double(12.34));
inputBuilder.setAttribute("aBoolean", Boolean.TRUE);
inputBuilder.setAttribute("aCharacter", 'c');
inputBuilder.setAttribute("aShortText", "some text");
inputBuilder.setAttribute("aLongText", "a longer test");
inputBuilder.setAttribute("aPassword", "secret");
inputBuilder.setAttribute("aHTML", "<html>text</html>");
inputBuilder.setAttribute("anXML", "<tag>text</tag>");
inputBuilder.setAttribute("aDate", new Date());
inputBuilder.setAttribute("aBinary", new Binary("some bytes".getBytes()));
inputBuilder.setAttribute("aPDF", (Binary) null);
inputBuilder.setAttribute("anImage", (Binary) null);
inputBuilder.setAttribute("aProperties", "name=value\nname2=value2");
inputBuilder.setAttribute("aSession", (Session) null);
inputBuilder.setAttribute("aCurrency", "USD");
inputBuilder.setAttribute("aCountry", "US");
inputBuilder.setAttribute("aLanguage", "en");
inputBuilder.setAttribute("aRefindKey", "Never use this a input");
```

L'exemple précédent utilise explicitement le nouveau `Long (42L)` et le nouveau `Double (12.34)`, bien que `42L` et `12.34` soient suffisants en raison du remplissage automatique. Notez également que nous devons convertir des valeurs nulles, car le compilateur Java ne peut pas autrement déterminer laquelle des méthodes `setAttribute` surchargées appeler. Toutefois, comme les attributs non configurés seront automatiquement nuls, vous n'avez jamais besoin de définir explicitement la valeur nulle.

Il est possible de spécifier explicitement l'attribut et le `AttributeType` lors de la création d'une entrée à l'aide de l'API. Cette approche n'est pas recommandée, mais peut être nécessaire dans de rares cas et ressemblerait à ce qui suit.

Utilisation erronée de setAttribute :

```
Request request = new Request("Library:/AllTypes.robot");
RQLObjectBuilder inputBuilder = request.createInputVariable("AllTypes");
inputBuilder.setAttribute(new Attribute("anInt", "42", AttributeType.INTEGER));
inputBuilder.setAttribute(new Attribute("aNumber", "12.34", AttributeType.NUMBER));
inputBuilder.setAttribute(new Attribute("aBoolean", "true", AttributeType.BOOLEAN));
inputBuilder.setAttribute(new Attribute("aCharacter", "c", AttributeType.CHARACTER));
inputBuilder.setAttribute(new Attribute("aShortText", "some text",
AttributeType.TEXT));
inputBuilder.setAttribute(new Attribute("aLongText", "a longer test",
AttributeType.TEXT));
inputBuilder.setAttribute(new Attribute("aPassword", "secret", AttributeType.TEXT));
inputBuilder.setAttribute(new Attribute("aHTML", "<html>bla</html>",
AttributeType.TEXT));
inputBuilder.setAttribute(new Attribute("anXML", "<tag>text</tag>",
AttributeType.TEXT));
inputBuilder.setAttribute(new Attribute("aDate", "2012-01-15 23:59:59.123",
AttributeType.DATE));
inputBuilder.setAttribute(new Attribute("aBinary",
Base64Encoder.encode("some bytes".getBytes()), AttributeType.BINARY));
inputBuilder.setAttribute(new Attribute("aPDF", null, AttributeType.BINARY));
inputBuilder.setAttribute(new Attribute("anImage", null, AttributeType.BINARY));
inputBuilder.setAttribute(new Attribute("aProperties", "name=value\nname2=value2",
AttributeType.TEXT));
inputBuilder.setAttribute(new Attribute("aSession", null, AttributeType.SESSION));
inputBuilder.setAttribute(new Attribute("aCurrency", "USD", AttributeType.TEXT));
```

```
inputBuilder.setAttribute(new Attribute("aCountry", "US", AttributeType.TEXT));
inputBuilder.setAttribute(new Attribute("aLanguage", "en", AttributeType.TEXT));
inputBuilder.setAttribute(new Attribute("aRefindKey", "Never use this a input",
AttributeType.TEXT));
```

Toutes les valeurs d'attribut doivent être fournies sous forme de chaînes. Les valeurs de chaîne sont ensuite converties en objets Java appropriés en fonction du type d'attribut fourni. Cela n'est utile que si vous créez d'autres API génériques en plus de l'API Kofax RPA Java.

Paramètres d'exécution

En plus de la méthode `createInputVariable`, la requête contient un certain nombre de méthodes qui contrôlent la façon dont le robot s'exécute sur `RoboServer`.

Méthodes de contrôle d'exécution sur requête

<code>setMaxExecutionTime (int seconds)</code>	Contrôle le temps d'exécution du robot. Une fois ce temps écoulé, le robot est arrêté par <code>RoboServer</code> . La temporisation ne démarre pas tant que le robot ne commence pas à s'exécuter, donc si le robot est mis en file d'attente sur <code>RoboServer</code> , cela n'est pas pris en compte.
<code>setStopOnConnectionLost (boolean)</code>	Si vrai (par défaut), le robot s'arrête si <code>RoboServer</code> découvre que la connexion à l'application cliente a été perdue. Il faut avoir une très bonne raison pour définir cette valeur sur faux ; si votre programme n'est pas écrit pour gérer cette valeur, votre application ne fonctionnera pas comme prévu.
<code>setStopRobotOnApiException (boolean)</code>	Si vrai (par défaut), le robot est arrêté par <code>RoboServer</code> après que la première exception API ait été déclenchée. Par défaut, la plupart des étapes d'un robot déclenchent une exception API si l'étape ne s'exécute pas. Configurez cette valeur dans l'onglet Gestion des erreurs de l'étape. Lorsqu'elle est définie sur faux, le robot continue de s'exécuter quelles que soient les exceptions d'API. Toutefois, à moins que votre application n'utilise le mode d'exécution en streaming <code>RequestExecutor</code> , une exception est toujours déclenchée par <code>execute()</code> . Soyez prudent lorsque vous la configurez sur faux.
<code>setUsername (String) , setPassword (String)</code>	Configure les identifiants <code>RoboServer</code> . <code>RoboServer</code> peut être configuré pour exiger une authentification. Lorsque cette option est activée, le client doit fournir des identifiants ou <code>RoboServer</code> rejette la requête.
<code>setRobotLibrary (RobotLibrary)</code>	Une bibliothèque de robot indique à <code>RoboServer</code> où trouver le robot identifié dans la requête. Pour plus d'exemples liés aux différents types de bibliothèque et à leur utilisation, voir Bibliothèques de robots .

setExecutionId(String)	Vous permet de configurer le <code>executionId</code> pour cette requête. Si vous n'en fournissez pas, RoboServer en génère un automatiquement. L'ID d'exécution est utilisé pour la journalisation et il est également nécessaire pour arrêter le robot par programmation. L'ID doit être globalement unique (dans le temps). Si deux robots utilisent le même ID d'exécution, les journaux seront incohérents.
setProject(String)	Cette méthode n'est utilisée qu'à des fins de journalisation. Management Console utilise ce champ pour lier les messages du journal au projet, afin que les vues du journal puissent être filtrées par projet. Si votre application n'utilise pas la <code>RepositoryRobotLibrary</code> , vous devez définir cette valeur pour informer le système de journalisation RoboServer à quel projet (le cas échéant) ce robot appartient.

Bibliothèques de robots

Dans Design Studio, les robots sont regroupés en projets. Si vous regardez dans le système de fichiers, vous pouvez voir que ces projets sont identifiés par un dossier nommé Library (voir la rubrique « Bibliothèques et projets de robot » dans l'*aide de Kofax RPA* pour plus d'informations).

Lorsque vous créez la requête d'exécution pour RoboServer, vous identifiez le robot par une URL de robot :

```
Request request = new Request("Library:/Input.robot");
```

Ici, `Library:/` est une référence symbolique à une bibliothèque de robot, dans laquelle RoboServer devra rechercher le robot. La `RobotLibrary` est spécifiée dans le constructeur :

```
request.setRobotLibrary(new DefaultRobotLibrary());
```

Trois implémentations de bibliothèque de robot différentes sont disponibles et votre sélection dépend de l'environnement de déploiement.

Bibliothèques de robots

Type de bibliothèque	Description
<code>DefaultRobotLibrary</code>	Cette bibliothèque configure RoboServer pour rechercher le robot dans le dossier du projet actuel, qui est défini dans l'application Paramètres. Si vous avez plusieurs RoboServers, vous devez déployer vos robots sur tous les RoboServers. Cette bibliothèque de robot n'est pas mise en cache, le robot est donc rechargé depuis le disque à chaque exécution. Cette approche rend la bibliothèque utilisable dans un environnement de développement où les robots changent souvent, mais elle n'est pas adaptée à un environnement de production.

Type de bibliothèque	Description
EmbeddedFileBasedRobotLibrary	<p>Cette bibliothèque est intégrée dans la requête d'exécution envoyée à RoboServer. Pour créer cette bibliothèque, vous devez créer un fichier zip contenant les robots et toutes ses dépendances (types, snippets et ressources). Utilisez le menu Outils > Créer un fichier de bibliothèque du robot dans Design Studio.</p> <p>La bibliothèque est envoyée avec chaque requête, ce qui ajoute une surcharge pour les grandes bibliothèques, mais les bibliothèques sont mises en cache sur RoboServer, ce qui offre les meilleures performances possibles.</p> <p>L'un des avantages est que les robots et le programme peuvent être déployés comme une seule unité, ce qui permet une migration propre d'un environnement QA vers un environnement de production. Toutefois, si les robots changent souvent, vous devrez les redéployer souvent.</p> <p>Vous pouvez utiliser le code suivant pour configurer la bibliothèque de robot intégrée pour votre requête.</p> <pre>Request request = new Request("Library:/Tutorials/ NewsMagazine.robot"); RobotLibrary library = new EmbeddedFileBasedRobotLibrary (new FileInputStream ("c:\\\\embeddedLibrary.robotlib")); request.setRobotLibrary(library);</pre>

Type de bibliothèque	Description
RepositoryRobotLibrary	<p>C'est la bibliothèque de robot la plus flexible. Cette bibliothèque utilise le répertoire intégré des Management Consoles comme bibliothèque de robots. Lorsque vous utilisez cette bibliothèque, RoboServer contacte le Management Console, qui envoie une bibliothèque de robot contenant le robot et ses dépendances.</p> <p>La mise en cache s'effectue par robot, à la fois dans Management Console et RoboServer. Dans Management Console, la bibliothèque générée est mise en cache en fonction du robot et de ses dépendances. Sur RoboServer, le cache est basé sur un délai d'attente, il n'est donc pas nécessaire de demander le Management Console pour chaque requête. En outre, la bibliothèque se charge entre RoboServer et Management Console utilisant la mise en cache HTTP publique/ privée, pour réduire davantage la bande passante.</p> <p>Si <code>NewsMagazine.robot</code> est chargé dans le Management Console, vous pouvez utiliser la bibliothèque de robot du répertoire lors de l'exécution du robot :</p> <pre>Request request = new Request("Library:/Tutorials/ NewsMagazine.robot"); RobotLibrary library = new RepositoryRobotLibrary("http:// localhost:50080", "Default Project", 60000); request.setRobotLibrary(library);</pre> <p>Cette commande demande à RoboServer de charger le robot depuis un Management Console local et de le mettre en cache pendant une minute avant de vérifier avec le Management Console pour voir si une nouvelle version du robot (son type et ses snippets) a changé.</p> <p>En outre, toute ressource chargée via le protocole <code>Library:/</code> entraîne RoboServer à demander la ressource directement à partir du Management Console.</p>

Java avancé

Cette section décrit les fonctionnalités avancées de l'API, y compris le streaming des données de sortie, la journalisation et la configuration SSL, ainsi que l'exécution en parallèle.

Répartition de la charge et du basculement

Dans le `RequestExecutor`, l'exécuteur reçoit un tableau de RoboServers. Au fur et à mesure que l'exécuteur est construit, il essaie de se connecter à chaque RoboServer. Une fois connecté, il envoie une requête ping à chaque RoboServer pour découvrir comment le serveur est configuré.

Exécuteur à charge équilibrée :

```
RoboServer prod = new RoboServer("prod.kapow.local", 50000);
RoboServer prod2 = new RoboServer("prod2.kapow.local", 50000);
Cluster cluster = new Cluster("Prod", new RoboServer[]{ prod, prod2}, false);
Request.registerCluster(cluster);
```

La charge est distribuée à chaque RoboServer en ligne dans le cluster, en fonction du nombre d'emplacements d'exécution inutilisés sur le RoboServer. La requête suivante est toujours distribuée au RoboServer avec le plus d'emplacements disponibles. Le nombre d'emplacements d'exécution disponibles est obtenu via la réponse ping initiale, et l'exécuteur effectue le suivi de chaque robot qu'il démarre et de qu'il arrête. Le nombre d'emplacements d'exécution sur un RoboServer est déterminé par le paramètre **Nombre maximum de robots simultanés** dans la section Management Console > Administration > RoboServers.

Si un RoboServer passe hors ligne, il ne reçoit aucune requête d'exécution de robot avant d'avoir répondu avec succès à la requête ping.

Règle du client unique

Par défaut, les connexions API sont limitées à 20 connexions. Toutefois, pour garantir les meilleures performances, nous vous recommandons de n'avoir qu'un seul client API utilisant un cluster donné de RoboServers. Si vous avez trop de machines virtuelles JVM exécutant des robots sur les mêmes RoboServers, cela entraînera une baisse des performances.

Bien que ce qui suit ne soit pas recommandé, si votre environnement nécessite la gestion d'un volume plus élevé, vous pouvez configurer la limite de connexion en ajustant la propriété système `kapow.max.multiplexing.clients` dans le fichier `common.conf`.

Executor Logger

Lorsque vous exécutez une requête, la méthode d'exécution déclenche une exception si un robot génère une erreur. D'autres types d'erreurs et d'avertissemens sont signalés via l'interface `ExecutorLogger`. Dans les exemples précédents, `ExecutionLogger` n'était pas fourni lors de l'exécution de robots, qui est l'implémentation par défaut qui écrit dans `System.out`.

Ce qui suit est un exemple de la façon dont le `ExecutorLogger` signale si l'un des RoboServers passe hors ligne. Dans cet exemple, un cluster est configuré avec un RoboServer qui n'est pas en ligne.

ExecutorLogger, exemple de serveur hors ligne :

```
RoboServer rs = new RoboServer("localhost", 50000);
Cluster cluster = new Cluster("name", new RoboServer[]{rs}, false);
Request.registerCluster(cluster);
```

Si vous exécutez cet exemple, il écrit ce qui suit dans la console.

ExecutorLogger, sortie de console RoboServer hors ligne :

```
RoboServer{host='localhost', port=50000} went offline.
Connection refused
```

Si vous n'avez pas besoin que votre application écrive directement dans `System.out`, vous pouvez fournir une implémentation `ExecutorLogger` différente lors de l'enregistrement du cluster.

Utiliser DebugExecutorLogger :

```
Request.registerCluster(cluster, new DebugExecutorLogger());
```

Cet exemple utilise le `DebugExecutorLogger()`, qui écrit également dans `System.out`, mais seulement si le débogage de l'API est activé. Vous pouvez fournir votre propre implémentation de `ExecutorLogger`, pour contrôler la manière dont les messages d'erreur sont gérés. Voir le JavaDoc `ExecutorLogger` pour plus d'informations.

Streaming des données

Si vous avez besoin de présenter les résultats d'une exécution de robot en temps réel, vous pouvez utiliser l'API pour renvoyer immédiatement les valeurs extraites, au lieu d'attendre que le robot termine son exécution et accède au `RQLResult`.

L'API offre la possibilité de recevoir un rappel chaque fois que l'API reçoit une valeur renvoyée par le robot. Faites-le via l'interface `RobotResponseHandler`.

Streaming de la réponse à l'aide de `AbstractFailFastRobotResponseHandler` :

```
public class DataStreaming {

    public static void main(String[] args) throws ClusterAlreadyDefinedException {

        RoboServer server = new RoboServer("localhost", 50000);
        Cluster cluster = new Cluster("MyCluster", new RoboServer[] {server}, false);
        Request.registerCluster(cluster);

        try {
            Request request = new Request("Library:/Tutorials/NewsMagazine.robot");
            RobotResponseHandler handler = new AbstractFailFastRobotResponseHandler();

            {
                public void handleReturnedValue(RobotOutputObjectResponse response,
                                                Stoppable stoppable) throws RQLEException {
                    RQLObject value = response.getOutputObject();
                    Long personId = (Long) value.get("personId");
                    String name = (String) value.get("name");
                    Long age = (Long) value.get("age");
                    System.out.println(personId + ", " + name + ", " + age);
                }
            };
            request.execute("MyCluster", handler);
        }
    }
}
```

L'exemple précédent utilise la deuxième méthode d'exécution de la requête, qui attend un `RobotResponseHandler` en plus du nom du cluster sur lequel exécuter le robot. Dans cet exemple, créez un `RobotResponseHandler` en étendant `AbstractFailFastRobotResponseHandler`, qui fournit une gestion des erreurs par défaut, pour ne gérer que les valeurs renvoyées par le robot.

La méthode `handleReturnedValue` est appelée chaque fois que l'API reçoit une valeur renvoyée depuis `RoboServer`. Le `AbstractFailFastRobotResponseHandler` utilisé dans cet exemple déclenche des exceptions de la même manière que la méthode d'exécution sans streaming. Cela signifie qu'une exception est levée en réponse à toutes les exceptions d'API générées par le robot.

Le `RobotResponseHandler` a plusieurs méthodes qui peuvent être regroupées en trois catégories.

Événements du cycle de vie du robot

Méthodes appelées lorsque l'état d'exécution du robot change sur RoboServer, par exemple lorsqu'il démarre et termine l'exécution.

Événements des données du robot

Méthodes appelées lorsque le robot renvoie des données ou des erreurs à l'API.

Gestion des erreurs supplémentaires

Méthodes appelées en raison d'une erreur à l'intérieur de RoboServer ou dans l'API.

RobotResponseHandler – événements du cycle de vie du robot

Nom de méthode	Description
void requestSent(RoboServer roboServer, ExecuteRequest request)	Appelée lorsque le RequestExecutor trouve le serveur qui exécute la requête.
void requestAccepted(String executionId)	Appelée lorsque le RoboServer trouvé accepte la requête et la place dans sa file d'attente.
void robotStarted(Stopable stoppable)	Appelée lorsque le RoboServer commence à exécuter le robot. Cela se produit généralement immédiatement après la mise en file d'attente du robot, à moins que le RoboServer ne soit soumis à une charge importante ou utilisé par plusieurs clients API.
void robotDone(RobotDoneEvent reason)	Appelée lorsque le robot a terminé son exécution dans RoboServer. Le RobotDoneEvent est utilisé pour spécifier si l'exécution s'est terminée normalement, en raison d'une erreur, ou si elle a été arrêtée.

RobotResponseHandler – événements des données du robot

Nom de méthode	Description
void handleReturnedValue(RobotOutputObjectResponse response, Stopable stoppable)	Appelée lorsque le robot exécute une activité Valeur de retour et que la valeur est renvoyée via le socket à l'API.
void handleRobotError(RobotErrorResponse response, Stopable stoppable)	Appelée lorsque le robot déclenche une exception API. Dans des circonstances normales, le robot cesse de s'exécuter après la première exception API. Ce comportement peut être remplacé en utilisant Request.setStopRobotOnApiException(false), auquel cas cette méthode est appelée plusieurs fois. Cette approche est utile si vous souhaitez qu'un robot de streaming des données continue de s'exécuter indépendamment des erreurs générées.
void handleWriteLog(RobotMessageResponse response, Stopable stoppable)	Appelée lorsque le robot exécute l'étape Écrire le journal. Ceci est utile pour fournir des informations de journalisation supplémentaires depuis un robot.

RobotResponseHandler – gestion des erreurs supplémentaires

Nom de méthode	Description
void handleServerError (ServerErrorResponse response, Stoppable stoppable)	Appelée si RoboServer génère une erreur. Par exemple, cela peut arriver si le serveur est trop occupé pour traiter des requêtes, ou si une erreur se produit à l'intérieur de RoboServer, ce qui l'empêche de démarrer le robot.
handleError (RQLEexception e, Stoppable stoppable)	Appelée si une erreur se produit dans l'API ou, plus fréquemment, si le client perd la connexion avec RoboServer.

De nombreuses méthodes incluent un objet `Stoppable`, qui peut être utilisé pour l'arrêt en réponse à une erreur spécifique ou à une valeur renvoyée.

Certaines méthodes vous permettent de déclencher une `RQLEexception`, ce qui peut avoir des conséquences. Le thread qui appelle le handler est le thread qui appelle `Request.execute()` et les exceptions déclenchées peuvent surcharger la pile d'appels. Si vous déclenchez une exception en réponse à `handleReturnedValue`, `handleRobotError` ou `handleWriteLog`, il est de votre responsabilité d'invoquer `Stoppable.stop()`, ou le robot peut continuer à s'exécuter même si l'appel à `Request.execute()` est terminé.

Le streaming des données est le plus souvent utilisé dans l'un des cas suivants.

- Applications web basées sur Ajax, où les résultats sont présentés à l'utilisateur en temps réel. Si les données ne sont pas diffusées, les résultats ne peuvent pas s'afficher tant que le robot n'a pas fini son exécution.
- Des robots qui renvoient tellement de données que le client ne pourrait pas tout garder en mémoire tout au long de l'exécution du robot.
- Processus qui doivent être optimisés pour que les valeurs extraites soient traitées parallèlement à l'exécution du robot.
- Processus qui stockent des données dans des bases de données sous un format personnalisé.
- Robots qui devraient ignorer ou nécessiter une gestion personnalisée des exceptions d'API (voir l'exemple suivant).

Collecte des réponses et des erreurs à l'aide de AbstractFailFastRobotResponseHandler :

```
public class DataStreamingCollectErrorsAndValues {
    public static void main(String[] args) throws ClusterAlreadyDefinedException {
        RoboServer server = new RoboServer("localhost", 50000);
        Cluster cluster = new Cluster("MyCluster", new RoboServer[] {server}, false);
        Request.registerCluster(cluster);

        try {
            Request request = new Request("Library:/Tutorials/NewsMagazine.robot");
            request.setStopRobotOnApiException(false); // IMPORTANT!!
            request.setRobotLibrary(new DefaultRobotLibrary());
            ErrorCollectingRobotResponseHandler handler =
                new ErrorCollectingRobotResponseHandler();
            request.execute("MyCluster", handler);

            System.out.println("Extracted values:");
            for (RobotOutputObjectResponse response : handler.getOutput()) {

```

```

RQLObject value = response.getOutputObject();
Long personId = (Long) value.get("personId");
String name = (String) value.get("name");
Long age = (Long) value.get("age");
System.out.println(personId + ", " + name + ", " + age);
}

System.out.println("Errors:");
for (RobotErrorResponse error : handler.getErrors()) {
    System.out.println(error.getErrorLocationCode() + ", " +
        error.getErrorMessage());
}
}
}

private static class ErrorCollectingRobotResponseHandler extends
    AbstractFailFastRobotResponseHandler {

    private List<RobotErrorResponse> _errors =
        new LinkedList<RobotErrorResponse>();
    private List<RobotOutputObjectResponse> _output =
        new LinkedList<RobotOutputObjectResponse>();
    public void handleReturnedValue
        (RobotOutputObjectResponse response, Stoppable stoppable)
        throws RQLEException {
        _output.add(response);
    }

    @Override
    public void handleRobotError(RobotErrorResponse response,
        Stoppable stoppable) throws RQLEException {
        // do not call super as this will stop the robot
        _errors.add(response);
    }

    public List<RobotErrorResponse> getErrors() {
        return _errors;
    }

    public List<RobotOutputObjectResponse> getOutput() {
        return _output;
    }
}
}

```

L'exemple précédent montre comment utiliser un `RobotResponseHandler` qui collecte les valeurs renvoyées et les erreurs. Ce type de handler est utile si le robot doit continuer à s'exécuter même en cas d'erreurs, ce qui peut être utile si le site web est instable et parfois expire. Notez que seules les erreurs du robot (exceptions d'API) sont collectées par le gestionnaire. Si la connexion à RoboServer est perdue, `Request.execute()` déclenche toujours une `RQLEException`, et le robot est arrêté par RoboServer.

Pour plus d'informations, voir le JavaDoc `RobotResponseHandler`.

SSL

L'API communique avec RoboServer via un `RQLService`, un composant RoboServer qui écoute les requêtes API sur un port réseau spécifique. Lorsque vous démarrez un RoboServer, vous spécifiez s'il doit utiliser le service SSL chiffré, ou le service socket brut, ou les deux (en utilisant deux ports

différents). Tous les RoboServers dans un cluster doivent exécuter le même RQLService (bien que le port puisse être différent).

En supposant que nous ayons démarré un RoboServer avec le RQLService SSL sur le port 50043 :

```
RoboServer -service ssl:50043
```

Le code suivant peut être utilisé.

Configuration SSL

```
RoboServer server = new RoboServer("localhost", 50043);
boolean ssl = true;
Cluster cluster = new Cluster("MyCluster", new RoboServer[] {server}, ssl);
Request.registerCluster(cluster);
```

Vous devez créer le cluster en tant que cluster SSL et spécifier le port SSL utilisé par chaque RoboServer. Désormais, toutes les communications entre RoboServer et l'API seront chiffrées.

Pour que cet exemple fonctionne, vous avez besoin de `not-yet-commons-ssl-0.3.17.jar` dans le `classpath` de votre application. Vous pouvez le trouver à côté du fichier API.jar dans votre installation Kofax RPA.

En plus du chiffrement des données, le protocole SSL offre la possibilité de vérifier l'identité de la partie distante. Ce type de vérification est très important sur Internet. Le plus souvent, votre client API et les RoboServers sont sur le même réseau local ; vous avez donc rarement besoin de vérifier l'identité de l'autre partie, mais l'API prend en charge cette fonctionnalité si cela s'avère nécessaire.

La vérification d'identité n'étant presque jamais utilisée, elle n'est pas décrite dans ce guide. Pour plus d'informations à ce propos, voir les exemples SSL inclus avec l'API Java.

Exécution en parallèle

Les deux méthodes d'exécution de la requête sont bloquantes, ce qui signifie qu'un thread est requis pour chaque exécution du robot. Les exemples des sections précédentes illustrent l'exécution directe du robot sur le thread principal, ce qui n'est généralement pas préférable car vous ne pouvez exécuter qu'un seul robot à la fois de manière séquentielle.

L'exemple suivant exécute deux robots de tutoriel en parallèle. Cet exemple utilise la bibliothèque `java.util.concurrent` pour le multithreading.

Exemple de multithreading

```
import com.kapowtech.robosuite.api.java.repository.construct.*;
import com.kapowtech.robosuite.api.java.rql.*;
import com.kapowtech.robosuite.api.java.rql.construct.*;
import com.kapowtech.robosuite.api.java.rql.engine.hotstandby.*;

import java.util.concurrent.*;

public class ParallelExecution {

    public static void main(String[] args) throws Exception {

        RoboServer server = new RoboServer("localhost", 50000);
        Cluster cluster = new Cluster("MyCluster", new RoboServer[] {server},
            false);
        Request.registerCluster(cluster);
```

```

int numRobots = 4;
int numThreads = 2;
ThreadPoolExecutor threadPool = new ThreadPoolExecutor(numThreads,
    numThreads, 10, TimeUnit.SECONDS, new LinkedBlockingQueue());
for (int i = 0; i < numRobots; i++) {
    Request request = new Request("Library:/Tutorials/NewsMagazine.robot");
    request.setRobotLibrary(new DefaultRobotLibrary());
    threadPool.execute(new RobotRunnable(request));
}
threadPool.shutdown();
threadPool.awaitTermination(60, TimeUnit.SECONDS);
}

// -----
// Inner classes
// -----
static class RobotRunnable implements Runnable {

    Request _request;

    RobotRunnable(Request request) {
        _request = request;
    }

    public void run() {

        try {
            RQLResult result = _request.execute("MyCluster");
            System.out.println(result);
        }
    }
}
}

```

L'exemple précédent crée un `ThreadPoolExecutor` avec deux threads, puis nous créons quatre `RobotRunnables` et nous les exécutons sur le pool de threads. Comme le pool de threads contient deux threads, deux robots commencent à s'exécuter immédiatement. Les deux robots restants sont parqués dans la `LinkedBlockingQueue` et exécutés dans l'ordre une fois que les deux premiers robots ont terminé leur exécution et que les threads du pool de threads deviennent disponibles.

Notez que la requête est modifiable et pour éviter de déclencher des conditions, la requête est clonée dans la méthode d'exécution. Comme une requête est modifiable, vous ne devez jamais modifier la même requête sur des threads séparés.

Intégration de répertoire

Dans le Management Console, vous spécifiez également les clusters de RoboServers, qui sont utilisés pour exécuter des robots planifiés, ainsi que des robots dont l'exécution s'effectue comme des services REST. L'API vous permet d'utiliser le `RepositoryClient` pour obtenir des informations sur le cluster depuis Management Console. Voir la documentation `RepositoryClient` pour plus d'informations.

Intégration de répertoire :

```

public class RepositoryIntegration {
    public static void main(String[] args) throws Exception {
        RepositoryClient client = RepositoryClientFactory.createRepositoryClient

```

```

        ("http://localhost:50080", null, null);
        Request.registerCluster(client, "Cluster 1");

        Request request = new Request("Library:/Tutorials/NewsMagazine.robot");
        request.setRobotLibrary(new DefaultRobotLibrary());
        RQLResult result = request.execute("MyCluster");
        System.out.println(result);
    }
}

```

L'exemple précédent montre comment créer un `RepositoryClient` qui se connecte à un Management Console déployé sur localhost. Pour que cet exemple fonctionne, `commons-logging-1.1.1.jar`, `commons-codec-1.4.jar` et `commons-httpclient-4.1.jar` doivent être inclus dans votre classpath.

L'authentification n'est pas activée, donc `nul` est envoyé comme nom d'utilisateur et mot de passe. Lorsque vous enregistrez le `RepositoryClient`, vous spécifiez le nom d'un cluster qui existe sur le Management Console. Il interroge ensuite le Management Console pour obtenir une liste des RoboServers configurés pour ce cluster et vérifie toutes les deux minutes pour voir si la configuration du cluster est mise à jour sur le Management Console.

Cette intégration vous permet de créer un cluster sur Management Console que vous pouvez modifier dynamiquement à l'aide de l'interface utilisateur Management Console. Lorsque vous utilisez un cluster Management Console avec l'API, l'utilisation doit être exclusive et vous ne devez pas l'utiliser pour planifier le robot, car cela enfreindrait la règle du client unique.

Sous le capot

Cette section explique ce qui se passe sous le capot lorsque vous enregistrez un cluster et exécutez des requêtes.

Lorsque vous enregistrez un cluster avec la requête, un `RequestExecutor` est créé en arrière-plan. Ce `RequestExecutor` est stocké sur une carte en utilisant le nom de cluster comme clé. Lorsqu'une requête est exécutée, le nom de cluster fourni est utilisé pour rechercher le `RequestExecutor` associé et exécuter la requête.

Exécution normale

```

public static void main(String[] args) throws InterruptedException,
    RQLEException {

    RoboServer server = new RoboServer("localhost", 50000);
    Cluster cluster = new Cluster("MyCluster", new RoboServer[]{ server}, false);
    Request.registerCluster(cluster);
    Request request = new Request("Library:/Tutorials/NewsMagazine.robot");
    request.setRobotLibrary(new DefaultRobotLibrary());
    RQLResult result = request.execute("MyCluster");
    System.out.println(result);
}

```

Maintenant, écrivez le même exemple en utilisant directement le `hiddenRequestExecutor`.

Exécution sous le capot :

```
public static void main(String[] args) throws InterruptedException,
```

```

RQLEException {
    RoboServer server = new RoboServer("localhost", 50000);
    Cluster cluster = new Cluster("MyCluster", new RoboServer[]{ server}, false);
    RequestExecutor executor = new RequestExecutor(cluster);

    Request request = new Request("Library:/Tutorials/NewsMagazine.robot");
    request.setRobotLibrary(new DefaultRobotLibrary());
    RQLResult result = executor.execute(request);
    System.out.println(result);
}

```

Le `RequestExecutor` est masqué par défaut, vous n'avez donc pas à le suivre. Vous ne pouvez créer qu'un `RequestExecutor` par cluster, donc si vous l'utilisez directement, vous devez stocker une référence dans toute votre application. L'utilisation de `Request.registerCluster(cluster)` signifie que vous pouvez ignorer les règles du cycle de vie et de `RequestExecutor`.

Le `RequestExecutor` contient l'état et la logique nécessaires, qui fournissent les fonctionnalités d'équilibrage de charge et de remplacement en cas de panne. L'utilisation directe de `RequestExecutor` offre également quelques fonctionnalités supplémentaires.

Fonctionnalités RequestExecutor

Lorsque le `RequestExecutor` n'est pas connecté à un répertoire, vous pouvez ajouter ou supprimer dynamiquement les RoboServers en appelant `addRoboServer(...)` et `removeRoboServer(...)`. Ces méthodes modifient la liste de distribution utilisée dans le `RequestExecutor`.

`RequestExecutor.getTotalAvailableSlots()` renvoie le nombre d'emplacements d'exécution inutilisés sur tous les RoboServers dans la liste de distribution interne.

En utilisant ces méthodes, vous pouvez ajouter dynamiquement des RoboServers à votre `RequestExecutor` une fois que le nombre d'emplacements d'exécution disponibles devient faible.

Lorsque vous créez le `RequestExecutor`, vous pouvez éventuellement fournir un `RQLEngineFactory`. Le `RQLEngineFactory` vous permet de personnaliser quel `RQLProtocol` est utilisé lors de la connexion à un RoboServer. Cela n'est nécessaire que dans de rares circonstances, telles que l'utilisation d'un certificat client pour augmenter la sécurité. Voir le chapitre Certificats dans le *Guide de l'administrateur de Kofax RPA* pour plus d'informations.

Applications web

Le `RequestExecutor` contient un certain nombre de threads internes utilisés pour envoyer et recevoir des requêtes aux RoboServers, ainsi que pour envoyer un ping à chaque RoboServer connu à intervalles réguliers. Ces threads sont tous marqués comme `daemon`, ce qui signifie qu'ils n'empêchent pas la JVM de s'arrêter lorsque le thread principal existe. Voir le JavaDoc `Thread` pour plus d'informations sur les threads de `daemon`.

Si vous utilisez le `RequestExecutor` à l'intérieur d'une application web, la JVM a une durée de vie plus longue que votre application web, et vous pouvez déployer et annuler le déploiement de votre application web pendant que le conteneur web est en cours d'exécution. Cela signifie qu'une application web est responsable de l'arrêt de tous les threads qu'elle a créés. Si l'application web n'arrête pas un thread, une fuite de mémoire est créée lorsque vous annulez le déploiement de

l'application web. Cette fuite de mémoire se produit car les objets référencés par les threads en cours d'exécution ne peuvent pas être récupérés jusqu'à ce que les threads s'arrêtent.

Si vous utilisez le `RequestExecutor` à l'intérieur d'une application web, votre programme est responsable de l'extinction de ces threads internes, ce qui se fait en appelant `Request.shutdown()` ou `RequestExecutor.shutdown()` si votre programme a créé explicitement le `RequestExecutor`.

Cet exemple montre comment utiliser un `ServletContextListener` pour arrêter correctement l'API lorsqu'une application web n'est pas déployée. Vous devez définir le Context listener dans votre application web.xml.

Arrêt correct dans l'application web :

```
import com.kapowtech.robosuite.api.java.repository.construct.*;
import com.kapowtech.robosuite.api.java.rql.*;
import com.kapowtech.robosuite.api.java.rql.construct.*;

import javax.servlet.*;

public class APIShutdownListener implements ServletContextListener {
    public void contextInitialized(ServletContextEvent servletContextEvent) {
        RoboServer server = new RoboServer("localhost", 50000);
        Cluster cluster = new Cluster("MyCluster", new RoboServer[]{ server },
            false);
        try {
            Request.registerCluster(cluster);
        }
        catch (ClusterAlreadyDefinedException e) {
            throw new RuntimeException(e);
        }
    }

    public void contextDestroyed(ServletContextEvent servletContextEvent) {
        Request.shutdown();
    }
}
```

`contextDestroyed` est appelé lorsque le conteneur web annule le déploiement de l'application. `Request.shutdown()` est appelé pour s'assurer que tous les threads internes dans le `RequestExecutor` masqué sont arrêtés correctement.

Comme `contextInitialized` ne peut pas déclencher d'exceptions non cochées, vous devez encapsuler la `ClusterAlreadyDefinedException` dans une `RunTimeException`. En raison de la hiérarchie du chargeur de classe dans les conteneurs web Java, il est possible de déclencher cette exception si l'application est déployée deux fois. Cela se produit uniquement si le fichier .jar de l'API a été chargé par un chargeur de classe commun et non par le chargeur de classe de l'application individuelle.

Débogage API

L'API peut fournir des informations supplémentaires à des fins de débogage. Pour activer le débogage de l'API, vous devez configurer la propriété système `DEBUG_ON`. La valeur de cette propriété doit être un nom de classe/paquet dans l'API.

Par exemple, si vous êtes intéressé par les transmissions de données entre l'API et RoboServer, vous pouvez demander des informations de débogage pour le paquet `com.kapowtech.robosuite.api.java.rql.io`. Pendant le développement, vous pouvez le faire en configurant directement la propriété système dans le programme :

Activation du débogage :

```
System.setProperty("DEBUG_ON", "com.kapowtech.robosuite.api.java.rql.io");
RoboServer server = new RoboServer("localhost", 50000);
Cluster cluster = new Cluster("MyCluster", new RoboServer[]{ server}, false);
Request.registerCluster(cluster);
```

Si vous déboguez une application en production, vous pouvez définir la propriété système via la ligne de commande.

Activation du débogage :

```
java -DDEBUG_ON=com.kapowtech.robosuite.api.java.rql.io
```

Si vous êtes intéressé par un débogage de plusieurs paquets, séparez les noms des paquets par des virgules. Au lieu d'un nom de paquet, vous pouvez saisir l'argument `ALL`, pour afficher le débogage de tous les paquets.

API Repository

L'API Repository vous permet d'interroger le répertoire de Management Console pour récupérer une liste de projets, des robots et l'entrée requise pour appeler un robot. Elle vous permet également de déployer par programmation des robots, des types et des fichiers de ressources.

Dépendances

Pour utiliser l'API Repository, ajoutez tous les fichiers .jar du dossier `API/robosuite-java-api/lib` situé dans le dossier d'installation Kofax RPA sur le `classpath` de votre projet.

Utilisez Java 8 ou ultérieur.

Client répertoire

La communication avec le répertoire se fait via le `RepositoryClient` dans le `com.kapowtech.robosuite.api.java.repository.engine`.

Créer un `RepositoryClient` :

```
public static void main(String[] args) {

    String username = "admin";
    String password = "admin";
    try {
        RepositoryClient client = RepositoryClientFactory.
            createRepositoryClient("http://localhost:50080/",
            username, password);
        Project[] projects = client.getProjects();
        for (Project project : projects) {
            System.out.println(project.getName());
        }
    }
}
```

```

        }
    }
    catch(
        RepositoryClientException e)
    {
        e.printStackTrace();
    }
}

```

Ici, un `RepositoryClient` est configuré pour se connecter au répertoire de Management Console sur `http://localhost:50080/`, avec un nom d'utilisateur et un mot de passe.

Une fois le `RepositoryClient` créé, la méthode `getProjects()` est utilisée pour demander au répertoire une liste de projets. Notez que lors de l'appel de l'une des méthodes `RepositoryClient`, une `RepositoryClientException` est déclenchée si une erreur se produit.

Le `RepositoryClient` dispose des méthodes suivantes.

Méthodes de `RepositoryClient` :

Signature de méthode	Description
<code>void deleteResource(String projectName, String resourceName, boolean silent)</code>	Supprime une ressource d'un projet. Si <code>silent</code> est vrai, aucune erreur n'est générée si la ressource n'existe pas. L'argument <code>resourceName</code> utilise le chemin complet de la ressource.
<code>void deleteRobot(String projectName, String robotName, boolean silent)</code>	Supprime un robot d'un projet. L'argument <code>robotName</code> utilise le chemin complet du robot.
<code>void deleteSnippet(String projectName, String snippetName, boolean silent)</code>	Supprime un snippet d'un projet. L'argument <code>snippetName</code> utilise le chemin complet du snippet.
<code>void deleteType(String projectName, String modelName, boolean silent)</code>	Supprime un type d'un projet. L'argument <code>modelName</code> utilise le chemin complet du type.
<code>void deployLibrary(String projectName, EmbeddedFileBasedRobotLibrary library, boolean failIfExists)</code>	Déploie une bibliothèque sur le serveur. Les robots, les types et les ressources sont remplacés à moins que <code>failIfExists</code> soit vrai.
<code>void deployResource(String projectName, String resourceName, byte[] resourceBytes, boolean failIfExists)</code>	Déploie une ressource dans un projet. Si une ressource avec le nom donné existe déjà, elle peut être remplacée en configurant <code>failIfExists</code> sur faux. L'argument <code>resourceName</code> utilise le chemin complet de la ressource.
<code>void deployRobot(String projectName, String robotName, byte[] robotBytes, boolean failIfExists)</code>	Déploie un robot dans un projet. Si un robot avec le nom donné existe déjà, il peut être remplacé en configurant <code>failIfExists</code> sur faux. L'argument <code>robotName</code> utilise le chemin complet du robot.
<code>void deploySnippet(String projectName, String snippetName, byte[] snippetBytes, boolean failIfExists)</code>	Déploie un snippet dans un projet. Si un snippet avec le nom donné existe déjà, il peut être remplacé en configurant <code>failIfExists</code> sur faux. L'argument <code>snippetName</code> utilise le chemin complet du snippet.
<code>void deployType(String projectName, String typeName, byte[] typeBytes, boolean failIfExists)</code>	Déploie un type dans un projet. Si un type avec le nom donné existe déjà, il peut être remplacé en configurant <code>failIfExists</code> sur faux. L'argument <code>typeName</code> utilise le chemin complet du type.

Signature de méthode	Description
Project[] getProjects ()	Renvoie les projets présents dans ce répertoire.
Cluster[] getRoboServerClusters ()	Renvoie une liste de clusters et de RoboServers en ligne (valides) qui sont enregistrés avec le Management Console exécutant le répertoire.
Cluster[] getRoboServerClusters (boolean onlineRoboServer)	Renvoie une liste de clusters et de RoboServers enregistrés avec le Management Console. Utilisez l'indicateur <code>onlineRoboServer</code> pour indiquer si la liste des clusters doit inclure seulement les RoboServers qui sont en ligne ou tous les RoboServers.
Cluster addRoboServer(String clusterName, int portNumber, String host)	Ajoute un nouveau RoboServer à un cluster.
Robot[] getRobotsInProject (String projectName)	Renvoie le chemin d'accès complet des Robots à Moteur Basique disponibles dans le projet.
RobotSignature getRobotSignature (String projectName, String robotName)	Renvoie la signature du robot avec le chemin complet du robot, ainsi que les variables d'entrée requises pour exécuter ce robot, et une liste des types qu'il peut renvoyer ou stocker.
RepositoryFolder getProjectInventory (String projectName)	Renvoie l'arborescence entière des dossiers et fichiers du répertoire.
RepositoryFolder getFolderInventory (String projectName, String folderPath)	Renvoie les dossiers et fichiers du sous-dossier dans le projet spécifié depuis le répertoire.
RepositoryFolder getFileInventory (String projectName, String folderPath, String fileName, RepositoryFile.Type fileType)	Récupère le fichier et les fichiers référencés à partir de la Management Console. Notez que l'inventaire des fichiers est encapsulé dans <code>RepositoryFolder</code> pour récupérer les références.
void deleteFile (RepositoryFile file)	Supprime le fichier spécifié du répertoire.
Date getCurrentDate ()	Renvoie la date et l'heure actuelles du Management Console.
byte[] getBytes (RepositoryFile file)	Renvoie la taille en octets du fichier spécifié dans le répertoire.
void updateFile (RepositoryFile file, byte[] bytes)	Met à jour le fichier spécifié dans le répertoire avec de nouveaux octets.
void moveFile (RepositoryFile sourceFile, String destFolderPath)	Déplace le fichier spécifié du répertoire vers un dossier spécifié dans <code>destFolderPath</code> .
void renameRobot (RepositoryFile robotFile, String newName)	Renomme le fichier robot spécifié.
void deleteFolder (String projectName, String folderPath)	Supprime le dossier spécifié dans le répertoire.
void deleteRoboServer (String clusterName, RoboServer roboServer)	Supprime un RoboServer.

Signature de méthode	Description
Map<String, String> getInfo()	<p>Renvoie des informations sur le Management Console et l'API Repository</p> <p>La méthode renvoie un mappage des éléments suivants :</p> <ul style="list-style-type: none"> • « application » à la version de Management Console contenant la version principale, la version mineure et la version point pour, par exemple, 11.4.0 • « répertoire » à l'ID de la dernière DTD utilisée par l'API Repository, par exemple : //Kapow Technologies//DTD Repository 1.5//EN • « rql » à l'ID de la dernière DTD utilisée par l'API Robot Query Language, par exemple : //Kapow Technologies//DTD RoboSuite Robot Query Language 1.13//EN
pingRepository()	Envoie un ping au serveur de répertoire et renvoie null en cas de succès, ou une chaîne d'erreur dans le cas contraire.
deployConnector(String projectName, String connectorName, byte[] connectorBytes, boolean failIfExists, AdditionalInfo additionalInfo)	Déploie un connecteur dans le répertoire.
deleteConnector(String projectName, String connectorName, boolean silent, AdditionalInfo additionalInfo)	Supprime un connecteur du répertoire.
getRobotsByTag(String projectName, String tag)	Récupère les robots avec une balise spécifiée.

 Le chemin complet est relatif à votre dossier de projet.

Les serveurs proxy doivent être spécifiés explicitement lors de la création du `RepositoryClient`. Les serveurs proxy http standard sans authentification sont pris en charge. Les serveurs proxy NTLM avec authentification sont également pris en charge.

Voir le JavaDoc `RepositoryClient` pour plus d'informations.

Déploiement avec le client répertoire

L'exemple suivant montre comment déployer un robot et un type depuis le système de fichiers local à l'aide de `RepositoryClient`.

Déploiement à l'aide de `RepositoryClient` :

```
String user = "test";
String password = "test1234";
RepositoryClient client = new RepositoryClient("http://localhost:50080", user,
    password);
try {
    FileInputStream robotStream = new FileInputStream
```

```

("c:\\MyRobots\\Library\\Test.robot");
FileInputStream typeStream = new FileInputStream
("c:\\MyRobots\\Library\\Test.type");

// Use the Kapow Java APIs StreamUtil to convert InputStream to byte[].
// For production we recommend IOUtils.toByteArray(InputStream i)
// in the commons-io library from apache.
byte[] robotBytes = StreamUtil.readStream(robotStream).toByteArray();
byte[] typeBytes = StreamUtil.readStream(typeStream).toByteArray();

// we assume that no one has deleted the Default project
client.deployRobot("Default project", "Test.robot", robotBytes, true);
client.deployType("Default project", "Test.type", typeBytes, true);
}
catch (FileNotFoundException e) {
    System.out.println("Could not load file from disk " + e.getMessage());
}
catch (IOException e) {
    System.out.println("Could not read bytes from stream " + e.getMessage());
}
catch (FileAlreadyExistsException e) {
    // either the type or file already exist in the give project
    System.out.println(e.getMessage());
}
}

```

API Repository REST

L'API Repository est en fait un groupe d'URL et de services RESTful où les données peuvent être publiées.

Toutes les méthodes du client répertoire, qui récupèrent des informations depuis le répertoire, envoient du XML au répertoire et le répertoire répond en XML. Toutes les méthodes de déploiement envoient des octets dans le répertoire (informations encodées dans l'URL) et le répertoire répond en XML pour accuser réception. Le format du XML envoyé et reçu est régi par une DTD disponible sur www.kapowtech.com.

Voici un exemple de toutes les requêtes XML. Tous les messages doivent commencer par la déclaration suivante :

```

<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE repository-request PUBLIC "-//Kapow Technologies//"
DTD Repository 1.5//EN" "http://www.kapowtech.com/robosuite/
repository_1_5.dtd">

```

Si Management Console est déployé sur `http://localhost:8080/ManagementConsole`, les requêtes doivent être publiées sur `http://localhost:8080/ManagementConsole/secure/RepositoryAPI?format=xml`

Snippets

Un certain nombre de snippets XML sont utilisés dans l'API et les snippets suivants sont utilisés dans les exemples. Nous vous recommandons d'étudier la DTD pour comprendre la structure des données.

Lors de l'envoi des requêtes, nous avons souvent besoin de décrire un fichier. The constructs have been added to the 1.5 DTD to assist in project synchronization between Design Studio and Management Console.

Nom du snippet	Code
répertoire-fichier-requête	<pre><repository-file-request> <project-name>Default project</project-name> <name>ExName</name> <type>snippet</type> <path>subfolder</path> <last-modified>2019-02-01 19:26:12.321</last-modified> <last-modified-by>username</last-modified-by> <checksum>a342ddaf</checksum> </repository-file-request></pre>
répertoire-fichier	<pre><repository-file><name>filename</name> <type>ROBOT</type> <last-modified>2019-02-01 19:26:12.321</last-modified> <last-modified-by>username</last-modified-by> <checksum>a342ddaf</checksum> <dependencies><dependency><name>exsnippet</name><type>snippet</type></dependency> </dependencies> </repository-file></pre>

Opérations REST

Méthode	Exemple de requête	Exemple de réponse
delete-file (robot)	<pre><repository-request> <delete-file file-type="robot" silent="true"> <project-name>Default project</project-name> <file-name>InputA.type</file-name> </delete-file> </repository-request></pre>	<pre><repository-response><delete-successful/></repository-response></pre>
delete-file (type)	<pre><repository-request> <delete-file file-type="type" silent="false"> <project-name>Default project</project-name> <file-name>InputA.type</file-name> </delete-file> </repository-request></pre>	<pre><repository-response><error type="file-not-found">Could not find a Type named InputA.type in project 'Default project'</error></repository-response></pre>
delete-file (snippet)	<pre><repository-request> <delete-file file-type="snippet" silent="true"> <project-name>Default project</project-name> <file-name>InputA.type</file-name> </delete-file> </repository-request></pre>	<pre><repository-response><delete-successful/></repository-response></pre>
delete-file (resource)	<pre><repository-request> <delete-file file-type="resource" silent="true"> <project-name>Default project</project-name> <file-name>InputA.type</file-name> </delete-file> </repository-request></pre>	<pre><repository-response><delete-successful/></repository-response></pre>

Méthode	Exemple de requête	Exemple de réponse
get-projects	<repository-request> <get-projects/> </repository-request>	<repository-response><project-list><project-name>Default project</project-name></project-list></repository-response>
get-robots-in-project	<repository-request> <get-robots-in-project> <project-name>Default project</project-name> </get-robots-in-project> </repository-request>	<repository-response><robot-list><robot><robot-name>DoNothing.robot</robot-name><version>10.7</version><last-modified>2019-10-11 18:24:12.648</last-modified></robot></robot-list></repository-response>
get-robot-signature	<repository-request> <get-robot-signature> <project-name>Default project</project-name> <robot-name>DoNothing.robot</robot-name> </get-robot-signature> </repository-request>	<repository-response><robot-signature><robot-name>DoNothing.robot</robot-name><version>10.7</version><last-modified>2019-10-11 18:24:12.648</last-modified><input-object-list><input-object><variable-name>InputA</variable-name><type-name>InputA</type-name><input-attribute-list><input-attribute><attribute-name>aString</attribute-name><attribute-type>Short Text</attribute-type></input-attribute><input-attribute><attribute-name>aNumber</attribute-name><attribute-type>Number</attribute-type></input-attribute><input-attribute><attribute-name>aSession</attribute-name><attribute-type>Session</attribute-type></input-attribute><input-attribute><attribute-name>aBoolean</attribute-name><attribute-type>Boolean</attribute-type></input-attribute><input-attribute><attribute-name>aDate</attribute-name><attribute-type>Date</attribute-type></input-attribute><input-attribute><attribute-name>aCharacter</attribute-name><attribute-type>Character</attribute-type></input-attribute><input-attribute>

Méthode	Exemple de requête	Exemple de réponse
		<pre>><attribute-name>anImage</attribute-name><attribute-type>Image</attribute-type></input-attribute></input-attribute-list></input-object><input-object><variable-name>InputB</variable-name><type-name>InputB</type-name><input-attribute-list><input-attribute required="true"><attribute-name>aString</attribute-name><attribute-type>Short Text</attribute-type></input-attribute><input-attribute required="true"><attribute-name>anInt</attribute-name><attribute-type>Integer</attribute-type></input-attribute><input-attribute required="true"><attribute-name>aNumber</attribute-name><attribute-type>Number</attribute-type></input-attribute><input-attribute required="true"><attribute-name>aSession</attribute-name><attribute-type>Session</attribute-type></input-attribute><input-attribute required="true"><attribute-name>aBoolean</attribute-name><attribute-type>Boolean</attribute-type></input-attribute><input-attribute required="true"><attribute-name>aDate</attribute-name><attribute-type>Date</attribute-type></input-attribute><input-attribute required="true"><attribute-name>aCharacter</attribute-name><attribute-type>Character</attribute-type></input-attribute><input-attribute required="true"><attribute-name>anImage</attribute-name><attribute-type>Image</attribute-type></input-attribute></input-attribute-list></input-object></input-object-list><returned-type-list><returned-type><type-name>OutputA</type-name><returned-attribute-list><returned-attribute><attribute-name>aString</attribute-name><attribute-type>Short Text</attribute-type></returned-attribute></returned-attribute></pre>

Méthode	Exemple de requête	Exemple de réponse
		-list></returned-type></returned-type-list><stored-type-list/></robot-signature></repository-response>
get-clusters	<repository-request> <get-clusters/> </repository-request>	<repository-response><clusters><cluster name="Cluster 1" ssl="false"><roboserver host="localhost" port="50000"/></cluster></clusters></repository-response>
get-current-date	<repository-request> <get-current-date/> </repository-request>	<repository-response> <current-date>2019-02-01 19:26:12.321</current-date> </repository-response>
get-bytes	<repository-request> <get-bytes> <repository-file-request>EXAMPLE</repository-file-request> </get-bytes> </repository-request>	<repository-response> <file-content> <file-bytes></file-bytes> </file-content> </repository-response>
get-project-inventory	<repository-request> <get-project-inventory> <project-name>Default project</project-name> </get-project-inventory> </repository-request>	<repository-response> <repository-folder> <path></path> <sub-folders> -- repository-folders (recursively) -- </sub-folders> <files> -- zero, one or more repository-file elements -- </files> <references> -- zero, one or more repository-file elements needed by robots in folder -- </references> </repository-folder> </repository-response>
get-folder-inventory	<repository-request> <get-folder-inventory> <project-name>Default project</project-name> <path>subfolder</path> </get-folder-inventory> </repository-request>	<repository-response> <repository-folder> <path></path> <sub-folders> -- repository-folders (recursively) -- </sub-folders> <files> -- zero, one or more repository-file elements -- </files> <references> -- zero, one or more repository-file elements needed by robots in folder -- </references> </repository-folder> </repository-response>

Méthode	Exemple de requête	Exemple de réponse
get-file-inventory	<pre><repository-request> <get-file-inventory> <project-name>Default project</project-name> <path>subfolder</path> <name>robotname</name> <type>robot</type> </get-file-inventory> </repository-request></pre>	<pre><repository-response> <repository-folder> <path></path> <sub-folders> -- repository-folders (recursively) -- </sub-folders> <files> -- zero, one or more repository-file elements -- </files> <references> -- zero, one or more repository-file elements needed by robots in folder -- </references> </repository-folder> </repository-response></pre>
update-file	<pre><repository-request> <update-file> <repository-file-request>...</repository-file-request> <file-bytes></update-file> </repository-request></pre>	<pre><repository-response> <update-successful/> </repository-response></pre>
get-clusters	<pre><repository-request> <get-clusters online-roboserver='true'/'> </repository-request></pre>	<pre><repository-response> <clusters> <cluster name='ClusterName' ssl='false'> <roboserver host='localhost' port='50000' primary='true'/'> </cluster> </clusters> </repository-response></pre>
add-roboserver	<pre><repository-request> <add-roboserver> <cluster name='ClusterName' ssl='false'> <roboserver host='localhost' port='50000' primary='true'/'> </cluster> <roboserver host='localhost' port='50001' primary='true'/'> </add-roboserver> </repository-request></pre>	<pre><repository-response> <clusters> <cluster name='ClusterName' ssl='false'> <roboserver host='localhost' port='50000' primary='true'/'> <roboserver host='localhost' port='50001' primary='true'/'> </cluster> </clusters> </repository-response></pre>
delete-roboserver	<pre><repository-request> <add-roboserver> <cluster name='ClusterName' ssl='false'> <roboserver host='localhost' port='50000' primary='true'/'> <roboserver host='localhost' port='50001' primary='true'/'> </cluster> <roboserver host='localhost' port='50001' primary='true'/'> </add-roboserver> </repository-request></pre>	<pre><repository-response> <cluster name='ClusterName' ssl='false'> <roboserver host='localhost' port='50000' primary='true'/'> </cluster> </repository-response></pre>

Méthode	Exemple de requête	Exemple de réponse
delete-folder	<repository-request> <delete-folder> <project-name>Default project</project-name> <path>path/to/empty/folder</path> </delete-folder> </repository-request>	<repository-response> <delete-successful/> </repository-response>
move-file	<repository-request> <move-file> <repository-file-request>...</repository-file-request> <path>new/destination/path</path> </move-file> </repository-request>	<repository-response> <update-successful/> </repository-response>
Rename-robot	<repository-request> <rename-robot> <repository-file-request>...</repository-file-request> <file-name>newnameofrobot</file-name> </rename-robot> </repository-request>	<repository-response> <update-successful/> </repository-response>

! Les noms de robot, de type, de snippet et de ressource doivent être spécifiés sous la forme d'un chemin complet. Le chemin complet est relatif à votre dossier de projet.

Le déploiement se fait en publiant les octets bruts (le flux d'octets est envoyé en tant que corps de publication) aux URL suivantes. Voici un exemple où le répertoire est déployé sur `http://localhost:8080/ManagementConsole`.

Méthodes des opérations de déploiement :

Opération	URL
deploy robot	<code>http://localhost:8080/ManagementConsole/secure/RepositoryAPI?format=bytes&operation=deployRobot& projectName=Default project&fileName=DoNothing.robot&failIfExists=true</code>
deploy type	<code>http://localhost:8080/ManagementConsole/secure/RepositoryAPI?format=bytes&operation=deployType& projectName=Default project&fileName=InputA.type&failIfExists=true</code>
deploy Snippet	<code>http://localhost:8080/ManagementConsole/secure/RepositoryAPI?format=bytes&operation=deploySnippet& projectName=Default project&fileName=A.snippet&failIfExists=true</code>
deploy resource	<code>http://localhost:8080/ManagementConsole/secure/RepositoryAPI?format=bytes&operation=deployResource& projectName=Default project&fileName=resource.txt&failIfExists=true</code>
deploy library	<code>http://localhost:8080/ManagementConsole/secure/RepositoryAPI?format=bytes&operation=deployLibrary& projectName=Default project&fileName=NA&failIfExists=true</code>

Si l'authentification est activée sur Management Console, l'URL `http://localhost:8080/ManagementConsole/secure/RepositoryAPI` est protégée par l'authentification de base. Cela vous permet d'inclure des identifiants dans l'URL de la manière suivante : `http://username:password@localhost:8080/ManagementConsole/secure/RepositoryAPI`.

Chapitre 2

Guide du programmeur .NET

Ce chapitre décrit comment exécuter des robots à l'aide de l'API Kofax RPA .NET. Ce guide suppose que vous savez comment écrire des robots simples et que vous connaissez le langage de programmation C#.

Vous pouvez trouver des informations sur des classes .NET spécifiques dans l'aide compilée, `robosuite-dotnet-api.chm`, située dans votre dossier de documentation hors ligne. Pour plus d'informations, voir le *Guide d'installation Kofax RPA*.

.NET basique

En utilisant l'API .NET, toute application .NET peut devenir un client pour un RoboServer. La version 4.0 du .NET Framework est prise en charge. Si une version plus récente est installée sur votre ordinateur, vérifiez qu'elle est rétrocompatible, telle que .NET 4.5.

En plus d'exécuter des robots qui stockent des données dans une base de données, vous pouvez également demander aux robots de renvoyer des données directement à l'application cliente. Voici quelques exemples :

- Utilisez plusieurs robots pour effectuer une recherche qui regroupe les résultats de plusieurs sources en temps réel.
- Exécutez un robot en réponse à un événement à l'arrière-plan de votre application. Par exemple, exécutez un robot lorsqu'un nouvel utilisateur s'inscrit, pour créer des comptes sur des systèmes web non intégrés directement à l'arrière-plan de votre application.

Ce guide présente les classes principales et explique comment les utiliser pour exécuter des robots. Nous décrirons également comment fournir des entrées aux robots et contrôler leur exécution sur RoboServer.

L'API .NET est un fichier .dll /API/robosuite-dotnet-api/lib/robosuite-dotnet-api.dll qui se trouve dans le dossier d'installation Kofax RPA (voir la rubrique « Dossiers importants dans Kofax RPA » dans le *Guide d'installation* pour plus d'informations). Tous les exemples de ce guide se trouvent dans /API/robosuite-dotnet-api/examples. log4net.dll est une bibliothèque tierce requise située à côté du fichier de l'API .NET.

Premier exemple

Voici le programme requis pour exécuter le robot nommé `NewsMagazine.robot`, qui se trouve dans le dossier Tutoriels du projet par défaut. Le robot produit ses résultats à l'aide de l'activité d'étape Valeur de retour, ce qui facilite la gestion des données de sortie par programmation à l'aide de l'API. D'autres robots (généralement ceux exécutés dans une planification par Management

Console) stockent leurs données directement dans une base de données à l'aide de l'activité d'étape Stocker dans la base de données, auquel cas les données collectées par le robot ne sont pas renvoyées au client API.

Dans l'exemple suivant, le robot NewsMagazine est exécuté et les données de sortie sont traitées par programmation.

Exécuter un robot sans entrée :

```
using System;
using System.Collections.Generic;
using System.Text;
using Com.KapowTech.RoboSuite.Api;
using Com.KapowTech.RoboSuite.Api.Repository.Construct;
using Com.KapowTech.RoboSuite.Api.Construct;

namespace Examples
{
    class Program
    {
        static void Main(string[] args)
        {
            var server = new RoboServer("localhost", 50000);
            var ssl = false;
            var cluster = new Cluster("MyCluster", new RoboServer[]{ server}, ssl);

            Request.RegisterCluster(cluster); // you can only register a cluster
                                              once per application

            var request = new Request("Library:/Tutorials/NewsMagazine.robot");
            request.RobotLibrary = new DefaultRobotLibrary();
            RqlResult result = request.Execute("MyCluster");

            foreach (RqlObject value in result.GetOutputObjectsByName("Post"))
            {
                var title = value["title"];
                var preview = value["preview"];
                Console.WriteLine(title + ", " + preview);
            }
            Console.ReadKey();
        }
    }
}
```

Le tableau suivant répertorie les classes impliquées et leurs responsabilités.

RoboServer	Il s'agit d'un objet de valeur simple qui identifie un RoboServer qui peut exécuter des robots. Chaque RoboServer doit être activé par un Management Console et le KCU doit lui être assigné avant utilisation.
Cluster	Un cluster est un groupe de RoboServers fonctionnant comme une seule unité logique.
Request	Cette classe est utilisée pour construire la requête du robot. Avant de pouvoir exécuter des requêtes, vous devez enregistrer un cluster avec la classe Request.
DefaultRobotLibrary	Une bibliothèque de robot indique à RoboServer où trouver le robot identifié dans la requête. Des exemples ultérieurs explorent les différents types de bibliothèque de robot et quand/comment les utiliser.

RQLResult	Celui-ci contient le résultat d'une exécution du robot. Le résultat contient des réponses de valeur, avec des messages de journalisation et du serveur.
RQLObject	Chaque valeur renvoyée par un robot à l'aide de l'activité Valeur de retour est accessible en tant que RQLObject.

La première ligne indique à l'API que notre RoboServer s'exécute sur le port localhost 50000.

```
var server = new RoboServer("localhost", 50000);
```

Les lignes suivantes définissent un cluster avec un seul RoboServer. Le cluster est enregistré avec la classe Request, vous permettant d'exécuter des requêtes sur ce cluster. Chaque cluster ne peut être enregistré qu'une seule fois par application, ce qui s'effectue pendant l'initialisation de l'application.

Enregistrement d'un cluster :

```
var ssl = false;
var cluster = new Cluster("MyCluster", new RoboServer[]{ server}, ssl);
Request.RegisterCluster(cluster);
```

Le programme suivant crée une requête qui exécute le robot nommé `NewsMagazine.robot` et situé dans `Library:/Tutorials/Library:/`, faisant référence à la bibliothèque de robot configurée pour la requête. Ici est utilisée la `DefaultRobotLibrary`, qui demande à RoboServer de rechercher le robot dans le système de fichiers local des serveurs. Voir [Bibliothèques de robots](#) pour plus d'informations sur l'utilisation des bibliothèques de robots.

```
var request = new Request("Library:/Tutorials/NewsMagazine.robot");
request.RobotLibrary = new DefaultRobotLibrary();
```

La ligne suivante exécute le robot sur le cluster nommé `MyCluster` (le cluster précédemment enregistré) et renvoie le résultat une fois l'exécution du robot terminée. Si une erreur se produit pendant l'exécution du robot, une exception est déclenchée ici.

```
RqlResult result = request.Execute("MyCluster");
```

Finalement, nous traitons les valeurs extraites. Tout d'abord, nous récupérons toutes les valeurs extraites du type nommé `Post` et les parcourons. Pour chaque `RQLObject`, nous accédons aux attributs du type `Post` et nous affichons le résultat. Les attributs et les mappages sont traités dans une section ultérieure.

```
foreach (RqlObject value in result.GetOutputObjectsByName("Post")) {
    var title = value["title"];
    var preview = value["preview"];
    Console.WriteLine(title + ", " + preview);
```

Entrée de robot

La plupart des robots exécutés via l'API sont paramétrés par une entrée, telle qu'un mot-clé de recherche ou des identifiants de connexion. L'entrée dans un robot fait partie de la requête envoyée à RoboServer et est fournie à l'aide de la méthode `createInputVariable` sur la requête.

Entrée utilisant `RQLObjectBuilder` implicite

```
var request = new Request("Library:/Tutorials/Input.robot");
request.CreateInputVariable("userLogin").SetAttributeEntry
    ("username", "scott").SetAttributeEntry("password", "tiger");
```

Dans le programme précédent, nous créons une `Request` et utilisons `CreateInputVariable` pour créer une variable d'entrée nommée `userLogin`. Nous utilisons ensuite `setAttribute` pour configurer les attributs Nom d'utilisateur et Mot de passe de la variable d'entrée.

L'exemple précédent est une notation abrégée courante, mais peut également être exprimé plus en détail à l'aide de `RqlObjectBuilder` :

```
var request = new Request("Library:/NewsMagazine.robot");
RqlObjectBuilder userLogin = request.CreateInputVariable("userLogin");
userLogin.SetAttributeEntry("username", "scott");
userLogin.SetAttributeEntry("password", "tiger");
```

Les deux exemples sont identiques. Le premier invoque la méthode en cascade sur le `RqlObjectBuilder` anonyme et est donc plus court.

Lorsque RoboServer reçoit cette requête, les événements suivants se produisent :

- RoboServer charge `Input.robot` (à partir d'une `RobotLibrary` configurée pour la requête).
- RoboServer vérifie que le robot a une variable nommée `userLogin` et que cette variable est marquée comme entrée.
- RoboServer vérifie ensuite que les attributs que nous avons configurés en utilisant `setAttribute` sont compatibles avec le type de variable `userLogin`. Par conséquent, le type doit avoir des attributs nommés Nom d'utilisateur et Mot de passe et ils doivent tous deux être des attributs textuels (la section suivante décrit le mappage entre l'API et les attributs Design Studio).
- Si toutes les variables d'entrée sont compatibles, RoboServer démarre l'exécution du robot.

Si un robot nécessite plusieurs variables d'entrée, vous devez toutes les créer pour exécuter le robot. Vous n'avez qu'à configurer les attributs requis ; tout attribut non requis que vous ne configurez pas via l'API aura juste une valeur nulle. Si vous avez un robot qui nécessite une connexion à Facebook et Twitter, vous pouvez définir l'entrée comme suit.

```
Request request = new Request("Library:/Input.robot");
request.CreateInputVariable("facebook").SetAttributeEntry
    ("username", "scott").SetAttributeEntry("password", "facebook123");
request.CreateInputVariable("twitter").SetAttributeEntry
    ("username", "scott").SetAttributeEntry("password", "twitter123");
```

Types d'attribut

Lorsque vous définissez un nouveau type dans Design Studio, sélectionnez un type d'attribut pour chaque attribut. Certains attributs peuvent contenir du texte, comme du texte court, du texte long, un mot de passe, du HTML, du XML, et lorsqu'ils sont utilisés à l'intérieur d'un robot, il peut être nécessaire de stocker du texte dans ces attributs. Si vous stockez du texte dans un attribut XML, ce texte doit être un document XML valide. Cette validation se produit lorsque le type est utilisé dans un robot, mais comme l'API ne sait rien sur ce type, elle ne valide pas les valeurs d'attribut de la même manière. Par conséquent, l'API n'a que huit types d'attributs et Design Studio a 19 types disponibles. Ce tableau montre le mappage entre les types d'attribut Design Studio et l'API.

Mappage API sur Design Studio

Type d'attribut API	Type d'attribut RoboServer
Texte	Texte court, texte long, mot de passe, HTML, XML, propriétés, langue, pays, devise, clé de recherche

Type d'attribut API	Type d'attribut RoboServer
Entier	Entier
Booléen	Booléen
Nombre	Nombre
Caractère	Caractère
Date	Date
Session	Session
Binaire	Binaire, Image, PDF

Les types d'attributs API sont ensuite mappés sur .NET de la manière suivante.

Types .NET pour les attributs

Type d'attribut API	Classe Java
Texte	System.String (chaîne)
Entier	System.Int64
Booléen	System.Boolean (booléen)
Nombre	System.Double (double)
Caractère	System.Char (caractère)
Date	System.DateTime
Session	Com.Kapowtech.Robosuite.Api.Construct.Session
Binaire	Com.Kapowtech.Robosuite.Api.Construct.Binary

La méthode `RqlObjectBuilder.setAttribute` est surchargée, vous n'avez donc pas besoin de spécifier explicitement le type d'attribut lors de la configuration d'un attribut via l'API, tant que la classe .NET correcte est utilisée comme argument. Voici un exemple qui montre comment définir les attributs d'un objet avec tous les types d'attributs Design Studio possibles.

Utilisation recommandée de `setAttribute`:

```
RqlObjectBuilder inputBuilder = request.CreateInputVariable("AllTypes");
inputBuilder.SetAttributeEntry("anInt", 42L);
inputBuilder.SetAttributeEntry("aNumber", 12.34d);
inputBuilder.SetAttributeEntry("aBoolean", true);
inputBuilder.SetAttributeEntry("aCharacter", 'c');
inputBuilder.SetAttributeEntry("aShortText", "some text");
inputBuilder.SetAttributeEntry("aLongText", "a longer text");
inputBuilder.SetAttributeEntry("aPassword", "secret");
inputBuilder.SetAttributeEntry("aHTML", "<html>text</html>");
inputBuilder.SetAttributeEntry("anXML", "<tag>text</tag>");
inputBuilder.SetAttributeEntry("aDate", DateTime.Now);
inputBuilder.SetAttributeEntry("aBinary", (Binary) null);
inputBuilder.SetAttributeEntry("aPDF", (Binary) null);
inputBuilder.SetAttributeEntry("anImage", (Binary) null);
inputBuilder.SetAttributeEntry("aProperties", "name=value\nname2=value2");
inputBuilder.SetAttributeEntry("aSession", (Session) null);
inputBuilder.SetAttributeEntry("aCurrency", "USD");
inputBuilder.SetAttributeEntry("aCountry", "US");
```

```
inputBuilder.SetAttributeEntry("aLanguage", "en");
inputBuilder.SetAttributeEntry("aRefindKey", "Never use as input");
```

Dans l'exemple précédent, nous devons convertir des valeurs nulles car le compilateur Java ne peut pas autrement déterminer laquelle des versions surchargées de la méthode `SetAttributeEntry` appeler. Toutefois, comme les attributs non configurés seront automatiquement nuls, vous n'avez jamais besoin de définir explicitement la valeur nulle.

Il est possible de spécifier explicitement le `Attribute` et le `AttributeType` lors de la création d'une entrée à l'aide de l'API. Cette approche n'est pas recommandée, mais peut être nécessaire dans de rares cas et ressemblerait à ce qui suit.

Utilisation non recommandée de `setAttribute`

```
RqlObjectBuilder inputBuilder = request.CreateInputVariable("alltypes");
inputBuilder.SetAttributeEntry(new AttributeEntry("anInt", "42",
    AttributeEntryType.Integer));
inputBuilder.SetAttributeEntry(new AttributeEntry("aNumber", "12.34",
    AttributeEntryType.Number));
inputBuilder.SetAttributeEntry(new AttributeEntry("aBoolean", "true",
    AttributeEntryType.Boolean));
inputBuilder.SetAttributeEntry(new AttributeEntry("aCharacter", "c",
    AttributeEntryType.Character));
inputBuilder.SetAttributeEntry(new AttributeEntry("aShortText", "some text",
    AttributeEntryType.Text));
inputBuilder.SetAttributeEntry(new AttributeEntry("aLongText", "a longer text",
    AttributeEntryType.Text));
inputBuilder.SetAttributeEntry(new AttributeEntry("aPassword", "secret",
    AttributeEntryType.Text));
inputBuilder.SetAttributeEntry(new AttributeEntry("aHTML", "<html>text</html>",
    AttributeEntryType.Text));
inputBuilder.SetAttributeEntry(new AttributeEntry("anXML", "<tag>text</tag>",
    AttributeEntryType.Text));
inputBuilder.SetAttributeEntry(new AttributeEntry("aDate",
    "2012-01-15 23:59:59.123", AttributeEntryType.Date));

inputBuilder.SetAttributeEntry(new AttributeEntry("aBinary", null,
    AttributeEntryType.Binary));
inputBuilder.SetAttributeEntry(new AttributeEntry("aPDF", null,
    AttributeEntryType.Binary));
inputBuilder.SetAttributeEntry(new AttributeEntry("anImage", null,
    AttributeEntryType.Binary));
inputBuilder.SetAttributeEntry(new AttributeEntry("aProperties",
    "name=value\nname2=value2", AttributeEntryType.Text));
inputBuilder.SetAttributeEntry(new AttributeEntry("aCurrency", "USD",
    AttributeEntryType.Text));
inputBuilder.SetAttributeEntry(new AttributeEntry("aCountry", "US",
    AttributeEntryType.Text));
inputBuilder.SetAttributeEntry(new AttributeEntry("aLanguage", "en",
    AttributeEntryType.Text));
inputBuilder.SetAttributeEntry(new AttributeEntry("aRefindKey",
    "Never use this as input", AttributeEntryType.Text));
```

Toutes les valeurs d'attribut doivent être fournies sous forme de chaînes. Les valeurs de chaîne sont ensuite converties en objets .NET appropriés en fonction du `AttributeEntryType` fourni. Cela n'est utile que si vous créez d'autres API génériques en plus de l'API Kofax RPA .NET.

Paramètres d'exécution

En plus de la méthode `CreateInputVariable`, la requête contient un certain nombre de propriétés qui contrôlent la façon dont le robot s'exécute sur un RoboServer.

Méthodes de contrôle d'exécution sur requête

MaxExecutionTime	Contrôle le temps d'exécution du robot en secondes. Une fois ce temps écoulé, le robot est arrêté par RoboServer. La temporisation ne démarre pas tant que le robot ne commence pas à s'exécuter, donc si le robot est mis en file d'attente sur RoboServer, cela n'est pas pris en compte.
StopOnConnectionLost	Si vrai (par défaut), le robot s'arrête si RoboServer découvre que la connexion à l'application cliente est perdue. Si vous configurez cette valeur sur faux et que votre programme n'est pas écrit pour gérer cette valeur, votre application ne fonctionnera pas comme prévu.
StopRobotOnApiException	Si vrai (par défaut), le robot est arrêté par RoboServer après que la première exception API ait été déclenchée. Par défaut, la plupart des étapes d'un robot déclenchent une exception API si l'étape ne s'exécute pas. Configurez cette valeur dans l'onglet Gestion des erreurs de l'étape. Lorsqu'elle est définie sur faux, le robot continue de s'exécuter quelles que soient les exceptions d'API. Toutefois, à moins que votre application n'utilise le <code>IRobotResponseHandler</code> pour diffuser en streaming les résultats, une exception est toujours déclenchée par <code>Execute()</code> . Soyez prudent lorsque vous la configurez sur faux.
Username, Password	Configure les identifiants RoboServer. RoboServer peut être configuré pour exiger une authentification. Lorsque cette option est activée, le client doit fournir des identifiants ou RoboServer rejette la requête.
RobotLibrary	Assigne une <code>RobotLibrary</code> à la requête. Une bibliothèque de robot indique à RoboServer où trouver le robot identifié dans la requête. Pour plus d'exemples liés aux différents types de bibliothèque et à leur utilisation, voir Bibliothèques de robots .

ExecutionId	<p>Vous permet de configurer le <code>executionId</code> pour cette requête. Si vous n'en fournissez pas, RoboServer en génère un automatiquement. L'ID d'exécution est utilisé pour la journalisation et il est également nécessaire pour arrêter le robot par programmation. L'ID doit être globalement unique (dans le temps). Si deux robots utilisent le même ID d'exécution, les journaux seront incohérents.</p> <p>La définition de cette propriété est utile si vos robots font partie d'un workflow plus large et que vous avez déjà un identifiant unique dans votre application cliente, car cela vous permet de joindre les journaux du robot au reste du système.</p>
setProject (String)	<p>Cette méthode n'est utilisée qu'à des fins de journalisation. Management Console utilise ce champ pour lier les messages du journal au projet, afin que les vues du journal puissent être filtrées par projet.</p> <p>Si votre application n'utilise pas la <code>RepositoryRobotLibrary</code>, vous devez définir cette valeur pour informer le système de journalisation RoboServer à quel projet (le cas échéant) ce robot appartient.</p>

Bibliothèques de robots

Dans Design Studio, les robots sont regroupés en projets. Si vous regardez dans le système de fichiers, vous verrez que ces projets sont représentés par un dossier avec la seule contrainte qu'il doit contenir un dossier nommé `Bibliothèque`.

Lorsque vous créez la requête d'exécution pour RoboServer, vous identifiez le robot par une URL de robot :

```
Request request = new Request("Library:/Input.robot");
```

Ici, `Library:/` est une référence symbolique à une bibliothèque de robot, dans laquelle RoboServer devra rechercher le robot. Le `RobotLibrary` est spécifié dans le constructeur de la façon suivante :

```
request.setRobotLibrary(new DefaultRobotLibrary());
```

Trois implémentations de bibliothèque de robot différentes sont disponibles. Celle à sélectionner dépend de votre environnement de déploiement.

Bibliothèques de robots

Type de bibliothèque	Description
DefaultRobotLibrary	<p>Cette bibliothèque configure RoboServer pour rechercher le robot dans le dossier du projet actuel. Ce dossier est défini dans l'application Paramètres.</p> <p>Si vous avez plusieurs RoboServers, vous devez déployer vos robots sur tous les RoboServers.</p> <p>Cette bibliothèque de robot n'est pas mise en cache, le robot est donc recharge depuis le disque à chaque exécution. Cette approche rend la bibliothèque utilisable dans un environnement de développement où les robots changent souvent, mais elle n'est pas adaptée à un environnement de production.</p>
EmbeddedFileBasedRobotLibrary	<p>Cette bibliothèque est intégrée dans la requête d'exécution envoyée à RoboServer. Pour créer cette bibliothèque, vous devez créer un fichier zip contenant les robots et toutes ses dépendances (types, snippets et ressources). Pour ce faire, utilisez le menu Outils > Créer un fichier de bibliothèque du robot dans Design Studio.</p> <p>La bibliothèque est envoyée avec chaque requête, ce qui ajoute une surcharge pour les grandes bibliothèques, mais les bibliothèques sont mises en cache sur RoboServer, ce qui offre les meilleures performances possibles.</p> <p>L'un des avantages est que les robots et le programme peuvent être déployés comme une seule unité, ce qui permet une migration propre d'un environnement QA vers un environnement de production. Toutefois, si les robots changent souvent, vous devrez les redéployer souvent.</p> <p>Vous pouvez utiliser le code suivant pour configurer la bibliothèque de robot intégrée pour votre requête.</p> <pre>var request = new Request ("Library:/Tutorials/NewsMagazine. robot"); var stream = new FileStream ("c:\embeddedLibrary.robotlib", FileMode.Open); request.RobotLibrary = new EmbeddedFileBasedRobotLibrary (stream);</pre>

Type de bibliothèque	Description
RepositoryRobotLibrary	<p>C'est la <code>RobotLibrary</code> la plus flexible.</p> <p>Cette bibliothèque utilise le répertoire intégré de Management Console comme bibliothèque de robot. Lorsque vous utilisez cette bibliothèque, RoboServer contacte le Management Console qui envoie une bibliothèque de robot contenant le robot et ses dépendances.</p> <p>La mise en cache s'effectue par robot, à la fois dans Management Console et RoboServer. Dans Management Console, la bibliothèque générée est mise en cache en fonction du robot et de ses dépendances. Sur RoboServer, le cache est basé sur un délai d'attente, il n'est donc pas nécessaire de demander le Management Console pour chaque requête. En outre, la bibliothèque se charge entre RoboServer et Management Console utilisant la mise en cache HTTP publique/ privée, pour réduire davantage la bande passante.</p> <p>Si <code>NewsMagazine.robot</code> est chargé dans le Management Console, vous pouvez utiliser la bibliothèque de robot du répertoire lors de l'exécution du robot :</p> <pre>var request = new Request ("Library:/Tutorials/NewsMagazine. robot"); request.RobotLibrary = new RepositoryRobotLibrary ("http://localhost:50080", "Default Project", 60000);</pre> <p>Cette commande demande à RoboServer de charger le robot depuis un Management Console local et de le mettre en cache pendant une minute avant de vérifier avec le Management Console pour voir si une nouvelle version du robot (son type et ses snippets) est disponible.</p> <p>En outre, toute ressource chargée via le protocole <code>Library:/</code> entraîne RoboServer à demander la ressource directement à partir du Management Console.</p>

.NET avancé

Cette section décrit les fonctionnalités avancées de l'API, y compris le streaming des données de sortie, la journalisation et la configuration SSL, ainsi que l'exécution en parallèle.

Charger la distribution

Dans le `RequestExecutor`, l'exécuteur reçoit un tableau de RoboServers. Au fur et à mesure que l'exécuteur est construit, il essaie de se connecter à chaque RoboServer. Une fois connecté, il envoie une requête ping à chaque RoboServer pour découvrir comment le serveur est configuré.

Exécuteur à charge équilibrée

```
RoboServer prod = new RoboServer("prod.kapow.local", 50000);
RoboServer prod2 = new RoboServer("prod2.kapow.local", 50000);
Cluster cluster = new Cluster("Prod", new RoboServer[]{ prod, prod2}, false);
Request.RegisterCluster(cluster);
```

La charge est distribuée à chaque RoboServer en ligne dans le cluster, en fonction du nombre d'emplacements d'exécution inutilisés sur le RoboServer. La requête suivante est toujours distribuée au RoboServer avec le plus d'emplacements disponibles. Le nombre d'emplacements d'exécution disponibles est obtenu via la réponse ping initiale, et l'exécuteur effectue le suivi de chaque robot qu'il démarre et qu'il arrête. Le nombre d'emplacements d'exécution sur un RoboServer est déterminé par le paramètre **Nombre maximum de robots simultanés** dans la section Management Console > Administration > RoboServers.

Si un RoboServer passe hors ligne, il ne reçoit aucune requête d'exécution de robot avant d'avoir répondu avec succès à la requête ping.

Règle du client unique

Par défaut, les connexions API sont limitées à 20 connexions. Toutefois, pour garantir les meilleures performances, nous vous recommandons de n'avoir qu'un seul client API utilisant un cluster donné de RoboServers. Si vous avez trop de machines virtuelles JVM exécutant des robots sur les mêmes RoboServers, cela entraînera une baisse des performances.

Bien que ce qui suit ne soit pas recommandé, si votre environnement nécessite la gestion d'un volume plus élevé, vous pouvez configurer la limite de connexion en ajustant la propriété système `kapow.max.multiplexing.clients` dans le fichier `common.conf`.

Streaming des données

Si vous avez besoin de présenter les résultats d'une exécution de robot en temps réel, vous pouvez utiliser l'API pour renvoyer immédiatement les valeurs extraites, au lieu d'attendre que le robot termine son exécution et accède au `RqlResult`.

L'API offre la possibilité de recevoir un rappel chaque fois que l'API reçoit une valeur renvoyée par le robot. Faites-le via l'interface `IRobotResponseHandler`.

Streaming de la réponse à l'aide de `AbstractFailFastRobotResponseHandler`

```
using System;
using Com.KapowTech.RoboSuite.Api;
using Com.KapowTech.RoboSuite.Api.Repository.Construct;
using Com.KapowTech.RoboSuite.Api.Construct;
using System.IO;
using Com.KapowTech.RoboSuite.Api.Engine.Hotstandby;

namespace Examples
```

```

{
    public class DataStreaming {

        public static void Main(String[] args)  {

            var server = new RoboServer("localhost", 50000);
            var cluster = new Cluster("MyCluster", new RoboServer[] { server },
                false);
            Request.RegisterCluster(cluster);

            var request = new Request("Library:/Tutorials/NewsMagazine.robot");
            IRobotResponseHandler handler = new SampleResponseHandler();
            request.Execute("MyCluster", handler);
        }
    }

    public class SampleResponseHandler : AbstractFailFastRobotResponseHandler
    {
        override public void HandleReturnedValue(RobotOutputObjectResponse
            response, IStoppable stoppable)
        {
            var title = response.OutputObject["title"];
            var preview = response.OutputObject["preview"];
            Console.WriteLine(title + ", " + preview);
        }
    }
}

```

L'exemple précédent utilise la deuxième méthode d'exécution de la requête, qui attend un `RobotResponseHandler` en plus du nom du cluster sur lequel exécuter le robot. Dans cet exemple, créez un `IRobotResponseHandler` en étendant `AbstractFailFastRobotResponseHandler`, qui fournit la gestion des erreurs par défaut, pour gérer les valeurs renvoyées par le robot.

La méthode `handleReturnedValue` est appelée chaque fois que l'API reçoit une valeur renvoyée depuis `RoboServer`. Le `AbstractFailFastRobotResponseHandler` utilisé dans cet exemple déclenche des exceptions de la même manière que la méthode d'exécution sans streaming. Cela signifie qu'une exception est levée en réponse à toutes les exceptions d'API générées par le robot.

Le `IRobotResponseHandler` a plusieurs méthodes qui peuvent être regroupées en trois catégories.

Événements du cycle de vie du robot

Méthodes appelées lorsque l'état d'exécution du robot change sur `RoboServer`, par exemple lorsqu'il démarre et termine l'exécution.

Événements des données du robot

Méthodes appelées lorsque le robot renvoie des données ou des erreurs à l'API.

Gestion des erreurs supplémentaires

Méthodes appelées en raison d'une erreur à l'intérieur de `RoboServer` ou dans l'API.

RobotResponseHandler – événements du cycle de vie du robot

Nom de méthode	Description
void requestSent (RoboServer roboServer, ExecuteRequest request)	Appelée lorsque le RequestExecutor trouve le serveur qui exécute la requête.
void requestAccepted (String executionId)	Appelée lorsque le RoboServer trouvé accepte la requête et la place dans sa file d'attente.
void RobotStarted (IStoppable stoppable)	Appelée lorsque le RoboServer commence à exécuter le robot. Cela se produit généralement immédiatement après la mise en file d'attente du robot, à moins que le RoboServer ne soit soumis à une charge importante ou utilisé par plusieurs clients API.
void robotDone (RobotDoneEvent reason)	Appelée lorsque le robot a terminé son exécution dans RoboServer. Le RobotDoneEvent est utilisé pour spécifier si l'exécution s'est terminée normalement, en raison d'une erreur, ou si elle a été arrêtée.

RobotResponseHandler – événements des données du robot

Nom de méthode	Description
void HandleReturnedValue (RobotOutputObjectResponse response, IStoppable stoppable)	Appelée lorsque le robot exécute une activité Valeur de retour et que la valeur est renvoyée via le socket à l'API.
void HandleRobotError (RobotErrorResponse response, IStoppable stoppable)	Appelée lorsque le robot déclenche une exception API. Dans des circonstances normales, le robot cesse de s'exécuter après la première exception API. Ce comportement peut être remplacé en utilisant Request.StopRobotOnApiException = false, auquel cas cette méthode est appelée plusieurs fois. Cette approche est utile si vous voulez qu'un robot de streaming des données continue à s'exécuter indépendamment des erreurs générées.
void HandleWriteLog (RobotMessageResponse response, IStoppable stoppable)	Appelée si le robot exécute l'activité Écrire le journal. Ceci est utile pour fournir des informations de journalisation supplémentaires depuis un robot.

RobotResponseHandler – gestion des erreurs supplémentaires

Nom de méthode	Description
void HandleServerError (ServerErrorResponse response, IStoppable stoppable)	Appelée si RoboServer génère une erreur. Par exemple si le serveur est trop occupé pour traiter des requêtes, ou si une erreur se produit à l'intérieur de RoboServer, ce qui l'empêche de démarrer le robot.
void handleError (RQLEException e, IStoppable stoppable)	Appelée si une erreur se produit dans l'API. Plus fréquemment, si le client perd la connexion avec RoboServer.

De nombreuses méthodes incluent un objet `IStoppable`, qui peut être utilisé pour l'arrêt en réponse à une erreur spécifique ou à une valeur renvoyée.

Certaines de ces méthodes vous permettent de déclencher une `RQLEException`. Le thread qui appelle le handler est le thread qui appelle `Request.Execute()`, ce qui signifie que les exceptions déclenchées peuvent surcharger la pile d'appels. Si vous déclenchez une exception en réponse à `handleReturnedValue`, `handleRobotError` ou `handleWriteLog`, il est de votre responsabilité d'invoquer `Stoppable.stop()`, ou le robot peut continuer à s'exécuter même si l'appel à `Request.Execute()` est terminé.

Le streaming des données est le plus souvent utilisé dans l'un des cas suivants.

- Application web basée sur Ajax, où les résultats sont présentés à l'utilisateur en temps réel. Si les données ne sont pas diffusées, les résultats ne peuvent pas s'afficher tant que le robot n'a pas fini son exécution.
- Des robots qui renvoient tellement de données que le client ne pourrait pas tout garder en mémoire tout au long de l'exécution du robot.
- Processus qui doivent être optimisés pour que les valeurs extraites soient traitées parallèlement à l'exécution du robot.
- Processus qui stockent des données dans des bases de données sous un format personnalisé.
- Robots qui devraient ignorer ou nécessiter une gestion personnalisée des exceptions d'API (voir l'exemple suivant).

Collecte des réponses et des erreurs à l'aide de `AbstractFailFastRobotResponseHandler` :

```
using System;
using System.Collections;
using System.Collections.Generic;
using Com.KapowTech.RoboSuite.Api;
using Com.KapowTech.RoboSuite.Api.Repository.Construct;
using Com.KapowTech.RoboSuite.Api.Construct;
using System.IO;
using Com.KapowTech.RoboSuite.Api.Engine.Hotstandby.Interfaces;

namespace Examples
{
    public class DataStreaming
    {

        public static void Main(String[] args)
        {

            var server = new RoboServer("localhost", 50000);
            var cluster = new Cluster("MyCluster", new RoboServer[] { server },
                false);
            Request.RegisterCluster(cluster);

            var request = new Request("Library:/Tutorials/NewsMagazine.robot");
            request.StopRobotOnApiException = false; // IMPORTANT!!

            ErrorCollectingRobotResponseHandler handler =
                new ErrorCollectingRobotResponseHandler();
            request.Execute("MyCluster", handler); // blocks until robot is
                done, or handler throws an exception

            Console.WriteLine("Extracted values:");
            foreach (RobotOutputObjectResponse response in handler.
                GetOutput())
            {

```

```

        var title = response.OutputObject["title"];
        var preview = response.OutputObject["preview"];
        Console.WriteLine(title + ", " + preview);
    }

    Console.WriteLine("Errors:");
    foreach (RobotErrorResponse error in handler.GetErrors())
    {
        Console.WriteLine(error.ErrorLocationCode + ", " + error.
            ErrorMessage);
    }
}

public class ErrorCollectingRobotResponseHandler :
    AbstractFailFastRobotResponseHandler {

    private IList<RobotErrorResponse> _errors =
        new List<RobotErrorResponse>();
    private IList<RobotOutputObjectResponse> _output =
        new List<RobotOutputObjectResponse>();

    override public void HandleReturnedValue(RobotOutputObjectResponse
        response, IStoppable stoppable) {
        _output.Add(response);
    }

    override public void HandleRobotError(RobotErrorResponse response,
        IStoppable stoppable) {
        // do not call super as this will stop the robot
        _errors.Add(response);
    }

    public IList<RobotErrorResponse> GetErrors() {
        return _errors;
    }

    public IList<RobotOutputObjectResponse> GetOutput() {
        return _output;
    }
}
}

```

L'exemple précédent montre comment utiliser un `IRobotResponseHandler` qui collecte les valeurs renvoyées et les erreurs. Ce type de handler est utile si le robot doit continuer à s'exécuter même en cas d'erreurs, ce qui peut être utile si le site web est instable et parfois expire. Notez que seules les erreurs du robot (exceptions d'API) sont collectées par le gestionnaire. Si la connexion à RoboServer est perdue, `Request.Execute()` déclenche toujours une `RQLEException`, et le robot est arrêté par RoboServer.

Pour plus d'informations, voir la documentation `IRobotResponseHandler`.

SSL

L'API communique avec RoboServer via un `RQLService`, un composant RoboServer qui écoute les requêtes API sur un port réseau spécifique. Lorsque vous démarrez un RoboServer, vous spécifiez s'il doit utiliser le service SSL chiffré, ou le service socket brut, ou les deux (en utilisant deux ports

différents). Tous les RoboServers dans un cluster doivent exécuter le même RQLService (bien que le port puisse être différent).

En supposant qu'un RoboServer est démarré avec le RQLService SSL sur le port 50043 :

```
RoboServer -service ssl:50043
```

Le code suivant peut être utilisé.

```
RoboServer server = new RoboServer("localhost", 50043);
boolean ssl = true;
Cluster cluster = new Cluster("MyCluster", new RoboServer[] {server}, ssl);
Request.RegisterCluster(cluster);
```

Vous devez créer le cluster en tant que cluster SSL et spécifier le port SSL utilisé par chaque RoboServer. Désormais, toutes les communications entre RoboServer et l'API seront chiffrées.

En plus du chiffrement des données, le protocole SSL offre la possibilité de vérifier l'identité de la partie distante. Ce type de vérification est très important sur Internet. Le plus souvent, votre client API et les RoboServers sont sur le même réseau local ; vous avez donc rarement besoin de vérifier l'identité de l'autre partie, mais l'API prend en charge cette fonctionnalité si cela s'avère nécessaire.

Voir [Exemples](#) pour savoir comment compiler et exécuter l'exemple SSL inclus.

Intégration de répertoire

Dans Management Console, vous spécifiez également les clusters de RoboServers, qui sont utilisés pour exécuter des robots planifiés, ainsi que des robots dont l'exécution s'effectue comme des services REST. L'API vous permet d'utiliser le `RepositoryClient` pour obtenir des informations sur le cluster depuis la Management Console. Pour plus d'informations, voir la documentation `RepositoryClient`.

Intégration de répertoire

```
using System;
using Com.KapowTech.RoboSuite.Api;
using Com.KapowTech.RoboSuite.Api.Construct;
using Com.KapowTech.RoboSuite.Api.Repository.Engine;

namespace Examples
{
    public class RepositoryIntegration
    {
        public static void Main(String[] args)
        {
            string userName = "admin";
            string password = "admin";
            RepositoryClient client = new RepositoryClient
                ("http://localhost:50080", userName, password);

            Request.RegisterCluster(client, "Production");
            var request = new Request("Library:/Tutorials/NewsMagazine.robot");
            var result = request.Execute("Production");
            Console.WriteLine(result.ToString());
        }
    }
}
```

L'exemple précédent montre comment créer un `RepositoryClient` qui se connecte à une Management Console déployé sur le port localhost 50080.

Si le Management Console requiert une authentification, vous devez saisir un nom d'utilisateur et un mot de passe, sinon nul pour les deux. Lorsque vous enregistrez le `RepositoryClient`, vous spécifiez le nom d'un cluster qui existe sur la Management Console. Il interroge ensuite la Management Console pour obtenir une liste des RoboServers configurés pour ce cluster et vérifie toutes les deux minutes pour voir si la configuration du cluster est mise à jour sur la Management Console.

Cette intégration vous permet de créer un cluster sur la Management Console que vous pouvez modifier dynamiquement à l'aide de l'interface utilisateur de la Management Console. Lorsque vous utilisez un cluster Management Console avec l'API, l'utilisation doit être exclusive et vous ne devez pas l'utiliser pour planifier le robot, car cela enfreindrait la règle du client unique.

Executor Logger

Lorsque vous exécutez une requête, la méthode d'exécution déclenche une exception si un robot génère une erreur. D'autres types d'erreurs et d'avertissemens sont signalés via l'interface `ExecutorLogger`. Dans les exemples précédents, `ExecutionLogger` n'était pas fourni lors de l'exécution de robots, qui est l'implémentation par défaut qui écrit dans `System.out`.

Ce qui suit est un exemple de la façon dont le `ExecutorLogger` signale si l'un des RoboServers passe hors ligne. Dans cet exemple, un cluster est configuré avec un RoboServer qui n'est pas en ligne.

ExecutorLogger, exemple de serveur hors ligne :

```
RoboServer rs = new RoboServer("localhost", 50000);
Cluster cluster = new Cluster("name", new RoboServer[]{rs}, false);
Request.RegisterCluster(cluster);
```

Si vous exécutez cet exemple, il écrit ce qui suit dans la console.

ExecutorLogger, sortie de console RobotServer hors ligne :

```
RoboServer[Host=localhost, Port=50000]' went offline.
Com.KapowTech.RoboSuite.Api.Engine.UnableToConnectException:.....
```

Si vous n'avez pas besoin que votre application écrive directement dans `System.out`, vous pouvez fournir une implémentation `IExecutorLogger` différente lors de l'enregistrement du cluster :

Utiliser DebugExecutorLogger :

```
Request.RegisterCluster(cluster, new DebugExecutorLogger());
```

Cet exemple utilise le `DebugExecutorLogger()`, qui écrit également dans `System.out`, mais seulement si le débogage de l'API est activé. Alternativement, vous pouvez fournir votre propre implémentation de `ExecutorLogger`, pour contrôler la manière dont les messages d'erreur sont gérés.

Sous le capot

Cette section explique ce qui se passe sous le capot lorsque vous enregistrez un cluster et exécutez des requêtes.

Lorsque vous enregistrez un cluster avec la requête, un `RequestExecutor` est créé en arrière-plan. Ce `RequestExecutor` est stocké sur une carte en utilisant le nom de cluster comme clé. Lorsqu'une requête est exécutée, le nom de cluster fourni est utilisé pour rechercher le `RequestExecutor` associé et exécuter la requête.

Exécution normale

```
public static void Main(String[] args)
{
    RoboServer server = new RoboServer("localhost", 50000);
    Cluster cluster = new Cluster("MyCluster", new RoboServer[]{ server}, false);
    Request.RegisterCluster(cluster);

    var request = new Request("Library:/Tutorials/NewsMagazine.robot");
    request.RobotLibrary = new DefaultRobotLibrary();
    var result = request.Execute("MyCluster");
    Console.WriteLine(result);
}
```

Maintenant, écrivez le même exemple en utilisant directement le `hiddenRequestExecutor`.

Exécution sous le capot :

```
public static void Main(String[] args)
{
    RoboServer server = new RoboServer("localhost", 50000);
    Cluster cluster = new Cluster("MyCluster", new RoboServer[]{ server}, false);
    RequestExecutor executor = new RequestExecutor(cluster);

    var request = new Request("Library:/Tutorials/NewsMagazine.robot");
    request.RobotLibrary = new DefaultRobotLibrary();
    var result = executor.Execute(request);
    Console.WriteLine(result);
}
```

Le `RequestExecutor` est masqué par défaut, vous n'avez donc pas à le suivre. Vous ne pouvez créer qu'un `RequestExecutor` par cluster, donc si vous l'utilisez directement, vous devez stocker une référence dans toute votre application. L'utilisation de `Request.RegisterCluster(cluster)` signifie que vous pouvez ignorer les règles du cycle de vie et de `RequestExecutor`.

Le `RequestExecutor` contient l'état et la logique nécessaires, qui fournissent les fonctionnalités d'équilibrage de charge et de remplacement en cas de panne. L'utilisation directe de `RequestExecutor` offre également quelques fonctionnalités supplémentaires.

Fonctionnalités RequestExecutor

Lorsque le `RequestExecutor` n'est pas connecté à un répertoire, vous pouvez ajouter ou supprimer dynamiquement les RoboServers en appelant `AddRoboServer(...)` et `RemoveRoboServer(...)`. Ces méthodes modifient la liste de distribution utilisée dans le `RequestExecutor`.

La propriété `RequestExecutor.TotalAvailableSlots` contient le nombre d'emplacements d'exécution inutilisés sur tous les RoboServers dans la liste de distribution interne.

En utilisant ces méthodes, vous pouvez ajouter dynamiquement des RoboServers à votre `RequestExecutor` une fois que le nombre d'emplacements d'exécution disponibles devient faible.

Lorsque vous créez le `RequestExecutor`, vous pouvez éventuellement fournir un `IRqlEngineFactory`. Le `IRqlEngineFactory` vous permet de personnaliser quel `RQLProtocol` est utilisé lors de la connexion à un RoboServer. Cela n'est nécessaire que dans de rares circonstances, telles que l'utilisation d'un certificat client pour augmenter la sécurité. Voir le chapitre Certificats dans le *Guide de l'administrateur de Kofax RPA* pour plus d'informations.

API Repository

L'API Repository vous permet d'interroger le répertoire de Management Console pour récupérer une liste de projets, des robots et l'entrée requise pour appeler un robot. Elle vous permet également de déployer par programmation des robots, des types et des fichiers de ressources.

Client répertoire

La communication avec le répertoire s'effectue via le `RepositoryClient` dans l'espace de nom `Com.KapowTech.RoboSuite.Api.Repository.Engine`.

Récupérer des projets depuis le répertoire

```
string UserName = "admin";
string Password = "admin1234";
RepositoryClient client = new RepositoryClient("http://localhost:50080/", UserName,
Password);
Project[] projects = client.GetProjects();
foreach(Project p in projects) {
Console.WriteLine(p);
}
```

Ici, un `RepositoryClient` est configuré pour se connecter au répertoire de Management Console sur `http://localhost:50080/`, avec un nom d'utilisateur et un mot de passe. Si le Management Console n'est pas protégé par mot de passe, vous devez saisir nul pour le nom d'utilisateur et le mot de passe.

Une fois le `RepositoryClient` créé, la méthode `GetProjects()` est utilisée pour demander au répertoire une liste de projets. Notez que lors de l'appel de l'une des méthodes `RepositoryClient`, une `RepositoryClientException` est déclenchée si une erreur se produit.

Le `RepositoryClient` dispose des onze méthodes suivantes.

Méthodes de `RepositoryClient` :

Signature de méthode	Description
<code>void DeleteResource(string projectName, string resourceName, boolean silent)</code>	Supprime une ressource d'un projet. L'argument <code>resourceName</code> utilise le chemin complet de la ressource.

Signature de méthode	Description
void DeleteResource(string projectName, string resourceName, boolean silent, AdditionalInfo additionalInfo)	Supprime une ressource d'un projet. L'argument <code>resourceName</code> utilise le chemin complet de la ressource. Cette méthode inclut le paramètre <code>AdditionalInfo</code> pour saisir des commentaires qui s'affichent dans la colonne Message de validation sur la page Ressources de la Management Console.
void DeleteRobot(string projectName, string robotName, boolean silent, AdditionalInfo additionalInfo)	Supprime un robot d'un projet. L'argument <code>robotName</code> utilise le chemin complet du robot.
void DeleteType(string projectName, string typeName, boolean silent, AdditionalInfo additionalInfo)	Supprime un type d'un projet. L'argument <code>typeName</code> utilise le chemin complet du type.
void DeleteSnippet(string projectName, string snippetName, boolean silent, AdditionalInfo additionalInfo)	Supprime un snippet d'un projet. L'argument <code>snippetName</code> utilise le chemin complet du snippet.
void DeployLibrary(string projectName, EmbeddedFileBasedRobotLibrary library, boolean failIfExists, AdditionalInfo additionalInfo)	Déploie une bibliothèque sur le serveur. Les robots, les types et les ressources sont remplacés à moins que <code>failIfExists</code> soit vrai.
void DeployResource(string projectName, string resourceName, byte[] resourceBytes, boolean failIfExists, AdditionalInfo additionalInfo)	Déploie une ressource dans un projet. Si une ressource avec le nom donné existe déjà, elle peut être remplacée en configurant <code>failIfExists</code> sur faux. L'argument <code>resourceName</code> utilise le chemin complet de la ressource.
void DeployRobot(string projectName, string robotName, byte[] robotBytes, boolean failIfExists, AdditionalInfo additionalInfo)	Déploie un robot dans un projet. Si un robot avec le nom donné existe déjà, il peut être remplacé en configurant <code>failIfExists</code> sur faux. L'argument <code>robotName</code> utilise le chemin complet du robot.
void GetRobotsByTag (projectName, tag)	Renvoie une liste de robots avec la balise spécifiée du projet sélectionné.
void DeployType(string projectName, string typeName, byte[] typeBytes, boolean failIfExists, AdditionalInfo additionalInfo)	Déploie un type dans un projet. Si un type avec le nom donné existe déjà, il peut être remplacé en configurant <code>failIfExists</code> sur faux. L'argument <code>typeName</code> utilise le chemin complet de la ressource.
void DeploySnippet(string projectName, string snippetName, byte[] snippetBytes, boolean failIfExists, AdditionalInfo additionalInfo)	Déploie un snippet dans un projet. Si un snippet avec le nom donné existe déjà, il peut être remplacé en configurant <code>failIfExists</code> sur faux. L'argument <code>snippetName</code> utilise le chemin complet du snippet.
void DeployConnector(string projectName, string connectorName, byte[] connectorBytes, boolean failIfExists, AdditionalInfo additionalInfo)	Déploie un connecteur dans un projet. Si un connecteur avec le nom donné existe déjà, il peut être remplacé en configurant <code>failIfExists</code> sur faux.
void DeleteConnector(string projectName, string connectorName, boolean silent, AdditionalInfo additionalInfo)	Supprime un connecteur d'un projet. L'argument <code>connectorName</code> utilise le chemin d'accès complet du connecteur.
Project[] GetProjects()	Renvoie les projets présents dans ce répertoire.

Signature de méthode	Description
Cluster[] GetRoboServerClusters()	Renvoie une liste de clusters et de RoboServers en ligne (valides) enregistrés sur le Management Console exécutant le répertoire.
Cluster[] GetRoboServerClusters (boolean onlineRoboServer)	Renvoie une liste de clusters et de RoboServers enregistrés sur le Management Console. Utilisez l'indicateur <code>onlineRoboServer</code> pour indiquer si la liste des clusters doit inclure seulement les RoboServers qui sont en ligne ou tous les RoboServers.
Cluster AddRoboServer(String clusterName, int portNumber, String host)	Ajoute un nouveau RoboServer à un cluster.
Robot[] GetRobotsInProject (String projectName)	Renvoie les chemins d'accès complets des Robots à Moteur Basique disponibles dans le projet.
RobotSignature GetRobotSignature (String projectName, String robotName)	Renvoie la signature du robot avec le chemin complet du robot, ainsi que les variables d'entrée requises pour exécuter ce robot, et une liste des types qu'il peut renvoyer ou stocker.
RepositoryFolder GetProjectInventory (String projectName)	Renvoie l'arborescence entière des dossiers et fichiers du répertoire.
RepositoryFolder GetFolderInventory (String projectName, String folderPath)	Renvoie les dossiers et fichiers du sous-dossier dans le projet spécifié depuis le répertoire.
RepositoryFolder GetFileInventory (String projectName, String folderPath, String fileName, RepositoryFile.Type fileType)	Récupère le fichier et les fichiers référencés à partir de Management Console. Notez que l'inventaire des fichiers est encapsulé dans <code>RepositoryFolder</code> pour récupérer les références.
Void DeleteFile (RepositoryFile file, boolean silent, AdditionalInfo additionalInfo)	Supprime le fichier spécifié du répertoire.
Date GetCurrentDate()	Renvoie la date et l'heure actuelles du Management Console.
byte[] GetBytes (RepositoryFile file)	Renvoie la taille en octets du fichier spécifié dans le répertoire.
ComputeChecksum (byte[] bytes)	Renvoie la somme de contrôle du fichier spécifié pour vérifier l'intégrité des données.
void UpdateFile (RepositoryFile file, byte[] bytes, AdditionalInfo additionalInfo)	Met à jour le fichier spécifié dans le répertoire avec de nouveaux octets.
void MoveFile (RepositoryFile sourceFile, String destFolderPath)	Déplace le fichier spécifié du répertoire vers un dossier spécifié dans <code>destFolderPath</code> .
void RenameRobot (RepositoryFile robotFile, String newName)	Renomme le fichier robot spécifié.
void DeleteFolder (String projectName, String folderPath, AdditionalInfo additionalInfo)	Supprime le dossier spécifié dans le répertoire.

Signature de méthode	Description
<code>void DeleteRoboServer(String clusterName, RoboServer roboServer)</code>	Supprime un RoboServer.
<code>Map<String, String> getInfo()</code>	<p>Renvoie des informations sur la Management Console et l'API Repository</p> <p>La méthode renvoie un mappage des éléments suivants :</p> <ul style="list-style-type: none"> « application » à la version de Management Console contenant la version principale, la version mineure et la version point pour, par exemple, 11.4.0 « répertoire » à l'ID de la dernière DTD utilisée par l'API Repository, par exemple : //Kapow Technologies//DTD Repository 1.5//EN « rql » à l'ID de la dernière DTD utilisée par l'API Robot Query Language, par exemple : //Kapow Technologies//DTD RoboSuite Robot Query Language 1.13//EN

i Le chemin complet est relatif à votre dossier de projet.

Le paramètre `AdditionalInfo` contient des commentaires qui apparaissent dans la colonne Message de validation de la page Ressources de la Management Console.

Si l'authentification est activée sur le répertoire, la requête peut être refusée si les identifiants fournis ne disposent pas d'un accès suffisant.

Le répertoire est accessible via http. Lors de l'utilisation de la version .NET de l'API Repository, tous les serveurs proxy configurés pour le système seront utilisés par l'API Repository.

Déploiement avec le client répertoire

L'exemple suivant montre comment déployer un robot et un type depuis le système de fichiers local à l'aide de `RepositoryClient`.

Déploiement dans le répertoire

```
string user = "test";
string password = "test1234";
RepositoryClient client = new RepositoryClient("http://localhost:50080", user,
password);

byte[] robotBytes = File.ReadAllBytes("c:\\MyRobots\\Library\\Test.robot");
byte[] typeBytes = File.ReadAllBytes("c:\\MyRobots\\Library\\Test.type");

// we assume that no one has deleted the Default project
client.deployRobot("Default project", "Test.robot", robotBytes, true);
client.deployType("Default project", "Test.type", typeBytes, true);
```

API Repository as REST

Le répertoire est également accessible via des [services RESTful](#).

Affichage des tâches Management Console à l'aide d'une API

Utilisez la vue des tâches sur la Management Console pour mettre en file d'attente les exécutions du robot à l'aide d'une API. Au lieu d'exécuter un robot sur un RoboServer, le robot est placé dans une file d'attente sur la Management Console.

Sachez que certaines fonctionnalités telles que la définition de l'ID d'exécution, la définition des connexions à la base de données, la définition de la durée d'exécution maximale et le fait de forcer un robot à s'arrêter sur les exceptions de l'API, ne peuvent pas être contrôlées à l'aide de la file d'attente de l'API Management Console. Voici les avantages de l'utilisation de la file d'attente de l'API Management Console :

- Les robots seront mis en file d'attente même lorsque des ressources telles que les dispositifs ne sont pas disponibles.
- Les robots sont dirigés vers le bon RoboServer lorsque plusieurs versions de RoboServers sont disponibles.
- Vous n'avez pas besoin de gérer les clusters lors de la création d'une application. Les clusters sont gérés au niveau du projet dans la Management Console.

Pour créer une requête QueuedRequest de l'API Java, procédez comme suit :

- Téléchargez un fichier JSON JAR (org.json.jar), qui est requis pour exécuter le code.
- Compilez le code.
- Ajoutez l'emplacement du fichier .jar à `classpath` lors de l'exécution du code.
- Configurez la requête. Voir les informations détaillées et les exemples qui suivent.

Mettre en file d'attente une exécution de robot

Utilisez la classe `QueuedRequest` pour mettre en file d'attente un robot exécuté sur la Management Console.

 Le code est sensible à la casse.

Constructeurs

Constructeur	Description
<code>QueuedRequest (String robot)</code>	Pour une API .NET, crée une nouvelle requête <code>QueuedRequest</code> avec le robot spécifié. Voir l'exemple après ce tableau.

Constructeur	Description
<code>QueuedRequest(String robotURL, String projectName, RepositoryHttpClientProvider httpClientProvider)</code>	<p>Pour une API Java, crée une nouvelle requête <code>QueuedRequest</code> pour le robot spécifié.</p> <p>Les entrées sont composées de trois paramètres spécifiant l'URL du robot, le nom du projet et le répertoire.</p> <p>Voir l'exemple après ce tableau.</p>

Propriétés

Propriété	Description
<code>getPollingIntervalMillis() / setPollingIntervalMillis(int pollingIntervalMillis)</code>	<p>Met à jour le statut d'exécution du robot. Renvoie les valeurs et les erreurs d'exception de l'API par interrogation. Cette fonction définit l'intervalle en millisecondes entre les interrogations. Des intervalles courts provoquent plus de charge sur la Management Console mais augmentent le taux de mises à jour par demande. La valeur par défaut est 1000 (1 seconde).</p>
<code>getRobotURL() / setRobotURL(String robotName)</code>	<p>Le nom du robot à exécuter, y compris un chemin d'accès si le robot réside dans un dossier.</p> <p>i La partie URL de la propriété n'est pas une URL réelle. Le nom a été choisi pour des raisons de rétrocompatibilité.</p>
<code>getRobotLibrary() / setRobotLibrary(RobotLibrary robotLibrary)</code>	<p>La bibliothèque de robots à utiliser dans la requête. Vous devez définir une bibliothèque de robots avant d'appeler <code>execute()</code>.</p>
<code>getPriority() / setPriority(Priority priority)</code>	<p>La priorité de la requête dans la file d'attente : MINIMUM, BAS, MOYEN, HAUT ou MAXIMUM. La valeur par défaut est MOYEN.</p>
<code>setStopRobotOnApiException(boolean stopOnError)</code>	<p>Lorsque cette propriété est définie, le robot sera arrêté sur le RoboServer après que la première exception de l'API aura été renvoyée au client. La valeur par défaut est <code>true</code>.</p>
<code>getTimeout() / setTimeout(long timeout)</code>	<p>La durée maximale, en secondes, pendant laquelle le robot patiente dans la file d'attente en attendant qu'un RoboServer exécute le robot. La valeur par défaut est 600 (10 minutes).</p>

Méthodes

Méthode	Description
<code>createInputVariable(String name)</code>	Crée un nouvel objet Entrée avec le nom spécifié et renvoie un <code>RQLObjectBuilder</code> qui peut être utilisé pour construire l'objet.

Méthode	Description
createInputVariable(String name, RQLObject rqlObject)	Crée un nouvel objet d'entrée avec le nom spécifié, à partir de <code>rqlObject</code> . Cette objet est utile si les réponses d'un robot sont utilisées comme entrée pour un autre robot.
createOAuthInputVariable(String name, String userName, String applicationName)	Crée une variable d'entrée pour un type d'entrée OAUTH qui recherchera les détails OAuth dans la Management Console, en fonction du nom d'utilisateur et du nom d'application. Assurez-vous que dans la Management Console, l'application et l'utilisateur OAuth sont associés au même projet que le robot dont vous demandez l'exécution.
execute()	Place le robot dans la file d'attente pour exécution, puis attend qu'il se termine.
execute(RobotResponseHandler handler)	Place le robot dans la file d'attente pour exécution et appelle le gestionnaire à différents moments du traitement de la requête.

Exemple d'API .NET

```
QueuedRequest request = new QueuedRequest("myfolder/myrobot.robot");
request.setRobotLibrary(new RepositoryRobotLibrary(
    "http://localhost:50080/", "Default Project",
    60000, "admin", "admin"));
request.setPriority(QueuedRequest.Priority.HIGH);
RQLResult result = request.execute();
```

Exemple d'API JAVA

```
RepositoryHttpClientProvider repository = new RepositoryRobotLibrary("http://
localhost:50080/", "Default project", 60000, "admin", "admin");
QueuedRequest request = new QueuedRequest("myfolder/myrobot.robot", "Default project",
repository);
request.setPriority(QueuedRequest.Priority.HIGH);
RQLResult result = request.execute();
```

Chapitre 3

API Management Console REST

Ce chapitre fournit des informations sur les services Management Console REST fournis avec le produit. Les services REST sont accessibles à partir de l'interface utilisateur Swagger à l'aide de l'exemple d'URL suivant :

<http://localhost:8080/ManagementConsole/api/swagger-ui.html>

Les services REST suivants sont disponibles dans Kofax RPA 11.4.0.

Service REST	Objectif
tâches	Mise en file d'attente des tâches du robot. Avec ce service, vous pouvez obtenir un exemple de structure d'entrée du robot nécessaire pour exécuter le robot, mettre en file d'attente les tâches du robot et récupérer le résultat de l'exécution du robot.

Tâches

Il s'agit du service REST pour la mise en file d'attente des tâches du robot.

Méthodes

POST robotInputExample

Utilisé pour obtenir un exemple de structure des valeurs d'entrée du robot nécessaire pour faire fonctionner le robot. Ce sont les valeurs que vous configurez à l'étape « Configurer une entrée » lors de la création de la planification.

Dans la section **Paramètres**, modifiez le corps de la requête pour spécifier les propriétés `projectName` et `robotName`, puis cliquez sur **Exécuter**. La réponse contiendra l'exemple de structure de l'entrée de votre robot que vous pouvez utiliser pour créer une requête de mise en file d'attente des tâches du robot.

POST queueRobot

Utilisé pour la mise en file d'attente des tâches du robot.

Dans la section **Paramètres**, modifiez le corps de la requête comme indiqué ici :

1. Dans la propriété `priority`, spécifiez le niveau de priorité le plus approprié : `MINIMUM`, `BAS`, `MOYEN`, `HAUT` ou `MAXIMUM`. Les tâches qui ont une priorité plus élevée ont accès aux ressources requises et sont exécutées plus tôt que celles qui ont une priorité inférieure. Voir « Mise en file d'attente des travaux de planification » dans l'*Aide de Kofax RPA* pour plus d'informations.
2. Configurez les propriétés `projectName` et `robotName`.

3. Dans la propriété `robotInputConfig`, collez l'exemple de structure d'entrée que vous avez obtenu avec la méthode `robotInputExample` et modifiez les valeurs d'entrée comme il convient.
4. Dans la propriété `timeout`, spécifiez le délai d'attente après lequel les tâches doivent arrêter la mise en file d'attente.
5. Cliquez sur **Exécuter**.

GET getRobotOutput/{ticket}

Utilisé pour obtenir le résultat de l'exécution du robot, comme les données de sortie du robot, l'état de la file d'attente et les informations d'erreur.

Lorsque la requête POST `queueRobot` est exécutée, un ticket d'exécution unique est généré pour cette requête. Copiez le ticket de la réponse `queueRobot` et collez-le dans la section **Paramètres** de la requête `getRobotOutput`. Cliquez sur **Exécuter**.

La réponse contiendra l'état et le résultat de l'exécution du robot. Si le robot contient des données de sortie, elles sont écrites dans la propriété `values`.