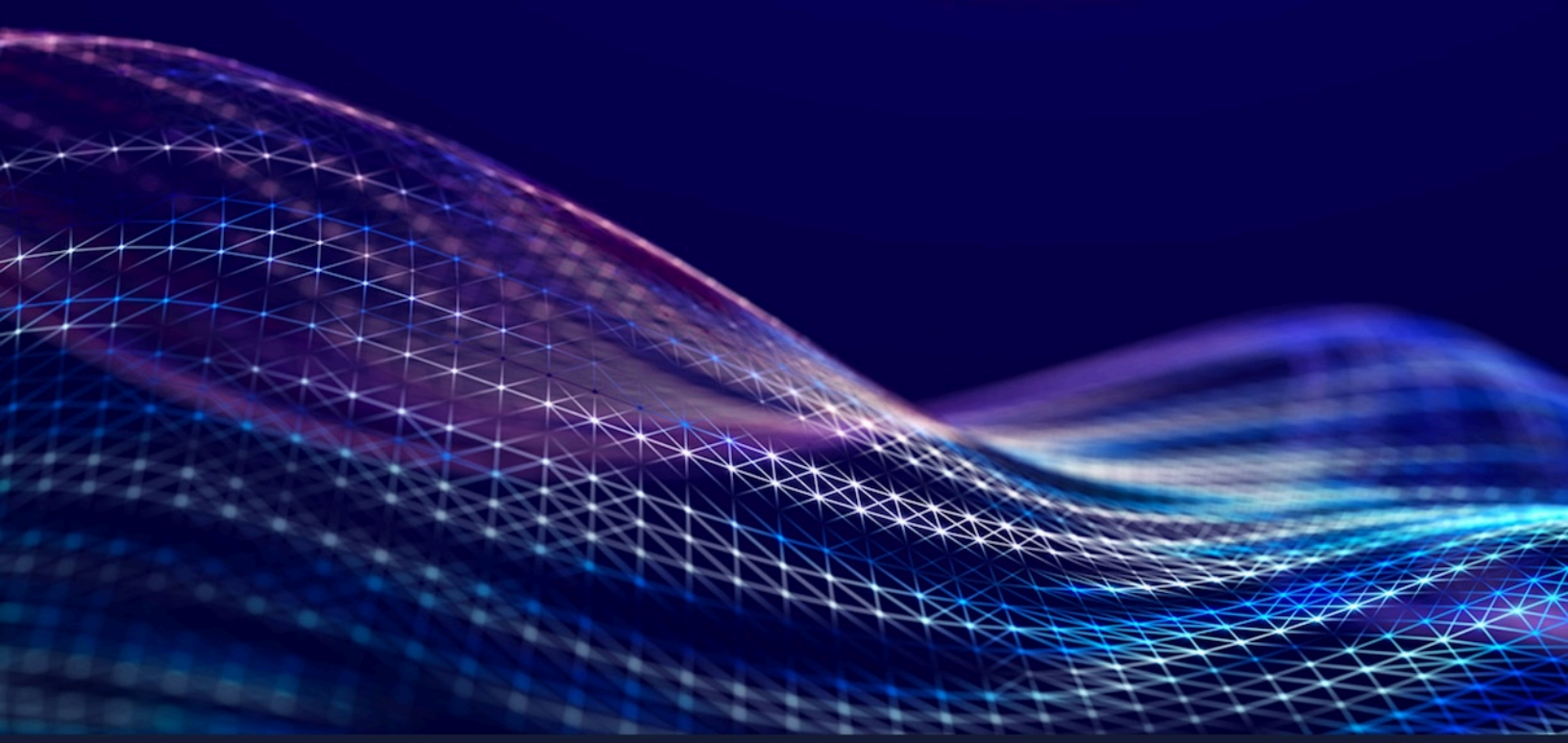

Tungsten RPA
開発者ガイド
2026.1

TUNGSTEN AUTOMATION

目次

はじめに

API を使用して、Management Console で次のようなタスクを実行します。

- ユーザーとグループの作成と管理
- ロボットの管理と実行
- ユーザー API キーの作成と管理
- ライセンスと認証トークンの管理
- バックアップの作成と管理

このガイドは次のパートで構成されています:

- 「Java」では、Java プログラムで使用できる API について説明します。
- 「.NET」では、C# プログラムを含む .NET アプリケーションで使用する API について説明します。
- 「REST サービス」では、Management Console で提供される REST API サービスについて説明します。

RPA 用のクライアント アプリケーションを開発するには、JSON を用いた REST の使用が推奨されます。クライアント アプリケーションを開発する際は、多くの場合、Management Console REST API を使用する方が便利かつ簡単です。

Java および .NET API リファレンス ドキュメントは、オフライン ドキュメント フォルダにあります。詳細については、『Tungsten RPA インストール ガイド』を参照してください。

製品ドキュメント

Tungsten RPA のドキュメント セットは、次の場所から入手できます。

https://docshield.tungstenautomation.com/Portal/Products/RPA/2026.1-cqvh2o7vk9/RPA.htm

完全なドキュメント セットにオンラインでアクセスするには、インターネットに接続する必要があります。

インターネットに接続せずにアクセスする方法については、オフライン ドキュメントを参照してください。

ドキュメント セットには、次のようなリソースがアルファベット順で含まれています。

***Tungsten RPA* 管理者ガイド**
Tungsten RPA での管理タスクについて説明します。

***Tungsten RPA* のベストプラクティス ガイド**

Tungsten RPA 環境でロボット ライフサイクル マネジメントを使用しながらパフォーマンスを最適化し、成功を確実にするために推奨される方法とテクニックを提供します。

***Tungsten RPA Desktop Automation* サービス ガイド**

リモート コンピューターで Desktop Automation を使用するために必要な Desktop Automation サービスを設定および管理する方法について説明します。

***Tungsten RPA* 開発者ガイド**

Java、.NET、および REST API を使用して Management Console 上でロボットを実行するための情報と手順が記載されています。

***Tungsten RPA* ロボット構築スタート ガイド**

Tungsten RPA を使用したロボット構築処理を示すチュートリアルが記載されています。

***Tungsten RPA* のヘルプ**

Tungsten RPA の使用方法について説明しています。ヘルプは Tungsten RPA 製品内から利用できます。

***Tungsten RPA* インストール ガイド**

Tungsten RPA およびそのコンポーネントを開発環境にインストールする方法について説明します。

***Tungsten RPA Java API documentation (Tungsten RPA Java API* ドキュメント)**

開発者が Tungsten RPA で使用できる Tungsten RPA Java API パッケージおよびクラスへのアクセスを提供します。

Tungsten RPA API は、元の製品名である「RoboSuite」に対する詳細な参照を含んでいます。RoboSuite の名前は下位互換性を確保するために残されています。API ドキュメントの中では、RoboSuite という用語は Tungsten RPA と同じ意味で使われています。

***Tungsten RPA* リリース ノート**

他の Tungsten RPA ドキュメントからは入手できない最新の詳細やその他の情報が含まれています。

***Tungsten RPA* 技術仕様**

サポートされるオペレーティング システムおよびその他のシステム要件に関する情報が含まれています。

***Tungsten RPA* アップグレード ガイド**

Tungsten RPA やそのコンポーネントを新しいバージョンにアップグレードする手順が含まれています。

***Tungsten RPA* ユーザー ガイド**

Tungsten RPA とそのコンポーネントの使用手順が記載されています。Tungsten RPA のヘルプ のトピックとともに、ヘルプには含まれていない詳細な内容が記載されています。

トレーニング

Tungsten Automation は、製品を最大限に活用できるように、オンデマンド トレーニングおよびインストラクター主導のトレーニングを提供しています。トレーニング コースとスケジュールの詳細については、Tungsten Automation Learning Cloud を参照してください。

Tungsten RPA ドキュメント サイトには、コンポーネントの理解および RPA でのロボット作成に関する基礎の確認のためのチュートリアルが用意されています。https://docshield.tungstenautomation.com/Portal/Products/RPA/2026.1-cqvh2o7vk9/RPA.htm を参照してください。

Tungsten Automation 製品のヘルプの入手

[Tungsten Automation Knowledge Portal (Tungsten Automation ナレッジ ポータル)] リポジトリにある記事の内容は定期的に更新され、Tungsten Automation 製品の最新情報について参照することができます。製品に関してご不明の点がある場合は、Knowledge Portal (ナレッジ ポータル) で情報を検索することをお勧めします。

[Tungsten Automation Knowledge Portal] にアクセスするには次のリンクを使用してください。

https://knowledge.tungstenautomation.com/

Knowledge Portal は Google Chrome、Mozilla Firefox、または Microsoft Edge 向けに最適化されています。

Knowledge Portal では次のような機能を利用できます。

- 強力な検索機能で必要な情報をすぐに見つけることができます。
  [Search (検索)] ボックスに目的の語句を入力し、検索アイコンを選択してください。
- 製品情報、設定の詳細、リリース情報などのドキュメント。
  記事を見つけるには、Knowledge Portal のホームページにアクセスし、製品に該当するソリューション ファミリーを選択するか、[View All Products (すべての製品を表示)] ボタンを選択します。

Knowledge Portal のホームページからは、次の操作を実行できます。

- Tungsten Automation Community (Tungsten Automation コミュニティ) へのアクセス (全カスタマー)。
  [Tungsten Automation Resources (Tungsten Automation リソース)] メニューで、**[Community (コミュニティ)]** リンクを選択します。
- Tungsten Automation Customer Portal (Tungsten Automation カスタマー ポータル) へのアクセス (一部のカスタマーのみ)。

[Support Portal Information (サポート ポータルの情報)] ページに移動し、**[Log in to the Tungsten Automation Customer Portal (Tungsten Automation カスタマー ポータルにログイン)]** を選択します。

- Tungsten Automation Partner Portal (Tungsten Automation パートナー ポータル) へのアクセス (一部のパートナーのみ)。

  [Support Portal Information] ページに移動し、**[Log in to the Tungsten Automation Partner Portal (Tungsten Automation パートナー ポータルにログイン)]** を選択します。

- サポート コミットメント、ライフサイクル ポリシー、電子フルフィルメントの詳細、およびセルフサービス ツールへのアクセス。

  [Support Details (サポートの詳細)] ページに移動し、適切な記事を選択します。

第 1 章

Java

この章では、Tungsten RPA Java API を使用してロボットを実行する方法について説明します。この情報は、簡単なロボットの記述方法を把握し、Java プログラミング言語に精通している開発者を対象としています。

この章では、コア クラスとそれらのクラスを使用してロボットを実行する方法について説明します。また、ロボットに入力を提供する方法についても説明します。

Java API は、Tungsten RPA インストール フォルダ内の `\API\rpa-api-java\lib\` にある `rpa-api-java.jar` ファイルです。『Tungsten RPA インストール ガイド』の「重要なファイルとフォルダ」を参照してください。

Java API ファイルと一緒に、API が依存している他の JAR ファイルも配置されます。

Java クラスに関する情報は、Tungsten RPA 製品ドキュメント サイトの「API と開発者ドキュメント」リンクで参照することができます。https://docshield.tungstenautomation.com/Portal/Products/RPA/2026.1-cqvh2o7vk9/RPA.htm。

Java クラスに関する情報は、オフライン ドキュメント フォルダにある `\API\rpa-api-java` ヘルプ フォルダを参照してください。

Java の基本

Management Console で実行されるロボットは、Java API を使用してキューに格納されます。これにより、ユーザーは特定のロボットを実行するように指示するリクエストを送信できるようになります。

API を使用することで、Java ベースのアプリケーションはロボットを Management Console のキューに格納することができます。データベースにデータを保存するロボットを実行するだけではなく、ロボットがクライアント アプリケーションに直接データを返すようにすることもできます。ここではいくつかの例を示します:

- 複数のロボットを使用して、複数のソースからの結果をリアルタイムで集約する統合検索を実行します。
- アプリケーションのイベントに応じてロボットを実行します。たとえば、新しいユーザーがサインアップしたときにロボットを実行して、バックエンドに直接統合されていない Web ベースのシステムにアカウントを作成します。

Java チュートリアルの例

以下のコードは、デフォルトのプロジェクトの Tutorials フォルダの最初の例である `NewsMagazine.robot` という名前のベーシック エンジン ロボットを実行するために必要なコードです。ロボットは、値返却アクション ステップを使用して結果を書き込みます。これにより、API を使用してプログラムで出力を簡単に処理することができます。他のロボット (通常は Management Console のスケジュールで実行) は、データベース データ登録アクション ステップを使用してデータを直接データベースに保存します。この場合、ロボットによって収集されたデータが API クライアントに返されることはありません。

次の例では、サンプル チュートリアル `NewsMagazine.robot` が実行され、出力がプログラムで処理されます。`"{ api key }"` という値は、適切なユーザー API キーに置き換える必要があります。

入力なしでロボットを実行します

```
import com.kapowtech.robosuite.api.java.rest.*;

import java.net.*;

 /**
 * Example that shows you how to execute NewsMagazine.robot from tutorial 1
 */
 public class Tutorial1 {

      public static void main(String[] args) {
        String apiKey = "{ api key }"; // To be filled out
        try {
            QueuedRequest request = new QueuedRequest(
                    new URL("http://localhost:8080/ManagementConsole"),
                    "Tutorials/NewsMagazine.robot",
                    "Default project",
                    apiKey);
            ExecutionResult result = request.execute();
            for (RpaObject o : result.getOutputObjectsByName("Post")) {
                String title = (String) o.get("title");
                String preview = (String) o.get("preview");
                System.out.println(title + ": " + preview );
            }
        }
        catch (ApiException | MalformedURLException e) {
            e.printStackTrace();
         }
    }
 }
```

次の表に、関係するクラスとその責任を示します。

クラス	説明
`QueuedRequest`	このクラスを使用してロボット リクエストを構築します。
`ExecutionResult`	ロボットの実行結果を含めるには、このクラスを使用します。結果には、値の応答とエラー メッセージが含まれます。
`RpaObject`	戻り値アクションを使用してロボットから返される各値には、`RpaObject` としてアクセスできます。

次の行により、API に対して Management Console が `localhost` のポート `8080` で実行されていることが通知されます。

```
String mcUrl = "http://localhost:8080/ManagementConsole";
```

次のコードにより、`Tutorials` にある `NewsMagazine.robot` という名前のロボットを実行するリクエストを作成します。

```
QueuedRequest request = new QueueRequest (new URL(mcUrl, robot, project, apiKey);
```

次の行により、ロボットを実行し、ロボットが完了した後に結果が返されるようにします。デフォルトでは、ロボットが API 例外を生成すると、`execute` が例外をスローします。

```
ExecutionResult result = request.execute()
```

コマンド `request.execute()` はブロックしており、ロボットが実行されるか例外がスローされるまで戻りません。または、データ ストリーミング で説明されているデータ ストリーミング方式を使用します。

次に、抽出した値を処理します。まず、Post という名前のタイプのすべての抽出値を取得し、それらを反復処理します。各 `RpaObject` について、Post タイプの属性にアクセスし、結果を出力します(属性とマッピングについては、後のセクションで説明します)。

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

Java ロボット入力

Java API を介して実行されるほとんどのロボットについては、検索キーワードやログイン資格情報などの入力パラメータを通じて設定を行います。ロボットへの入力は、Management Console のリクエストの一部であり、リクエストの `createInputVariable` メソッドを使用して提供します。

暗黙的な `RpaObjectBuilder` を使用した入力

```
QueuedRequest request = new QueuedRequest(mcUrl, robot, project, apiKey);
 request.createInputVariable("userLogin").setAttribute("username", "scott")
 .setAttribute("password", "tiger");
```

この例では、`QueuedRequest` を作成し、`createInputVariable` によって `userLogin` という名前の入力変数を作成します。次に、`setAttribute` によって、入力変数のユーザー名とパスワードの属性を設定します。

上記の例は一般的な略記法ですが、`RpaObjectBuilder` オブジェクトを使用してより詳細に表現することもできます

明示的な `RQLObjectBuilder` を使用した入力

```
QueuedRequest request = new QueuedRequest(mcUrl, robot, project, apiKey);
 RpaObjectBuilder userLogin = request.createInputVariable("userLogin");
 userLogin.setAttribute("username", "scott");
 userLogin.setAttribute("password", "tiger");
```

2 つの例は同じです。1 つ目は、匿名の `RpaObjectBuilder` でカスケード メソッド呼び出しを使用するため、より短くなります。

Management Console がこのリクエストを受信すると、次のような処理が発生します。

- Management Console はリクエストをキューに格納して、可能な場合は利用可能な RoboServer に送信します。
- RoboServer は `Input.robot` を読み込みます。
- RoboServer はロボットに `userLogin` という名前の変数があり、この変数が入力としてマークされていることを確認します。
- RoboServer は、 `setAttribute` を使用して設定された属性が変数の `userLogin` のタイプと互換性があるかを検証します。その結果、このタイプには `username` および `password` という名前の属性 (どちらもテキスト ベースの属性である必要があります) が必要となります(Java API と Design Studio 属性間のマッピングについては、このガイドの次のセクションで説明します)。
- すべての入力変数に互換性がある場合、RoboServer はロボットの実行を開始します。

ロボットが複数の入力変数を必要とする場合、それらをすべて作成してロボットを実行する必要があります。必要な属性のみを設定する必要があります。Java API を通じて設定しないオプションの属性には、デフォルトで null 値が割り当てられます。

次の例では、Facebook および Twitter にログインする必要があるロボットがある場合に、入力を次のように定義します。

```
QueuedRequest request = new QueuedRequest(mcUrl, robot, project, apiKey);
 request.createInputVariable("facebook").setAttribute("username", "scott")
 .setAttribute("password", "facebook123");
 request.createInputVariable("twitter").setAttribute("username", "scott")
 .setAttribute("password", "twitter123");
```

Java 属性タイプのマッピング

このセクションでは、Java API と Design Studio 間のマッピングについて説明します。Design Studio で新しいタイプを定義するとき、各属性のタイプを選択します。

一部の属性には、ショート テキスト、ロング テキスト、パスワード、HTML、XML などのテキストが含まれています。ロボット内で使用する場合は、これらの属性にテキストを保存することが必要になる場合があります。

XML 属性にテキストを保存する場合、テキストは有効な XML ドキュメントである必要があります。この検証は、ロボット内でタイプが使用されている場合に発生しますが、Java API ではタイプについて何も検出されないため、同じ方法による属性値の検証は行われません。そのため、Java API には属性のタイプが 7 個あり、Design Studio には 20 個のタイプがあります。

次の表に、API 属性タイプと Design Studio 属性タイプのマッピングを示します。

Java API から Design Studio へのマッピング

Java クラス	API 属性タイプ	Design Studio 属性タイプ
`java.lang.String`	テキスト	ショート テキスト、ロング テキスト、パスワード、HTML、XML、プロパティ、言語、国、通貨、JSON
`java.lang.Long`	整数	整数
`java.lang.Boolean`	ブール値	ブール値
`java.lang.Double`	数	数
`java.lang.Character`	文字	文字
`java.util.Date`	日付	日付
`com.kapowtech.robosuite.api.java.rest.Binary`	バイナリ	バイナリ、画像、PDF、Excel、セッション

`RpaObjectBuilder` の `setAttribute` メソッドはオーバーロードされたメソッドであるため、適切な Java class が引数として使用されている場合、API を介して属性を設定する際に属性タイプを明示的に指定する必要はありません。

オブジェクトの属性に可能な Design Studio 属性タイプをすべて設定する方法を次の例に示します。

`setAttribute` の推奨される使用方法

```
QueuedRequest request = new QueuedRequest(mcUrl, robot, project, apiKey);
 RpaObjectBuilder 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 text");
 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", (Binary) null);
 inputBuilder.setAttribute("aCurrency", "USD");
 inputBuilder.setAttribute("aCountry", "US");
 inputBuilder.setAttribute("aLanguage", "en");
 inputBuilder.setAttribute("aJSON", "json text");
 inputBuilder.setAttribute("anExcel", (Binary) null);
```

上記の例では、新しい `Long(42L)` と新しい `Double(12.34)` を明示的に使用していますが、自動ボクシングのためには 42L と 12.34 で十分です。また、Java コンパイラーはオーバーロードされた `setAttribute` メソッドを呼び出す判断ができないため、null 値をキャストする必要があることに注意してください。ただし、未設定の属性は自動的に null になるため、null を明示的に設定する必要はありません。

お勧めはしませんが、API を使用して入力を作成するときに、`setAttribute` と `AttributeType` を明示的に指定できます。この方法はまれに必要になる場合があり、次の例のようになります。

`setAttribute` の誤った使用方法

```
QueuedRequest request = new QueuedRequest(mcUrl, robot, project, apiKey);
 RpaObjectBuilder 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 text",
 AttributeType.TEXT));
 inputBuilder.setAttribute(new Attribute("aPassword", "secret", AttributeType.TEXT));
 inputBuilder.setAttribute(new Attribute("aHTML", "<html>text</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.BINARY));
 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("aJSON", "json text", AttributeType.TEXT));
 inputBuilder.setAttribute(new Attribute("anExcel", null, AttributeType.BINARY));
```

すべての属性値は、文字列の形式で提供する必要があります。文字列値は、属性タイプに基づいて適切な Java オブジェクトに変換されます。この値は、Tungsten RPA Java API 上に他の汎用 API を構築する場合に役立ちます。

Java 実行パラメータ

`createInputVariable` メソッドに加えて、`QueuedRequest` にはロボットをキューに格納する方法を制御するメソッドが含まれています。

リクエストに応じた実行制御メソッド

パラメータ	説明
`setTimeout(long timeout)`	ロボットの実行最大キュー時間を秒数で設定します。この時間が経過してもロボットが起動しない場合はタイムアウト エラーが発生します。 デフォルトでは 600 秒、つまり 10 分です。

パラメータ	説明
`setStopRobotOnApiException(boolean)`	true (デフォルト) の場合、最初の API 例外が発生した後にロボットは停止します。 デフォルトでは、ロボットのほとんどのステップは、ステップの実行に失敗すると API 例外を発生させます。ステップの [エラー処理] タブでこの値を設定します。 false の場合、API 例外は破棄され、ロボットは実行を続行します。
`setPollingIntervalMillis(int milliSeconds)`	キューに格納されたロボットからの更新をリクエストする間隔を設定します。値が小さいほど更新は頻繁になりますが、Management Console へのリクエストの数が増えます。 デフォルトでは 1000 ミリ秒、つまり 1 秒です。
`setPriority(QueuedRequest.Priority priority)`	キュー内のリクエストの優先度を設定します。 優先度は、MINIMUM、LOW、MEDIUM、HIGH、および MAXIMUM です。 優先度が大きいほど、RoboServer のスロットが使用可能になったときのロボット実行の優先度が高くなります。 デフォルトの優先度は MEDIUM です。

Java SSL 証明書

デフォルトでは、API コールではサーバーの自己署名証明書 (SSL) を検証するときにシステム Java ストアが利用されます。証明書を検証できない場合は、PKIX エラーが発生します。Management Console に SSL などの信頼されていない証明書を使用した場合は、API コールを実行する前に、信頼された `keystore` を指定する必要がある場合があります。

信頼された `keystore` を指定するには、次の手順を実行します。

1. システム プロパティで、信頼された `keystore` を設定します。

```
System.setProperty("javax.net.ssl.trustStore", "<keystore file>");

System.setProperty("javax.net.ssl.trustStorePassword", "<keystore password>");
```

2. *`<keystore file>`* を `keystore` ファイルへのパスに置き換えます。このファイルは、通常 `.jks` ファイルです。
3. `keystore` ファイルがロックされている場合は、password プロパティを使用し、*`<keystore password>`* をパスワードに置き換えます。
   次の詳細な例を参照してください。

証明書の検証をオフ (無効) にするには、次の手順を実行します。

1. システム プロパティで API の検証を無効にします。
   `System.setProperty("kapow.apiIgnoreCertIssues", "true");`
   サーバー証明書の検証は実行されなくなりますが、暗号化が使用されます。

2. (推奨) 検証の代わりに `keystore` をロードします。

例: SSL 証明書を検証

```
import com.kapowtech.robosuite.api.java.repository.engine.RepositoryClientFactory;
import com.kapowtech.robosuite.api.java.rest.QueuedRequest;

public class Example {

  public static void main(String[] args) {

    try {

      System.setProperty("javax.net.ssl.trustStore", "C:\.ssl\keystore.jks");

      System.setProperty("javax.net.ssl.trustStorePassword", "password");


      URL mcUrl = new URL("https://localhost:8443/ManagementConsole");

      String project = "Default project";

      String robot = "Unnamed";

      String apiKey = "{api key}";


      QueuedRequest request = new QueuedRequest(mcUrl, robot, project, apiKey);

      request.execute();


      var repoClient = RepositoryClientFactory.createRepositoryClient(mcUrl, apiKey);

      repoClient.getProjects();

    } catch (Exception e) {

      System.out.println(e.getMessage());

    }

  }

}
```

Java の高度な機能

このセクションでは、出力データのストリーミングを含む高度な API 機能について説明します。

データ ストリーミング

ロボットの実行結果をリアルタイムで表示する必要がある場合は、ロボットが実行を終了して `ExecutionResult` にアクセスするまで待機する代わりに、Java API を使用して、抽出した値をロボットがすぐに返すようにすることができます。

Java API は、ロボットから返された値を API が受け取るたびにコールバックを受信するようにすることができます。これは、`RobotResponseHandler` インターフェイスを通じて行います。

次のレスポンス ハンドラは API の一部ではありませんが、サンプル ライブラリに含まれています。

`AbstractFailFastRobotResponseHandler` を使用した応答ストリーミング

```
import com.kapowtech.robosuite.api.java.rest.*;

/** * An abstract RobotResponseHandler that implements a FailFast behavior, so you only
 need to implement
* response handling.
*
*
/
public abstract class AbstractFailFastRobotResponseHandler implements
 RobotResponseHandler {

protected AbstractFailFastRobotResponseHandler() {    }

   public void requestAccepted(String executionId) {}

   public void robotStarted(Stoppable stoppable) {}

   public void robotCompleted() {}

   public void robotCanceled() {}

   public void robotFailed() {}

   public void queueTimedOut() {}

   /**
    * Throws the exception.
    *
    * @param e         the exception.
    * @param stoppable an object that allows you to stop the robot.
    * @throws ApiException always throws the exception given as argument.
    */
   public void handleError(ApiException e, Stoppable stoppable) throws ApiException {
       stoppable.stop();
       throw e;
   }


   /**
    * Throws RobotErrorResponseException in response to an error (stops the robot).
    *
    * @param response  the robot error response.
    * @param stoppable an object that allows you to stop the robot.
    * @throws ApiException is always thrown.
    */
```

```
    public void handleExecutionError(RobotErrorResponse response, Stoppable stoppable)
 throws ApiException {
        stoppable.stop();
        throw new ApiException(response.toString());
    }

   /**
    * Throws a ServerErrorResponseException.
    *
    * @param response  the server error response.
    * @param stoppable an object that allows you to stop the robot.
    * @throws ApiException if server has error.
    */
    public void handleServerError(ServerErrorResponse response, Stoppable stoppable)
 throws ApiException {
        stoppable.stop();
        throw new ApiException(response.toString());
    }
}
```

この例では、別のリクエスト実行メソッドを使用しています。このメソッドでは、`RobotResponseHandler` が必要です。

次の例では、デフォルトのエラー処理を提供する `AbstractFailFastRobotResponseHandler` を拡張して `RobotResponseHandler` を作成し、ロボットが返す値のみを処理します。`AbstractFailFastRobotResponseHandler` は、Java API の例の一部です。

`RobotResponseHandler` を使用して `AbstractFailFastRobotResponseHandler` を拡張するレスポンス ストリーミング

```
import com.kapowtech.robosuite.api.java.rest.*;

import java.net.*;

/**
 * This example shows how to stream data through the RobotResponseHandler
 * in real-time. Extends the AbstractFailFastRobotResponseHandler in
 * order to process returned values, if more control is desired, you can
 * implement RobotResponseHandler to access the various error handling methods.
 */
public class DataStreaming {

    public static void main(String[] args) {
        String apiKey = "{ api key }"; // To be filled out
        try {
            QueuedRequest request = new QueuedRequest(
                    new URL("http://localhost:8080/ManagementConsole"),
                    "Tutorials/NewsMagazine.robot",
                    "Default project",
                    apiKey);
            RobotResponseHandler handler = new AbstractFailFastRobotResponseHandler() {
                public void handleReturnedValue(RobotOutputObjectResponse response,
 Stoppable stoppable) {
                    RpaObject value = response.getOutputObject();
                    String title = (String) value.get("title");
                    String preview = (String) value.get("preview");
                    System.out.println(title + ": " + preview );
                }
            };
            request.execute(handler);
        }
```

```
        catch (ApiException | MalformedURLException e) {
            e.printStackTrace();
        }
    }
}
```

`handleReturnedValue` メソッドは、RoboServer から Java API が戻り値を受け取るたびに呼び出されます。この例で使用されている `AbstractFailFastRobotResponseHandler` は、非ストリーミング実行メソッドと同じ方法で例外をスローします。これは、ロボットによって生成された API 例外に応じて例外がスローされることを意味します。

`RobotResponseHandler` メソッドは 3 つのカテゴリに分類することができます。

ロボットのライフ サイクル イベント
実行の開始時と終了時など、ロボットの実行状態が変化した場合に呼び出されるメソッド。

ロボット データ イベント
ロボットが Java API にデータまたはエラーを返した場合に呼び出されるメソッド。

追加のエラー処理
実行エラーまたは Java API のエラーによって呼び出されたメソッド。

`RobotResponseHandler` - ロボットのライフ サイクル イベント

メソッド名	説明
`void requestAccepted(String executionId)`	Management Console がリクエストを受け入れて、キューに格納した場合に呼び出されます。
`void robotStarted(Stoppable stoppable)`	ロボットが実行を開始した場合に呼び出されます。これは通常、Management Console に使用可能な RoboServer がない場合でない限り、ロボットがキューに格納された直後に発生します。
`void robotCompleted()`	ロボットが正常に完了した場合に呼び出されます。
`void robotCanceled()`	ロボットがキャンセルされた場合に呼び出されます。
`void robotFailed()`	ロボットが失敗した場合に呼び出されます。エラー イベントが先行します。
`void queueTimedOut()`	ロボットがキューで待機中にタイムアウトになった場合に呼び出されます。

`RobotResponseHandler` - ロボット データ イベント

メソッド名	説明
`void handleReturnedValue(RobotOutputObjectResponse response, Stoppable stoppable)`	ロボットが値返却アクション ステップを実行し、値が Java API に返された場合に呼び出されます。

メソッド名	説明
`void handleExecutionError(RobotErrorResponse response, Stoppable stoppable)`	ロボットが API 例外を発生させたときに呼び出されます。 通常の状況では、ロボットは最初のAPI例外の後に実行を停止します。 この動作は `QueuedRequest.setStopRobotOnApiException(false)` を使用してオーバーライドすることができます。この場合、このメソッドは複数回呼び出されます。このアプローチは生成されたエラーに関係なく、データ ストリーミング ロボットの実行を継続する場合に役立ちます。

`RobotResponseHandler` - 追加のエラー処理

メソッド名	説明
`void handleServerError(ServerErrorResponse response, Stoppable stoppable)`	サーバーでエラーが生成された場合に呼び出されます。 たとえば、サーバーがビジー状態でリクエストを処理できない場合、またはサーバー内でエラーが発生してロボットを起動できない場合などが挙げられます。
`handleError(ApiException e, Stoppable stoppable)`	API 内でエラーが発生した場合、最も一般的には、クライアントが接続を失った場合に呼び出されます。

メソッドの多くには `Stoppable` オブジェクトが含まれており、特定のエラーまたは返された値に応じて停止するために使用できます。

一部のメソッドでは `ApiException`をスローでき、帰結が得られる場合があります。ハンドラーを呼び出すスレッドは、`QueuedRequest.execute()` を呼び出すスレッドです。スローされた例外は、呼び出しスタックをオーバーロードする可能性があります。

`handleReturnedValue` または `handleExecutionError` の応答として例外をスローした場合は、`Stoppable.stop()` をユーザーが呼び出しを行います。ユーザーが呼び出しを行わない場合、ロボットは `QueuedRequest.execute()` への呼び出しが完了しても実行し続ける可能性があります。

データ ストリーミングは、次の使用例のいずれかで最もよく使用されます。

- 結果がリアルタイムでユーザーに表示される Ajax ベースの Web アプリケーション。データがストリーミングされない場合、ロボットの実行が完了するまで結果は表示されません。
- クライアントがロボットの実行中にすべてをメモリに保持することができない程大量のデータを返すロボット。
- 抽出された値がロボットの実行と並行して処理されるように最適化する必要があるプロセス。
- カスタム形式でデータベースにデータを保存するプロセス。
- API 例外のカスタム処理を無視または必要とするロボット（次の例を参照）。

`DataStreamingCollectErrorsAndValues` を使用したレスポンスとエラーの収集

```
import com.kapowtech.robosuite.api.java.rest.*;
```

```
import java.net.*;
import java.util.*;

/**
 * This example shows how to stream responses to the client using a
 RobotResponseHandler,
 * in this example the robot will continue to execute even if the Robot generates API
 * exceptions. The handler stores all output and error and these can be inspected
 after
 * the robot is done executing.
 * <b>Note</b> that <code>setStopRobotOnApiException(false)</code>, if this is not
 done
 * the robot will automatically stop after the first exception has been returned.
 *
 */
public class DataStreamingCollectErrorsAndValues {
    public static void main(String[] args) throws MalformedURLException {
        String apiKey = "{ api key }"; // To be filled out

        try {
            QueuedRequest request = new QueuedRequest(
                    new URL("http://localhost:8080/ManagementConsole"),
                    "Tutorials/NewsMagazine.robot",
                    "Default project",
                    apiKey);

            request.setStopRobotOnApiException(false);  // IMPORTANT!!
            ErrorCollectingRobotResponseHandler handler = new
 ErrorCollectingRobotResponseHandler();
            request.execute(handler);

            System.out.println("Extracted values:");
            for (RobotOutputObjectResponse response : handler.getOutput()) {
                RpaObject value = response.getOutputObject();
                String title = (String) value.get("title");
                String preview = (String) value.get("preview");
                System.out.println(title + ": " + preview );
            }

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

    }
    private static class ErrorCollectingRobotResponseHandler extends
 AbstractFailFastRobotResponseHandler {
        private final List<RobotErrorResponse> _errors = new LinkedList<>();
        private final List<RobotOutputObjectResponse> _output = new LinkedList<>();
        public void handleReturnedValue(RobotOutputObjectResponse response, Stoppable
 stoppable) {
            _output.add(response);
        }

        @Override
        public void handleExecutionError(RobotErrorResponse response, Stoppable
 stoppable) {
            // 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;
        }
    }
}
```

上記の例は、返された値とエラーを収集する `RobotResponseHandler` の使い方を示します。このタイプのハンドラは、エラーが発生した場合でもロボットが実行を継続するようにする必要がある場合に役立ちます。これは、Web サイトが不安定でタイムアウトするような場合に役立ちます。

ロボット エラー（API 例外）のみがハンドラーによって収集されることに注意してください。接続が失われた場合、`QueuedRequest.execute()` は引き続き `ApiException` をスローし、ロボットは停止します。

詳細については、`RobotResponseHandler` の JavaDoc を参照してください。ドキュメントについては、「Java」および「製品ドキュメント」を参照してください。

並列実行

標準の `QueuedRequest.execute()` メソッドのみが同期的な実行を行います。(コード例については、データ ストリーミング を参照してください。)

並列で実行する場合は、デフォルトの `execute()` メソッドに別のスレッドを使用するか、独自に完了を通知するカスタム ハンドラを作成します。

リクエストは変更が可能であり、条件の発生を避けるため、リクエストは実行メソッド内で複製されます。リクエストは変更が可能ですが、異なるスレッドで同じリクエストを変更しないようにしてください。

非同期バリアント `QueuedRequest.execute(RobotResponseHandler handler)` は、実行を開始する前にすぐに値を返します。

Java デバッグ

Java API はデバッグを目的とした追加の情報を提供します。API デバッグを有効にするには、システム プロパティ `DEBUG_ON` を設定します。このプロパティの値は、Java API のパッケージ/クラス名である必要があります。

たとえば、API と Management Console の間での状態の送信を確認する場合は、パッケージ `com.kapowtech.robosuite.api.java.rest.internal` にデバッグ情報を要求することができます。

開発中に、コードでシステム プロパティを直接設定することでこの操作を実行します。

システム デバッグを有効にする

```
System.setProperty("DEBUG_ON", "com.kapowtech.robosuite.api.java.rest.internal");
```

本番環境でアプリケーションをデバッグする場合は、コマンド ラインからシステム プロパティを定義できます。

アプリケーション デバッグを有効にする

```
java -DDEBUG_ON=com.kapowtech.robosuite.api.java.rest.internal
```

複数のパッケージからデバッグする場合は、パッケージ名をコンマで区切ります。パッケージ名の代わりに、引数 `ALL` を指定すると、すべてのパッケージからのデバッグを出力することができます。

Java リポジトリ API

リポジトリ API を使用して Management Console のリポジトリを照会し、プロジェクト、ロボット、およびロボットを呼び出すために必要な入力のリストを取得します。また、ロボット、タイプ、リソース ファイルをプログラムでデプロイする場合にもリポジトリ API を使用します。

リポジトリ API を使用するには、Tungsten RPA インストール フォルダにある `\API\rpa-api-java\lib` フォルダ内のすべての `.jar` ファイルをプロジェクトの `classpath` に追加します。

Java リポジトリ クライアント

リポジトリとの通信は、`RepositoryClient` を通じて `com.kapowtech.robosuite.api.java.repository.engine` で行われます。

`RepositoryClient` の作成

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

import java.net.*;

/**
* This example shows how to create a RepositoryClient that connects to a repository.
*
*/
public class RepositoryClientExample {

    public static void main(String[] args) {
        String apiKey = "{ api key }"; // To be filled out
        try {
            RepositoryClient client = RepositoryClientFactory.createRepositoryClient
                (new URL("http://localhost:8080/ManagementConsole"), apiKey);
            Project[] projects = client.getProjects();
            for (Project project : projects) {
                System.out.println(project.getName());
            }
        }
        catch (RepositoryClientException | MalformedURLException e) {
            e.printStackTrace();
        }
    }
}
```

コード例では、`http://localhost:8080/ManagementConsole/` 上の Management Console のリポジトリに接続するように `RepositoryClient` が設定され、API キーを使用しています。

`RepositoryClient` が作成された後、 `getProjects()` メソッドは、プロジェクトのリストのリポジトリを照会するために使用されます。`RepositoryClient` メソッドのいずれかを呼び出すときにエラーが発生した場合、`RepositoryClientException` がスローされることに注意してください。

`RepositoryClient` には次のメソッドがあります。

`RepositoryClient` のメソッド

メソッド署名	説明
`void deleteResource(String projectName, String resourceName, boolean silent)`	プロジェクトからリソースを削除します。 silent が true の場合、リソースが存在しなくてもエラーは生成されません。 `resourceName` 引数はリソースのフル パスを使用します。
`void deleteRobot(String projectName, String robotName, boolean silent)`	プロジェクトからロボットを削除します。 `robotName` 引数はロボットのフル パスを使用します。
`void deleteSnippet(String projectName, String snippetName, boolean silent)`	プロジェクトからスニペットを削除します。 `snippetName` 引数はスニペットのフル パスを使用します。
`void deleteType(String projectName, String modelName, boolean silent)`	プロジェクトからタイプを削除します。 `modelName` 引数は、タイプのフルパスを使用します。
`void deployResource(String projectName, String resourceName, byte[] resourceBytes, boolean failIfExists)`	リソースをプロジェクトにデプロイします。指定された名前のリソースがすでに存在する場合、`failIfExists` を false に設定することで上書きできます。 `resourceName` 引数はリソースのフル パスを使用します。
`void deployRobot(String projectName, String robotName, byte[] robotBytes, boolean failIfExists)`	ロボットをプロジェクトにデプロイします。指定された名前のロボットがすでに存在する場合、`failIfExists` を false に設定することで上書きできます。 `robotName` 引数はロボットのフル パスを使用します。
`void deploySnippet(String projectName, String snippetName, byte[] snippetBytes, boolean failIfExists)`	スニペットをプロジェクトにデプロイします。指定された名前のスニペットがすでに存在する場合、`failIfExists` を false に設定することで上書きできます。 `snippetName` 引数はスニペットのフル パスを使用します。
`void deployType(String projectName, String typeName, byte[] typeBytes, boolean failIfExists)`	タイプをプロジェクトにデプロイします。指定された名前のタイプがすでに存在する場合、`failIfExists` を false に設定することで上書きできます。 `typeName` 引数は、タイプのフルパスを使用します。
`Project[] getProjects()`	このリポジトリに存在するプロジェクトを返します。

メソッド署名	説明
`Robot[] getRobotsInProject(String projectName)`	プロジェクトで使用可能なベーシック エンジン ロボットのフルパスを返します。
`RobotSignature getRobotSignature(String projectName, String robotName)`	ロボットの完全なパス、ロボットの実行に必要な入力変数、およびロボットが返す可能性のあるタイプまたは保存する可能性のあるタイプのリストとともに、ロボットの署名を返します。
`RepositoryFolder getProjectInventory(String projectName)`	リポジトリからフォルダとファイルのツリー全体を返します。
`RepositoryFolder getFolderInventory(String projectName, String folderPath)`	指定したプロジェクトのサブ フォルダのフォルダとファイルをリポジトリから返します。
`RepositoryFolder getFileInventory(String projectName, String folderPath, String fileName, RepositoryFile.Type fileType)`	Management console からファイルと参照ファイルを取得します。 ファイル インベントリは、参照を取得するために `RepositoryFolder` でラップされることに注意してください。
`void deleteFile(RepositoryFile file)`	指定されたファイルをリポジトリから削除します。
`Date getCurrentDate()`	Management Console の現在の日付と時刻を返します。
`byte[] getBytes(RepositoryFile file)`	リポジトリ内の指定されたファイルのサイズをバイト単位で返します。
`void updateFile(RepositoryFile file, byte[] bytes)`	リポジトリ内の指定されたファイルを新しいバイトで更新します。
`void moveFile (RepositoryFile sourceFile, String destFolderPath)`	指定されたファイルをリポジトリから `destFolderPath` で指定されているフォルダに移動します。
`void renameRobot(RepositoryFile robotFile, String newName)`	指定したロボット ファイルの名前を変更します。
`void deleteFolder(String projectName, String folderPath)`	リポジトリ内の指定されたフォルダを削除します。
`Map<String, String> getInfo()`	Management Console およびリポジトリ API についての情報を返します。 このメソッドは、次のマッピングを返します。 • 「アプリケーション」に対する、メジャー バージョン、マイナー バージョン、およびドット バージョンを含む Management Console のバージョン (例: `2026.1`) • 「リポジトリ」に対する、リポジトリ API で使用される最新の DTD の ID (例: `//Kapow Technologies//DTD Repository 1.5//EN`)

メソッド署名	説明
`pingRepository()`	リポジトリ サーバーに対して ping を実行すると、成功した場合は `null` を返し、失敗した場合はエラー文字列を返します。
`deployConnector(String projectName, String connectorName, byte[] connectorBytes, boolean failIfExists, AdditionalInfo additionalInfo)`	リポジトリにコネクタを展開します。
`deleteConnector(String projectName, String connectorName, boolean silent, AdditionalInfo additionalInfo)`	リポジトリからコネクタを削除します。
`getRobotsByTag(String projectName, String tag)`	指定したタグを持つロボットを取得します。

フル パスはプロジェクト フォルダからの相対パスです。

詳細については `RepositoryClient` JavaDoc を確認してください。

Java ロボットの展開

次の例は、`RepositoryClient` を使用して、ローカル ファイル システムからロボットとタイプを展開する方法を示しています。

`RepositoryClient` を使用したロボットの展開:

```
import com.kapowtech.robosuite.api.java.repository.engine.*;
import com.kapowtech.robosuite.api.java.repository.engine.FileAlreadyExistsException;

import java.io.*;
import java.net.*;
import java.nio.file.*;

public class DeployRobot {
    public static void main(String[] args) {
        String apiKey = "{ api key }";
        String mcUrl = "http://localhost:8080/ManagementConsole";
        String libraryPath = "c:\\MyRobots\\Library\\";
        String robotName = "Test.robot";
        String typeName = "Test.type";
        String projectName = "Default project";
        boolean failIfExists = true;

        try {
            RepositoryClient client = new RepositoryClientImpl(new URL(mcUrl), apiKey);
            client.deployRobot(
                    projectName,
                    robotName,
                    Files.readAllBytes(Paths.get(libraryPath + robotName)),
                    failIfExists
            );
            client.deployType(
                    projectName,
                    typeName,
```

```
                    Files.readAllBytes(Paths.get(libraryPath + typeName)),
                    failIfExists
            );
        }
        catch (IOException | FileAlreadyExistsException | RepositoryClientException e)
 {
            System.out.println(e.getMessage());
        }
    }
}
```

第 2 章

.NET

この章では、Tungsten RPA .NET API を使用して、ロボットを実行する方法について説明します。ロボットを実行し、.NET API アプリケーションから Management Console リポジトリにアクセスします。この情報は、簡単なロボットの記述方法を把握し、C# プログラミング言語に精通している開発者を対象としています。

.NET API は、Tungsten RPA インストール フォルダ内の `API\robosuite-dotnet-api\lib` にある `robosuiteapi.dll` ファイルです。詳細については、『Tungsten RPA インストール ガイド』の「重要なファイルとフォルダ」を参照してください。

`Newtonsoft.JSon.dll` は .NET API ファイルに含まれる必須のサードパーティ ライブラリです。

.NET API を使用するプログラムには、`System.Security.Permissions` パッケージを含める必要がある場合があります。そのパッケージのバージョンは、使用している .NET のバージョンに応じて異なります。

.NET API クラスに関する情報は、オフライン ドキュメント フォルダにある `api\robosuite-dotnet-api` ヘルプ フォルダを参照してください。詳細については、『Tungsten RPA インストール ガイド』を参照してください。

.NET の基本

任意の .NET ベースのアプリケーションで .NET API を使用して、そのアプリケーションを RPA インスタンスのクライアントにします。.NET API は .NET Standard 2.0 をターゲットとしており、.NET (Core) 2.0 ～ 6.0、または .NET Framework 4.7.2 以降のアプリケーションをサポートします。

ロボットをプログラミングしてデータベースにデータを保存するだけでなく、ロボットをプログラミングしてクライアント アプリケーションに直接データを返すようにすることもできます。ここではいくつかの例を示します:

- 複数のロボットを使用して、複数のソースからの結果をリアルタイムで集約する検索を実行します。
- アプリケーションのバックエンドのイベントに応じてロボットを実行します。たとえば、新しいユーザーがサインアップしたときにロボットを実行して、バックエンドに直接統合されていない Web ベースのシステムにアカウントを作成します。

.NET 認証

API キーを使用して、Management Console で API を認証します。

Management Console でユーザーの API キーを作成します。『Tungsten RPAユーザー ガイド』を参照してください。

認証後に、API キーが提供されたユーザー アカウントの認証を使用して .NET API が実行されます。

Management Console では、.NET API を認証するためにユーザー名とパスワードを使用することはできません。

.NET ログ

.NET API は、主にトラブルシューティングのためにログ データを提供します。.NET API 自体がログ メッセージを書き込むことはありません。アプリケーションが .NET API ログ記録を独自のログと統合できるように、プラグインが提供されます。アプリケーションは、.NET API がログ メッセージとともに呼び出すコールバックまたはインターフェイスを登録します。

ログ機能は、`KofaxRpa.Logging` 名前空間にあります。

- 最も単純な形式のロギングにアクセスするには、デリゲート関数 `public delegate void Logger(Level level, string msg);` を `SimpleLogger.SetLogger(Logger logger)` に登録します。

  API がログに記録する内容がある場合、デリゲートが呼び出され、ログ レベルとメッセージが渡されます。メッセージのレベルが `LogLevel` のしきい値を下回っている場合、デリゲートが呼び出されることはありません。
- ログ レベルを設定するには、`SimpleLogger.LogLevel` プロパティを使用して、値を `Fatal`、`Error`、`Warn`、`Info`、および `Debug` に設定します。

`SimpleLogger` の機能によって必要な内容が提供されない場合は、`ILogFactory` インターフェイスの実装をアプリケーションの `Registered.Logger` に登録します。このインターフェイスは、リクエストに応じて、特定のタイプ (API のコンポーネント) の `ILog` インターフェイスを実装したオブジェクトを返します。詳細については、オフライン ドキュメントのヘルプ フォルダを参照してください。

ログの記録に `log4net` を使用するアプリケーションは、`API/lib/log4netlogging` ディレクトリにある `log4netlogging.dll` アセンブリを使用できます。

`Com.KapowTech.RoboSuite.Api.Log4NetLogging` 名前空間の `Logging` クラスは、`log4net` ログを登録するために次の 2 つのメソッドを提供します。

- `Logging.Install(string config_file)` は、設定ファイルから設定された `log4net` ログ記録を開始します。
- `Install()` には、アプリケーションによる `log4net` 自体の設定が必要です。

ログの記録方法が登録されていない場合、.NET API はすべてのログ データを暗黙的に破棄します。

.NET リポジトリ API

.NET リポジトリ API を使用して Management Console のリポジトリを照会し、ロボットを呼び出すために必要なプロジェクト、ロボット、および入力のリストを取得します。また、ロボット、タイプ、リソース ファイルをプログラムでデプロイする場合にもリポジトリ API を使用します。

.NET リポジトリ クライアント

リポジトリとの通信は、`RepositoryClient` を通じて `Com.KapowTech.RoboSuite.Api.Repository.Engine` のネームスペースで実現されます。

`RepositoryClient` からのプロジェクトの取得

```
String mcUrl = "http://localhost:8080/ManagementConsole";
String apiKey = "{ api key }";
RepositoryClient client = new RepositoryClient(mcUrl, apiKey);
Project[] projects = client.GetProjects();
foreach (Project p in projects) Console.WriteLine(p);
```

ここで、認証用の API キーを使用して `http://localhost:8080/ManagementConsole` 上の Management Console のリポジトリに接続するように、`RepositoryClient` を設定する必要があります。

`RepositoryClient` が作成された後、 `GetProjects()` メソッドは、プロジェクトのリストのリポジトリを照会するために使用されます。`RepositoryClient` メソッドのいずれかを呼び出すときにエラーが発生した場合、`RepositoryClientException` がスローされることに注意してください。

`RepositoryClient` には、次のようなオブジェクト タイプをリポジトリに展開するメソッドや、リポジトリから削除するメソッドがあります。コネクタ、ファイル、フォルダ、リソース、ロボット、スニペット、およびタイプ。また、メソッドにより、ファイル、フォルダ、プロジェクトのインベントリを取得することもできます。詳細については、オフライン ドキュメントのヘルプ フォルダを参照してください。

リポジトリには `http` 経由でアクセスします。.Net バージョンのリポジトリ API を使用する場合は、システム用に設定されたプロキシ サーバーがリポジトリ API で使用されます。

.NET ロボットの展開

次の例は、`RepositoryClient` を使用して、ローカル ファイル システムからロボットとタイプを展開する方法を示しています。

`RepositoryClient` を使用したロボットの展開

```
String mcUrl = "http://localhost:8080/ManagementConsole";
String apiKey = "{ api key }";
RepositoryClient client = new RepositoryClient(mcUrl, apiKey);
byte[] robotBytes = File.ReadAllBytes(@"c:\MyRobots\Library\Test.robot");
byte[] typeBytes = File.ReadAllBytes(@"c:\MyRobots\Library\Test.type");
```

```
// Assume that the Default project still exists
client.DeployRobot("Default project", "Test.robot", robotBytes, true);
client.DeployType("Default project", "Test.type", typeBytes, true);
```

Management Console の .NET

Management Console で、ロボットの実行をキューに格納します。ロボットを RoboServer で直接実行するのではなく、ロボットを Management Console のキューに配置できます。

実行 ID の設定、データベース接続の定義、最大実行時間の設定、API 例外時のロボットの強制停止など、一部の機能は、Management Console API キューを使用して制御できないことに注意してください。

Management Console API キューを使用する利点は次のとおりです。

- 特定のリソース (デバイスなど) が利用できない場合でも、ロボットはキューに登録されます。
- 複数のバージョンの RoboServer が利用可能な場合、ロボットは正しい RoboServer にルーティングされます。
- アプリケーションの構築中にクラスタを管理する必要がなくなります。クラスタ設定は、Management Console でプロジェクト レベルで管理されます。

Management Console にアップロードされたロボットの場合、ロボットを呼び出す方法を示したコード スニペットを生成することができます。『Tungsten RPAユーザー ガイド』を参照してください。

ロボットの実行のキューへの格納

.NET API を使用して `QueuedRequest` クラスを使用し、Management Console でロボットを実行します。

例:

```
string mcUrl = "http://localhost:8080/ManagementConsole";
string apiKey = "{ api key }";
QueuedRequest request = new QueuedRequest(mcUrl, "Default project", "myfolder/
myrobot.robot", apiKey);
request.Priority = QueuedRequestPriority.HIGH;
ExecutionResult result = request.Execute();
```

このコード スニペットは、ロボットを実行キューに格納し、実行が完了するまで待機します。ロボットを待機せずにキューに格納するには、ロボットの実行が開始および停止したときに通知を受け取り、結果を収集するハンドラを `Execute()` 関数に渡します。

また、リクエストを実行する前に、入力オブジェクトをリクエストに追加します。`CreateInputVariable()` 関数は、`RpaObject` 型の空の入力オブジェクトを返します。このオブジェクトには、さまざまなタイプの属性を追加するための `Add()` メソッドがあります。

例:

```
request.CreateInputVariable("person")
```

```
        .Add("ID", 42)
        .Add("name", "Jane Doe")
        .Add("age", 37)
        .Add("isMale", false);
```

詳細については、オフライン ドキュメントのヘルプ フォルダを参照してください。

.NET 属性タイプのマッピング

Design Studio で新しいタイプを定義するとき、各属性のタイプを選択します。一部の属性には、ショート テキスト、ロング テキスト、パスワード、HTML、XML などのテキストを含めることができます。

ロボット内で使用する場合、これらの属性に保存されるテキストには要件がある場合があります。XML 属性にテキストを保存する場合、テキストは有効な XML ドキュメントである必要があります。この検証は、ロボット内でタイプが使用されている場合に発生しますが、.NET API ではタイプについて何も検出されないため、同じ方法による属性値の検証は行われません。その結果、.NET API では Design Studio よりも少ない数の属性タイプが使用されます。

この表は、.NET API と Design Studio 属性タイプのマッピングを示しています。

.NET API から Design Studio 属性タイプへのマッピング

API 属性タイプ	Design Studio 属性タイプ
TEXT	ショート テキスト、ロング テキスト、パスワード、HTML、XML、プロパティ、言語、国、通貨、および RefindKey
INTEGER	Integer
BOOLEAN	Boolean
NUMBER	Number
CHARACTER	Character
DATE	日付
BINARY	Binary、Image、PDF

API 属性タイプは、次の方法で C# にマッピングします。

.NET API から C# 属性タイプへのマッピング

API 属性タイプ	C# 属性タイプ
TEXT	string
INTEGER	long
BOOLEAN	bool
NUMBER	double
CHARACTER	char
DATE	DateTime
BINARY	Binary (KofaxRpa 名前空間のクラス)

.NET 属性プロパティ

RpaObject 内の属性には、Attributes プロパティを介してアクセスできます。これは、属性名をキーとする Dictionary です。これには、属性の名前、タイプ、および値へのアクセスを提供する RpaObject.Attribute オブジェクトが含まれています。

値は静的型 object であり、属性タイプに応じてキャストする必要があることに注意してください。「.NET 属性タイプのマッピング」を参照してください。

あるいは、RpaObject の [] インデックス演算子を使用すると、属性の値に直接アクセスすることができます。演算子の添字は属性名です。

ロボットの入力を準備する場合、RpaObject には複数の方法で属性を追加できます。

RpaObject.Add(Attribute) メソッドは、名前、タイプ、値から構築された Attribute を受け取ります。RpaObject.Add(string, AttributeType, object) は Attribute を作成して、それを追加します。

あるいは、提供された値に基づいて Attribute を作成する、すべての属性タイプに対するオーバーロードがあります。たとえば、Add(string, bool) は、タイプ BOOLEAN の Attribute を作成し、それを RpaObject に追加します。

詳細については、オフライン ドキュメントのヘルプ フォルダを参照してください。

Management Console コネクタ

MCConnector を使用して、Management Console に接続するオブジェクトを構築します。このトピックでは次の 2つの例を示します。

- RepositoryClient と QueuedRequest
- RemoteCertificateValidationCallback

RepositoryClient と QueuedRequest

MCConnector を作成して、Management Console の URL と [API キー] を 1 つのオブジェクトにバンドルします。

Management Console の URL と [API キー] を RepositoryClient または QueuedRequest オブジェクトに個別に渡す代わりに、MCConnector を使用して両者に渡します。MCConnector オブジェクトを作成した後は、このオブジェクトを繰り返し使用することができます。

MCConnector を使用したバンドルの作成

```
string mcUrl = "http://localhost:8080/ManagementConsole"
string apiKey = "{ api key }";
MCConnector connector = new MCConnector(mcUrl, apiKey);
...
RepositoryClient client = new RepositoryClient(connector);
```

```
...
QueuedRequest request = new QueuedRequest(connector, "Default project", "myfolder/
myrobot.robot");
...
```

RemoteCertificateValidationCallback

独自の証明書検証を実装し、その検証を `RemoteCertificateValidationCallback` の形式で `MCConnector` コンストラクタに渡します。このコールバックは、デフォルトのシステムによる証明書の処理が要件を満たしていない場合に役立つ可能性があります。

Management Console へのアクセスが HTTPS 経由である場合は、その証明書の有効性がチェックされます。`RemoteCertificateValidationCallback` を使用すると、検証の結果とともに `MCConnector` が呼び出され、証明書を受け入れて接続を確立する必要があるかどうかが判断されます。

`MCConnector` を使用したリモート証明書検証の作成

```
string mcUrl = "https://localhost:8081/ManagementConsole"
string apiKey = "{ api key }";
MCConnector connector = new MCConnector(mcUrl, apiKey, new
 RemoteCertificateValidationCallback(MyValidationFunction));
```

`RemoteCertificateValidationCallback` の詳細については、Microsoft .NET のドキュメントを参照してください。API には、システムの証明書ストアにリストされていない機関によって署名された証明書を受け入れる方法を示す実装例が含まれています。

第 3 章

REST サービス

この章では、Tungsten RPA で提供される Management Console REST サービスについて説明します。

- 一部のシナリオでは、たとえば大量のスケジュールを実行するために、Management Console の REST API にアクセスして編集する必要がある場合があります。
- RPA 用のクライアント アプリケーションを開発するには、JSON を用いた REST の使用が推奨されます。クライアント アプリケーションを開発する際は、多くの場合、Management Console REST API を使用する方が便利かつ簡単です。
- Java、.NET、および XML を使用した REST など、RPA 用クライアント アプリケーションを開発するためのその他のオプションもサポートされていますが、将来のリリースではサポートの範囲が縮小される可能性があります。

Tungsten RPA Management Console REST API が変更される可能性があります。Tungsten RPA のそれぞれのリリースでは、REST API のカスタマイズされたプログラミングを調整する必要がある変更が発生する可能性があります。

JSON を使用した REST

Management Console REST API で動作する REST API クライアント アプリケーションを作成するために、Management Console は JSON API 仕様ファイルをダウンロードするためのエンドポイントを提供します。便宜上、サードパーティの API エクスプローラ アプリケーションを使用して REST API を操作します。

Management Console で REST サービスを使用するには認証が必要です。

次の構文の URL パスを使用して、JSON API ファイルをダウンロードし、API エクスプローラにインポートします。

```
protocol://<host>:<port>/<context-path>/api/mc/api-docs
```

例:

```
http://localhost:8080/ManagementConsole/api/mc/api-docs
```

XML を使用した REST

リポジトリ API は、Restful サービスとデータをポストできる URL のグループです。

リポジトリから情報を取得するすべてのリポジトリ クライアント メソッドは、XML をリポジトリに送信し、リポジトリは XML で応答します。すべてのデプロイメソッドは、バイトをリポジトリ (URL でエンコードされた情報) にポストし、リポジトリは確認のために XML を返します。送受信される XML の形式は、ドキュメント タイプ定義 (DTD) によって制御されます。

REST 操作

ロボット、タイプ、スニペット、およびリソース名は、完全なパスで指定する必要があります。フル パスはプロジェクト フォルダからの相対パスです。

REST 操作

メソッド	リクエスト例	応答例
delete-file (robot)	<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>	<repository-response><delete-successful/></repository-response>
delete-file (type)	<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>	<repository-response><error type="file-not-found">Could not find a Type named InputA.type in project 'Default project'</error></repository-response>
delete-file (snippet)	<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>	<repository-response><delete-successful/></repository-response>
delete-file (resource)	<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>	<repository-response><delete-successful/></repository-response>

メソッド	リクエスト例	応答例
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>anInt</attribute-name><attribute-type>Integer</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><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 requir

メソッド	リクエスト例	応答例
		ed="true"><attribute-name>aStrin g</attribute-name><attribute-typ e>Short Text</attribute-type></inp ut-attribute><input-attribute req uired="true"><attribute-name>anI nt</attribute-name><attribute-typ e>Integer</attribute-type></input- attribute><input-attribute requir ed="true"><attribute-name>aNumbe r</attribute-name><attribute-typ e>Number</attribute-type></input- attribute><input-attribute requir ed="true"><attribute-name>aSessi on</attribute-name><attribute-typ e>Session</attribute-type></input- attribute><input-attribute requir ed="true"><attribute-name>aBoole an</attribute-name><attribute-typ e>Boolean</attribute-type></input- attribute><input-attribute requir ed="true"><attribute-name>aDate</ attribute-name><attribute-type>Dat e</attribute-type></input-attrib ute><input-attribute required="tru e"><attribute-name>aCharacter</att ribute-name><attribute-type>Charac ter</attribute-type></input-attrib ute><input-attribute required="tru e"><attribute-name>anImage</attrib ute-name><attribute-type>Image</ attribute-type></input-attribute ></input-attribute-list></input- object></input-object-list><return ed-type-list><returned-type><typ e-name>OutputA</type-name><return ed-attribute-list><returned-attrib ute><attribute-name>aString</att ribute-name><attribute-type>Short Text</attribute-type></returned- attribute></returned-attribute-lis t></returned-type></returned-typ e-list><stored-type-list/></robot- signature></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>

メソッド	リクエスト例	応答例
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>
get-file-inventory	<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>	<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>
update-file	<repository-request> <update-file> <repository-file-request>...</repository-file-request> <file-bytes></update-file> </repository-request>	<repository-response> <update-successful/> </repository-response>
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>

メソッド	リクエスト例	応答例
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>

XML ベースのリクエストの例

以下は、すべての XML ベースのリクエストの例です。すべてのメッセージは、次の宣言で始まる必要があります。

```
<?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">
```

Management Console が `http://localhost:8080/ManagementConsole` に展開されている場合、リクエストは `http://localhost:8080/ManagementConsole/secure/RepositoryAPI?format=xml` にポストする必要があります。

XML スニペット

API 全体を通じて、例では XML スニペットが使用されています。DTD を調べて、データの構造を理解することをお勧めします。

リクエストを送信する場合は、通常ファイルを記述する必要があります。同様に、応答にはファイルに関するデータが含まれます。次の表は、例で短縮されているスニペットを示しています。Design Studio と Management Console.のプロジェクト同期を支援するために構成要素が 1.5 DTD に追加されました。

スニペット名	コード
repository-file-request	<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>
repository-file	<repository-file><name>filename</name> <type>ROBOT</name><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>

ロボット、タイプ、スニペット、およびリソース名は、完全なパスで指定する必要があります。フル パスはプロジェクト フォルダからの相対パスです。

展開は、生のバイト (オクテット ストリームがポスト本文として送信されます) を次の URL にポストすることで実行します。以下は、リポジトリが `http://localhost:8080/ManagementConsole` で展開されている場合の例です。

展開操作のメソッド:

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