Oracle® Endeca Web Acquisition Toolkit

使用ガイド

リリース3.1.0　2013年10月

ORACLE

著作権および免責事項

OracleおよびJavaはOracle Corporationおよびその関連企業の登録商標です。その他の名称は、それぞれの所有者の商標または登録商標です。UNIXは、The Open Groupの登録商標です。

このソフトウェアおよび関連ドキュメントの使用と開示は、ライセンス契約の制約条件に従うものとし、知的財産に関する法律により保護されています。ライセンス契約で明示的に許諾されている場合もしくは法律によって認められている場合を除き、形式、手段に関係なく、いかなる部分も使用、複写、複製、翻訳、放送、修正、ライセンス供与、送信、配布、発表、実行、公開または表示することはできません。このソフトウェアのリバース・エンジニアリング、逆アセンブル、逆コンパイルは互換性のために法律によって規定されている場合を除き、禁止されています。

ここに記載された情報は予告なしに変更される場合があります。また、誤りが無いことの保証はいたしかねます。誤りを見つけた場合は、オラクル社までご連絡ください。

このソフトウェアまたは関連ドキュメントを、米国政府機関もしくは米国政府機関に代わってこのソフトウェアまたは関連ドキュメントをライセンスされた者に提供する場合は、次の通知が適用されます。

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations.As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs.No other rights are granted to the U.S. Government.

このソフトウェアもしくはハードウェアは様々な情報管理アプリケーションでの一般的な使用のために開発されたものです。このソフトウェアもしくはハードウェアは、危険が伴うアプリケーション(人的傷害を発生させる可能性があるアプリケーションを含む)への用途を目的として開発されていません。このソフトウェアもしくはハードウェアを危険が伴うアプリケーションで使用する際、安全に使用するために、適切な安全装置、バックアップ、冗長性(redundancy)、その他の対策を講じることは使用者の責任となります。このソフトウェアもしくはハードウェアを危険が伴うアプリケーションで使用したことに起因して損害が発生しても、オラクル社およびその関連会社は一切の責任を負いかねます。

このソフトウェアまたはハードウェア、そしてドキュメントは、第三者のコンテンツ、製品、サービスへのアクセス、あるいはそれらに関する情報を提供することがあります。オラクル社およびその関連会社は、第三者のコンテンツ、製品、サービスに関して一切の責任を負わず、いかなる保証もいたしません。オラクル社およびその関連会社は、第三者のコンテンツ、製品、サービスへのアクセスまたは使用によって損失、費用、あるいは損害が発生しても一切の責任を負いかねます。

目次

表一覧

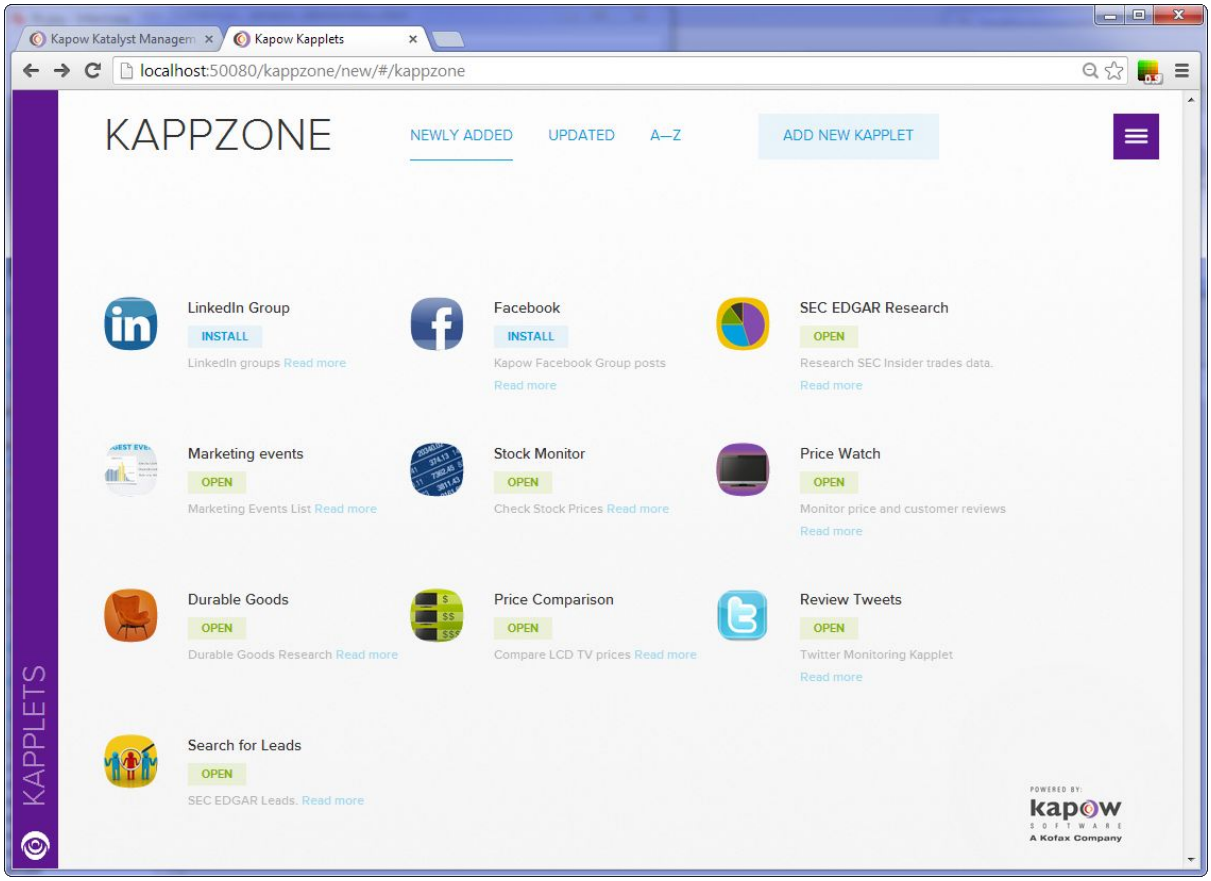
第1章 9.3の新機能

Kapow Katalyst 9.3の新機能の重要な部分は、次のとおりです。

- 「KappZone™ 9.3のすぐに使用できるビジュアルなData Exploration」
- 「Webアプリケーションからの統合APIの迅速な作成」
- 「RESTおよびSOAP Webサービス用のXMLおよびJSONペイロードのより簡単な構成」
- 「ユーザー管理」
- 「追加の機能および改善」
- 「サポート期限の通知」

KappZone™ 9.3のすぐに使用できるビジュアルなData Exploration

データから最大価値を引き出すには、よりインテリジェントで、情報に基づいた意思決定を促進するようにデータを使用可能にする必要があります。バージョン9.3の新しいKapplet CompositionおよびData Exploration機能を使用したKappZone™の再設計によって、データを必要なときにユーザーの手元に直接視覚的に配信し、それらをソート、フィルタおよび処理できるようにするすばやいソリューションを提供しています。



Kapplet Composition

ユーザーがKappletデータセットのデータを1行以上選択して、選択した行をロボットへの入力として使用する処理を実行できる機能を備えた、複数のページおよび様々なページ・レイアウトがあるKappletsを設計します。

Kapplet Data Exploration

データ結果は、表、グラフ、または表とグラフの組合せを組み込むことができる優れたビジュアル・ページにレンダリングされ、ユーザーに対して、ビジュアル表現とドリルダウンのためのスマート・フィルタを使用した、Kappletのデータセットのインタラクティブな参照を提供します。グラフには、ヒストグラム、折れ線グラフ、円グラフ、バブル・チャートおよび散布図が含まれます。

Webアプリケーションからの統合APIの迅速な作成

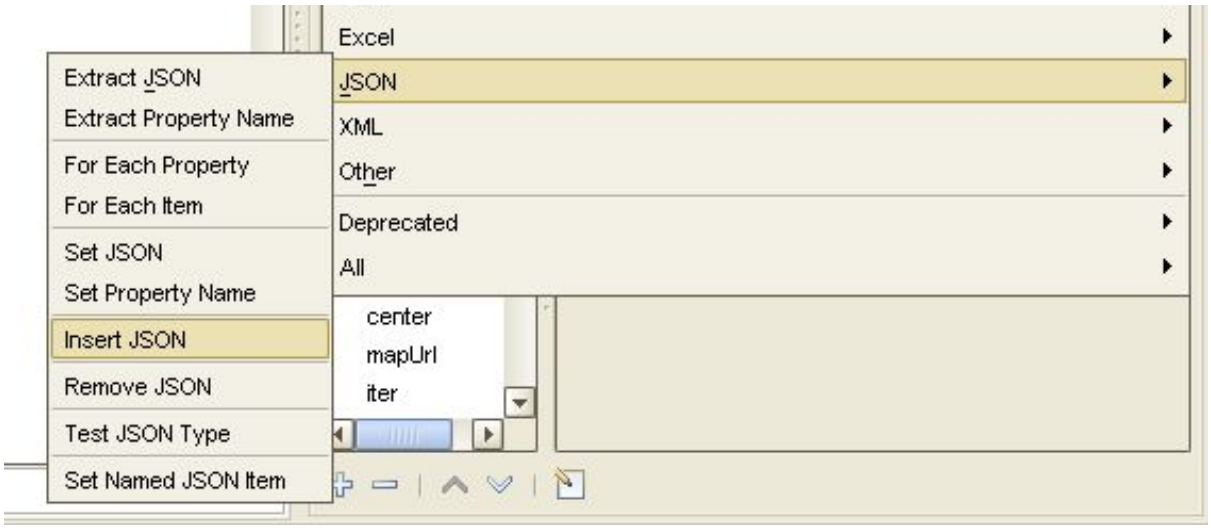
9.3では、新しい埋込みのWebKitブラウザ・エンジンを導入しており、これによって、データ統合設計用のターゲット・サイトおよびアプリケーションのより高速で効果的なレンダリングが可能になります。これは、HTML5および高度なJavaScriptフレームワークを使用するインタラクティブWebアプリケーションをサポートするように最適化されています。また、静的HTMLおよびレガシーWebアプリケーション用に最適化された、標準のブラウザ・エンジンも引き続きサポートしています。

既存のロボットを新しいWebKitブラウザ・エンジンを使用するように移行するために、プロジェクト・ビューで直接使用可能な移行機能があります。

RESTおよびSOAP Webサービス用のXMLおよびJSONペイロードのより簡単な構成

Kapow Katalyst 9.3では、Design Studioで、XMLおよびJSON編集機能を使用してWebサービスとネイティブに対話する機能が向上しています。

- XMLまたはJSON変数を直接編集のためにメインのワークスペース・ウィンドウで開きます。
- 複合タイプの変数をXMLまたはJSONに直接変換します。
- XMLおよびJSONで操作するための新しい抽出および編集ステップ。



ユーザー管理

9.3では、ユーザー認証を管理するためのLDAP/Active Directoryを使用するかわりのオプションとして、Management Consoleで組込みのユーザー管理が提供されています。ロールおよび権限に基づいてKappletsの配布が制御されます。

Management Consoleのユーザー管理タブでは、次が提供されます。

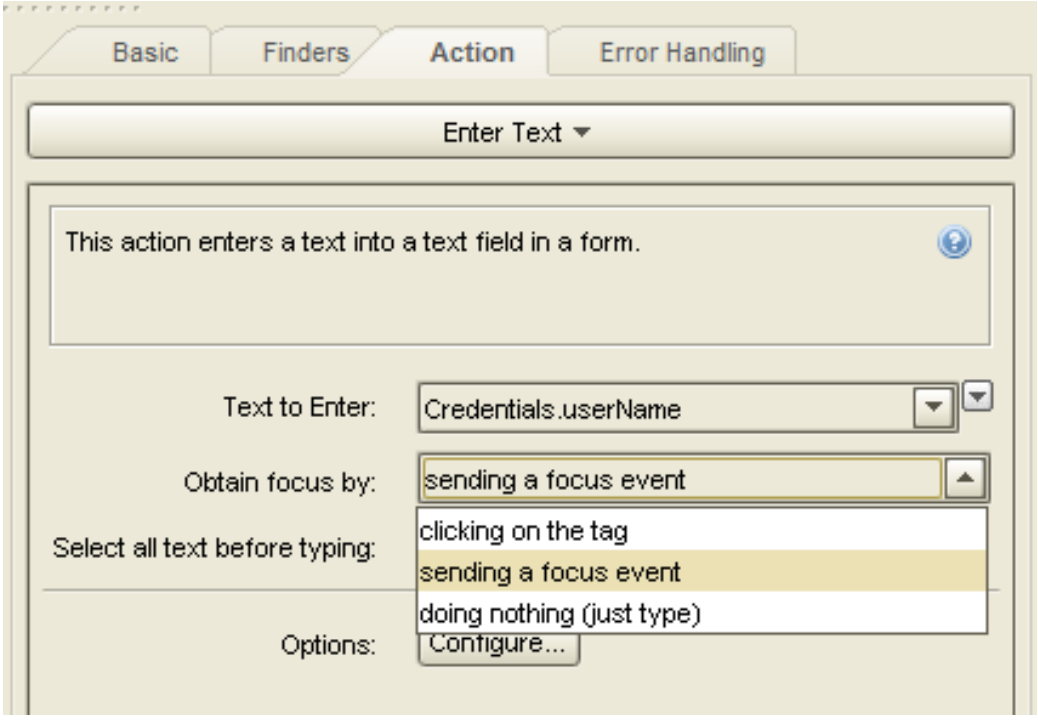
・ ユーザーの追加、編集および削除

・ セキュリティ・グループの追加、編集および削除

組込みのユーザー管理は、RoboServerのSettingsで有効化します。

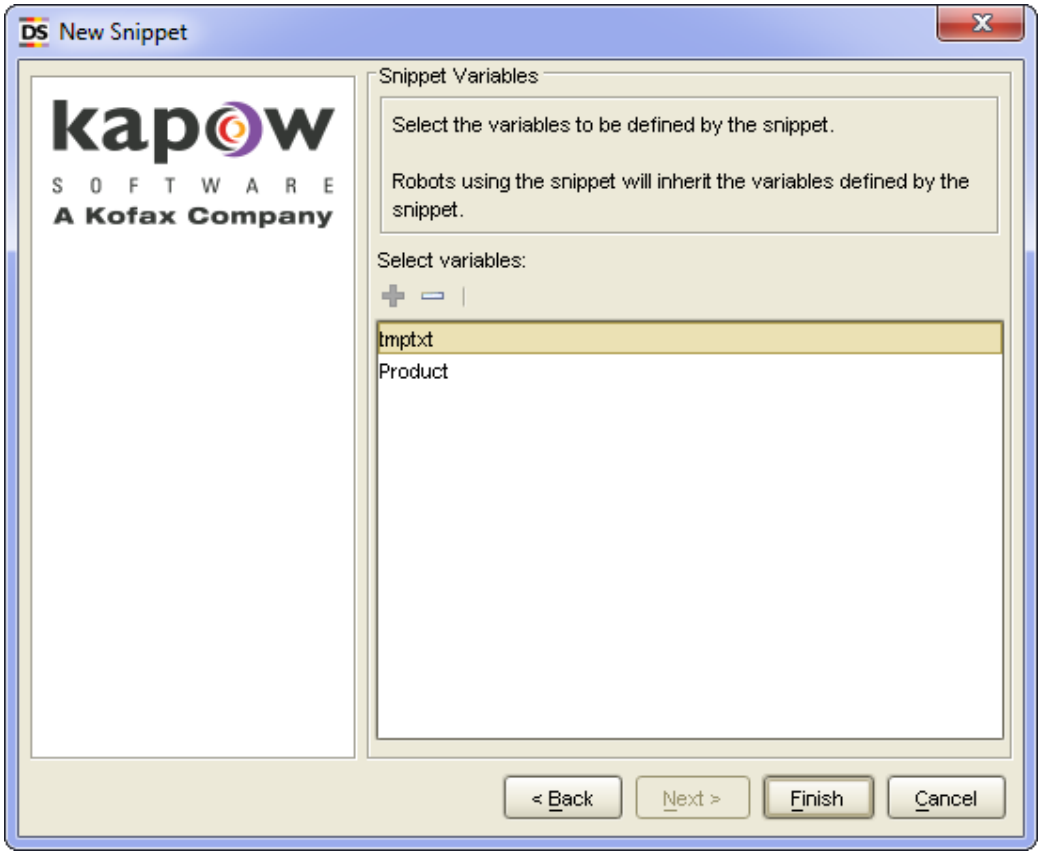
追加の機能および改善

テキストの入力: JavaScriptでアクティブ化される入力フィールドのサポート。テキスト入力前にテキスト・フィールドのフォーカスを構成するためのオプション。



キーの押下: JavaScriptの特殊なキー押下イベント([Enter]、[Tab]など)の使用のサポート。

スニペットの作成: 関連変数をスニペットに組み込むオプションが提供されています。



パスの抽出: 変数に対する完全なタグ・パスを抽出します。これは、後続のタグ・ファインダで同じタグを再度識別するために使用できます。

サポート期限の通知

9.3では、次のサポート期限の通知があります。

JMS RoboServerサービスのサポート期限

JMS RoboServerサービスは非推奨であり、2014年の最初のメジャー・リリースで削除されます。このサービスを使用しているお客様は、ソケット・ベースのサービスに移行する必要があります。

古い記憶域システムのサポート期限

バージョン7.2 (2010年春リリース)で、Kapow Katalystに新しい記憶域システムとロギング・システムを導入しました。古いシステムは、お客様が新しい改善されたシステムに移行できるように保持されました。

2014年の最初のメジャー・リリースで、古いシステムはサポート期限となり、削除されます。

削除の対象は、RoboManager、データベース記憶域環境、ファイル記憶域環境、複数ファイル記憶域環境、データベース・メッセージ環境、電子メール・メッセージ環境、ファイル・メッセージ環境、オブジェクトの再検索ステップ、キーの再検索属性タイプ、およびデータベース出力タイプです。

まだ古いシステムを使用しているお客様に対しては、移行を計画することをお薦めします。詳細は、サポートに問い合せてください。

第2章 概要

Kapow Katalyst製品ファミリへようこそ。ドキュメントのこの入門部分の内容は、次のとおりです。

- Kapow Katalystの目的および機能の概要、そのコンポーネントの簡単な紹介、およびドキュメントの残りの部分の手がかりとなるガイドライン。
- 一般的な用語で構成についても説明するインストール・ガイド。
- 製品の重要な変更を強調し、新しいバージョンへの切替えに役立つ、5章「以前のバージョンのユーザーに対する注意事項」。
- Kapow Katalystの使用開始に役立つ多数の入門ビデオ・チュートリアル。
- 特定のトピックおよびテクニックをさらに詳しく説明する上級ビデオ・チュートリアル。
- 様々な種類のKapow Katalyst ライセンス・キーの概要。
- Kapow Compute Units (KCU)。

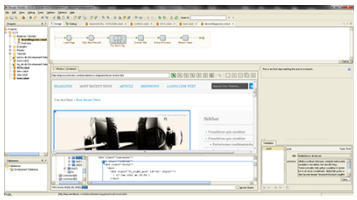
以前のバージョンのKapow Katalystからアップグレードしている場合は、5章「以前のバージョンのユーザーに対する注意事項」を読むことをお薦めします。

サポートまたはヘルプを取得する他の方法を探している場合は、主要なネットベースのリソースに対するリスト参照を確認してください。

第3章 概要

Kapow Katalystは、アプリケーション統合およびプロセス自動化のためのプラットフォームです。接続するように作成されなかったアプリケーションを統合し、このような異機種間システム、自社システムを使用したクラウド/SaaSアプリケーション、最新のWebアプリケーションを含むレガシー・システム、パートナWebサイトを含むバックオフィス・システム全体でプロセスを自動化できます。

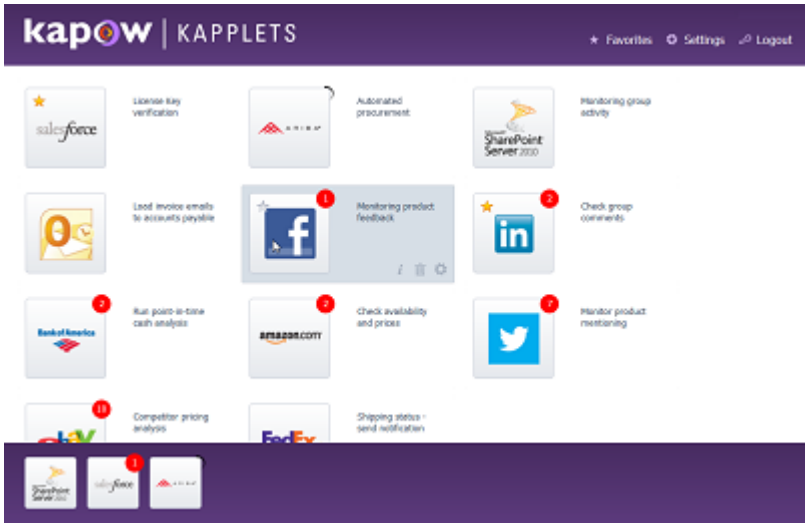
ビジュアル・エディタDesign Studio（図を参照）を使用して、統合して自動化ワークフローを作成するアプリケーションおよびデータ・ソースをクリック・スルーします。



Kapow Katalyst ビジュアル・エディタ: Design Studio

Kapow Katalystでは、これらのワークフローはロボットと呼ばれます。ロボットを作成すると、それらを統合したとおりにアプリケーションを自由にナビゲートできます。アプリケーションへのログイン、ページのデータ部分の抽出、フォームまたは検索ボックスへのデータの入力、メニュー選択、複数ページのスクロールを実行できます。ロボットは、データベース、ファイル、APIおよびWebサービスへのアクセス、あるアプリケーションからのデータのエクスポートおよび別のアプリケーションへのロード、その間の必要に応じたデータの変換も実行できます。

作成したロボットはManagement Console内のリポジトリにアップロードされます。ここから、RoboServerでバッチ実行用にスケジュールするか、あるいはJavaまたはC# APIを介して要求に応じて実行でき、ロボットがリポジトリに追加されるとすぐに使用可能になるRESTサービスを調整でき、Kappletsと呼ばれる特殊な目的のエンド・ユーザーWebアプリケーションとして公開できます。



Kapplets

Management Consoleには、ロード・バランシング、フェイルオーバー、RoboServerの状態の監視、ユーザー・ロールおよび権限の管理の役割もあります。

次のステップ

まだ実行していない場合は、すぐにKapow Katalystをインストールする必要があります。

一連のチュートリアルに従って開始することをお薦めします。チュートリアルでは、最初のロボットの作成、それらのManagement Consoleへのデプロイ、およびKappletsとしての公開を順に学習します。

さらに詳細な範囲については、関連アプリケーションのユーザーズ・ガイドの各項またはトピックを参照してください。

- Design Studio
- Management Console
- Kapplets
- JavaおよびC# API

第4章 インストール・ガイド

Kapow Katalystには、すべてのソフトウェアをインストールする1つの統合インストーラがあり、ソフトウェアのそれぞれ異なる部分がサーバー上およびクライアント・コンピュータ上で実行される場合でもすべてインストールします。特定のインストールのロール(サーバー、クライアントなど)は、ソフトウェアの基本インストール後に行われるステップで決定されます。

以降の項では次の内容を説明します。

- インタラクティブまたはサイレント(ユーザー操作なし)にKapow Katalystをインストールする方法。
- Kapow Katalystを起動できるように、ライセンス情報を入力する方法。
- Kapow Katalystの構成方法。
- コンピュータの再起動時に、Kapow Katalystサーバー・アプリケーションを自動的に起動するように設定する方法。
- サポートされているプラットフォームおよびシステム要件。

以前のバージョンからのアップグレード

安定性のために、異なるバージョンのKapow Katalystを同じコンピュータ上に相互に影響を与えることなく一緒にインストールできます(同時に実行する場合に、Management Consoleに対して異なるポート番号を使用するように構成する必要がある場合を除く)。つまり、新しいバージョンをインストールしてそれに慣れるまでの間、古いバージョンで今までどおり日常業務を遂行できます。

このことはまた、アップロードしたロボット、実行スケジュールなどの重要なデータが、同じコンピュータ上にインストールした異なるバージョン間で共有されないということでもあります。したがって、ある時点で、古いKapow Katalystインストールのこのデータを新しいインストールにコピーする必要があります。(5章「以前のバージョンのユーザーに対する注意事項」で説明されているDesign Studioプロジェクトの問題と似ていますが別の問題です。)

このデータをあるバージョンのManagement Consoleから別のバージョンにコピーするには、古いインストールのバックアップを作成し、それを新しいインストールに復元します。このプロセスは、ここで説明しています。バックアップの作成は、Kapow Katalystの様々なバージョンで多少異なる方法で実行されるため、関連する古いバージョン用のオンライン・ドキュメントを確認して、バックアップの作成方法を学習してください。ドキュメントはそれぞれ次の場所で入手します。9.0 8.2 8.1 8.0 7.2 7.1 7.0

Management ConsoleをTomcat Webサーバー(次を参照)で実行している場合は、新しいバージョンのKapow Katalyst用にTomcat構成を変更する前に、必ずバックアップを作成してください。

TomcatでのManagement Console

Management ConsoleはKapow Katalystの主要部分であり、RoboServerアプリケーション内の埋込みコンポーネントとしてデフォルトで実行されます。ただし、スタンドアロンのTomcat Webサーバーに通常のWebアプリケーションとしてインストールすることもできます。これを実行する(または実行するよう切り替えることを後で決定した)場合は、20章「TomcatでのManagement Console」の項の説明に従って、多数の追加手動ステップを実行する必要があります。

RoboServerアプリケーションで埋込みを実行する場合、Management Consoleでは埋込みDerbyデータベースが使用されます。顧客側では、埋込みManagement Consoleで100スケジューラまたは500ロボットを超える場合に、パフォーマンスが低下することを確認しています。この状況が発生した場合は、別のデータベースを選択できるTomcatにManagement Consoleをデプロイすることをお薦めします。

Tomcat WebサーバーにManagement Consoleをインストールしている、以前のバージョンのKapow Katalystからアップグレードしている場合は、新しいバージョンをTomcatにインストールする前に必ずバックアップを作成してください(前述を参照)。

Kapow Katalystのインストール

この項では、Windowsでのインタラクティブ・インストールとサイレント・インストール、およびLinuxでの通常インストールとヘッドレス・インストールについて説明します。この項ではさらに、インストール時に作成される最も重要なフォルダもリストします。

WindowsまたはLinuxにインストールする前に、32ビットまたは64ビットのいずれのバージョンをインストールするかを決定する必要があります。考慮する主要な問題は次のとおりです。

- オペレーティング・システムでいずれかのバージョンのみをサポートできること。64ビットのWindowsでは両方がサポートされており、Linuxではインストールと一致する一方を使用することをお薦めします。
- 64ビット・バージョンは主に、多くのRAMを使用するRoboServerを実行するサーバーで有効です。(インストール後に、「RAM割当の変更」の項の説明に従って、RAMの許容量を構成する必要があります。)

同じコンピュータにKapow Katalystの32ビットと64ビットの両方のバージョンをインストールでき、これらは、ユーザー側で特別な作業をしなくても構成およびロボットを共有します。ただし、いずれかのKapow Katalystアプリケーションを起動しようとするたびに、注意して正しいバージョン(32ビットまたは64ビット)を起動する必要があります。さらに、Linuxでは、32ビット・バージョンと64ビット・バージョンに対して、必ず異なるインストール・フォルダを選択してください(Windowsではこれは自動です)。

Windowsでのインストール

注記

Windowsにインストールするには管理者権限が必要です。

インストーラのダウンロード方法の指示を受信します。**KapowKatalyst_9.3.0.msi**(32ビット)または**KapowKatalyst_9.3.0_x64.msi**(64ビット)のファイルをダウンロードしてハード・ディスクに保存します。ダウンロードの完了後、ファイルを実行してインストールを開始し、インストーラの手順およびダイアログに従います。次に、「ライセンス情報の入力」の説明に従ってライセンス情報の入力に進みます。

Windowsでのサイレント・インストール

サイレント・インストーラは、ユーザー操作なしで実行できます。これは、たとえば、インストール・プロセスをスクリプトで自動化する必要がある場合に便利です。Kapow Katalystのサイレント・インストールを実行する場合は、管理権限を使用して次のコマンドを実行する必要があります。

```
msiexec /qn /i KapowKatalyst_9.3.0.msi
```

この結果、デフォルトの場所にプログラムがインストールされます。別の場所を指定する場合は、かわりに次を実行します。

```
msiexec /qn /i KapowKatalyst_9.3.0.msi INSTALLDIR="dir"
```

ここで、**dir**はインストール先の場所です。インストール後は、「ライセンス情報の入力」の説明に従って、さらにライセンス情報の入力に進む必要があります。

Linuxでのインストール

注記

Linuxでは権限が与えられていないユーザーとしてインストールできます。

適切なファイルのダウンロード方法の指示を受信します。インストール用の**.tar.gz**ファイルをダウンロードしてハード・ディスクに保存します。このファイルの名前は、Linux 32ビット・バージョンの場合は**KapowKatalyst_9.3.0.tar.gz**、Linux 64ビット・バージョンの場合

は**KapowKatalyst_9.3.0_x64.tar.gz**です。次の指示はLinux 32ビット・バージョンを表していますが、もう一方もファイル名以外は同じです。

ダウンロードの完了後、ファイルの内容を解凍することによってインストールが実行されます。ほとんどのLinuxディストリビューションで、これは、ファイルを右クリックして、適切な解凍オプションを選択することによって実行できます。ファイルは、次のようにコマンドラインから解凍することもできます。

```
$tar xzf KapowKatalyst_9.3.0.tar.gz
```

または、ファイルを特定のディレクトリに解凍するには、次のコマンドを使用できます。

```
$tar xzf KapowKatalyst_9.3.0.tar.gz -C /destination_directory
```

ファイルの解凍後、「ライセンス情報の入力」の説明に従ってライセンス情報の入力に進みます。

Kapow Katalystの重要なフォルダ

理解しておく必要があるインストール内のフォルダおよびファイルがいくつかあります。

インストール・フォルダ

インストール・フォルダは、Kapow Katalystがインストールされるフォルダです。Windowsでは、インストール・フォルダのデフォルトは次に設定されます。

```
C:\Program Files\Kapow Katalyst 9.3.0
```

Linuxでは、アーカイブを解凍したディレクトリ内のKapowKatalyst_9.3.0という名前のディレクトリになります。

インストール・フォルダには、次の重要なフォルダが含まれています。

bin Kapow Katalystに関するすべての実行可能プログラムが含まれています。

API Kapow Katalyst統合APIに関するファイルおよびドキュメントが含まれています。

Plugins Kapow Katalystプラグインが含まれています。

lib/jdbc インストール済のJDBCデータベース・ドライバが含まれています。これらは、Kapow Katalystアプリケーションで常に使用可能なドライバです。ただし、通常は、ここの説明に従ってJDBCドライバを管理する必要があります。

プロジェクト・フォルダ

プロジェクト・フォルダには、「Design Studioユーザーズ・ガイド」で説明されているように、ロボットおよびタイプのライブラリが含まれています。プロジェクト・フォルダの場所は、「Kapow Katalystの構成」の説明に従ってSettingsアプリケーションで構成できます。Windowsでは、デフォルトの場所は、Windowsのバージョンに応じて次のいずれかのようになります。

```
C:\Documents and Settings\username\My Documents\My Robots\9.3.0
```

```
C:\Users\username\Documents\My Robots\9.3.0
```

Linuxでは、プロジェクト・フォルダのデフォルトの場所は、次のとおりです。

```
~/KapowKatalyst/9.3.0
```

プロジェクト・フォルダには、**Library**という名前の単一のフォルダが含まれている必要があります。

アプリケーション・データ・フォルダ

アプリケーション・データ・フォルダには、Kapow Katalyst専用のファイルが含まれていますが、これは同じコンピュータのユーザーごとに異なります。Windowsでは、アプリケーション・データ・フォルダは(Windowsのバージョンに応じて)次のようになります。

```
C:\Documents and Settings\username\Local Settings\Application Data\
KapowKatalyst\9.3.0
```

```
C:\Users\username\AppData\Local\KapowKatalyst\9.3.0
```

Linuxでは、アプリケーション・データ・フォルダは次のようになります。

```
~/.KapowKatalyst/9.3.0
```

アプリケーション・データ・フォルダの場所を変更するには、インストール・フォルダの**bin**フォルダにある**common.conf**ファイルを編集できます。次の2行を追加する必要があります。

```
wrapper.java.additional.10=-Dkapow.applicationDataFolder=
"Folder containing Configuration folder"
```

```
wrapper.java.additional.10.stripquotes=TRUE
```

Kapow Katalystを実行しているユーザーに、フォルダに対する読取りおよび書込みアクセス権限があることを確認してください。

通常の状況では、このフォルダ内のファイルまたはフォルダは直接変更または削除せずに、かわりにGUIツールを使用してください。アプリケーション・データ・フォルダには、次の重要なフォルダが含まれています。

Certificates Kapow Katalystで認識されているHTTPS証明書が含まれています。詳細は、「証明書」の項を参照してください。

Configuration 構成ファイルが含まれ、これは、「Kapow Katalystの構成」の項の説明に従って保守されます。

Data Management Consoleで使用される埋込みDerbyデータベースが含まれています(Tomcat WebサーバーにWebアプリケーションとしてインストールされた場合を除く)。

DemoDatabase Management Console初心者用チュートリアルで使用される開発データベースが含まれ、これは、Kapow Katalystの導入時に小さいプロジェクトで使用することもできます。

Logs ログ・ファイルが含まれています。

ライセンス情報の入力

基本インストール後の最初のステップは、ライセンス情報の入力です。Management ConsoleおよびDesign Studioでライセンス・キーを入力する場所の情報は、それぞれ以降のページを参照してくだ

さい。RoboServerでは、Management Consoleから必要なライセンス情報が自動的に受信されるため、ライセンス情報を入力する必要はありません。

Management Console（ライセンス・サーバー）

Management Consoleにライセンス情報を入力する前に、起動する必要があります。

Windows　「スタート」メニューのManagement Consoleの起動を使用します。

Linux　コマンドラインからManagement Consoleを起動します。これはRoboServerプログラムの一部であり、インストールの下の**bin**ディレクトリにあります(「Kapow Katalystの重要なフォルダ」を参照)。次を使用して起動できます。

```
$./RoboServer -p 50000 -MC
```

(./の部分を除いた場合、これはWindowsでも実行できます。)

自動起動　別の方法として、「サーバーの自動起動」の説明に従って、後でManagement Consoleの自動起動を設定する予定の場合は、Management Consoleを手動で起動するかわりに、ここでその実行を選択できます。

Management Consoleを起動した後は(起動方法に関係なく)、そのGUIをブラウザで開く必要があります。Windowsでは、「スタート」メニューの「Management Console」項目を使用して実行できます。すべてのプラットフォームで、ブラウザを開いてhttp://localhost:50080/に移動できます。ライセンス条項を受け入れて、ライセンス・キーを含むライセンス情報を入力します。

Design Studio

Design Studioがまだ起動していない場合は、次のように起動します。

Windows　「スタート」メニューの「Design Studio」項目を使用します。

Linux　インストールの下の**bin**ディレクトリ(「Kapow Katalystの重要なフォルダ」を参照)にある**DesignStudio**プログラムを起動して、次のようにコマンドラインからDesign Studioを起動します。

```
$./DesignStudio
```

(./の部分を除いた場合、これはWindowsでも実行できます。)

次に、Design Studioにライセンスが必要であるため、次のようなダイアログが表示されます。

Kapow管理者またはシステム管理者(あるいはその両方)から提供された情報に応じて、3つの選択肢があります。

ライセンス・サーバー　管理者から、すべてのDesign Studioユーザーのライセンスを管理する中央のライセンス・サーバー(Management Console)へのURLを提供された場合は、ライセンス・サーバーを選択し、そのサーバーに対するURLと資格証明を入力するのみです。Design Studioを使用するには、ライセンス・サーバーが実行中で、使用可能なDesign Studioシートがある必要があります。

開発者ライセンス　開発者ライセンスは、Management ConsoleとDesign Studioの両方に対する結合ライセンス・キーです。ローカルのManagement Consoleでライセンス・キーを入力するか、またはDesign Studioでダイアログを使用できます。Design Studioを起動すると必ず、Management Consoleも自動的に起動します。

試用ライセンス　試用ライセンスは、開発者ライセンスと同様に機能しますが、試用/デモ目的の短期使用を対象としています。

Design Studioでライセンス・キーを入力する方法の詳細な例は、次のビデオを参照してください。

通常、Design Studioでは、最初に起動したときのみダイアログが表示されます。ただし、ライセンス・サーバーが通信できない場合(たとえば移動したか、実行中ではないため)は、ダイアログが再度表示されます。

Kapow Katalystの構成

Kapow Katalystのインストール後は、ニーズにあわせてインストールを構成する必要があります。一部の構成は、ガイドの18章「RoboServerの構成」の説明に従ってSettingsアプリケーションを使用して実行します。

その他の主にKapow Katalystシステムの管理者に関係する構成は、Management Consoleを介して、より具体的には「サーバー」タブ、「プロジェクト」タブおよび「オプション」タブを介して実行します。「サーバー」タブで、必要なRoboServerおよびクラスタを設定することが特に重要です。次の項では、これらのRoboServer (およびManagement Console)を自動的に起動する方法について説明します。

サーバーの自動起動

インストールにサーバー機能が含まれている場合は、必要なサーバーを自動的に起動するように構成する場合があります。これが関係しない1つの例は、たとえば、IT部門が設定したManagement Consoleに接続している間のみDesign Studioを実行する必要がある場合です。

サーバー機能という場合は、RoboServerとManagement Console (ライセンス・サーバー)を意味します。実際に、これらの2つの機能は、同じサーバー・プログラムである**RoboServer**によって、それが起動時に渡された引数に応じて提供されます。

「RoboServerの起動」の項で**RoboServer**プログラムに対するコマンドライン引数について詳細に説明しており、ここでは繰返し説明はしません。補足として、**-service**引数はRoboServerプログラムでロボットを実行できるようにする引数です。同様に、**-MC**引数は、Management Console機能を可能にする引数です(「Management Consoleの起動」も参照)。

これを前提として、**RoboServer**を自動的に起動する方法(あらゆる引数)を詳細に説明します。これは、WindowsとLinuxでかなり異なります。

Windowsでのサーバーの起動

RoboServerをWindows上で自動的に起動するには、それをWindowsサービスとして追加する必要があります。Kapow Katalystインストールに含まれている**ServiceInstaller.exe**を使用して、Windowsサービスを追加および削除する方法を示します。

Windowsサービスの追加

RoboServerをサービスとして実行するには、最初に**ServiceInstaller.exe**プログラムを使用してインストールする必要があります。次は、このプログラムに対するコマンドライン引数の概要を示す一般的な例です(ここでは複数行で表示されていますが、これは1行のコマンドです)。

```
ServiceInstaller.exe -i RoboServer.conf
wrapper.ntservice.account=Account
wrapper.ntservice.password.prompt=true
wrapper.ntservice.name=Service-name
wrapper.ntservice.starttype=Start-method
wrapper.syslog.loglevel=INFO wrapper.app.parameter.1="First-
Argument" wrapper.app.parameter.2="Second-argument"
```

wrapper.ntservice.account:	RoboServerを実行する必要があるユーザーのアカウント。Kapow Katalystでは、構成はユーザーのディレクトリに格納されており、正しい構成が設定されているユーザーを選択することが重要です。 RoboServerをドメイン・ユーザーで実行する必要がある場合は、アカウントを**domain\account**の形式で入力する必要があります。 RoboServerを一般ユーザーで実行する必要がある場合は、アカウントを**.\account**の形式で入力する必要があります。
wrapper.ntservice.password.prompt:	値trueは、アカウント用のパスワードの入力をユーザーに求めます。パスワードをコマンドラインで入力する場合は、かわりに**wrapper.ntservice.password=whatever-the-password-is**を使用します。
wrapper.ntservice.name:	インストールするサービスの名前。サービスの名前には空白を含めることはできません。
wrapper.ntservice.starttype:	システムの再起動時に、サービスを自動的に開始する場合は**AUTO_START**。 しばらくしてからサービスを開始する場合は**DELAY_START**。(Windows XPおよび2003ではサポートされていません。) サービスを手動で開始する場合は**DEMAND_START**。
wrapper.syslog.loglevel:	RoboServerからevent logにコンソール出力がリダイレクトされます。
wrapper.app.parameter.*:	RoboServerの引数。必要に応じた数を入力できます。

サービスがインストールされると、ユーザーに「サービスとしてログオン」権限が付与されます。サービスの開始に失敗した場合は、**gpedit.msc**を開いて権限が付与されていることを確認し、(Windows 7の場合)**Computer Configuration -> Windows Settings-> Security Settings -> Local Policies -> User Rights Assignment -> Log on as a service**に移動して、ユーザーを追加します。

例

この例では、ユーザー**bob**で実行される**RoboServer9.3.0**(注意: 空白なし)というサービスをインストールします。スクリプトの実行時に、ユーザーはパスワードの入力を求められます。RoboServerの引数は**-p 50000**です(ロボットを実行できることを意味します)。

```
ServiceInstaller.exe -i RoboServer.conf
wrapper.ntservice.account=.\bob
wrapper.ntservice.password.prompt=true
wrapper.ntservice.name="RoboServer9.3.0"
wrapper.ntservice.starttype=AUTO_START
wrapper.syslog.loglevel=INFO wrapper.app.parameter.1="-p"
wrapper.app.parameter.2="50000"
```

例

この例では、ドメイン**mydomain**のユーザー**bob**で実行される**RoboServer9.3.0**(注意:空白なし)というサービスをインストールします。RoboServerの引数は**-p 50000 -MC**です(ロボットとManagement Consoleの両方を実行できることを意味します)。

```
ServiceInstaller.exe -i RoboServer.conf
wrapper.ntservice.account=mydomain
\bob wrapper.ntservice.password=secret
```

```
wrapper.ntservice.name="RoboServer9.3.0"
wrapper.ntservice.starttype=AUTO_START
wrapper.syslog.loglevel=INFO wrapper.app.parameter.1="-p"
wrapper.app.parameter.2="50000" wrapper.app.parameter.3="-MC"
```

Windowsサービスの削除

サービスをアンインストールするには、次を実行できます。

```
ServiceInstaller.exe -r RoboServer.conf
wrapper.ntservice.name=Service-name
```

wrapper.ntservice.name: 削除するサービスの名前

例

この例では、**RoboServer9.3.0**というサービスを削除します。

```
ServiceInstaller.exe -r RoboServer.conf
wrapper.ntservice.name="RoboServer9.3.0"
```

Linuxでのサーバーの起動

Linuxで**RoboServer**を自動的に起動する最も単純な方法は、**crontab**を使用することです。次のコマンドを使用して、必要なユーザーに対するLinuxのスケジュール済ジョブのリストを作成または編集します(**root**またはその特定のユーザーのいずれかで実行すると最も簡単です)。

```
crontab -u someUser -e
```

スケジュール済ジョブのリストに、たとえば次を追加します。

```
@reboot $HOME/KapowKatalyst_9.3.0/bin/RoboServer -p 50000
```

または

```
@reboot $HOME/KapowKatalyst_9.3.0/bin/RoboServer -p 50000 -MC
```

このように、**RoboServer**プログラムは、再起動時に、指定したコマンドライン引数で起動します。実際のインストール・フォルダの下にある**bin**ディレクトリを識別する必要があることに注意してください。

サポートされているプラットフォーム

これは、サポートされているプラットフォームのリストです。リストされているバージョンは、Kapow Katalystがテストされたバージョンです。これは、他のバージョンで動作しないという意味ではありません。リストされていないバージョンで動作するかわからない場合、詳細はKapow Softwareに問い合せてください。

IDEプラットフォーム	
Windows	Windows Server 2003、Windows Server 2008、Windows Server 2008 R2、Windows XP、Windows Vista、Windows 7
Linux	Red Hat Enterprise Linux 6.0、Ubuntu 12.04 LTS
サーバー・プラットフォーム	
Windows	Windows Server 2003、Windows Server 2008、Windows Server 2008 R2、Windows XP、Windows Vista、Windows 7
Linux	Red Hat Enterprise Linux 5.6または6.0、CentOS 5.6、Debian 7.2

データベース	
Oracle*	バージョン11g
IBM DB2	9.7
Microsoft SQL Server	バージョン2005、2008および2008 R2
Sybase*	jConnect for JDBC 3.0バージョン6.0.5ドライバを使用したAdaptive Server Enterprise 15.0
MySQL*	バージョン5.x。RoboManagerはバージョン5.5以上ではサポートされません。
Postgres 9.2	publicスキーマを使用しない場合は、Postgresにアクセスするユーザー・アカウントに対してデフォルト・スキーマを設定する必要があります(ALTER USER <username> SET search_path to <schema>)。
HBase	HBaseテーブルに格納ステップはHBaseバージョン0.90.6で使用できます。
API	
Java	Java 1.5以上。Oracle WebLogic 11、IBM WebSphereおよびJBOSSなどのJ2EEサーバーを含みます。
JMX	Java 1.6
.NET	C#、.NETバージョン4.0
Management Consoleポータル	
Tomcat	バージョン5.5、6.0および7.0。最低Java 1.7。

* Oracle、SybaseまたはMySQLにデータを保存すると、データが失われる場合があることに注意してください。Oracleでは、空の文字列がnullに変換されます。Sybaseでは、空の文字列が" "(1つの空白)に変換されます。MySQLでは、日付の保存時にミリ秒の精度が失われます。詳細は、「ObjectKeyに関する注意」を参照してください。

システム要件

次の表は、様々なプラットフォームに対するシステム仕様を具体的に示しています。要件はアプリケーションに依存する場合があるため、これらは絶対的な数値としてではなくガイドラインとしてのみ使用してください。複雑なクリッピング・ソリューションでは、単純な収集ソリューションよりさらに高い能力を必要とする場合があります。サーバーに関する推奨は、1つのサーバーを対象としています。特定のアプリケーションに使用するサーバーの数(クラスタのサイズ)は完全に別の問題であり、他の場所で説明する方法を使用して見積る必要があります。

IDE要件:

	最低	推奨
Windows	Intel Core Duo 1.8 GHzのCPU (または同等のAMD)、2GBのRAM、260MBの空きディスク容量	Intel Core Duo 2.66 GHzのCPU (または同等のAMD)、4GBのRAM、260MBの空きディスク容量
Linux	Intel Core Duo 1.8 GHzのCPU (または同等のAMD)、2GBのRAM、260MBの空きディスク容量	Intel Core Duo 2.66 GHzのCPU (または同等のAMD)、4GBのRAM、260MBの空きディスク容量

サーバー要件:

	最低	推奨
Windows (32ビット)	Intel Xeon L5520 CPU (または同等のAMD Opteron)、CPUコアごとに1GBのメモリー	Intel Xeon X5680/X5677 CPU (または同等のAMD Opteron)、CPUコアごとに1.5GB

	最低	推奨
	+ OS用に512MB、500MBの空きディスク容量	のメモリー + OS用に1GB、500MBの空きディスク容量
Linux (32ビット)	Intel Xeon L5520 CPU (または同等のAMD Opteron)、CPUコアごとに1GBのメモリー + OS用に512MB、500MBの空きディスク容量	Intel Xeon X5680/X5677 CPU (または同等のAMD Opteron)、CPUコアごとに1.5GBのメモリー + OS用に1GB、500MBの空きディスク容量
Windows (64ビット)	Intel Xeon L5520 CPU (または同等のAMD Opteron)、CPUコアごとに1.5GBのメモリー + OS用に1GB、500MBの空きディスク容量	Intel Xeon X5680/X5677 CPU (または同等のAMD Opteron)、CPUコアごとに2GBのメモリー + OS用に1.5GB、500MBの空きディスク容量
Linux (64ビット)	Intel Xeon L5520 CPU (または同等のAMD Opteron)、CPUコアごとに1.5GBのメモリー + OS用に1GB、500MBの空きディスク容量	Intel Xeon X5680/X5677 CPU (または同等のAMD Opteron)、CPUコアごとに2GBのメモリー + OS用に1.5GB、500MBの空きディスク容量

リアルタイム・データ: ユーザーが結果をリアルタイムで待機しているソリューションがある場合、通常はCPU速度がボトルネックであり、ハードウェア・プラットフォームで使用可能な最速のCPUを購入する必要があります。

専用ハードウェア: パフォーマンスを最大にするために、RoboServerは常に専用ハードウェアで実行することをお薦めします。つまり、RoboServerと同じハードウェア上で、データベース・サーバーや他のサービスを実行しないでください。

オペレーティング・システム: Kapowでは、Javaアプリケーションの実行時の処理速度が現在はWindowsよりLinuxオペレーティング・システムの方が速いため、Linuxオペレーティング・システムを使用することをお薦めします。シナリオによっては、Linux使用時に最大33%高いスループットを測定しました。

第5章 以前のバージョンのユーザーに対する注意事項

安定性のために、異なるバージョンのKapow Katalystを同じコンピュータ上に相互に影響を与えることなく一緒にインストールできます。つまり、新しいバージョンをインストールしてそれに慣れるまでの間、古いバージョンで今までどおり日常業務を遂行できます。このように作業している間に、たいていの場合、古いバージョンのデフォルトのロボット・プロジェクトを新しいバージョンのDesign Studioでも開くことになります。ここで1つ注意が必要なのは、ロボットを新しいバージョンで保存すると、古いバージョンのDesign Studioを使用してそのロボットを開くことができなくなることです。したがって、かわりにプロジェクトのコピーを作成して、新しいバージョンのDesign Studioの習得中はそのコピーを使用することをお薦めします。

次に、アップグレード時に考慮する必要がある、Kapow Katalystの各バージョン間の最も重要な相違点をリストします。

バージョン9.2のKapow Katalystでは、Microsoft Excel™ファイルからデータを抽出するための新機能が導入されています。古い機能のかわりにこれらを使用する方法の詳細は、「9.2の新しいExcel機能」の項を参照してください。

バージョン9.1のKapow Katalystには、RoboRunnerアプリケーションが含まれなくなりました。かわりにManagement Consoleを使用する方法の詳細は、「9.1におけるRoboRunnerのサポート停止」の項を参照してください。

バージョン8.2のKapow Katalystでは、いくつかの小規模な変更が加えられ、まれに古いロボットの実行に影響を与える場合があります。これらの詳細およびその他の変更は、「8.2の変更点」の項を参照してください。

バージョン8.1のKapow Katalystでは、モデルとオブジェクトの概念がタイプと変数という概念に置き換えられました。この重要な変更の詳細は、「8.1における新しい変数およびタイプのシステム」の項を参照してください。

バージョン8.0のKapow Katalystでは、ロボットの制御フローおよびエラー処理を表示する新しい方法が導入されました。これは、「8.0の新しい制御フローおよびエラー処理」で説明するように、以前のバージョンで作成されたロボットも含め、すべてのロボットに影響を与えます。

バージョン7.2のKapow Katalystでは、大幅に使いやすくなった新しい記憶域システムが導入されました。以前のバージョンからアップグレードしているユーザーは、自分のペースで新しい記憶域システムに切り替えることができます。詳細は、「7.2の新しい記憶域システム」を参照してください。

バージョン7.1のKapow Katalystでは、Microsoft VistaおよびWindows 7のインストール・ルールに準拠するために特定のファイルの場所が変更されました。このリリースからは、Kapow Katalystに関連するファイルは、インストール・フォルダ、プロジェクト・フォルダおよびアプリケーション・データ・フォルダの3つのフォルダに分けられています。これらのフォルダについては、ここを参照してください。これらの変更によって、Design Studioの実行のためにWindowsの(「Program Files」フォルダに対する書込みアクセス権限がある)パワー・ユーザーになる必要がなくなりました。

9.2の新しいExcel機能

9.2でExcelが処理される方法は、以前のバージョンのDesign StudioでExcelが処理されていた方法と基本的に異なるため、ロボットにExcelが含まれている場合は、これらを以前のバージョンから9.2に簡単にアップグレードする方法はないことを知っておくことが重要です。かなりの量の再作成を伴う可能性があります。これは、以前のバージョンのExcelは、ロード時にHTMLに変換され、後続の抽出などがHTMLの場合と同様に実行されていましたが、9.2では、Excelは新しいスプレッドシート・ページ・ビューにロードされ、ここからデータ抽出用の専用ステップが使用されるためです。

ExcelをHTMLに変換というレガシー・オプション

古いロボットは、その動作を可能にする、ExcelをHTMLに変換というレガシー・オプションがあるため今までどおり動作し、このオプションがデフォルトでオンになっています。新しいExcel機能を使用するためにロボットをアップグレードする場合は、このオプションをオフにする必要があります。これは、ロボット構成で行い、構成オプション・ボタンをクリックし、構成オプション・ダイアログのレガシー・タブで、オプションの選択を解除します。しかし、この変更は、Excel上で処理する抽出またはループを実行しない場合を除き、ロボットを破損することになります。ロボット全体を調べ、ステップの一部を新しい専用のExcelステップに置換する必要があり、そこではロボットは常に、ロードしたExcelドキュメントのデータを処理します。このため、新しいExcel機能を使用するために古いロボットをアップグレードする相応な理由がある場合を除いて、アップグレードしないことをお薦めします。これは、他の9.2の機能を使用するためにロボットをアップグレードできないのではなく、単にレガシー・オプションをオンのままにしておく必要があるということです。

注意: ExcelをHTMLに変換オプションは、古いロボットにExcelからの抽出ステップまたは他の方法でExcelが含まれていなかった場合でも常に設定されます。これを覚えておくことが重要であるのは、古いロボットを9.2にロードして、これを出発点に新しいロボットを作成する場合であり、その理由は、ExcelをHTMLに変換オプションをオフにしないと、新しいExcel機能を利用できないためです。

Excelからの抽出ステップ・アクションの非推奨化

Excelからの抽出ステップ・アクションは非推奨であり、新しいロボットでは使用できず、かわりに、ページの作成ステップ・アクションを使用する必要があります。この方法は、「変数からExcelページをロードする方法」を参照してください。

9.1におけるRoboRunnerのサポート停止

以前に通知したとおり、RoboRunnerアプリケーションはサポートされなくなりました。このガイドでは、かわりの処理方法を示します。

特定の間隔でロボットをスケジュールするタスクは現在、Webベースのアプリケーションである Management Consoleで処理されます。様々な各RoboRunner実行シナリオ、およびこれらのタスクを Management Consoleを使用して処理する方法を説明しています。後で細かく参照することになるため、先に進む前に、11章「Management Consoleユーザーズ・ガイド」を読んでおくことをお薦めします。

ロボットはManagement Consoleでスケジュールされますが、実際に実行されるのはRoboServer上です。RoboServerの詳細は、ここを参照してください。RoboServerは、Control CenterまたはManagement Consoleのダッシュボードを使用して監視できます。

スケジュールの概念

RoboRunnerにはロボットを実行する方法が2通りあります。

-fileオプション	ロボットがRoboRunnerを使用して個別にスケジュールされていた場合は、ここで説明しているとおりに、各ロボットに対して希望の時間に実行されるように別々のスケジュールを作成する必要があります。ロボットをこのようにスケジュールすると、現在は入力オブジェクトを使用してロボットをパラメータ化できるため、さらに利点があります。 同時に実行する必要があるロボットが複数ある場合、スケジュールを作成するには、Management Consoleの「ロボット」タブでそれらのロボットを選択し、右クリックして、コンテキスト・メニューからスケジュールの作成を選択します。
-dirオプション	RoboRunnerの**-dir**オプションを使用してディレクトリ内の複数のロボットを実行していた場合は、ロボット・グループ・ジョブを使用します。これを使用すると、ロボットの名前に基づいて、指定した時間に多数のロボットをスケジュールできます。

つまり、一緒に実行する必要があるロボットが、次のいずれかになるようにする必要があります。

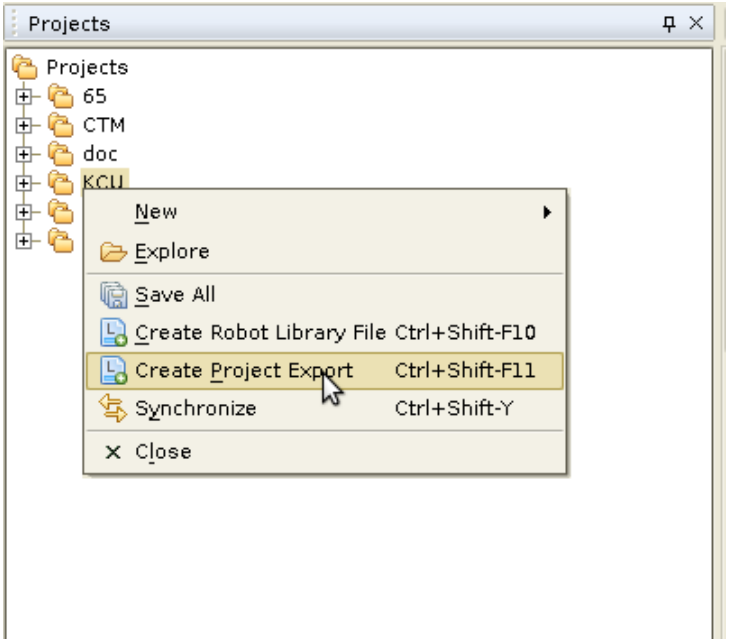
- batch1_bestbuy.robot、batch1_newegg.robotなど、共通の名前接頭辞または接尾辞を設定します。または、
- 独自のプロジェクト内に入れます。この結果、.*引数を含むパターン・ベースの基準を使用できます。

使用する方法を後で変更すると多くの作業が必要となる場合があるため、ロボットのアップロードを開始する前に、2つの方法のいずれを使用するかを選択する必要があります。

重要: FileStorageEnvironmentまたはFileMessageEnvironmentを使用するロボットは、Management Consoleでサポートされていません。ファイルの書込みステップ・アクションを使用するようにロボットを再作成するか、またはデータをデータベースに格納し、格納したデータをファイルに変換するスクリプトまたはプログラムを作成できます。

Management Consoleへの一括デプロイ

ロボットの構成が完了すると、それらをManagement Consoleにデプロイする準備が整います。これは、Design Studio内からプロジェクト・エクスポートを作成することによって実行します。これを行うには、プロジェクト・ビューでプロジェクト・フォルダを右クリックし、コンテキスト・メニューからプロジェクト・エクスポートの作成を選択します。



次に、プロジェクト・エクスポートを保存する場所を選択し、「OK」ボタンをクリックします。

ロボットのいずれかがバージョン8.1より前に開発された場合は、最初に、まだ存在している可能性がある**.model**ファイルを展開する必要があります。タイプとモデルの詳細は、ここを参照してください。また、プロジェクト内のロボットの名前は、別々のサブフォルダに格納されている場合でも一意である必要があります。名前の競合がある場合は、ダイアログで通知されます。

プロジェクト・エクスポートを作成した後は、Management Consoleを開き、「オプション」タブ(「管理」 =>「オプション」)に移動してプロジェクト・インポートを実行します。

ロボットのスケジュール

ロボットをアップロードした後は、ロボットを実行するスケジュールを作成できます。

ロボットで古いデータベース記憶域システム(データベース記憶域/メッセージ/情報環境)を使用している場合は、新規スケジュール・ダイアログのレガシー・タブでオプションを構成する必要がありま

す。ただし、これらのロボットは、7.2で導入された新しい記憶域方法を使用するように再作成することをお薦めします。新しい記憶域システムの詳細は、ここを参照してください。

新しい記憶域システムには、改善された新しいデータベース・ロギング・システムが含まれ、これはManagement Consoleに統合されており、ロボットで生成されたエラーにスケジュールから直接ジャンプできます。これは、レガシーのメッセージ環境と同時に機能しますが、記憶域/メッセージ環境は将来のある時点で製品から削除される予定であるため、ロボットを再作成することを改めてお薦めします。

8.2の変更点

この項は、古いバージョンのKapow Katalystのユーザーのみを対象としています。一部の変更が古いロボットの実行に影響を与える可能性がある、バージョン8.2で行われた変更を詳細に説明します。

- 試行ステップのすべての分岐で失敗すると、試行ステップの構成方法に応じて、状況が記録され、ロボットの呼出し側にAPI例外としてレポートが戻されます(この詳細は「エラーを処理する方法」で詳しく説明します)。8.2より前のバージョンでRoboServerでロボットを実行している場合は、各分岐の失敗原因の詳細がすべて記録またはレポートされました。ただし、試行ステップのすべての分岐が失敗したことは記載されず、また試行ステップの名前または場所も指定されませんでした(この情報はデバッガでのみ使用可能でした)。

  バージョン8.2からは、これは試行ステップへの参照も含め、試行ステップのすべての代替が失敗しましたという別のエラーとしてレポートまたは記録されます。このエラーは、失敗した分岐に関する詳細の前にレポートまたは記録されます。この変更は一部のAPIクライアントに影響を与える場合があり、ロボットの式で使用できるロボット・プロパティ**`Robot.executionErrors`**の内容に影響を与えることに注意してください。

- データベース・ロギングを使用している場合は、ログ表の1つを更新する必要があります。ログ・データベース内の**ROBOT_MESSAGE**に、現在は**PROJECTNAME**という追加の列があります。データベース・ロギングを使用する前に、**ROBOT_MESSAGE**表を更新する必要があります。2つの選択肢があり、1つは**ROBOT_MESSAGE**表を削除する方法で(以前の実行のすべてのエラー/メッセージが失われます)、RoboServer (またはManagement Console)の再起動時に表が再作成されます。以前の実行の履歴を保持する場合、またはRoboServerで使用される資格証明にALTER TABLE権限が含まれていない場合は、DBAに**PROJECTNAME**列の追加を依頼する必要があります。列のタイプは、**VARCHAR(255)** (Oracleでは**NVARCHAR2**、SybaseおよびMicrosoft SQL Serverでは**NVARCHAR**)です。

8.1における新しい変数およびタイプのシステム

この項は、古いバージョンのKapow Katalystのユーザーのみを対象としています。バージョン8.1における新しい変数およびタイプのシステムの理解をさらに深めていきます。

バージョン8.1の時点で、オブジェクトおよびモデルという用語は使用されなくなりました。ロボットには現在、多数の変数を含めることができ、各変数にタイプがあります。以前はオブジェクトという語がかなり漠然と使用されていたため、この新しい用語は、対象をより明確にすることを意図しています。全体的に、次の変更が行われました。

- 用語が変更されたため、モデル・エディタでオブジェクトと呼ばれていたものがタイプと呼ばれるようになり、ロボットでオブジェクトと呼ばれていたものが変数と呼ばれるようになりました。したがって、モデル・エディタは現在タイプ・エディタと呼ばれます。

- タイプと変数は、オブジェクトと同様には動作しません。タイプは、変数用のブループリントとしてのみ表示する必要があり、つまり、変数を特定のタイプから作成できます。古い用語は、オブジェクトが実際にはモデル・エディタで作成され、その後必要に応じてロボットに追加できることを示していました。これは、現在では行われません。したがって、理解する必要がある重要な相違点は、タイプでは変数は定義されず、ブループリントのみが定義され、ロボットでタイプを利用するには、そのタイプの変数を作成する必要があるということです。

- タイプ・ファイルでは、古いモデル・ファイルとは対照的に、単一のタイプのみが定義されます。以前は、モデル・ファイルでオブジェクトをいくつでも定義できたため、概要の保守が困難になっていました。

- 特定のタイプの変数を複数作成することは可能です。これは、モデル・ファイルで定義されたオブジェクトをロボットに1回しか追加できなかった以前とは対照的です。

- 多数の単純タイプが存在し、これは単一値の格納に使用できます。これらの目的は、特にスクラッチパッド・オブジェクトを置換することです。また、多くの場合、これらは、一時データの格納のためにのみタイプを作成するかわりに使用できるため、ロボットの作成を容易にします。区別できるように、タイプ・エディタで作成されたタイプは複合タイプと呼ばれます。複合タイプの変数のみをロボットへの入力として使用でき、出力として返すことができます。したがって、単純タイプの変数は内部ロボット変数と考える必要があります。

- 複合タイプは、入力に使用または出力に使用のいずれとしても定義されません。かわりに、変数で、入力として使用するかどうかを設定できます。出力という用語は使用されなくなり、すべての変数を出力にできます(前述のように、入力または出力のいずれとしても使用できない単純タイプの変数を除きます)。

- 古いモデル・ファイルも今までどおり使用できますが、編集はできなくなりました。モデル・ファイルの内容は、現在はオブジェクトとしてではなくタイプとして解釈されます。つまり、モデル・ファイルで定義されたタイプは、新しいタイプ・ファイルで定義されたタイプと同様に、変数の作成に使用できます。モデル・ファイルでは、オブジェクトは入力または出力のいずれであるか定義されていました。この情報は、入力は変数で定義されるため必要なくなりました。したがって、モデル・ファイルのロード時には単に破棄されます。これに注意する場合の例として、古いモデル・ファイルで定義されたタイプから変数を作成する場合があります。タイプの導出元のオブジェクトが以前は入力オブジェクトであった場合、対応するタイプから作成される変数は、自動的には入力に設定されないことに注意することが重要です。これは手動で実行する必要があります。

- モデル・ファイルをタイプ・ファイルに変換できます。この結果、モデル・ファイルで定義された各タイプ(オブジェクト)に対して単一タイプのファイルが作成され、その後これらを必要に応じて編集できます。モデルをタイプに変換するには、(Design Studioの)プロジェクト・ビューでモデルを右クリックし、コンテキスト・メニューからドメイン・モデル・ファイルの展開を選択します。

- グローバル変数の古い概念が変更されています。現在は、いずれの変数もグローバルとして設定できます。あらゆる種類の変数をグローバルにできることで、ロボットの構成方法の自由度が高まりました。古いグローバル変数を使用してロボットを開くと、これらは、単純タイプの新しい種類のグローバル変数である短いテキストに変換されます。

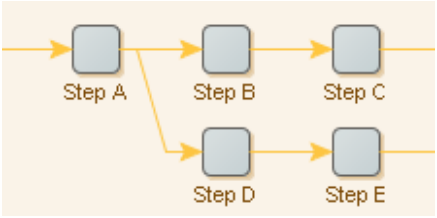
8.0の新しい制御フローおよびエラー処理

バージョン8.0のKapow Katalystでは、ロボットの制御フローを表示する新しい方法が導入され、エラーを処理する方法に大きな変更が加えられています。これは、以前のバージョンで作成されたロボットも含め、すべてのロボットに影響を与えるため、同じ方法で引き続き動作しますが、これらのロボットは、バージョン8.0以上で開いた場合は表示が異なります。以前のバージョンのKapow Katalystを使用するユーザーのために、この章では、相違点の例を示します。

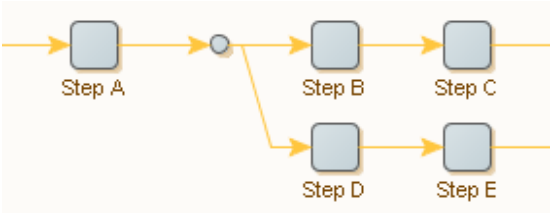
分岐点および試行ステップ

最もわかりやすい変更は、通常のステップの出力接続が1つのみになったことです。ステップの後に複数のステップが続く場合は、その後に明示的な分岐点(小さい円)または試行ステップが挿入されます。これらの新しいステップは、そこから複数の接続が出るように設定できる唯一のステップです。

これは、ステップ構成から分岐モード・プロパティがなくなったことも意味します。ステップが以前にすべての分岐の分岐モードであった場合、現在は次の例のように、分岐点が続きます。最初に、バージョン7.2で表示した場合のロボットのスニペットを示します。

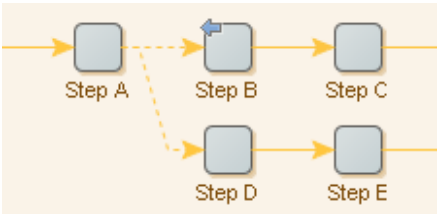


バージョン8.0では、同じロボットが現在は次のようになります。

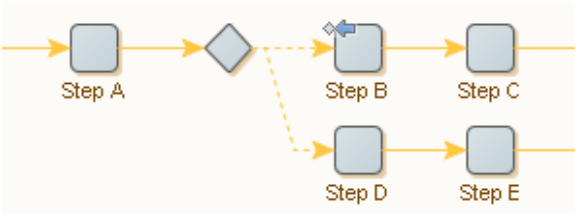


もちろん、ステップの後に分岐点が挿入されるのは、そのステップの後に続くステップが複数ある場合のみです。

ステップが以前に成功分岐までの分岐モードであった場合、今度は分岐点ではなく試行ステップが続きます。このステップは、エラーを処理する新しい方法の中心的なコンポーネントであり、「エラー処理」で説明しています。ここでは、その表示の例のみを示します。バージョン7.2ロボットの次の部分を考えてみます。



このロボットをバージョン8.0で開くと、次のようになります。



終了ステップ

終了ステップの導入は、小さいですが重要な変更です。終了ステップは何もしません(特にロボット実行を終了しません)が、自動的に挿入される明示的な最終ステップをロボット内の各分岐すべてに提供します。つまり、終了ステップを(設計モードで)クリックすると、分岐のすべてが実行されます。以前は、何もしないステップが同じ目的のために手動で追加されることがよくありました。これは必要なくなりました。

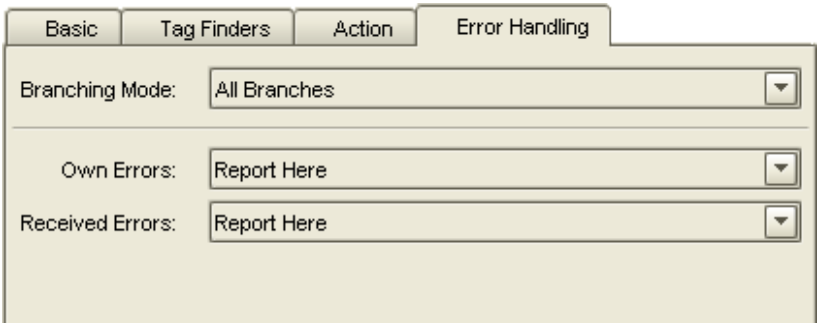
単一の明示的なロボットの開始マーカー(右向きの三角形)も各ロボットに表示されます。これは、単にバランスのために提供されており、ロボットをもう少しグループ・ステップ(別の新機能)に似た形式にするためです。

次の完了ロボットは、ロボットの終了ステップと開始時のマーカーの両方を示しています。バージョン7.2では、ロボットは次のように表示されました。

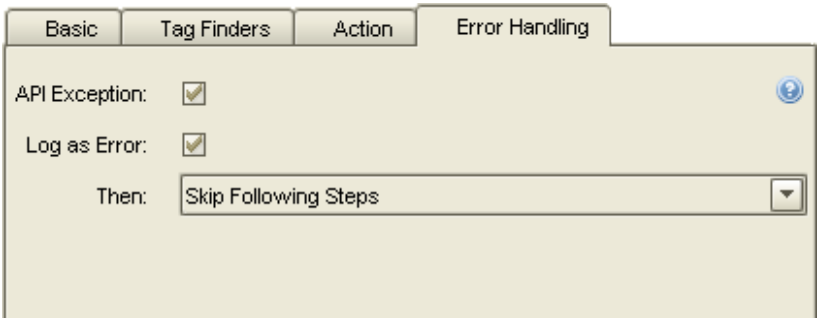
バージョン8.0で開くと、同じロボットが次のようになります。

エラー処理

ロボットのエラー処理の領域は全面的に改訂されました。これは、「エラーを処理する方法」の項で詳細に説明します。次では、エラー処理の古い方法が新しい方法にどのように変換されるかのみを示します。バージョン7.2では、エラー処理は次のプロパティを使用して構成しました。



これはすべて、後述する次のプロパティ・セットで置換されています。



エラーを処理方法に関係なくレポートまたは記録

独自のエラー・ステップ・プロパティでは、エラーをレポートするかどうか、およびエラーがロボットのその後の実行に与える影響(エラー処理)の両方を指定していました。これらの面が、現在は分けられ、独立して構成できます。独自のエラーのかわりに、現在はAPI例外、エラーとして記録、処理の3つのプロパティがあります。

エラーの処理方法(無視する場合も含む)に関係なく、問題をロボットの呼出し側にレポートできます(これはAPI例外と呼びます)。さらにこれとは関係なく、エラーを記録できます(エラーとして記録)。API例外とロギングの両方を選択することも、各ステップの選択を単独で解除することもできます。

以前の独自のエラー・プロパティのエラー処理の部分は、現在は処理プロパティで処理されます。

エラーの発生場所における処理

以前は、エラーを処理する1つの非常に重要な方法は、処理のためにエラーを(ロボット内で左側にあるステップに)戻す必要があることを指定することでした。(おそらく多数の)通過ステップにおける、受信したエラーおよび分岐モードの構成によって、エラーを最終的に処理する方法を決定していました。レポートして終了(結果的にAPI例外が発生)するか、ロボットの実行に影響を与えるか、無視するかであり、通過したすべてのステップの構成を考慮せずに理解することはできませんでした。

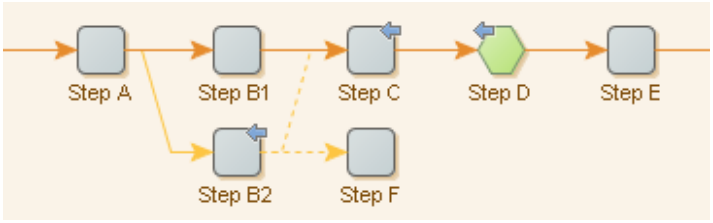
この「エラーを前に戻す」という全体的な考え方が、現在はより直接的な方法で置き換えられ、この場合、他のステップの構成に依存することなく、エラーが発生したステップで処理方法を必ず指定します。したがって、受信したエラー・プロパティは削除されました。

古いロボットを開くと、処理プロパティの割当方法を決定するために、前に戻すを使用する各ステップに対する分析が行われます。ほとんどの場合、結果は、前に戻す(ステップ上に  で表示)ではなく、ステップは、次の代替の試行()、またはAPI例外とエラーとして記録を組み合せた、後続ステップのスキップ()のいずれかを使用するようになります。

前に戻すは、エラーを処理する方法としてまだ使用可能ですが、非推奨です(処理プロパティの編集時には選択できません)。まだ使用可能である唯一の理由は、一部のロボットを以前のバージョンのKapow Katalystと完全に同じ方法で実行するために必要であるためです。これが必要になるのは少数のロボットでのみです。これらのロボットを検査する場合は、すべてのステップで、受信したエラーが前に戻すに設定されていると考える必要があります。

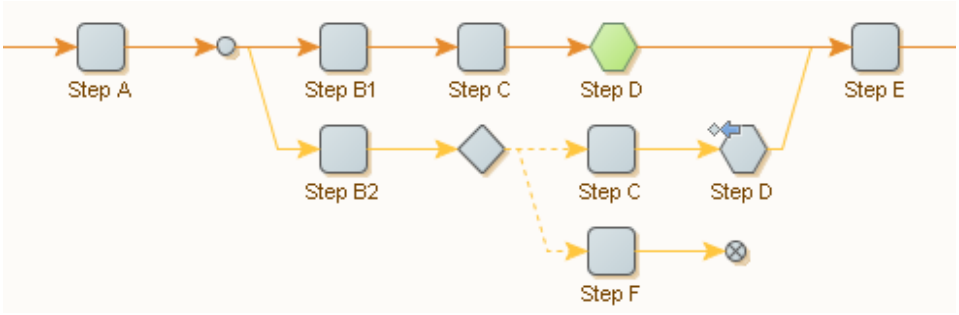
ロボット構造に対する小規模な変更

分岐がマージされるあまりないケースでは、前述の分析の結果が、ステップの処理プロパティを、ステップに続くパスに応じて様々な方法で構成する必要があるという結果になる場合があります。このような場合、ステップ(およびロボットの小さい部分の場合もあります)は、分岐のマージを解除するために、パスのこの依存性がなくなるように複製されます。この結果、予期しないロボットの拡張が発生する場合があります。次のロボットが例です(不自然に見えると思いますが、この動作を示すロボットは普通ではありません)。ロボットは、バージョン7.2ではこのようになります。



Step Dの実行時に、エラーがレポートされる場合があります。Step B1を経由する実行パスでそのステップに達した場合、エラーはB1に戻され、そこでレポートされます。ただし、Step B2経由でStep Dに達した場合は、レポートされません。逆に、実行が進んでStep Fに達することになります。

バージョン8.0で開いた場合、ロボットは自動的に次のように変更されます。



Step Dが複製され、2つのコピーが異なる方法で構成されていることがわかります。Step B1に関連付けられている方(緑色)は、API例外およびエラーとして記録がオンで、後続ステップのスキップを使用するのに対して、B2用の方は、ロギングなしですが、次の代替の試行を使用します。ステップを正しく接続できるように、Step Cも複製されます。

エラー・レポートの順序に対する小規模な変更

以前のKapow Katalystバージョンでは、前に戻すによって、API例外としてエラーをレポートするのを遅延できました。しかし、バージョン8.0では、エラーがレポートされる場合は、発生後すぐにレポートされます。一部のロボットに対しては、これは、以前と異なる順序で呼出し側にエラーのレポートが戻されること、およびそれらが異なる方法でAPI例外にグループ化される可能性があることも意味します。

7.2の新しい記憶域システム

バージョン7.2のKapow Katalystでは、データベースへのデータの格納が以前のバージョンを使用するよりも簡単になりました。新しい記憶域メカニズムでは、学習曲線が下がって冗長な作業が削減され、ロボットの作成により多くの時間を費やすことができます。

このガイドでは、古い記憶域システムと新しい記憶域システムの相違点、および古い記憶域システムから新しい記憶域システムへの移行に必要なアップグレード・パスについて説明します。

古い記憶域システム

最初のリリースのKapow Katalystから、データベースへのデータの格納が可能でした。その価値は証明されていますが、時間の経過とともに多数の問題が明らかになりました。

古い記憶域システムに関する問題点

- データベースへのデータの格納に特別なDatabaseOutputObjectが必要です。
- DatabaseOutputObjectにrefindKeyフィールドがあり、オブジェクトの構築時点でrefindKeyの重要性を理解するユーザーは少なく、データが一度格納されると、refindKeyの変更は面倒なプロセスでした。
- 各ロボットをRoboManagerデータベースに登録する必要があり、そのためにロボットにロボットIDがありました。
- 実行時に、DatabaseStorageEnvironment、DatabaseRobotInfoEnvironmentおよびDatabaseMessageEnvironmentの3つの環境を提供する必要がありました。
- オブジェクトの再検索アクションが複雑で、使用する際に間違うことがよくありました。

これらの問題から多くの予期しない危険性が生じ、経験のあるユーザーでさえ間違うことがありました。最もよくあるのがロボットIDの重複であり、すでに登録されているロボットがテンプレートとして使用された場合でも、コピーがRoboManagerに再登録されず、突然2つのロボットが同じロボットIDになりました。

新しい記憶域システム

バージョン7.2のKapow Katalystで導入された新しい記憶域システムでは、古い記憶域システムの短所を修正することを目的としています。

新しいシステムの利点

- データベース用の専用OutputObjectは不要であり、通常の出力オブジェクトが使用されます。
- ロボットの登録は必要ありません。
- データは、データベースに格納ステップ・アクションを使用して格納されます。
- 環境が不要で、ロギングはオプションであり、Settingsアプリケーションを使用して構成されます。
- RoboManagerが不要で、ログはWebインタフェースを使用して表示できます。
- データベースの検索およびデータベースでの削除という新しいステップ・アクションがあります。

新しいシステムでは、RoboManagerおよび環境の使用が完全に必要なくなり、データベースにデータを収集するロボットの実行プロセスが大幅に簡素化されます。

refindKeyのかわりに、現在はObjectKeyがありますが、モデルの構築時にこれを構成する以外に、ロボット内でも指定できます(また、ObjectKeyに対してデフォルトでフィールドが選択されていないため、オブジェクトIDを定義するフィールドを検討する必要があります)。

データベースに格納を使用してオブジェクトが格納されるとき、7つの追加(事前定義)フィールドが格納されます。これらは、次の変更点を除いて、古いDatabaseOutputObjectに含まれていた6つの非表示フィールドと同様です。

- ObjectKeyでrefindKeyを置換します。
- ExecutionIdでRobotRunIdを置換します(API実行と相互に関連付けできます)。
- RobotNameでRobotIdを置換します(このためロボット名が一意あることを確認してください)。
- LastUpdatedが追加され、これによって、このレコードが更新された最終時刻がわかります(古いChangeDateプラグインによって追加される機能と同様)。

レガシー・オプション

7.2の新規インストール後、古い記憶域システムは完全に表示されなくなります。これは新しいユーザーが混乱しないようにするためです。古いデータベース出力オブジェクトを作成する必要がある場合、またはデバッガで環境を使用してロボットを実行する場合は、Settingsでレガシー・プロパティを有効にする必要があり、これはレガシー・タブで実行します。

下位互換性

2つの記憶域システムには異なる表レイアウトが必要であるため、互換性はありません。ただし、ほとんどのシナリオで、2つのシステムは問題なく共存できるため、時間をかけてロボットを移行できます。1つのロボット内で新しいシステムと古いシステムの使用を混在させないでください。

ほとんどの場合、新しい記憶域システムを使用するロボットにDatabaseRobotInfo環境を提供していないかぎり、同じRoboServerで両方の記憶域システムを使用してロボットを実行できます。つまり、デプロイメント・ディスクリプタまたはAPIコードを使用して環境を構成している場合は、新しいシステムを使用するロボットと古いシステムを使用するロボット間のコードに区別を付ける必要があります。または、新しいロボットに一時的にロボットIDを付与して、RobotInfo/Message環境で実行できるようにし、その後すべてのロボットを移行してから環境を削除できます。

新しいロボットをデータベース記憶域環境で実行すると、ロボットに古いデータベース出力オブジェクトまたはオブジェクトを返すステップが含まれていないかぎり無視されます。

ロボットをデータベース・メッセージ環境と、RoboServerに対するデータベース・ロギングが有効な状態の両方で実行した場合は、2つの異なるログを取得します。1つはRoboManagerを使用してアクセスし、もう1つはManagement Consoleのログ・ビューを介してアクセスします。

移行

古い記憶域システムから新しい記憶域システムへの移行では、通常4つの別個の移行ステップが必要です。モデル、ロボット、データ、およびデータ表と対話するSQLスクリプトの移行です。

モデルの移行

モデル・タイプをDatabaseOutputObject (レガシー)からOutputObjectに変更し、6つの非表示フィールド
(robotId、robotRunId、refindKey、firstExtractionDate、latestExtractionDate、extractedInLatestRun)
を削除します。ObjectKeyの一部にするフィールドを検討します。

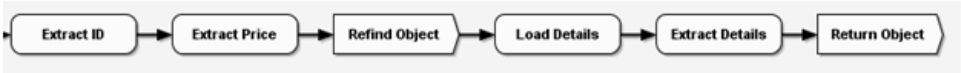
ロボットの移行

ロボットを移行するには、次のステップを完了する必要があります。

- オブジェクトを返すをデータベースに格納に置換して、適切に構成します。

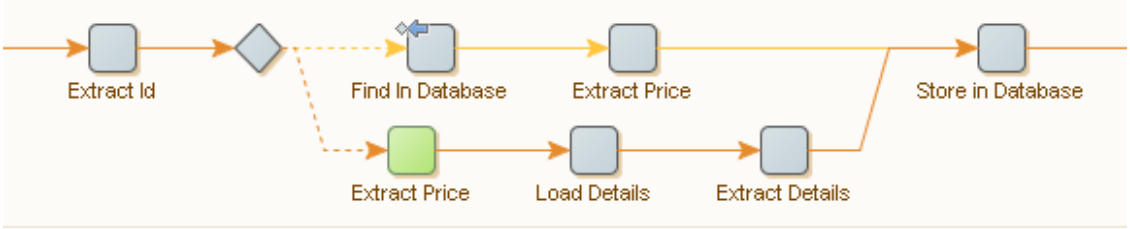
- オブジェクトの再検索をデータベースの検索に置換します。オブジェクトの再検索では、オブジェクトが更新され、オブジェクトが検出された場合は停止するため、ロボットの構造を多少変更する必要があります。

次に、オブジェクトの再検索を使用した標準的なシナリオを示します。オブジェクトIDが抽出され(この例でrefindKeyに使用される唯一の属性)、価格が抽出され(refindKeyの一部ではありません)、その後オブジェクトが再検索されます。オブジェクトが存在している場合は、価格が更新され(分岐の残りはスキップされ)、オブジェクトが存在しない場合は、詳細ページがロードされて詳細が抽出された後、オブジェクトが返されます。



オブジェクトの再検索の例

これは次のように変換する必要があります。



データベースの検索の使用

最初に、ID (オブジェクト・キー属性)を抽出した後、データベースの検索を使用してオブジェクトがすでに存在するかを確認します。オブジェクトが存在する場合は、現在のレコードが出力オブジェクトにロードされ、今度は以前の値の上に価格を抽出し、次にオブジェクトを再度格納します。データベースの検索でオブジェクトが見つからなかった場合は、エラーが生成され、次の代替の試行エラー処理によって実行が下位の分岐に戻され、そこで価格を抽出し、詳細をロードおよび抽出した後、オブジェクトを格納します。

新しいメソッドには、価格の抽出という追加ステップが必要になる場合がありますが、実行内容がかなりわかりやすくなり、オブジェクトの再検索による悪影響もありません。

ロボットに、データベースの問合せまたはSQLの実行という記憶域表を問い合せるステップがある場合は、これらも更新する必要がある場合があります。

既存のデータおよび表の移行

データの量が少ない場合、またはロボット実行のたびにすべてのデータを収集する場合、最も簡単な方法は、単に既存の表を削除して、更新したモデル・オブジェクトを使用して新しい表を作成することです。

データの量が多い場合、以前の実行の履歴データが必要な場合は、既存のデータ表を新しい表レイアウトに変換する必要があります。データベース・ベンダーのツールを使用して、フィールドの追加/削除および名前変更を実行することも可能ですが、この方法はお薦めしません。最適なソリューションは、ロボットを使用して古いデータをロードし、新しいフォーマットを使用してそれを格納することです。

これを実行するには、更新した出力オブジェクトから表を作成します。新しいバージョンのオブジェクトに別の名前を付与していない場合は、元のデータ表の名前を変更する必要があります。次に、データベースの問合せステップ・アクションを使用して既存のデータをロードするロボットを作成し、データベースに格納を使用してオブジェクトを格納します。

最初の抽出日や最終抽出日など、データ履歴が必要な場合は、これらを古い行から抽出し、オブジェクトを格納した後で、SQLの実行ステップを使用して更新できます。

refindKeyをKapow環境の外部に公開している場合は、既存のrefindKeyにマップするObjectKeyを追跡する必要があるため、外部にマッピング表を提供し、既存のrefindKeyを新しいObjectKeyに変換でき

るようにします。これは、ロボットIDが通常refindKey計算の一部であったため、以前のrefindKeyと完全に同じObjectKeyを必ず生成できるとはかぎらないため重要です。

SQLスクリプトの移行

外部SQLスクリプトの可能性および複雑性は事実上無限であるため、このようなスクリプトの移行方法について一般的なルールを適用することはできません。かわりに、いくつかガイドラインを示します。

収集データベースからの本番データの移動

収集表(ロボットによる収集先の表)に対するユーザー問合せの実行はお薦めしません。かわりに、本番データ用に別の表セットを設定してください。これらの表にはrefindKey/ObjectKeyが含まれますが、他の事前定義フィールドは含まれない場合もあります。本番表は通常、ロボットが正常に実行されるたびに更新される必要があります。

以前は、移動する行は通常、robotRunId列を使用して最新実行を検索することによって識別されましたが、この列は存在しなくなりました。かわりに、この行が最後に更新されたときのタイムスタンプを含む、LastUpdatedという列があります。このように、実行する必要があるのは、収集表からデータを最後に移動した後で更新されたすべての行を検出することのみです。同じ表にデータを収集するロボットが複数ある場合は、異なるロボットからのデータを識別するために、現在はrobotIdではなくrobotNameを使用する必要があります。

RoboManager表に対するスクリプト

新しいデータベース・ログでは、古いRoboManagerのログと比較して優れた情報が提供され、これらの一部(入力オブジェクトのロギングなど)は内部で使用されますが、残りはManagement Consoleのログ・ビューで参照できます。RoboManagerのRobotRunまたはMessage表に対する問合せを含むスクリプトがある場合、これらは新しいロギング・レイアウトと一致するように再作成する必要があります。

最も重要なことは、robotIdとrobotRunIdがRobotNameとExecutionIdに置換されたことです。executionIdの出現によって、以前はできなかった、APIを使用して作成された実行に対する特定の実行またはメッセージを追跡できるようになります。新しいROBOT_RUN表には、キュー時間、実行時間、合計実行時間など、特別な統計も多数含まれています。

第6章 初心者用チュートリアル

この項では、Kapow Katalystの概要の後に、Kapow Katalystでの最初のプロジェクトをガイドする3つのチュートリアルが続きます。これらのビデオに進む前に、Kapow Katalystが正しくインストールされ、設定されていることを確認してください。次のようなビデオをクリックして再生します。次のチュートリアル・ビデオに進む準備が整っている場合は、ページ右下隅のリンクをクリックします。

各ページには、ビデオのトランスクリプトも記載されています。

これは4つの初心者用チュートリアルの最初のチュートリアルで、Kapow Katalystでの最初のプロジェクトを滞りなくガイドします。この最初のビデオでは、Kapow Katalystを使用した場合のワークフローの概要を、Design Studioと呼ばれるメイン・プログラムを紹介しながら説明します。これらのチュートリアルに進む前に、Kapow Katalystが正しくインストールされ、設定されていることを確認してください。help.kapowsoftware.comで、ドキュメントの「概要」項の「インストール・ガイド」に記載されている関連部分に従ってください。

Kapow Katalystは、簡単に説明すると、マウスとキーボードを使用してブラウザで実行可能なプロセスを完全に自動化できるプラットフォームです。

発想から自動化プロセスまでの一般的な手順に従って、ゆっくりご覧ください。

すべては、自動化する必要があるプロセスについての発想から始まります。これらの初心者用チュートリアルでは、News MagazineというWebサイトから最新のニュースを自動的に抽出する必要があります。

最初のステップは、Webサイトの調査です。必要なものは何ですか。

実現する必要があるものについて確信を得たら、Design Studio (自動化プロセスの作成に使用するプログラム)を開きます。初めてDesign Studioを開いたときは、この初心者用チュートリアルにリンクしているようこそ画面が、他のドキュメントとともに表示されます。「OK」をクリックすると、Design Studioのメイン・ウィンドウが表示されます。

左側にはプロジェクト・ビューがあります。現時点では、デフォルトのプロジェクトのみが表示され、そのプロジェクトには見本ファイルの集合とTutorialsフォルダがあり、このフォルダには初心者用チュートリアルでこれから作成するファイルのサンプルが格納されています。

プロジェクト・ビューでPost.typeというファイルをダブルクリックすると、この種のファイルの編集と作成に使用するタイプ・エディタが開きます。タイプは、該当するタイプの変数に格納できるデータの種類を定義します。タイプと変数が理解できない場合、変数はテキストやイメージなどのオブジェクトを入れる容器、タイプはその容器のタイプを生み出す鋳型と考えることができます。

この特定のタイプは、News MagazineのWebサイトから抽出する情報を格納するように設計されています。タイプの作成プロセスには、最後のチュートリアルで触れます。

News Magazineからのニュースの抽出は、ロボットという自動化プロセスによって実行されます。ロボットは、Webブラウザで実行するプロセスの自動化と考えることができます。

また、TutorialsフォルダからNewsMagazine.robotというファイルをダブルクリックすると、ロボット・ファイルの編集と作成に使用するロボット・エディタが開きます。

エディタ上部のロボット・ビューにはロボットの構造が表示されます。各ステップは、ロボットが実行するアクションに対応しています。ロボット・ビューのステップに沿って、ロボットはNews Magazineをロードして最新の記事にナビゲートし、ループを使用して各ニュースのタイトルとプレビューを抽出します。ロボットは、収集した値を最後に返します。

ロボット・ビューの第2ステップをクリックすると、ページのロード・アクションが実行され、ロードされたNews Magazineが下部のブラウザ・ビューに表示されます。後でわかるように、ブラウザ・ビューによってロボットの作成が直観的になります。

次のチュートリアルでは、このロボットを自分で作成する方法を示します。

ロボットを使用した抽出のプロセスを自動化した後は、そのロボットをManagement Consoleにアップロードします。このManagement Consoleは、Kapow Katalystの操作面を管理するWebベースのアプリケーションです。ロボットからは、自分と他のユーザーが使用するアプリケーションとしてロボットを公開するKappletを作成できます。

完成品のKappletは、News Magazineから最新ニュースを自動的に抽出し、ユーザーが表示およびダウンロードできるようにその内容を返します。

これで最初のロボットを作成する準備が整いました。ロボット初心者用チュートリアルの視聴から初心者用チュートリアルを開始してください。

ロボット初心者用チュートリアル

後に続く内容は、ロボット初心者用チュートリアル・ビデオと、そのビデオのトランスクリプトです。

概略紹介

これは、Kapow Katalystでの最初のプロジェクトを最後までガイドする4つの初心者用チュートリアルの2番目です。このチュートリアルは、概要ビデオを視聴してから開始することをお薦めします。

KapowのDesign Studioでロボットを作成する方法について学習します。コンピュータの説明に従って自由に視聴してください。

具体的な紹介

Kapow Katalystのロボットは、Webブラウザで実行できるプロセスを自動化するために使用します。ロボットでは、手動で実行する必要があるマウスとキーボードによる一連の指示を再現および自動化できます。

この初心者用チュートリアルの目標は、これらのチュートリアル専用に作成されたサイトであるNews Magazineから、最新のニュースを抽出するプロセスを自動化することです。サイトへのリンクは、このビデオに関連するテキスト内にあります。(http://kapowsoftware.com/tutorial/news-magazine/index.html。)タブの下にあるMost Recent Newsには、News Magazineの最新記事が3件表示されています。ここから、そのタイトルと指定された短い記事プレビューを抽出します。このすべてをロボットで実行するように設計します。

新しいロボットの作成

実行しているDesign Studioで、デフォルトのプロジェクトを右クリックして、「新規」>>ロボット...の順に選択し、新しいロボットを作成します。ウィンドウが開き、新しいロボットの名前の入力が求められます。これが最初のロボットであるため、MyFirstRobot.robotに決定します。「次」をクリックします。ロボットをNews Magazineのフロント・ページ(http://kapowsoftware.com/tutorial/news-magazine/index.html)に関連付け、「終了」をクリックします。

ロボット・エディタが開き、News Magazineがロードされます。左側のプロジェクト・ビューで、新しいロボット・ファイルが自動的に選択されていることに注意してください。

ロボット・エディタ

ロボット・エディタには、5つの異なるメイン・ビューがあります。

ロードされたページを、ブラウザでの表示と同じように表示するブラウザ・ビューがあります。

このブラウザ・ビューの下には、ロードされたページのhtmlを表示するhtmlビューがあります。

最上部には、ロボットで実行されるアクションを確認できるロボット・ビューがあります。アクションは、リンクのクリックからデータベースへのデータ格納に至るすべてのアクションが対象となり

ます。アクティブなステップは緑色で強調表示され、アクティブなステップの左側にあるステップは実行済です。現時点では、小さな丸いステップの終了ステップがアクティブなステップで、ページのロード・ステップは実行済です。終了ステップの意味は、後で説明します。

右側のステップ・ビューは、アクティブ・ステップで実行されるアクションの構成に使用されます。現在のアクティブ・ステップは終了ステップであるため、構成するプロパティはなく、ステップ・ビューにはステップの説明のみが表示されています。

最後に、変数ビューでは、入力、出力または実行時にデータを格納するためにロボットが使用する変数を指定します。

ブラウザ・ビュー

抽出元のページにナビゲートするには、ブラウザ・ビューを使用します。ブラウザ・ビューでのダブルクリックは、通常のブラウザでのシングルクリックに相当します。抽出するニュースを取得するには、「Most Recent News」タブをダブルクリックします。

新しいステップが作成されてロボット・ビューで実行され、ブラウザ・ビューに「Most Recent News」ページがロードされていることに注意してください。

ページを下にスクロールすると、抽出する3つのニュース記事が表示されます。

ブラウザ・ビュー内でシングルクリックすると、クリックしたHTMLタグの周囲に緑色のボックスが表示されます。この緑色のボックスによって、ブラウザ・ビューで選択したタグにマークが付きます。タグを選択して右クリックすると、選択したタグに適用できるアクションを示すメニューが表示されます。今は、これを使用します。

タグのループ

次のステップは、3つの最新記事を繰り返すループを作成することです。そのためには、最初の記事のタグ内で任意のタグを単純に選択します。たとえば、写真を選択します。選択したタグで右クリックし、ループ>>For Each Tagの順に選択します。ロボット・ビューには、For Each Tagステップが表示され、ブラウザ・ビューには、ループの最初の記事の周りに青色のボックスが表示されます。

For Each Tagステップには、ループの繰返しに使用できる矢印があります。これらを使用して3つの記事を繰り返し、それらが正しく選択されていることを青色のボックスで確認します。矢印をクリックしてもロボットに変化はありませんが、ループが期待どおりの繰返しであることの確認には役立ちます。最初の記事に戻るには、最も左側の矢印を使用します。

最終ステップの機能が明らかになりました。ループでは、For Each Tagステップと終了ステップで折り返したすべてのステップをループします。終了ステップの後にはいかなるステップも追加できません。

変数の追加

現状、何かを抽出するには、変数を追加して、抽出対象のテキストを含める必要があります。紹介チュートリアルで触れたように、このロボットで使用する「post」というタイプがすでに用意されています。

右下隅の変数ビューで白色のボックスを右クリックし、複合タイプの変数の追加>>「post」の順に選択します。変数を構成するウィンドウが開きます。デフォルト設定のまま、「OK」をクリックします。

選択したタイプの変数が変数ビューに表示されます。「タイトル」と「プレビュー」の2つの属性があり、それぞれが抽出する対象に対応しています。

抽出

ブラウザ・ビューに目を戻すと、抽出の準備が整っています。青色のボックスでマークされた「post」のタイトルをクリックしてから右クリックし、抽出>>テキストの抽出>>「タイトル」の順に

選択します。指定タグと呼ばれる青色のボックス内を抽出すると、ループの他の繰返しによって、対応するタイトルとプレビューが他の投稿から抽出されることになります。

同じことを記事プレビューに対して実行します。プレビュー・テキストをクリックしてから右クリックし、抽出>>テキストの抽出>>「プレビュー」の順に選択します。抽出されたテキストが変数ビューに表示されていることに注意してください。ロボット・ビューに2つの抽出ステップが追加され、抽出されたことにも注意してください。変数ビューでの確認の際は、For Each Tagステップの矢印を使用して、他の投稿からもテキストが適切に抽出されていることを確認します。

値を返すステップ

収集したデータを出力するには、ロボットに値を返すステップを挿入する必要があります。終了ステップを右クリックしてサブメニューのステップを前に挿入に移動し、アクション・ステップを選択します。これによって新しいアクション・ステップが終了ステップの前に挿入されます。

右側のステップ・アクション・ビューのドロップダウンを使用し、このステップに対して値を返すアクションを選択します。ステップ・ビューが変更され、値を返すアクションのプロパティが表示されます。以前に追加した変数がデフォルトで選択されている状態で表示されます。ロボットはデバッガによるテストの準備が整っています。

デバッガ

デバッグ・モードには、ロボット・ビューの上にあるデバッグ・ボタンをクリックして切り替えます。

ロボット・エディタの下部が、ロボットの実行を監視する様々なツールが格納された複数のパネルに置換されます。メイン・パネルのデフォルトのタブには、「入力」および「出力」が表示されます。デバッグ・メニューから「実行」を選択し、ロボットを実行します。ロボットが正常に実行され、メイン・パネルに出力が表示されます。これで、3つの最新の記事をNews Magazine Webサイトから抽出するロボットが正常に作成されました。

なんらかの理由で実行に失敗した場合は、チュートリアルを再実行してすべてのステップをチェックするか、先ほど作成したロボットと同じロボットであるNewsMagazine.robotをチェックします。News Magazineロボットは、デフォルト・プロジェクトのTutorialsフォルダにあります。

次のステップは、Management Consoleへのロボットのアップロードと Kappletの作成です。Management Consoleチュートリアルに移動するか、特定のトピックについてヘルプが必要な場合はhelp.kapowsoftware.comに移動してください。

Kapplet初心者用チュートリアル

後に続く内容は、Kapplet初心者チュートリアル・ビデオと、そのビデオのトランスクリプトです。

これは4つの初心者用チュートリアルの3番目で、Kapow Katalystでの最初のプロジェクトを滞りなくガイドします。このチュートリアルは、概略ビデオとロボット・チュートリアルを視聴してから開始することをお薦めします。

Management ConsoleとKappZoneを使用してロボットをKappletsとして実行する方法を学習します。コンピュータの説明に従って自由に視聴してください。

このManagement Consoleは、Kapow Katalystの操作面を管理するWebベースのアプリケーションです。Management Consoleは最初に、ロボットとタイプのリポジトリとして機能します。Kappletsを作成して管理できるKappZoneも含まれています。Kappletsはロボットまたは複数ロボットの集合で、アプリケーションとして公開され、簡単に配布して実行できます。

このチュートリアルでは、News MagazineロボットをManagement Consoleにアップロードし、KappletとしてKappZoneに公開する方法を示します。

Design Studioを開き、ロボット初心者用チュートリアルで作成したロボットであるMyFirstRobotを開きます。次に、ロボット・エディタの左上隅にある「設計」ボタンをクリックし、ロボット・エディタが設計モードであることを確認します。

ロボットをManagement Consoleにアップロードするには、「ツール」メニューからリモート管理コンソールへのアップロードを選択します。すべてがすでに正しく構成されている場合は、単に「アップロード」を押します。ロボットが、関連付けられているすべてのタイプとともにManagement Consoleにアップロードされます。

表示されているManagement Consoleへのリンクをクリックします。これによって、Management Consoleのリポジトリにブラウザが開きます。

このリポジトリには、Management Consoleにアップロードされたロボット、タイプおよび他のファイルが格納されています。News Magazineロボットがリストに表示されていることを確認します。同様に、「タイプ」タブをクリックして、関連するタイプが正しくアップロードされていることを確認します。

ロボットからKappletを作成するには、ページ上部にあるKappZoneタブに移動します。これによって、KappZoneへのリンクを備えたページが開きます。クリックすると、新しいウィンドウにKappZoneが開きます。KappZoneとは、Kappletsを追加、削除および編集できる場所です。

Kappletを作成するには、リポジトリの上部にある新規Kappletの追加をクリックします。新しいKappletに「News Magazine」という名前を指定し、Kappletの作成をクリックします。Kappletが作成され、新しいKappletの構成ページが表示されます。

新しいKappletは、ページ上部のスイッチが示すように、現在のところ使用できません。これは、管理者のみがKappletを表示できることを意味します。Kappletは、構成の完了後、速やかに使用可能にします。

Kappletの構成には「アイデンティティ」と「ページ」の2つのページがあり、「アイデンティティ」が選択されています。「アイデンティティ」ページでは、Kappletの名前、説明およびアイコンを編集できます。説明として「Extracts the most recent news from News Magazine」と記載し、アイコンには以前に自分のコンピュータに保存したNews Magazineのロゴを使用します。

Kapplet構成の「ページ」セクションに移動し、Kappletが実際に処理する内容を構成する必要があります。Kappletの機能は複数のページで構成され、最も単純なKappletでも少なくとも2ページ、つまり、スタート・ページと結果履歴ページがあります。

スタート・ページでは、Kappletを即時に開始するか、特定の時間に自動的に開始するようにKappletをスケジュールできます。Kappletを開始すると、スタート・ページに追加されたすべてのロボットが実行されます。

結果履歴ページでは、実行したロボットが返した結果がアーカイブされます。

スタート・ページでアクションの追加をクリックしてから新規ロボットの追加をクリックし、アップロードしたロボットを選択してロボットの選択をクリックし、「OK」をクリックします。

これによって、Kappletの開始というラベルのボタンを備えたアクションが、スタート・ページに追加されます。また、Kappletへの投稿という新しいページも自動的に追加されます。左側の「投稿」ページをクリックします。ユーザーがこのページを結果履歴ページから選択していた場合は、Kappletの特定の結果が表示されます。

現在は、各結果によるタイトル属性の内容を格納した表が表示されます。表の「編集」をクリックし、プレビュー属性を表に追加して「OK」をクリックします。「投稿」ページに、News Magazineから抽出された各ニュースのタイトルとプレビューの両方が表示されます。

クリックして変更を適用し、Kappletを有効にして、右側のメニューに移動します。これは、右上隅の正方形にマウス・ポインタを重ねると表示されます。My KappZoneまたはKappZoneには、メニューから移動できます。KappZoneは、インストール可能なすべてのKappletsを保持するリポジトリと考えることができます。一方、My KappZoneには、各自のアカウントにインストールされたKappletsのみが格納されます。

KappZoneの下にある「すべて」をクリックします。KappZoneにNews MagazineのKappletが表示されます。マウス・ポインタを重ねると、Kappletをインストール、編集または削除できることがわかります。ここではインストールに進みます。Kappletはすぐに開くことができます。右側のメニュー・バーからMy KappZoneに移動して、News MagazineのKappletがMy KappZoneに追加されていることも確認できます。

アイコンをクリックしてKappletを開きます。Kappletのスタート・ページと結果履歴ページが表示されます。スタート・ページからは、Kappletをスケジュールしたり、Kappletの開始をクリックして1回のみ実行できます。ここではKappletの開始をクリックして先に進みます。

結果履歴ページに最初の結果が表示されます。結果をクリックし、結果を表示する「投稿」ページを開き、「投稿」ページの結果全体がブラウザに表示されるまで右にスクロールします。

これで、目標を達成しました。ロボットを使用して自動化プロセスを作成し、ユーザーが簡単にインストールして使用できるように、Kappletとして公開しました。

次のチュートリアルでは、Design Studioでタイプを作成する方法について説明します。

タイプ初心者用チュートリアル

後に続く内容は、タイプ初心者チュートリアル・ビデオと、そのビデオのトランスクリプトです。

概略紹介

これは、Kapow Katalystでの最初のプロジェクトをガイドする4つの初心者用チュートリアルの4番目です。このチュートリアルは、他の初心者用チュートリアルを終了してから開始することをお薦めします。

KapowのDesign Studioでタイプを作成する方法について学習します。コンピュータの説明に従って自由に視聴してください。

タイプの紹介

Kapow Katalystでは、変数はデータを格納するためにロボットによって使用されます。変数は、入力または出力として、あるいは実行時の一時的な値を格納するために使用されます。これらの変数はタイプ別に分類されます。タイプの例には、テキスト、イメージ、PDF、数字などがあります。所定の例はすべて単純タイプと呼ばれるものです。つまり、Design Studioに事前定義されているタイプです。

独自の複合タイプを作成することもできます。複合タイプは、複数の単純タイプの容器と考えることができます。

例を使用して説明します。

これらの初心者用チュートリアルでは、News Magazineから最新のニュースを抽出して格納するプロセスを自動化しました。これらの3つの記事から、タイトルと所定のテキストの短縮部分を抽出しました。タイトルは単純タイプである短いテキストの変数で、プレビューは単純タイプである長いテキストの変数で保持できます。これらの各タイプを格納するために、複合タイプを設計する必要があります。

新しいタイプの作成

起動したDesign Studioで、デフォルトのプロジェクトを右クリックして、「新規」>>「タイプ...」の順に選択し、新しい複合タイプを作成します。ウィンドウが開き、新しいタイプの名前の入力が求められます。これが最初のタイプであるため、MyFirstType.typeに決定します。「終了」をクリックして、タイプ・エディタを開きます。作成したタイプが、左側のプロジェクト・ビューで強調表示されていることに注意してください。

属性の追加

タイプ・エディタで最も重要な部分は、新しいタイプに関連付けられた属性のリストです。属性は、複合タイプの変数に格納できる様々な値を示します。各属性には、名前、タイプおよびその属性に関連する他のプロパティのリストがあります。

タイトル属性の作成を開始します。属性リストの左下隅にあるプラス記号をクリックして、新しい属性を追加します。新しいウィンドウが開き、このウィンドウで新しい属性を構成できます。属性にtitleという名前を指定し、そのタイプに対して短いテキストを選択します。短いテキストは単純タイプで、1行を超えないテキストを格納できます。「OK」をクリックして、属性を複合タイプに追加します。

同様に、短い記事プレビューを格納するpreviewという名前の属性を追加します。この属性は、1行を超える長いテキストを定義する、長いテキストというタイプにしてください。

ここで作成したのは、ロボットで変数になる複合タイプです。このため、MyFirstTypeタイプの変数は、titleとpreviewの2つの変数を保持できます。「ファイル」メニューから「保存」を選択して、タイプを保存します。

ロボットの変更

MyFirstType.typeは、ここでは、ロボット初心者用チュートリアルで使用した複合タイプのpost.typeとまったく同じです。MyFirstRobot.robotを開いた場合、変数のpost変数はタイプMyFirstTypeに切り替えることができます。

このためには、変数ビューでpost変数を右クリックし、変数の編集を選択します。表示された変数の編集ウィンドウで、MyFirstTypeをこの変数のタイプとして選択し、「OK」をクリックします。

最後のステップは、変更したロボットを保存してManagement Consoleにアップロードすることです。以前にアップロードしたバージョンと自動的に置き換わります。

これでKapow Katalystでの最初のプロジェクトがすべて完了しました。

第7章 上級チュートリアル

この項には、Kapow Katalystの高度なトピックに関するチュートリアルが記載されてます。

分岐、ロボット状態および実行フロー

後に続く内容は、分岐、ロボット状態および実行フローの概念を説明するビデオです。

ビデオ・トランスクリプト:

このチュートリアルでは、ロボットでの分岐の使用方法とその理由について説明します。その過程で、ロボット状態という概念を紹介し、一般的なロボット実行フローについて説明します。

初心者用チュートリアルを終了している場合は、ロボットの実行が最も左側のステップで開始し、終了ステップまで順番に続くことを理解しています。これは、分岐点がない直線的なロボットの例です。

分岐点を示す前に、ロボット状態の概念について紹介する必要があります。すべてのステップで、ロボットには、そのロボットの状態を形成する様々な要素があります。最も重要な要素は、現在開いているウィンドウとフレームおよび変数の現在値ですが、状態にはCookie、認証情報なども含まれます。

これらのすべての要素がロボット状態を形成します。

ロボット・ビューに戻り、分岐点について紹介します。分岐点は、Design Studioの「編集」メニューから、分岐の追加を選択して挿入されています。

ここで、ロボットは各分岐を上から下へ順に実行します。ロボットが終了ステップに達するたびに、実行は次の分岐から続行されます。

では、完全に直線的なロボットではなく、なぜ複数の分岐を使用した面倒な経路を使用するのでしょうか。

この質問には多くの答えがありますが、最も重要な理由は、実行が分岐点に戻るたびにロボットが前の状態に戻ることです。これまで説明したように、状態には開いているページ、変数値などが含まれ、ロボットが分岐点に戻るたびに、その分岐点を通過したときのページに戻り、分岐で発生したことはすべて忘れます。

使用方法の例を示します。

同じ出発都市から異なる3つの目的地への旅行に関して、Momondoというサイトを検索するロボットで現在作業しています。ロボットは、出発都市をフォームに入力し、3つの分岐に分かれ、異なる3つの目的地を入力し、クリックして検索します。分岐点をクリックして、複数の分岐に分かれる時点のロボットの状態を入力します。これは、新規の分岐を実行するたびに、ロボットが戻る状態です。これは、ロボットがサイトをロードして出発都市を3回入力する必要がないことを意味し、時間とCPUパワーを浪費しません。ロボットは、単純にロール・バックして中断したところから続行し、各分岐に対して新しい目的地をフォームに入力します。

このテクニックは、1ページを複数の異なる方法で処理する必要があるたびに使用できます。分岐の最後の部分ではすべて同じステップを使用するため、これらの3つの分岐を再度結合することもできます。

矢印の向きを変更するには、最初に、[Ctrl]を押したまま矢印を選択してからクリックします。次に、矢印の端をドラッグし、接続先のステップにドラッグします。

1つのステップの右側から別のステップの左側にドラッグし、新しい矢印を作成することもできます。

推測どおり、この方法ではかなり独創的なロボット・ツリーを作成できますが、不安になることはありません。実行フローは単純な単一のルールのみで判断されます。

実行が終了ステップに到達すると、実行は直前に到達した分岐点の次の分岐から続行されます。

繰り返して説明します。

実行が終了ステップに到達すると、実行は直前に到達した分岐点の次の分岐から続行されます。

実行が終了ステップに到達すると、実行は直前に到達した分岐点の次の分岐から続行されます。

慣れてしまうと、かなり直観的です。

ここで、伝えておくことがあります。実行フローを管理する別のルールがあり、これまでに説明したルールには1つの例外があります。それはFor Eachループです。

For Eachループには、For Each Tagアクション、For Each Windowアクション、For Each URLアクションなどがあります。

初心者用チュートリアルを終了している場合は、ロボットでFor Each Tagループを使用していたため、その機能を理解しています。ここでは、For Eachループに対する新しい考え方があります。For Eachループ・ステップは、ループの各繰返しが分岐に対応した分岐点と考えることができます。

つまり、実行が終了ステップに達すると、実行は直前に到達した分岐点の次の分岐点、または直前に到達したループ・ステップの次の繰返しのうち、いずれか早いほうから続行されます。

ループは、分岐とともに配置することで非常に効果的に使用できます。

他の側面もあり、ロボットの実行フローを非直線的に作成できます。最も顕著な1つに、エラー処理があります。エラーが特定のステップで発生すると、そのステップのエラー処理は、ロボットの実行をどの地点から続行するかを判断します。これを覚えておいてください。エラー処理の詳細を学習するには、エラー処理タブの右上隅の疑問符をクリックします。

ある分岐から次の分岐まで情報を保持した場合の問題は何ですか。すべての要素がロボット状態に保持されるわけではないため、ロボット状態についてさらに詳しく説明します。たとえば、グローバル変数は、ロボット実行の全体を通じて完全に直線的であるため、ロボットが以前の状態にロール・バックする際は初期の値に戻りません。これは、分岐間またはループの繰返し間で、情報を転送できることを意味します。

ロボットに変数を追加する場合は、「グローバル」チェック・ボックスを選択して、変数をグローバル変数に変換できます。

また、試行ステップは分岐点と似ていますが、同じではありません。試行ステップは、エラー処理によってのみアクティブ化されます。

ループの基本

次に続く内容は、ロボットに基本的なループを実装する方法を示すビデオと、そのビデオのトランスクリプトです。

ビデオ・トランスクリプト:

このビデオでは、ロボット内でのループを紹介します。特に、ブラウザ・ビューから直接アクセスできるループのタイプについて説明します。

ループについては、「初心者用チュートリアル」のビデオと、「分岐、ロボット状態および実行フロー」のビデオの両方で取り上げています。ループの使用経験がない場合は、このビデオの前に、そ

れらのビデオを視聴することをお薦めします。ループは実行の流れを変更する方法であることに特に注意してください。

多くの場合、最も有用なロボットとは、大量のアクションを単純に数多く処理するロボットです。これには、複数の類似する状況での同じ操作の実行が含まれる場合もあります。1つの例としては、初心者用チュートリアルで使用したNewsMagazineロボットがあります。このロボットは、For Each Tagステップを使用して、複数のブログ投稿からテキストを抽出します。For Each Tagステップは、同じロジックが適用される多数のループ・ステップの1つです。それらをなんらかの形ですべて使用して、複数の関連する状況で同じ手続きを実行します。

最も基本的なループ・ステップは、現在のウィンドウのタグをなんらかの形ですべてループするため、For Each Tagループに分類されます。これらの基本的なループを1つずつ説明します。

For Each Tag

このビデオで取り上げる3つのループ・ステップの最初は、For Each Tagと呼ばれる文字どおりのループです。

For Each Tagは、検出されたタグのすぐ内側に指定されている名前の各タグをループします。ループ内の最初のタグは、ソース・ビューのこのスクリーンショットで青色のボックスで示されています。ここで、ループの後続する繰返しを示すために、スクリーンショットに薄い青色のボックスを上に重ねます。

For Each Tagは、柔軟性と使い易さを兼ね備えているため、最も頻繁に使用しているループ・ステップです。初心者用チュートリアル・ビデオで使用したのもループ・ステップです。

このページに表示されている製品のように、選択した類似のタグをループする必要がある場合、最初に試みるのは、必要な最初の要素を右クリックし、ループ >> For Each Tagの順に選択することです。その結果、Design Studioでは、右クリックしたタグに類似したタグをループするFor Each Tagループが設定されます。

For Each Tagステップで矢印を使用してループを繰り返し、そのループによって形成されたすべての指定タグを表示できます。初心者用チュートリアル・ビデオで説明したように、ループで形成された青色のボックスは指定タグと呼ばれ、後続ステップのタグ・ファインダに対する起点として使用されます。このため、最初の製品の価格を抽出するステップを挿入した場合、ループの後続の繰返しでは、それに続く価格が同様に抽出されます。これが、このビデオで考察されているすべてのループの作動方法です。

ただし、vimeo.comのここで示しているように、ループを挿入することが問題となる場合があります。必要な最初の要素を右クリックし、For Each Tagループを選択しても、結果のループには最も上に表示されたビデオのみが挿入されます。これは、次の繰返しに進もうとしたときに確認できます。結果として、ループの最後の繰返しに到達したことを伝えるウィンドウが開きます。

これを修正するには、ループ・ステップの構成に直接移動し、この場合は、ループするクラスの仕様を削除する必要があります。Design Studioでは「最上位」クラスのタグのみが必要と推測されていますが、実際はクラスに関係なく、リストされている項目すべてのループを期待しています。クラス仕様を削除すると、ループは期待どおりに機能するようになります。

For Each Tagを効果的に使用するには、For Each Tagステップを構成する様々な方法を学習する必要があります。

For Each Table RowとFor Each Table Column

次のループは、相互に交換可能な場合が多いため、所定の状況を様々な方法で処理できます。タイプを賢明に選択できるように、各ループ・タイプの基本原則を説明しますが、タイプを選択する正しい単一の方法はありません。

次に示す2つのループ・ステップ・タイプは、両方ともFor Each Tagステップの派生ですが、使用方法はさらに限定されています。これらはFor Each Table RowおよびFor Each Table Columnと呼ばれ、それぞれ表の行および列をループします。For Each Tagステップと同様に、簡単に使用できるように右クリック・メニューに実装されています。

このスクリーンショットの最初の行は、1のマークが付いた指定タグとともに表示され、最初の列は2のマークが付いた指定タグです。このように、2つのループ・タイプを組み合せると、表内のあらゆる要素をループできます。

表の行または列全体にループを挿入するには、表の任意の要素を右クリックし、ループ・サブメニューから適切なアクションを選択します。挿入または除外するために、最初の行または列を選択できます。

最新のIkea家具の各列をループするループを挿入しました。ループ・ステップの名前がFor Each Tagであることに注意してください。表をループする一意のステップではなく、For Each Tagステップは表の列をループするように自動的に構成されています。

For Each Tag Path

For Each Tagループの次のタイプは、For Each Tag Pathと呼ばれます。For Each Tagと非常に類似しているため、簡単に説明します。

2つの相違は、For Each Tag Pathでは検出されたタグの内側にある任意のレベルのタグをループしますが、For Each Tagでは検出されたタグのすぐ内側にあるタグのみをループすることです。

ループするタグのすべてが1つの親タグのすぐ内側にない場合や、ループするタグがすべて異なるレベルにある場合は、For Each Tag Pathループを使用する必要があります。この例では、divタグがtdタグおよびtrタグ内ですべてループされ、検出されたタグのすぐ内側ではないことに注意してください。

For Each TagとFor Each Tag Pathのどちらを使用するかを判断する最も簡単な方法は、ソース・ビューでページの構造を確認することです。

For Tags with Class

For Each Table RowとFor Each Table ColumnがFor Each Tagループの派生であったように、For Tags with ClassはFor Each Tag Pathループの派生です。For Each Tag Pathループの例として、この派生バージョンの使用方法を示します。

ここでは、設定した表の列のループを削除して、For Tags with Classに注目します。このループは、クラス属性の値が同じタグすべてを繰り返しますが、これは、コンテンツが類似するタグに多く使用されます。今回は、右クリックする前に選択するタグが特定されているため、ソース・ビューにも注目してください。

通常、このループを使用すると、次のようになります。各製品が格納されたタグをクリックする際は、ソース・ビューを確認して、すべてのタグにproductContainerという同じクラスが設定されていることに注意します。これを理解した後は、タグの1つを単純に右クリックし、ループ >> For Tags with Class >> productContainerの順に選択します。その後、ループを繰り返して、指定タグが期待どおりになっているかをチェックします。

また、挿入されているステップがFor Tags with Classではなく、単に特定のタスクを実行するように構成されたFor Each Tag Pathであることにも注意してください。

For Each URL

最後に示すループはFor Each URLアクションで、これは単独での分類になります。

For Each URLは、検出されたタグ内の各URLを単純にループします。これは、リンクのコンテキストに関係なく、ページの特定の領域にあるすべてのリンクを抽出またはクリックする必要がある多くの場合に便利です。

For Each URLを最も簡単に挿入するには、ループするリンクが格納されているタグを選択し、選択したものを右クリックしてループ >> For Each URLの順に選択します。

これで、この記事の各URLを繰り返すループが設定されました。デフォルトでは、重複するURLはスキップされます。

For Each URLアクションについてはこのまま残します。For Each URLには、ステップ・ビューで変更可能な多数の構成可能性があることに注意してください。

最後に、ループを使用する際に役立つ2つの注意を示します。

注意1

「分岐、ロボット状態および実行フロー」のビデオで伝えた内容を明確に示すと、For Eachループ・ステップは、このビデオで登場したループ・ステップのように、ループのすべての繰返しに対してロボット・ビューで後続するすべてのステップを実行するため、ループ終了後にロボットでの実行を継続する必要がある場合は、ループ・ステップの前に別途の分岐を挿入する必要があります。この分岐はループ終了後に実行されます。

注意2

特定の条件に基づいてループを中断または繰返しをスキップできると好都合な場合があります。たとえば、ループ内でステップの1つを実行できない繰返しに達した場合は、多く場合、繰返しを完全にスキップすることが論理的です。

「分岐、ロボット状態および実行フロー」のビデオで説明したように、これは失敗したステップのエラー処理を調整することで対応できます。ステップのエラー処理ビューでは、このステップでエラーが発生した場合に、次の繰返しまたはループの中断を選択できます。

デフォルトでは、このオプションは後続ステップのスキップに設定されており、これは、ロボットが現在の実行位置で終了ステップに到達したことに相当します。つまり、このステップでエラーが発生した場合、ロボットは戻って、直前に到達した分岐点の次の分岐を実行するか、直前に到達したループ・ステップの次の繰返しを実行します。ただし、チェック・ボックスで指定されているため、API例外およびエラー・ログの記録も発生します。

ここまでは、単にループの基本です。さらに学習するには、「フォームでのループ」および「Repeat-Nextループ」のビデオを検証してください。また、ループをより深く理解するには、どうぞ、help.kapowsoftware.comにも移動してください。

フォームでのループ

後に続く内容は、フォームに適用するループ・ステップのいくつかを紹介するビデオ・デモと、そのビデオのトランスクリプトです。

ビデオ・トランスクリプト:

このビデオでは、フォームを使用している際に便利なループのタイプを説明します。これは「ループの基本」ビデオの続きで、そこで取得した知識をさらに拡大します。

このビデオで取り上げるループの多くは、「ループの基本」ビデオのFor Each Tagループと似たフォームですが、フォーム・ループはすべて、各繰返しに対して指定タグを割り当てない点でFor Each Tagループとは異なり、かわりに、作業する各要素に対して他のなんらかのアクションを実行します。

フォームでのFor Eachループ

2つのFor Eachループと、フォームで使用するように特に設計された他の1つのループがあります。それらは、それぞれFor Each Radio Button、For Each OptionおよびLoop Field Valuesと呼ばれます。

この図書館の検索ページでは、各フォーム・ループをデモします。

For Each Radio Buttonは、予想に違わない動作、つまり、各繰返しごとにグループ内の1つのラジオ・ボタンを選択します。ステップを挿入する最も簡単な方法は、ラジオ・ボタンを右クリックし、ループ・サブメニューからFor Each Radio Buttonを選択します。ループを繰り返すと、各ラジオ・ボタンが順番に選択されることがわかります。

次のフォーム固有のループはFor Each Optionと呼ばれます。これは、ドロップダウン・リストのオプションをループします。これまでと同様に、右クリック・メニューから簡単に選択できます。選択すると、ドロップダウンのいずれかのオプションをスキップするかどうかを尋ねるウィンドウが表示されます。これは、実行する内容に適切でないオプションをスキップする場合に便利です。ここでは、すべてのオプションをループします。ループ・ステップの矢印をクリックしたときに、各オプションが選択されるようになります。

フォームに関して取り上げる最後のループは、Loop Field Valuesと呼ばれます。これはFor Eachループではないため、これまでに説明した他のループとは若干異なって使用されます。このループは、リストされた値をループするためにテキスト入力フィールドで使用され、フィールドに挿入されます。ループの各繰返しに対して1つの値です。これまでと同様に、単に右クリックしてLoop Field Valuesオプションを選択します。これで、ループする値のタイプを選択できます。事前定義のオプションの1つを選択するか、ロボットで入力する独自の値リストを編集できます。今は、図書館を検索しているので、Hemingway、ShakespeareおよびPoeで構成される独自のリストを編集します。ループを繰り返すことで、記載されている著者がテキスト入力フィールドに入力されていることを確認できます。

これで、「フォームでのループ」に関するビデオは終了します。さらに学習するには、help.kapowsoftware.comに移動してください。

Repeat-Next

後に続く内容は、Repeat-Nextループを紹介するビデオ・デモと、そのビデオのトランスクリプトです。

ビデオ・トランスクリプト:

このビデオでは、高度ですが非常に便利なRepeat-Nextループと呼ばれるループを説明します。このループは特に、各ページが次の原因となるページをループする場合に効果的に機能します。

このビデオは、「ループの基本」ビデオで学習したスキルに基づいているため、そのビデオを確実に視聴しておいてください。

Repeat-Next

このループ・タイプは、ループ・ファミリの中では風変わりで、このループの設計は、Design Studioの他のすべてのループ・ステップとは全体的に異なっているため、ファミリの一部でもないと主張される可能性もあります。

第1に、Repeat Nextループは、あらゆる種類の効果を得るために連携して使用する必要がある2つの個別のステップで構成されます。

概念は実際にはかなり単純で、「次」ステップを配置した後のある地点に「繰返し」ステップを配置します。実行が「次」ステップに達すると、「繰返し」ステップに戻り、そこから実行が進んでループの1回の繰返しがマークされます。「繰返し」ステップの後で「次」ステップに達しない場合、ループは終了します。

ここでの問題と、さらにRepeat-Nextループに卓越した特質がある要因は、ほとんどのロボット状態が各繰返しの開始時に戻る一方、「次」ステップに到達したページは実際にはループの次の繰返しに移動する点です。このため、これまでの説明した、ロボット状態全体が各繰返しの開始時に戻る他のループと異なり、Repeat-Nextループの各繰返しでは異なるページを処理できます。

デモ

Repeat Nextループの典型的な使用例は、複数の検索結果ページをループする場合です。「繰返し」ステップを挿入した後、クリックして次のページに移動するステップを追加し、次に、「次」ステップを挿入します。すべてのページをループする場合でも簡単です。これで、「繰返し」ステップの矢印を使用してループを繰り返し、新しいページがすべての繰返しでロードされていることを確認できます。

当然ですが、まだ、ループを終了する方法は必要です。これは、クリック・ステップのエラー処理をループの中断に設定することで対応します。クリック・ステップで次のページへのリンクが検出されない場合は、最終ページに達したとみなして、ループを中断します。

各ページでなんらかのステップを実行する必要がある場合は、「繰返し」ステップの後に分岐ステップを追加し、新しい上側の分岐を追加できます。この新しい分岐には、ループや他のステップを、他の分岐でのクリックおよび「次」アクションによる影響を受けずに追加できます。たとえば、各ページに対する検索結果すべてのループを追加して、各結果から情報を抽出し、収集した情報を返すことができます。全体としては、検索したすべてのページからの結果をすべて抽出するロボットになります。デバッグ・モードでは、ロボットのシングルステップ実行によって実行フローに関する発想を得られます。すべての検索結果が1ページずつ抽出される様子がよくわかるように、ここでは記録速度を上げています。

ここでは、Repeat-Nextループを使用したフォームを覚えておいてください。上側の分岐では各ページで行うステップを実行し、下側の分岐では次のページをロードして、ループの次の繰返しを呼び出します。このタイプのループを使用する場合、この配置は有用で非常に一般的です。

ステップのエラー処理を次の繰返しに設定しても、Repeat-Nextループでは機能しないことに注意してください。次の繰返しに進むには、「次」ステップを実行する必要があります。

試行ステップ

このビデオでは、試行ステップを使用してロボットに条件とエラー処理を設定する方法について説明します。このビデオはきわめて高度な内容であることに注意してください。

ビデオ・トランスクリプト:

このビデオでは、Design Studioでロボットを作成する際に試行ステップを使用する方法をデモします。試行ステップの適用は、構造が日常的に変更される動的なWeb環境で機能する堅牢なロボットを作成するために重要です。使用事例には、Webサイトと相互作用したり、Webサイトから抽出するための複数の異なる方法の試みと、ロボットでの条件文の設定が含まれます。このビデオでは2つの例を説明し、2つの使用事例をデモしますが、最初は後者から開始します。

最初の例ではExtractFromTable.robotを使用しますが、これはNews MagazineのWebサイトの表からデータを抽出する基本的なロボットです。これはデフォルト・プロジェクトのExamplesフォルダ内のRobotsにあります。ロボットを開き、プロジェクト・ビューを非表示にします。

開いた後は、試行ステップと呼ばれる菱形のステップに到達するまで、ロボット・ビューでたどる経路は1つのみであることがわかります。試行ステップが行うことは、ロボットが試行するための複数の代替を設定することです。試行ステップには、試行ステップの右側から伸びる破線の矢印として表示される複数の代替を指定できます。代替が成功した場合は、試行ステップの他の代替を無視して通常どおりに続行されます。代替が成功しない場合は、試行ステップに戻り、次の代替が試行されます。

したがって、試行ステップを使用するには、代替の「成功」または「不成功」の意味を理解することが重要です。このロボットの例を使用してデモします。

ExtractFromTableは、ループを使用して表から個人情報を抽出する単純なロボットです。for each tagステップをクリックし、ロボットで抽出を行う表が表示されるまで下にスクロールします。表の行をループするこのループ内では、各個人に関する4つの情報が、個人変数の異なる属性に割り当てられています。試行ステップを使用し、表の最終列であるsexの値に基づいて、isMale属性にtrueまたはfalseを割り当てます。列にはMaleまたはFemaleの2つの値のいずれかを指定できますが、isMale属性にはtrueまたはfalseを割り当てます。

sex列の値をテストするために、最初の代替には、タグのテスト・ステップが挿入されています。タグのテストは、検出されたタグが指定のパターンに一致しているかどうかに基づいて、アクションを実行できるステップです。タグのテスト・ステップをクリックすることで、検出されたタグは現

在の繰返しで抽出している行のsex列のセルであること、照合するパターンは単純にmaleに設定されていること、テキストは大/小文字を区別しない照合のみであることがわかります。パターンに一致している場合は、エラー処理タブで指定した処理が実行されます。エラー処理では、ステップが次の代替の試行に設定されており、これがステップの左上隅にある小さいアイコンにも表されています。これは、性別が男性の場合は、そのためにパターンが一致し、タグのテスト・ステップでは、エラー処理に指定された処理が実行されて次の代替が試行されることを意味します。したがって、第2の代替にはisMale属性にtrueを割り当てるステップがあります。ただし、パターンが一致しない場合、タグのテスト・ステップでは何も処理されず、実行は通常のように次のステップに進み、isMale属性にfalseが割り当てられます。

つまり、代替は、次の代替を試行するように指定したエラーが検出されなかった場合は成功であり、そのようなエラーが検出された場合は不成功といえます。1つの代替が成功すると、他のすべての代替はスキップされます。

直前の例では、エラーがタグのテスト・ステップで生成されましたが、通常、エラーはそのアクションを実行できないステップで発生します。ほとんどの原因は、ステップで特定のタグを検出できないか、特定のコンテンツのロードがタイム・アウトになるためですが、ステップでアクションの実行が停止する原因にはあらゆる可能性があります。

例のロボットをデバッグ・モードで実行すると、正しい値がisMale属性に割り当てられたことを確認できます。trueの値はチェック・マークで表示され、falseの値は空欄で表されます。

試行ステップが機能する様子をよく理解するために、さらに複雑で、しかも一般的な使用事例を示します。Design Studioには、サイトにログインして、ログイン時にのみ入手可能ないくつかのデータを抽出する必要があるロボットがあります。このロボットが使用するページは、すでに見慣れているNews Magazineページです。このロボットはLoginと呼ばれ、examplesフォルダから使用できます。プロジェクト・ビューが非表示の場合は、「ウィンドウ」メニューから再度開くことができます。

ロボット・ビューで概要を確認してください。「ログイン」プロセスでのロボットの用途は、サイトにログインして最初の実行でセッションを保存する方法で、セッションを保存および復元することです。連続する実行では、ログイン手続きを再度実行せずに、保存済セッションをロードします。ある時点でセッションが期限切れになると、ロボットはログインして新しいセッションを保存する必要があります。大半のロボット実行は、このようにしてセッションを復元できるため、ログイン手続き全体をスキップできます。

ここに、テスト対象となる3つの異なるシナリオがあります。つまり、ロボットの初めての実行であるために保存されたセッションがない場合、すでにログインした保存済セッションがある場合、または保存されたセッションの期限が切れている場合の3つの異なるシナリオです。

ロボット・ビューには、保存済セッションを使用できるかどうかに応じて、ロボットがたどる2つの代替経路があります。各ケースを示すために、ロボットを介してクリックし、シミュレートしてみます。

ロボットの初めての実行では、保存済セッションはありません。ロボットは試行ステップから上側の代替を試行しますが、セッションの復元ステップを実行すると、復元するセッションがないため、失敗となります。このステップで失敗するとエラーとなり、次の代替の試行に設定されたエラー処理がトリガーされます。したがって、ロボットは2番目の代替を実行する必要があり、この下側の分岐をたどって、News Magazineをロードし、クリックしてログイン・ページに移動し、ユーザー名とパスワードを入力し、ログインをクリックし、ロボットの以降の実行に備えてセッションを保存し、再度クリックして抽出するデータに移動し、最後に特定のテキストを抽出します。これで、ロボットの最初の実行のシミュレートが終了しました。

ロボットの最初のステップに戻り、2回目の実行をシミュレートします。1回目の後のすべての実行では、セッションは正しく復元されるため、上側の代替をたどってセッションが復元され、ロボットがクリックしてデータを抽出するページを開き、最終的にデータが抽出され、単純です。

ここで、ログイン・セッションが期限切れになった後で、ロボットが実行される場合に発生する内容を再現します。そのためには、examplesフォルダからLogout.robotを開きます。ログアウト・ロボットは、保存済セッションを復元し、News Magazineページの「logout」をクリックします。このようにしてログイン・セッションの失効をエミュレートし、この特定のページでセッションがタイムアウ

トして期限切れになるための10分間を実際に待機することはありません。ログアウト・ロボットで終了ステップをクリックし、すべてのステップを実行します。

ログイン・ロボットに戻り、サイト・データのクリックというステップをクリックしてアクティブ・ステップにし、ツール・バーの「リフレッシュ」をクリックしてアクティブ・ステップまでのステップをすべて再実行します。すべてのステップが再実行された後は、News Magazineに引き続きログインしているかのように、ブラウザ・ビューにページが表示されていることに注意してください。これは、セッションが保存された時点のページのhtmlを含め、ロボット状態全体も保存済セッションに格納されているためです。したがって、セッションを復元しても、そのセッションの現在の状態が表示されるため、ページは正しくリフレッシュされません。一方、クリック・ステップを実行するために抽出ステップをクリックすると、ブラウザがリフレッシュされ、すでにログインしていない状態が表示されます。したがって、抽出ステップは必要なデータの抽出に失敗してエラーとなり、試行ステップから次の代替が試行され、ログインおよび以降の使用に備えた新しいセッションが保存されます。

このように、ログイン・ロボットは、3つの状況をすべて処理し、それによって、ログインしないと使用できないデータを抽出する前に、ロボットが常にログインするようになります。

このビデオで視聴した内容を要約すると、ロボットがたどる複数の代替ステップが、1つの試行ステップから生まれています。エラーの原因となるステップがなく、エラー処理タブに次の代替の試行が指定されている場合、代替は成功となります。成功となる最初の代替のみが実行され、その他は完全に無視されます。代替が成功しない場合は、戻って次の代替が試行されます。

次に、試行ステップを使用する際のヒントをいくつか示します。

最初の例で説明したように、名前に「テスト」が付いた条件ステップは、試行ステップとともに効果的に機能します。

試行ステップの代替が成功しない場合は、試行ステップ自体がエラーを生成します。試行ステップ自体のエラー処理タブに、必要なアクションを指定します。

ロボット・ビューで、試行ステップの下をダブルクリックすることで、試行ステップに名前を指定できます。

任意のステップのエラー処理で次の代替の試行を指定している場合は、その名前に基づいて戻る試行ステップを選択することもできます。これは、複雑なロボット構造を可能にするネストした試行ステップを作成できることを意味します。

これで、試行ステップのトピックに関するこのビデオは終了します。

このトピックの詳細は、help.kapowsoftware.comで検索してください。

スプレッドシート・ビュー

このビデオでは、スプレッドシート・ビューの機能を説明します。次に、このビデオのトランスクリプトを示します。

ビデオ・トランスクリプト:

現在のKapow Katalyst 9.2のロボットには、Excelドキュメントを処理するまったく新しい方法があります。Excelは、htmlページに変換せずに、Design Studioのページ・ビューにスプレッドシートとして直接表示されます。このビデオでは、スプレッドシート・ビューの機能について概要を説明し、Excelドキュメントから抽出するロボットの作成プロセスを示します。マシンの説明に従って自由に視聴してください。

Excelドキュメントをロードするロボットの例は、デフォルト・プロジェクトのexamplesフォルダにあり、excel.robotという名前が付いています。

ここではロボットを開きます。Excelドキュメントは、通常のページをロードする場合と同様に、URLから、または各自のマシンまたはサーバーのファイルからロードできます。このロボットの場合は、

リンクをクリックすることでロードされ、これはPersonData.xlsxのクリックというステップによって実行されます。

行のループ・ステップをクリックし、ドキュメントを表示します。ページ・ビューに、Excelでよく使用しているような名前と配列で、行と列があるスプレッドシートが表示されます。このドキュメントには、個人データの表が格納されています。

2つの異なるシートがあり、それぞれ100件ずつ登録されています。この2つのシートは、ページ・ビューの左下隅で切り替えることができます。最も左側のタブをクリックすることで、ドキュメント情報も参照できます。ただし、このビデオではSheet1に専念します。

ドキュメントの書式設定されていない値または未加工の式は、ページ・ビューの右下隅にあるドロップダウン・メニューから表示できます。この表示を変更すると、新しいステップの挿入時に抽出される値にも影響があるため、書式設定された値が表示されるように戻しておきます。

セルはクリックして選択し、クリックしてドラッグすることでセルの範囲を選択したり、行または列の名前をクリックすることで行または列全体を選択できます。左上隅ではスプレッドシート全体を選択します。

ここでは、ロボットからループ・ステップとすべての抽出ステップを削除して、どのくらい簡単にExcelドキュメントから抽出できるかを示します。ステップをドラッグして選択し、キーボードの[Delete]ボタンを押します。

新しいスプレッドシート・ビューには、セルをループして抽出し、テストできる新しい一連のステップ・アクションのまとまりが挿入されます。ブラウザ・ビューの場合と同様に、これらの機能は右クリック・メニューから使用可能で、次に、その操作方法を示します。

表のすべての行をループするループ・ステップを最初に挿入します。最初にスプレッドシート・ビューの左上隅をクリックし、スプレッドシート全体を選択します。次に、選択した領域内、つまり、スプレッドシート・ビュー内のいずれかの場所を右クリックし、ループ>>表の行のループ>>1行目を除外の順に選択します。1行目はヘッダー値で関係ないため、除外しています。

Excelでのループ・ステップでは、ループする1行目が指定範囲として設定されます。ループ・ステップの矢印をクリックすると、別の行が選択される様子が表示されます。ループであるため、これで、指定範囲からの抽出が可能になり、他のすべての行からも対応する値が抽出されます。

最初にIDを抽出するために、IDの値を右クリックし、抽出>>数字の抽出>>「ID」の順に選択します。すでに正しく構成されて、表示されているウィザードで「OK」をクリックします。

同様に、名をテキストとしてname変数に抽出し、年齢を数字としてage変数に抽出し、性別をブールとしてisMale変数に抽出できます。性別値は、男性の場合はtrue、女性の場合はfalseです。

表全体が抽出可能であるたことを表示するために、左上隅のアイコンをクリックし、ツール・バーの「実行」をクリックして、デバッグ・モードに切り替えます。結果リストには、Excelドキュメントから100件の値すべてが表示されます。

データ変換

後に続く内容は、ロボットでデータ変換を実行する方法を説明するビデオです。

ビデオ・トランスクリプト:

ロボットを設計しているときは、単純または複雑な変換ルールを使用したテキストまたは数字の変換が必要になることがあります。問題は、これをDesign Studioでどのようにして簡単に実行するかです。

このビデオでは、次のフィールドについて説明します。

ロボットを作成する際、この機能は平凡に見えますが実際は大変強力なツールであり、Design Studioインタフェースのあらゆるところでポップアップします。

このフィールドはデータ・コンバータ・リストと呼ばれていますが、今のところはブラック・ボックスとみなしてください。このブラック・ボックスは、変数や抽出されたテキストなどから、値の形式で入力を受け入れる場合がありますが、入力は必須ではありません。入力がある場合、その入力はデータ・コンバータ・リストの配置によって指定され、常にリストの上方に記載されます。

データ・コンバータ・リストには厳密に1つの出力があり、その出力は変数に格納されるか、格納されない場合は、リストの下方で処理されます。

要約すると、データ・コンバータ・リストは、ロボットでテキストおよび数字を操作する際に使用されます。ただし、データ・コンバータ・リストではテキストと数字の両方の操作を実行できますが、入力と出力は厳密には常にテキストとして解釈されます。

使用例には、電子メール・アドレスからの名前の抽出、

数字を2倍にする、

所定の日付に3日と2時間を加えるなどがあります。

内部を説明します。

データ・コンバータ・リストという名前が示すように、このフィールドには、プラス・アイコンをクリックして様々な異なるコンバータを追加できるリストがあります。これらすべてのコンバータは、なんらかの方法で入力を操作します。

次に、Kapow Software WebサイトのURLを入力として使用する例を示します。コンバータ・リストにコンバータがない状態で開始しているため、出力は入力と同じです。抽出コンバータを追加することで、URLの中央部を抽出できます。ここでは、コンバータがどのように機能するかについては気にしないでください。それついては後で戻って説明します。

次に、テキストを大文字に変換するコンバータを追加します。

2つのコンバータは、ともに連続してつながり、最初の出力が2番目の入力に流れるように移動します。

コンバータがリストされている順番は、コンバータの実行順序を表します。順番は矢印をクリックして変更できます。この場合、最終結果への影響はありません。

コンバータはマイナスをクリックして削除できます。

ペンと用紙のアイコンをクリックすると、コンバータの構成に使用するウィンドウが開きます。このウィンドウは、コンバータがリストに初めて追加された際にも開きます。

コンバータ構成ウィンドウには、入力のテストおよび出力のテストという2つの重要なフィールドが常に表示され、名前が示すように、指定されたコンバータの入力と出力を示します。

別のコンバータの使用方法を学習するには、構成ウィンドウの疑問符をクリックします。クリックすると、ドキュメントが開きます。ここでは、式とパターンについても確認できます。これらの使用方法を理解することは、データ・コンバータを使用する際の強力なスキルになります。

たとえば、算術演算を使用して数字を操作する場合は、式が必要です。式の評価というコンバータを使用すると、入力を2倍にできます。この操作の実行に使用する式について詳細を説明します。この操作では、入力を数字に変換してから、2を乗算します。コンバータからの入出力はすべてテキストであるため、入力に算術演算を適用する前に、その入力は数字に変換する必要があることを思い出してください。

Design Studioでのデータ・コンバータ・リストの使用について、いくつかの例を示します。

最も有用であることがわかるデータ・コンバータ・リストは、おそらく抽出ステップ・アクションに配置されているリストです。このリストでは、抽出されたテキストを変換してから変数に格納できます。

使用方法の例を示します。

今は、LinkedInからの求人を抽出するロボットを作成しています。現在、このロボットは、単純にページをロードして、仕事の説明が記載されたタグをループしています。各仕事の場所を抽出したいのですが、その場所は、他の2つの情報、具体的には会社名と日付と同じタグ内に記載されています。3つの情報は、単純にハイフンで区切られています。

テキスト全体をlocationという変数に抽出することから着手します。次に、ロボット・ビューで抽出ステップを選択し、抽出コンバータを、抽出ステップのデータ・コンバータ・リストに追加します。抽出コンバータでは、パターンを使用して、入力のどの部分を抽出して、出力として使用するかを判断します。この場合は、1つ目のハイフンと2つ目のハイフンの範囲内すべてを抽出します。

パターンを終了すると、抽出しようとしたテキストが出力のテスト・フィールドに反映されます。

終了ステップを選択してループを繰り返すことで、各仕事の説明から場所が正常に抽出されたことを確認できます。

このように、コンバータは速やかに実装され、非常に効果的です。

最後に、データ・コンバータを使用する際のヒントをいくつか示します。

ヒント1: Design Studioの多くのフィールドは、データ・コンバータ・リストに変更できます。これを行うには、フィールドの右側にある値セレクタからコンバータを選択します。

ヒント2: 最も有用なコンバータの1つは、パターンの置換コンバータです。このコンバータは、式とパターンを組み合せて、強力なテキスト操作を提供します。

ヒント3: データ・コンバータ・リストの制限された1つの入力に加え、式を使用することで追加の変数をフェッチできます。これによって、複数の変数からデータを組み合せることが可能です。

このデータ・コンバータに関する簡単な紹介を視聴いただき、ありがとうございました。

使用可能なコンバータの詳細は、help.kapowsoftware.comのドキュメントで、参照 >> Design Studio>>データ・コンバータの順に選択して参照してください。

パターン

後に続く内容は、パターンとそのパターンのDesign Studioでの使用に関するビデオです。前半は構文に関する講義式のプレゼンテーションで、後半はDesign Studioでのいくつかの使用事例について詳細を示します。ビデオの中で与えられる問題に対する解答は、ページの下部にあります。

ビデオ・トランスクリプト:

こんにちは。このビデオでは、Kapow Katalystでパターンと呼ばれている正規表現について詳細を説明します。ビデオの前半は、ワイルドカード、セット、サブパターン、反復演算子、代替サブパターン、サブパターン参照などの構文に関する講義式のプレゼンテーションです。後半では、Design Studioでの3つの例を介して、パターンを使用した条件の作成、タグ・ファインダおよびデータ変換の実行について説明します。正規表現をすでに理解している場合は、例に直接スキップしてもかまいません。

前述のように、正規表現はKapow Katalystではパターンと呼ばれているため、このビデオの残りの部分でもパターンと呼びます。

ワイルドカード

パターンは、文字列を表現するための記号を使用して、文字列をさらに汎用的な用語で表現する方法です。この概念は、ワイルドカード記号を使用して任意の文字を表現できるコンピュータでの検索か

ら、すでに把握しているかもしれません。'ca*'（この例ではアスタリスクがワイルドカード)の検索を実行すると、"cap"、"car"、"can"などがすべて返ります。パターンでは、この同じ概念を採用し、ここで示すように、さらに広範な構文にその概念を拡張しています。

Kapow Katalystでは、このパターンにPerl5の構文を使用しています。この構文では、ワイルドカード文字が'.'(ドットまたはピリオド)記号で表され、すべての記号、空白、考えられるその他特殊文字を含めた任意の1文字に対応しています。この対応は一致と呼ばれ、そのため、'ca.'というパターンは、たとえば、"cap"、"car"、"can"、または"c"の後に"a"が続き、その後に任意の1文字が続く他の文字列と一致すると言えます。同様に、'.a.'というパターンは、"nap"、"tan"、"sad"、または3文字で中央の文字が"a"の文字列と一致します。ただし、"an"は、パターンの各'.'が厳密に1文字に対応して一致する必要があるため、一致ではありません。同様に、"cans"は、パターンが文字列の一部ではなく全体と一致する必要があるため、一致ではありません。

パターン・エディタを使用すると、パターンが指定の文字列と一致しているかどうかをDesign Studioで直接テストできます。パターン・エディタは、たとえば、タグのテスト・ステップをロボットに挿入し、ステップ・アクション・ビューのパターン・フィールドの下で「編集…」をクリックすると表示されます。パターン・エディタには、3つのセクションがあります。上部にはパターンを入力でき、そのパターンが左側の「入力」フィールドに入力した文字列と照合されます。「テスト」ボタンをクリックするか、[Ctrl]+[Enter]のショートカットを使用すると、パターンが入力と一致しているかどうかどうかがわかります。

パターン・フィールドに'.a.'、「入力」フィールドに"can"と入力してみます。次に、[Ctrl]+[Enter]のショートカットを使用します。「出力」フィールドに、パターンは入力と一致します。が表示されます。ここでは、残りの出力は無視しても差し支えありません。一方、「入力」フィールドに"an"と入力して[Ctrl]+[Enter]を押すと、パターンは入力と一致しません。というメッセージが表示されます。パターン構文をさらに検討するために、パターン・エディタを体験して資料の理解をテストします。

明記されていませんが、これで、2つの異なる方法で一致させることが可能で、文字対文字の一致(つまり、パターン'a'と文字列"a"の一致)か、ワイルドカード記号'.'を使用して任意の文字と一致させることができます。追加の直接的な文字の一致には、この表にリストされているパターンが含まれます。

パターン	一致する文字列
'¥n'	改行文字。
'¥r'	復帰改行文字。
'¥t'	タブ文字。
'¥.'	"."
'¥¥'	"¥"

前にバックスラッシュ'¥'を付けることで、パターン構文で使用される他の記号も明示的に一致させることができます。

セット

次のステップは、候補文字との一致です。候補というのは、文字が文字セットの中の1文字のみと一致することです。文字セットは'[]'（大カッコ)を使用してパターンに記載します。1つの例として、'[abc]'（a、b、cのセット)では、"a"、"b"または"c"のいずれかと一致しますが、この3つ以外の文字とは一致しません。

文字の範囲をセットに指定する場合は、'-'（ダッシュまたはハイフン)を使用します。したがって、'[abc]'は'[a-c]'（aからcの文字セット)で記載できます。'[a-c]'を使用すると、"a"から"c"までの範囲の任意の文字と一致することを意味します。セットを定義する2つの方法を組み合せると、'[a-dkx-z]'（aからd、kおよびxからzのセット)のように指定可能で、これは、'[abcdkxyz]'（それらの文字すべてをセット内に指定)と記述するのと同じで、このように記述しない場合は、"a"から"d"の範囲内、"k"、または"x"から"z"の範囲内のいずれかである文字と一致することを示すようになります。

'[^]'（セットの先頭のキャレット）を使用して、否定のセットを定義することもできます。1つの例として、'[^a-c]'（aからcの否定セット）では、"a"、"b"および"c"を除く1文字と一致します。

パターン・エディタでは、セットを使用して、(1)数字、(2)空白文字、(3)数字でないものとの一致をそれぞれ試行してください。解答を確認する前に、これらの問題について考える時間が必要な場合は、ビデオを一時停止できます(解答はすべてこのページの下部にあります)。

セットに使用できる特定のショートカットがあり、これらはよく使用されます。次の表に、重要なショートカットのいくつかを示します。

簡易形式	セット
'¥d'	'[0-9]'（任意の数字）
'¥D'	'[^0-9]'（数字以外）
'¥s'	'[¥n¥r¥t]'（任意の空白文字）
'¥S'	'[^ ¥n¥r¥t]'（空白文字以外）
'¥w'	'[a-zA-Z0-9_]'（任意の英数字）
'¥W'	'[^a-zA-Z0-9_]'（英数字以外）

簡易形式はセット内にも使用できることに注意してください。たとえば、'[¥d¥w]'にはすべての数字と空白文字が含まれます。

サブパターン

次に、パターン内のサブパターンについて説明する必要があります。文字'a'、セット'[abc]'、エスケープ文字'¥d'、ワイルドカード'.'など、これまで説明してきた各用語は、サブパターンとみなすことができます。あるいは、'()'を使用して他のサブパターンをまとめてグループ化して、独自のサブパターンを作成できます。たとえば、'([ctb]an)'と記述して、'[ctb]an'からサブパターンを作成できます。

これから演算子をいくつか紹介しますが、これらの演算子は後に続くサブパターン全体に機能するため、十分な認識が重要です。

反復演算子

パターンの演算子では、次の表にある演算子の1つを使用して、後に続くサブパターンの反復を照合できます。

反復演算子	意味
'{m,n}'（この場合、n ≥ m）	先行するサブパターンのmからnの間(両端を含む)での反復と一致します。
'{m,}'	先行するサブパターンのm回以上の反復と一致します。

たとえば、パターン'a{1,}'は、文字列"a"、"aa"、"aaa"、または'a'の任意の数の反復と一致します。パターン'([bn]a){3,3}'は、'banana'、'babana'、'nabana'、あるいは後に"a"が続く"b"または"n"のいずれかが3回繰り返された他の文字列と一致します。各自で試してください。

セットに関しては、この表に示すように、最も有用な反復演算子の簡易バージョンもあります。

簡易演算子	対応する記述
'{m}'	'{m,m}'
'?'	'{0,1}'
'*'	'{0,}'

簡易演算子	対応する記述
'+'	'{1,}'

これまでに学習した事項を使用して、(4)任意、(5) "u"なしで綴られた"color"または"u"ありで綴られた"colour"、(6) 4桁の数字との一致をそれぞれ試行してください。解答は次のとおりです(ページの下部)。

よく使用されるパターンの1つに'.*'があり、これはあらゆるもの、つまり、空の場合でも任意の文字列と一致します。

これを拡張して、(7)少なくとも1つの数字を含むテキスト、(8) 1つの数字のみを含むテキストと一致するパターンを見つけ出してください。ここに、必要となる可能性がある構文のリストを示します(ビデオのみ)。

解答で使用している構文は、文字列の中で特定のサブパターンと照合する場合に非常に役立ちます。

代替サブパターン

これまでに代替の文字と一致させる方法を説明しましたが、代替のサブパターンとの一致はどうですか。'p1'から'pN'のNサブパターンがある場合は、'(p1|p2|…|pN)' (ここで示すカッコと縦棒)を使用してこれらのサブパターンの1つと一致できます。たとえば、ここに指定されているパターン'(abc|a{5}|¥d)'は、"abc"、"aaaaa"または任意の数字と一致します。

代替サブパターンを使用して、(9) 1つのみの数字は含まない文字列と一致するパターンの作成を試行してください。ここにも、必要となる可能性がある構文を示します。解答は次のとおりです(ページの下部)。

構文に演算子はありませんが、かわりに、解答では2つの代替を使用しています。最初の代替は数字がない文字列と一致し、2番目の代替は少なくとも2つの数字が含まれる文字列と一致します。

サブパターン参照

構文をカバーする最後の重要な部分は、サブパターン参照です。あるパターンのカッコで囲まれたサブパターン'(p1)'から'(pN)'によって一致した従属文字列"s1"から"sN"は、'¥1'から'¥N'を使用して参照でき、各サブパターンには、パターンに記載された順に左から右に番号が付きます。たとえば、'([chm])(at)'と"cat"の照合では、"c"を参照する際は'¥1'、"at"を参照する際は'¥2'の参照を使用できます。

パターン全体は、常に'¥0'で照合できます。

ここでは、サブパターン自体ではなく、そのサブパターンによって一致する文字列を参照していることに注意してください。当然ですが、サブパターン'(abc)'への参照では'abc'が生じますが、サブパターン'(¥d)'への参照では、当初のサブパターンによって一致した数字のみと一致します。

1つの例として、パターン'.*(['"]).*¥1.*' (何かの後に一重または二重引用符が続き、その後に何かが続き、その後に参照が続き、さらに何かが続く)を使用して、引用符が含まれている文字列の照合を考えてみます。これは複雑に見えますが、実際に注意する必要があるのは、参照は、サブパターンによって一致した同じタイプの引用符と一致するということのみです。つまり、このパターンは、二重引用符を使用したHe said "hello"と一重引用符を使用したHe said 'hello'のどちらの文字列とも一致します。混乱を避けるために、ここでは意図的に2つの文字列に引用符を付加していません。

後述のDesign Studioで示すように、サブパターンは、パターンの外側では特定の式で記載することもできます。これは、一致した文字列の特定の部分を抽出する場合に便利です。引用符の例として、'.*(['"])(.*)¥1.*'のように、引用符で囲まれたサブパターンの周囲にカッコを追加できます。これで、Design Studioで引用符を抽出できるようになりました。

ここで、別の問題です。サブパターン参照を使用して、(10)同じ数字4つ、(11)少なくとも2つの文字が同じ文字列と一致するように試行してください。解答は次のとおりです(ページの下部)。

より少ない反復

サブパターン参照を使用する場合は、次のことを把握していると役立ちます。デフォルトで、反復パターン演算子(*, +, {...})は、可能なかぎり多くの先行パターンの反復と一致します。かわりに、可能なかぎり少ない反復と一致するように、反復演算子の後に"?"を配置できます。

(12)文字列の最初の数字の出現と一致するようなサブパターンを試行してください。解答は次のとおりです(ページの下部)。

'?'を削除した場合は、サブパターンは文字列の最後の数字出現と一致することになります。

Design Studioでのパターンの使用

パターンの構文の学習を終えたところで、Design Studioでの様々な使用事例を説明します。

条件

条件の作成は、ロボットでパターンを賢く使用する最初の方法です。タグのテスト・ステップ・アクションは、このコンテキストに特に関連があるため、一般的な使用事例を検討します。

ここに1つのロボットがあり、デンマークを対象にエンジニアリングのすべての求人がリストされたLinkedInから抽出を実行します。このロボットは、ループを使用して各求人からURL、タイトルおよび会社名を抽出し、その情報をユーザーに返します。しかし、ここでは、おそらくコペンハーゲンではソフトウェア・エンジニアを探していることを示すために、Copenhagenおよびsoftwareという語が含まれた求人に限定して抽出する必要があるとします。

最初に、For Eachステップの後に新しいステップを挿入して、その新しいステップをクリックして選択し、ステップ・アクション・ビューのドロップダウンからタグのテストを選択することで、そのステップにタグのテスト・アクションを割り当てます。タグ・ファインダで、ループの現在の繰返しから求人投稿全体を検索するようにします。次に、設定している基準と一致する求人を検出するまでループを繰り返します。これによって、記述したパターンが実際に機能することを簡単にテストできるようになります。

ステップ・ビューで「アクション」タブに移動し、最初にテキストのみ(HTML全体ではなく)と照合するように選択し、パターンで「編集」を押します。パターン・エディタが表示され、一致するパターンを入力できます。softwareおよびCopenhagenという2つの語が現れる順序は不明なため、2つの代替サブパターンを作成する必要があります。最初の代替では、Copenhagenの後に何かが続き、その後にsoftwareが続くように指定しました。2番目の代替では、同じことを逆順で記述しました。最後に、代替の前後に任意のテキストを追加し、[Ctrl]+[Enter]を押してパターンが一致するかどうかをテストします。一致しました。

パターン・エディタを閉じ、タグのテストステップを、検出されたタグとパターンが一致しない場合はその後のステップをスキップするように設定します。これによって、指定された2つのワードがない求人投稿がスキップされます。

先に進み、デバッグ・モードでロボットを実行します。予想したとおりに少数の結果のみが抽出され、そのすべてにSoftwareおよびCopenhagenの語が含まれています。

タグ・ファインダ

パターンは、タグ・ファインダにも使用できます。これは、検索する情報の構造は把握しているが、その情報がページのどこに記載されているかを把握していない場合に大変役立ちます。たとえば、このロボットは、複数の異なるサイトに移動して、特定のヘッドフォンの価格を抽出します。ページのどこに価格があるのかを把握できないため、これを正確に判断するためにパターンが重要な役割を果たします。

抽出ステップの設定方法を示します。表示されているステップを削除して、新しいステップを挿入し、そのステップに対して抽出アクションを選択します。ステップを構成するためには、抽出するテキストから数字を抽出する数字コンバータを挿入することから開始します。次に、このロボット用に作成した変数のprice属性に抽出するように選択します。

ステップ・ビューでファインダ・タブに移動し、プラス記号をクリックしてタグ・ファインダを追加します。ページで価格を確認します。価格はそれ自体のタグから何もない状態で隔離されていることがわかります。この場合のパターン一致には、これが典型的です。ファインダ・ビューに、タグ・パターンというフィールドがあります。パターン'¥$[¥d¥.]+' (ドル記号の後に1つ以上の数字またはドットが続く)を直接記述できます。パターンは、ドル記号とその後に続く小数のみが含まれるタグと一致するように設計されています。ページ・ビューの右上隅にある拡大鏡をクリックすると、タグ・ファインダによって検出される内容が表示されます。残念なことに、ヘッドフォンの価格ではなくカート残高が検出されました。この種のサイトではカート残高は常に$0であるため、この間違いを回避するために、タグの最初の数字がゼロでないようにします。ヘッドフォンの法外な価格の場合、幸い価格がゼロで開始することはありません。パターンを'¥$[1-9][¥d¥.]+' (ドル記号の後に、ゼロでない数字が続き、その後に1つ以上の数字またはドットが続く)に書き直し、拡大鏡をクリックしたときに正しい価格がページに表示されるようにします。

ロボットをテストする前に、抽出ステップのエラー処理タブに移動し、エラーに対して無視して続行を選択します。タグ・ファインダによるページでの価格の検出に失敗した場合は、price属性のデフォルト値である-1を設定して単純に返すようにします。これによって、ロボットが価格を検出できなかったことが明確にわかります。デバッグ・モードに移動し、このロボットの初期の実行結果で、多くの価格が正しく抽出されていることを確認します。当然、この方法には不備がありますが、以外にも効果的な場合があります。

データ変換

パターンの最終的な用途は、あるフォームから別のフォームにデータを変換することです。それには、ステップに埋め込まれたデータ変換リストの1つを使用するか、専用の変数の変換ステップを使用できます。

この非常に単純な例では、ブログ投稿から著者と日付を抽出します。残念なことに、2つの情報はテキストの同じ文字列に格納されているため、抽出ステップではまとめて抽出されます。データ・コンバータでパターンを使用して、この2つの情報を切り離す方法を示します。

抽出ステップには、ステップ・アクション・ビューに配置されたデータ・コンバータ・リストがあります。抽出されたテキストは、変数に割り当てる前にデータ・コンバータ・リストを使用して変換できます。プラス記号をクリックして抽出を選択し、文字列の一部を抽出できるデータ・コンバータを挿入します。新しいウィンドウが開き、そこで抽出データ・コンバータを構成できます。パターン・エディタの場合と同様に、上部にはpatternがあり、下部には入力のテストおよび出力のテストがあります。抽出コンバータの要点は、入力文字列全体が一致するパターンを記述し、次にカッコを使用して抽出するサブパターンを指定することです。デフォルトでは、文字列全体を一致させて抽出すると、入力と出力の文字列は同一になります。

抽出された文字列から一部を除外する場合は、その内容をサブパターンの外側に記述する必要があります。サブパターンの前に'.* by ' (任意のテキストの後に空白、b、y、空白が続く)を配置します。文字列全体が一致しますが、著者の名前のみは従属文字列の一部であるため、著者名は出力のテスト・フィールドに表示されているように抽出されます。プレーン・テキスト' by 'によって、'.*' (任意のテキスト)の2つのインスタンスがそれぞれ日付と著者名に一致するようになります。

これで、構成ウィンドウを閉じて、抽出ステップを実行できます。著者名が変数に正しく割り当てられています。

抽出ステップに戻り、パターンを使用するもう1つのコンバータをデモします。抽出を削除し、かわりに拡張抽出コンバータを追加します。次に、両方の'.*' (任意のテキスト)インスタンスからサブパターンを作成する以外は、前に使用した抽出と同じパターンを記述します。出力のテストは、依然として入力のテストと同じです。これは、拡張抽出では、出力式フィールドにサブパターン参照を使用することで、抽出する必要があるサブパターンを選択できるようになっているためです。

式では、'$'記号の後に参照番号を指定して使用することで、サブパターン参照が作成されます。式は、一致するパターン全体を参照しますが、この式を'$1'に変更すると、日付を抽出する最初のサブパターンのみが取得され、'$2'を記述すると、2番目のサブパターンによって一致した著者名のみが取得されます。

式フィールドを使用すると、テキストを追加し、サブパターンを結合して、単純な文字列操作を実行することもできます。たとえば、2つの従属文字列を逆順に再結合する式を記述できます。式の詳細は、式フィールドの横にある疑問符をクリックしてください。

最後に、文字列の指定のパターンのインスタンスを置換するパターンの置換データ・コンバータもお薦めします。

パターンに関する説明はこれで終了します。ビデオの有用な部分は自由にレビューし、さらに多くの解答を確認するには、help.kapowsoftware.comを参照してください。

問題の解答

問題番号	解答
(1)	'[0-9]'
(2)	'[¥n¥r¥t]'
(3)	'[^0-9]'
(4)	'.*'
(5)	'colou?r'
(6)	'¥d{4}'
(7)	'.*¥d.*'
(8)	'¥D*¥d¥D*'
(9)	'(¥D*|.*¥d.*¥d.*)'
(10)	'(¥d)¥1{3}'
(11)	'.*(.).*¥1.*'
(12)	'.*?(¥d).*'

スニペット

後に続く内容は、スニペットを使用してロボット間でステップを共有する方法を説明するビデオです。

ビデオ・トランスクリプト:

複雑なロボットを作成すると、事柄が即座に雑然となり、貴重な画面資産を占有することがあります。複数のグループへのステップの配置は、そのようなロボットを整理するための簡単な方法で、最終結果に導くステップを単純化します。

ステップをグループ化するには、グループ化するステップを単純に選択し、ロボット・エディタの上にあるツール・バーの「グループ」ボタンを押します。

作成する際は各グループに名前を指定できます。これによって、グループの具体的な機能を簡単に把握できるようになります。

複雑なロボットの独立した部分を単純なグループに集めることで、自分自身および他のユーザーがロボットをより簡単に理解できます。

グループに名前を付けることは、内部を思い出す上で重要です。たとえば、サイトにログインし、複雑な変換を実行し、値をデータベースに格納したり、オンラインで何かを参照するグループを設定できます。

考えてみると、実際に自分のロボットの1つには、他のいくつかのロボットでも使用するログイン・グループがあります。当然ですが、このグループは自分の他のいくつかのロボットに対してコピー・

アンド・ペーストを開始できます。これで当面の問題は解決しますが、今後ログイン手続きを変更する必要が生じた場合は、変更したグループ・ステップを自分のすべてのロボットに再度コピーする必要があります。

スニペットの概念を紹介します。スニペットはグループ・ステップとして作成および編集されますが、個別のファイルに格納され、ロボットではカスタム・ステップとして必要な数だけ使用できます。

ロボットからファイルを切り離してステップ情報を格納することは、1つのロボットでスニペットを変更すると、他のロボットもすべてそのスニペットが変更されることを意味します。これによって、多くの時間が節約され、ロボットが大幅に単純化されます。

ステップの格納に加え、スニペットには、そのスニペットによって使用される変数のリストも格納できます。

たとえば、ユーザー名とパスワードを格納するログイン変数です。この変数は、このスニペットを使用するすべてのロボットで自動的に使用可能になります。

スニペットの作成

このすべてがDesign Studioでどのように機能するかを示します。

ここに、Zoho CRMにログインして、なんらかの連絡先情報を参照し、その情報を抽出して返すロボットがあります。このロボットのログイン手続きをグループ化し、Zohoでタスクを実行する予定がある他のロボットにも役立つように仕上げます。

ログイン・シーケンスを実行するステップからのスニペットの作成は、これより簡単にはなりません。すでに作成したグループを単純に選択します。次に、ロボット・エディタの上のツール・バーからスニペットへのグループの変換を選択します。

Design Studioに名前の入力を求めるプロンプトが表示され、すでにグループの名前である**LoginZoho**という名前を選択します。

スニペットが作成され、ボックスの左下隅の小さいスニペット・アイコンとして表示されます。

さらに、新規に作成されたスニペット・ファイルがプロジェクト・ビューで選択され、スニペットがスニペット・エディタの新しいタブに開きます。スニペット・エディタは、ロボットでスニペットを変更するたびに開きます。

LoginZohoスニペット・エディタに切り替えます。スニペット・エディタは、ロボット・エディタの外観とよく似ています。スニペットのステップを表示するスニペット・ビュー、スニペット・ビューのアクティブ・ステップによって実行されるアクションを表示するステップ・ビュー、およびロボット・ビューの際に理解した変数ビューがあります。スニペット・エディタには、ブラウザ・ビューではなく構成ビューがあり、スニペットの説明を記述できます。

スニペット・エディタでは、変更は構成ビューと変数ビューに対してのみ可能であることに注意してください。スニペットのステップを変更する場合は、ロボット・エディタのコンテキストで操作する必要があります。

スニペット・ビューが表示されているときに気付いたように、Login変数を使用する2つのステップにはエラー・インジケータのマークが付き、変数がスニペットに存在していないことを示します。

したがって、スニペットを完成するために最後に必要な操作は、Login変数をスニペットに追加することです。これによって、このスニペットを使用するあらゆるロボットで、ログイン資格証明が使用できるようになります。

これは、ロボットでの操作と同様に、変数ビューを単純に右クリックし、適切なタイプを選択することで追加します。表示された変数構成ウィンドウで、変数をロボットの変数と正確に一致するように構成する必要があります。変数名は正しいため、入力として使用を選択し、属性にデフォルト値を追加した後、「OK」をクリックします。

これで、完全に機能する**LoginZoho**スニペットになりました。

スニペットの使用

スニペットを別のロボットに追加する方法を示します。ここに、Zohoでなんらかのタスクを実行する必要がある新しいロボットがあります。スニペット・ステップは、ツール・バーからスニペット・ステップを前に挿入を選択することで挿入されます。

ロボットに未定義のスニペット・ステップが表示されます。次に、スニペット・ステップのステップ・ビューのドロップダウンから**LoginZoho**スニペットを選択します。

スニペットがロボットに挿入され、終了ステップをクリックすると実行されます。

変数ビューに表示されているように、スニペットからのログイン変数が、ロボットに使用できるようになっています。変数の左側にある小さいスニペット・アイコンは、この変数がスニペットに属していることを示します。これは、スニペット・エディタ以外では変数を直接編集できないことを意味します。これは、Loginスニペットをロボットから削除すると、変数も削除されることも意味します。

変数がロボット内に永続的に存在する必要がある場合は、その変数をロボットに手動で追加し、スニペットからのlogin変数と同じ名前、タイプおよび構成を指定する必要があります。新しい変数がスニペット変数のかわりになります。

いずれかのインスタンスでスニペットを変更した場合は、他のすべてのインスタンスも変更されます。たとえば、ユーザー名の入力ステップとパスワードの入力ステップの位置を入れ換え、最初のロボットに戻ると、ここでも2つのステップの順序は同様に変更されて表示されます。

ヒント

このビデオの終了にあたって、スニペットを使用する際のヒントをいくつか示します。

ヒント1

結果として、該当するロボットに必要なすべてが格納された大量の変数を保持するロボットとなる可能性があります。スニペットを作成する際は、そのスニペットに必要な属性のみがスニペット変数に含まれるように、これらの変数を最終的に分割する必要があります。

通常は、ロボット固有にタイプを作成するのではなく、機能に基づいたタイプの作成を試行してください。

ヒント2

スニペットが可能なかぎり残りのロボットから独立するように、デフォルトでないロボット構成を、必要なスニペットのステップに直接配置してください。

つまり、ここをクリックして、ここでスニペットのステップに重要な変更を加えた場合は、スニペットのステップのステップ・ビューで、ここをクリックして同じ変更を適用する必要があります。

ヒント3

スニペット構成ビューに説明を記述することで、スニペットのコンテキストを常に文書化してください。

このビデオはこれですべてです。スニペットについて詳細を学習する場合
は、help.kapowsoftware.comでドキュメントを参照できます。

日付抽出 – 単純な場合

後に続く内容は、記事の日付を抽出するためにNews Magazineロボットを変更する方法を示すビデオと、そのビデオのトランスクリプトです。

ビデオ・トランスクリプト:

Kapow Katalystを使用していただき感謝します。

このチュートリアルでは、ロボットの日付抽出ステップを使用して、Webサイトから日付を簡単に抽出する方法を示します。チュートリアルは2部構成です。最初の部分は単純な例に沿って視聴するチュートリアルで、2番目の部分はより慎重を要する例を示す実例のシナリオです。

コンピュータの説明に従って、最初の部分について自由に視聴してください。

初心者用チュートリアルを完了している場合は、News Magazineロボットを覚えている可能性があります。このロボットは、チュートリアル用に作成されたサイトであるNews Magazineから最新のニュースを抽出します。最新の記事から日付と時間も抽出するように、ロボットを変更します。

Design Studioを起動し、デフォルト・プロジェクトのBeginner Tutorialsフォルダからpostというタイプを開きます。dateという新しい属性を追加し、タイプに「日付」を指定します。ここで、タイプを保存してタイプ・エディタを閉じ、同じフォルダからNewsMagazineロボットを開きます。

ロボット・ビューで値を返すステップをクリックします。このステップが実行されます。プロジェクト・ビューとソース・ビューを閉じて、さらに大きい作業スペースを確保します。

ブラウザ・ビューで下にスクロールし、各写真の上にある日付と時間を確認します。日付と時間を3つの各記事から抽出するステップを追加します。

そのためには、時間と日付が含まれているタグを右クリックし、抽出 >> 日付の抽出の順に選択し、次に変数post.dateを選択します。

新しいウィンドウが開きます。このウィンドウは、「日付」という単純なタイプ、つまり、post.date変数が受け入れる唯一の書式である標準日付書式による日付の抽出に役立ちます。

日付の抽出の構成ウィンドウの重要な部分について説明します。入力のテスト・フィールドには、抽出されたままの未処理のテキストが表示されます。パターンというフィールドには、抽出したテキストに記載されているとおりに日付のパターンを記述します。現在のパターンは「dd MM yyyy」で、これは日、月および年を空白で区切って日付が構成されることを意味します。

これが、抽出したテキストの日付書式に対応していることを確認します。パターンはDesign Studioによって推定されているため、日付を単に抽出する場合は何も変更する必要はありません。出力のテスト・フィールドには、入力のテスト・フィールドから抽出された日付が標準日付書式で表示されます。

ここでは、さらに、各記事が公開された正確な時間も抽出する必要があるとします。これは、この情報も同様に取得するにはパターンを拡張する必要があることを意味します。入力のテストに注目しながら、パターンに「 * hh:mm」を追加します。時間を取り込むように、出力のテストが変化したことに注意してください。

追加したパターンについて説明します。空白とコロンは、入力のテスト内のそれぞれの文字に対応しています。アスタリスクは、入力のテストのatに対応していますが、一般的には空白以外の任意数の文字を表すために使用されます。hhは時間を、mmは分を意味します。パターン・フィールドに配置する事項に関する完全なリストを取得するには、ウィンドウの右上にある疑問符をクリックします。

「OK」をクリックします。これで、各記事から時間と日付が正常に抽出されます。デバッグ・モードでロボットをテストして確認します。

日付抽出 - 慎重を要する場合

後に続く内容は、日付情報が複数の場所に点在している場合に日付を抽出する方法を示すビデオ・デモと、そのビデオのトランスクリプトです。

ビデオ・トランスクリプト:

このビデオでは、日付情報が複数の場所に点在しているサイトで日付と時間を抽出する方法を示します。このビデオは、単純な日付抽出のチュートリアルを完了してから視聴することをお薦めします。

このロボットはSkypeの呼び出し履歴を抽出するために作成されました。このロボットは、Skypeにログインし、履歴が格納されている表をループします。唯一の問題は、日時の列に年の記載がなく、上方にある青色のバーにのみ記載されていることです。なんとかして、2つの情報を結合する必要があります。幸運にも、これはコンバータを使用して簡単に実行されます。

ループに入る前に、上方のバーのテキストをYearという変数に抽出するステップを追加します。ループ内に、新しいステップを挿入し、そのステップを選択して抽出アクションを実行します。次に、表の1行目で日付タグを選択し、アドレス・バーの右側にある黄色い正方形のボタンを使用して、その日付タグを抽出ステップで使用します。

この抽出をYear変数と結合するために、抽出ステップの「アクション」タブの下にコンバータを追加します。式の評価というコンバータを選択します。このコンバータを使用すると、変数Yearから抽出したデータに値を追加できます。

式の評価の構成ウィンドウが開きます。入力のテスト・フィールドには、抽出された状態の日付が表示されます。「式」フィールドには、単純にINPUTと大文字で記述します。これが抽出された日付です。この後にプラス記号と二重引用符で囲んだ空白を入力し、これによって日付の後に空白が追加されます。次に、もう1つプラス記号を追加した後、以前に抽出した年が格納された変数を表すYearを入力します。

出力のテストで、1つの抽出が1つのテキストに結合されていることを確認します。式の詳細を学習するには、「式」フィールドの横にある疑問符をクリックします。「OK」をクリックします。

ここで、日付の抽出というもう1つのコンバータを追加します。これは、単純な日付抽出のチュートリアルで使用したコンバータと同じです。前のコンバータの出力が、このコンバータの入力として使用されます。

日付の抽出コンバータの構成ウィンドウが開き、新しい「形式」パターンを挿入し、Design Studioによって入力されたデフォルトのパターンを削除します。ここで、単純な日付抽出のチュートリアルの場合と同様に、入力のテストから日付を抽出し、コンバータで日付を標準日付書式に変換するように、パターンを追加します。MM dd hh:mm MM yyyyです。月が2回指定されていますが、それは問題ではありません。最初の出現が単純に無視されます。

「OK」をクリックし、Date変数への抽出を選択し、日付と時間が正常に抽出されたことを確認します。

これで、日付の抽出に関するこのチュートリアルとデモは終了します。

第8章 Kapow Compute Units (KCU)

9.1ではKapow Katalystの使用許諾に大きな変更があり、CPUコア・ベースの価格設定からKapow Compute Units (KCU)に基づいた能力ベースの価格設定に移行し、選択したハードウェア構成とはまったく無関係になりました。

KCUの概要

KCUはKapow Compute Unitの略で、Kapow Katalyst RoboServerが1秒間に実行可能な操作数(またはステップ数)の単位として定義されています(基礎となるサーバー能力とは無関係)。

ステップはRoboServer内で実行可能なアクションの最小単位です。ステップの例には、Webページのロード、データベースへのデータ・レコードの書込み、データ要素に対する変換の実行などがあります。

1つのKCUは、1秒当たり合計5000 KCUポイントを表します。1つのKCUを構成するKapowステップの数は、各ステップ・タイプごとに異なる量のKCUが消費されるため、関係するKapowステップのタイプによって異なります。ステップは複数のグループに分割され、ここには、最重要グループがリストされています。

1. JavaScriptのI/Oと実行の両方を行うステップは10000 KCUポイント
   例: 4 KCUで1秒当たり2ページのロード

2. JavaScriptのI/Oまたは実行のいずれかを行うステップは1000 KCUポイント
   例: 4 KCUで1秒当たり20回のREST Webサービスの呼出し

3. 抽出および変換のステップは1 KCUポイント
   例: 4 KCUで1秒当たり40,000の抽出または割当ステップ

(完全なリストは、Design Studioで「ヘルプ」->KCU情報の順にメニュー項目をクリックして入手できます)

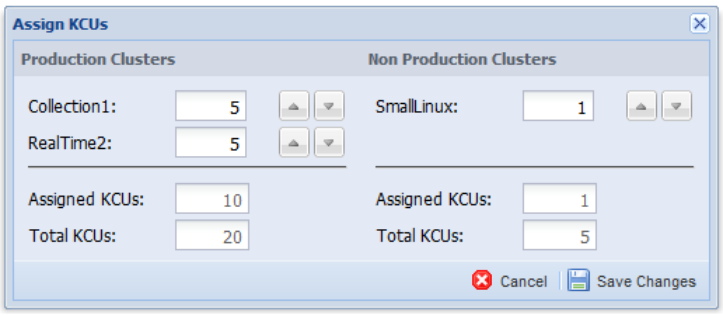
前述の内容は、十分なCPUパワーがあり、ソース・サーバーからの応答時間が十分に小さい場合です。(注意: 最も訪問数が多い23000のWebサイトについて、強力なCPUでの平均ページ・ロード時間は経験的に6.7秒と測定しています)。

ロボットで使用するKCU合計数は、ロボット実行後、Design Studioデバッガ・サマリー情報で確認できます。

KCUのデプロイと割当て

KCUを使用すると、ターゲット・ハードウェア環境や物理的なCPUとは関係なく、必要なコンピューティング能力を確保できます。また、これによって、CPUの割当数を制御せずに、クラウドベースまたはオンプレミス、仮想または物理的なCPU環境を柔軟に選択できます。

使用可能なKCUは、Management ConsoleのRoboServerクラスタに割り当てられ、クラスタ内で使用可能なRoboServer間に自動的に配分されます。



第9章 ライセンス・キー

Kapow Katalystインストールの使用を開始するには、「Management Console（ライセンス・サーバー）」の説明に従って、ライセンス・キーをManagement Consoleに入力する必要があります。これらのキーはKapowから受け取ることになります。

異なる3種類のライセンス・キーがあります。

本番キー　Kapow Katalystシステムの本番利用が認められます。

非本番キー　テストやステージングなどの本番以外の用途でのKapow Katalystシステムの使用が認められます。

開発者シート・キー　特別な種類の非本番キーで、自分のコンピュータ上のすべてのKapow Katalystプログラムを同じインストールの一部として実行できます。ただし、開発専用の用途または試験的なインストールを意図しているため、パフォーマンス上の制限がいくつかあります。

また、ライセンス・キー(種類に関係なく)には、保有しているKapow Katalystの機能、および購入したKCUの数も記載されます。開発者シート・キーには常に1 KCUが含まれ、他の2つの種類にはさらに多く含まれている可能性があります。

本番キーと非本番キーの両方を保有している場合は、それらを同じKapow Katalystシステム(つまり、同じManagement Console)にインストールできます。各キーに1つのシステムを設定するように選択することもできます。いずれの場合も、それぞれ本番および非本番として構成された少なくとも2つの異なるクラスタを設定する必要があります(単一のシステムで、または2つの各システムに1つのクラスタ)。これで、これらのクラスタにライセンスのKCUを割り当てることができます。

いずれの場合も、1つのキーを複数のManagement Consoleに入力しないでください。

第10章 Design Studioユーザーズ・ガイド

Design Studioは、ロボットおよびタイプを作成するためのアプリケーションです。また、Design Studioでは、ロボットをデバッグしたり、データベースに格納する必要がある各種タイプのデータベース表を作成できます。

Design Studioは、ロボット開発のための統合開発環境(IDE)です。これは、Design Studioを使用すると、ロボットやタイプを十分に設計できることを意味します。

ロボットは、独自の構文(構造)とセマンティック(意味)を使用して、わかりやすいビジュアル・プログラミング言語でプログラムされています。ロボットの構築をサポートするために、Design Studioは、インタラクティブなビジュアル・プログラミング、完全なデバッグ機能、プログラム状態の概観、コンテキスト依存のオンライン・ヘルプへの簡単なアクセスなど、強力なプログラミング機能を備えています。

Design Studioでは、データの抽出や入力のためのロボット変数で使用されるタイプも作成できます。Design Studioのタイプ・エディタを使用すると、実際のデータの後にモデル化されるタイプを設計できます。ほとんどの場合、タイプは、ロボットがデータ・ソースから抽出するデータを保持するように設計されます。

構成

このガイドは次のように構成されています。最初に、Design Studioの重要な概念を説明します。次に、ユーザー・インタフェースのツアーに進み、ロボットのコアとなる構成要素の概要を説明します。基礎をしっかりと理解した上で、Design Studioを使用して有用なロボットの作成方法を説明するチュートリアルに進みます。このチュートリアルは少しずつ高度になり、最後には、ユーザーが決定したタスクを実行するロボットを作成できるようになります。チュートリアルはこのユーザーズ・ガイドの中核で、作業を始める前にチュートリアルをマスターすることが重要です。

ガイドの残りの部分は、様々な「...する方法」というトピックに分かれています。簡単にこれらに目を通して、どのような内容が説明されているかを把握しておいてください。後で必要になったときに、該当するトピックの情報を読み返すと役立ちます。

読み始める前に

始める前に、2章「概要」の項を読むことをお薦めします。この項では、Design Studioと使用されるコンテキストを紹介し、Kapow Katalystのインストールについて説明して、次に2つの基本的なチュートリアルに進みます。

また、始める前に、読者が次の要件を満たしていることを確認してください。

- プログラミングの基本的な理解。
- HTMLの基本的な理解。
- JavaScriptの基本的な理解。

その他のリソース

Design Studioに関する追加情報は、Design Studioの「ヘルプ」メニューから簡単にアクセスできる参照ドキュメントから入手できます。

Design Studioの概念

Design Studioは、ロボットを作成したりタイプを設計するためのプログラミング環境です。ロボットは、独自の構文とセマンティックを使用する、特別な目的のプログラミング言語を使用して作成さ

れます。他のプログラミング環境と同様に、Design Studioでは、ロボット設計者がDesign Studioの機能を完全に把握するために理解する必要があるいくつかの概念が使用されています。この項では最も重要な概念を説明しているため、理解できない概念に直面したときはいつでもこの項を参照することをお薦めします。ここで説明するDesign Studioのすべての概念をすぐには理解できなくても、Design Studioを体験してロボットの作成を開始するうちに明確になっていきます。

ロボット

Design Studioの最も重要な概念はロボットです。ロボットは、データ・ソース(Webサイトが一般的ですが、Excelドキュメントやデータベースの場合もあります)に関連するタスクを実行するように設計されたプログラムです。通常、1つのデータ・ソースに対する1つのタスクごとに、1つのロボットを記述します。たとえば、http://cnn.comからニュースを抽出するロボット、http://yahoo.comからニュースを抽出するロボット、オンライン製品カタログから製品情報を抽出するロボットなどを作成します。

基本的に、ロボットは、ユーザーがブラウザで実行可能なすべての作業を(自動的に)実行するようにプログラムでき、データベースやExcelドキュメントからデータを抽出して、そのデータをデータベースやファイルなどに保存されているデータと結合することもできます。

ロボット状態

ロボットは、実行されるときにロボット状態に従って実行されます。ロボット状態は、主に次の4つの要素で構成されます。

- ウィンドウ
- 変数
- Cookie
- 認証

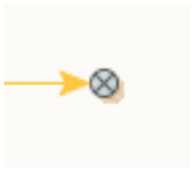
ウィンドウ要素とは現在開いているウィンドウで、各ウィンドウに1つのページが含まれます。このページはHTMLページ、スプレッドシート、XMLページなどがあり、ウィンドウにロードされるページのタイプに応じてページにページ・タイプが指定され、ページ・ビューの外観およびロボットに挿入できるステップはこのタイプによって決まります。少なくとも1つのウィンドウが常に開いており、1つのウィンドウが現在のウィンドウとしてマークされます。変数要素には、変数の現在の値が含まれます。Cookieと認証要素はそれぞれHTTPのCookieと認証で、Webサーバーとの通信時に受け取ります。

ステップ

ロボットはステップで構成されます。ステップはロボット・プログラムの構成要素です。ステップには、アクション・ステップ、試行ステップ、グループ・ステップおよび終了ステップの4タイプがあります。

ステップ

ステップはロボット状態に従って実行され、ステップの構成に応じて処理されます。したがって、ステップには入力ロボット状態があり、出力ロボット状態が生成されます。これの唯一の例外は終了ステップです。終了ステップは、ロボットの終了ではなく、ロボット内の分岐の終了をマークします(つまり、ロボットは必ずしも終了ステップ後に実行を終了するわけではありません)。ロボットのステップの中で出力接続がないのは終了ステップのみです。



終了ステップ

ステップには、ステップ名、タグ・ファインダのリスト、ステップ・アクション、エラー処理などのプロパティを設定できます。アクション・ステップにはこれらすべてのプロパティがあり、その他のタイプのステップには一部のプロパティのみあります。

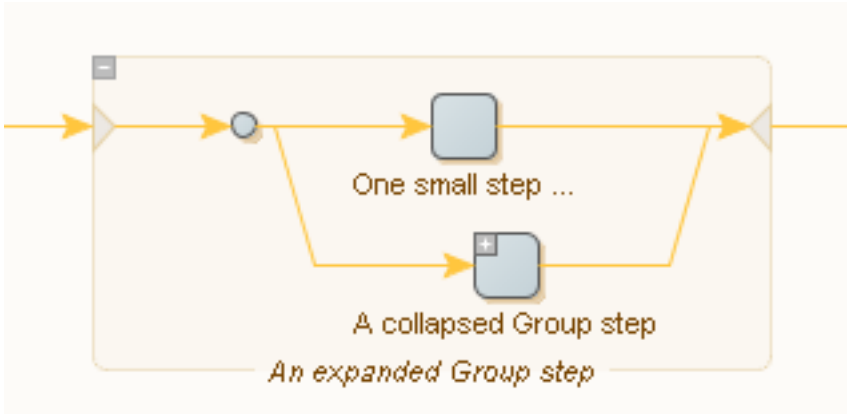
ステップ名は、「ヘッドラインの抽出」や「検索ページのロード」のように、そのステップを象徴する名前です。前述のロボットのステップ名は「MyStep」です。

ファインダは、ステップ・アクション(後述)が実行されるページの要素(HTML/XMLタグ、Excelセルなど)を検索します。ステップ・アクションによっては、1つの要素が必要な場合や、複数の要素を処理できる場合があります。要素を受け入れないステップ・アクションもあります。ファインダには、HTMLまたはXMLページでタグを検索するタグ・ファインダ、およびExcelページでセルを検索する範囲ファインダの2種類があります。

ステップ・アクションは、ステップが実行するアクションです。アクションはステップの中核で、ロボットを記述する際は適切なステップ・アクションを選択することが重要です。たとえば、抽出アクションは、HTMLページのタグからテキストを抽出して変数に格納します。クリック・アクションは、<a>タグに含まれるURLをロードして、ロボット状態の現在のウィンドウのページを、新たにロードされたHTMLページに置換します。通常は、アクションによってロボット状態が変更されます。たとえば、抽出アクションによって変数が変更され、クリック・アクションによってページ/ウィンドウ、Cookieおよび認証が変更される場合があります。

ステップは実行できます。実行されるステップは、ロボット状態を入力として受け取り、ファインダおよびステップ・アクションを適用して出力ロボット状態を生成します。その出力ロボット状態は次のステップに渡されて、そのステップの入力ロボット状態になります。ループ・アクションを呼ばれるステップ・アクションがあります(このようなアクションを含むステップはループ・ステップと呼ばれます)。ループ・ステップによってゼロまたは1つ以上の出力ロボット状態が生成され、それぞれについて次のステップが1回実行されます。

ステップはグループ・ステップにグループ化できます。グループ・ステップは展開または縮小できます。次の図に、展開されたグループ・ステップ、およびその内部にある縮小されたグループ・ステップの例を示します。グループ・ステップの左上隅にあるアイコン⊞と⊟を使用して、グループ・ステップを展開または縮小します。



グループ・ステップ

ステップは、実行を試行できるように適切に構成された場合に有効です。たとえば、ステップにアクションがない場合は、実行を試行できないため無効です。

さらに、ステップ定義ではステップのエラー処理も指定しますが、これは複雑であるため、後の項で別途説明します。

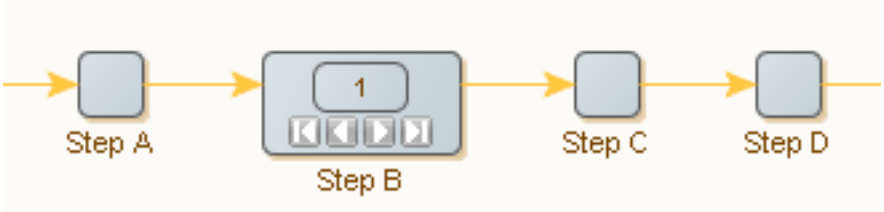
接続および実行フロー

ステップは接続を介して他のステップに接続でき、接続はステップ間の実行のフローに影響を与えます。次の単純なロボットを考えてみます。

このロボットは、Step A、Step BおよびStep Cという3つのステップで構成されています。エラーが発生せず、各ステップで出力ロボット状態が1つのみ生成されると仮定すると、このロボットは次のように実行されます。当初のロボット状態が生成され、そのロボット状態がStep A（最初のステップ）への入力として使用されます。Step Aで出力ロボット状態が生成されます。この出力ロボット状態がStep Bの入力ロボット状態になります。同様に、Step Bでロボット状態が生成され、これがStep Cの入力ロボット状態になります。Step Cが実行されて出力ロボット状態が生成されると、実行が完了します。要約すると、このステップの実行は、「A、B、C」のように記述できます。

実行時に、ステップで出力ロボット状態が生成されない場合があります。これが発生するのは、エラーが発生した場合、またはテスト・ステップがロボットの他の場所で継続して実行される場合です（「条件およびエラー処理」の項を参照）。

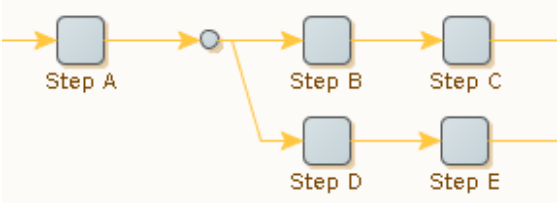
ループ・アクションが含まれるステップでは、入力状態が複数回生成され、毎回異なるロボット状態が出力される場合があります。Step Bにループ・アクションが含まれる次のロボットを考えてみます。



エラーやテスト・ステップが発生せず、Step Bで3つのロボット状態が出力され、他のすべてのステップではロボット状態が1つのみ出力されると仮定すると、ステップは「A、B[1]、C、D、B[2]、C、D、B[3]、C、D」の順に実行されます(B[N]は、Step Bに含まれるループ・アクションのN回目の繰返しを示します)。Step Bで生成される出力ロボット状態は異なるロボット状態、つまり、繰返しごとに新しいロボット状態が出力されることに注意してください。したがって、Step Cは、実行されるたびに新しいロボット状態を受け取ります。

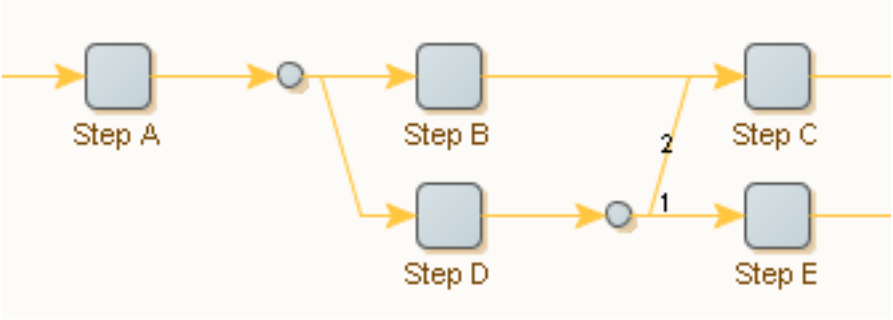
分岐、ロボット状態および実行パスに関するビデオ・チュートリアル

1つのステップが複数のステップに接続できます。これは分岐と呼ばれます。次のロボットを考えてみます。



このロボットは、Step Aの後に分岐点が続き、接続が2つの分岐に分かれています。1つの分岐はStep BとStep Cで構成され、もう1つはStep DとStep Eで構成されています。分岐点から後のすべての分岐は次々に実行されます。したがって、制御フローが変わるようなエラーまたはテスト・ステップが発生せず、各ステップで出力ロボット状態が1つのみ生成されると仮定すると、このロボットは「A、B、C、D、E」の順に実行されます。ただし、Step BとStep Dはそれぞれ、Step Aで生成された出力ロボット状態の同じコピーを受け取ることに注意することが重要です。

分岐は、複雑な方法でマージできます。次のロボットを考えてみます。



このロボットは、接続を明示的に順序付ける方法を示しています。このロボットでは、Step Dの分岐は番号で指定された順序で実行されるため、Step EはStep Cの前に実行されます。順序が番号で指定されていない場合、接続はトップダウンで実行されます。したがって、テスト・ステップがなく、エラーが発生せず、各ステップで出力ロボット状態が1つのみ生成されると仮定すると、このロボットは「A、B、C、D、E、C」の順に実行されます。Step Cは1回目に実行されるときにStep Bで生成された出力ロボット状態を受け取り、2回目に実行されるときにStep Dで生成された出力ロボット状態を受け取ります。

状況に応じて、複数の分岐の中から1つのみを選択(実行)する場合があります。次の項では、この方法について説明します。

条件およびエラー処理

ロボットは、様々なケースで異なるアプローチをする必要があります。ケースは、明示的なテスト(つまり、条件の評価)に基づく場合と、エラーが発生して処理が必要になる場合とで区別できます。

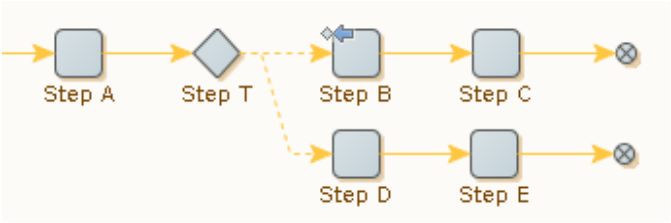
入力ロボット状態の内容(HTMLページ内の特定タグの有無など)に基づき、条件によって実行のフローが変更されます。エラー処理の目的は、特定のエラーが発生したときに(たとえば、HTMLページに必要なアンカー・タグが見つからないためにクリックできない場合)、実行のフローを変更することです。ただし、多くの場合、状況は2つの方法で判断され、アンカー・タグが見つかった場合はクリックされ(これが条件です)、見つからない場合はロボットがアンカー・タグのクリックを試みてエラーを処理します。条件として一般的と考えられる場合でも(たとえば、「この特定のページがエラーなしでロードできた場合」という条件)、複雑すぎて記述できない場合があります。このような場合は何も実行されませんが、ページのロードは試行され、条件が失敗したことを示すエラーとして処理されます。

その他のエラーは、ロボットまたはアクセス先のWebサイトに実際に問題が発生したサインです。たとえば、Webサイトが停止してページのロード・エラーが発生した場合や、HTMLページのページ・レイアウトが大幅に変更されたためにタグ・ファインダが必要なタグを検出できない場合です。要約すると、特定のエラーは、状況によって、条件の失敗または実際のエラーとみなすことができます。この解釈はロボットによって決まります。

条件による実行とエラー処理は区別が明確でないため、Design Studioには両方の機能が統合された方法で用意されています。すべてのステップは、エラーが発生した場合にどのように処理するかを構成できます。さらに、(なんらかの条件に基づく)テスト・アクションを含むステップでは同じアプローチを再利用し、条件が満たされない場合、(デフォルトの)アクションはエラー発生の場合と同様に動作します。

ロボット内のステップごとに、エラーに対する任意の対応を構成できます。ここでは、最も有用な2つのエラー処理オプションについて説明します(その他のオプションについては、「エラーを処理する方法」を参照)。最初に説明するオプションは、試行ステップに密接にリンクされています。

試行ステップは、このステップから複数の分岐が出る場合があるため、分岐点に似ています。最初の分岐から先にある分岐は、前の分岐上のステップでエラーが発生した場合のみ実行され、次の代替の試行オプションを使用して処理されるため、分岐点と異なります。通常の各ステップはロボット状態を1つのみ出力すると仮定して、次のロボットを考えてみます。



アイコンは、Step Bが次の代替の試行によってエラーを処理するように構成されていることを示します。

Step Bが正常に実行されると、ステップの実行は「A、T、B、C」の順になります。Tから出る最初の分岐はエラーなしで実行されたため、2番目の分岐は何も実行されません。

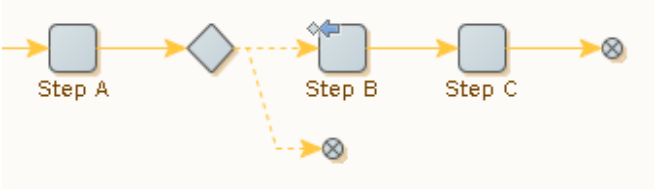
これに対して、Step Bでエラーが発生すると、ステップの実行は「A、T、B、T、D、E」の順になります。Step Bでのエラーが処理された後、実行は後続のステップからではなく、試行ステップから出る次の分岐の開始から続行されます。

試行ステップから出る各分岐は、その位置から進むことができる1つの方向を表しています。各分岐の開始近くのステップは、分岐に沿った実行が可能なアプローチかどうかを調査し(そうでない場合は次の代替の試行に影響します)、現在のケースに対して正しい分岐であることが判明した場合は後続のステップが実際の処理を実行します。分岐の開始近くの調査ステップは、テスト・ステップの場合や、(ステップでエラーが発生した場合に)この分岐は進む方向でないことを示すステップの場合があります。試行ステップからは、このような分岐が複数出る場合があります。

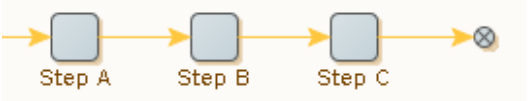
Java、JavaScript、C#などの一般的なプログラミング言語を理解している読者は、前述のロボットが「if-then-else」構造に似ていることに気づくはずです。試行ステップ後の最初の分岐には条件(「if」パート)と「then」パートが含まれ、最後の分岐には「else」パートが含まれています。複数の分岐がある場合、最初と最後の分岐の間にある分岐は「else-if」パートに似ています。

最初の分岐が、エラー発生の可能性があるアクションを実行する場合、この例は「try-catch」構造にたとえることもできます。最初の分岐は「try」パートで、2番目の分岐は「catch」パートに似ています。

別のエラー処理オプションである後続ステップのスキップを使用すると、次のロボットの例に示すように、一般的な特殊ケースをコンパクトに表すことができます。エラーが発生する可能性があるステップは最初の分岐の最初のステップで、2番目の分岐は何も処理しません。



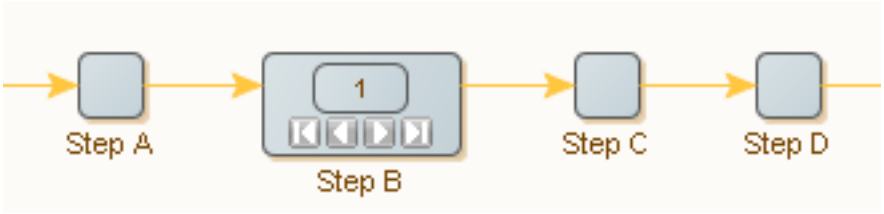
この結果、Step Bでエラーが発生した場合は、後続のステップの実行が単にスキップされます。次のように、エラー処理オプションである後続ステップのスキップ(デフォルト)を使用すると、試行ステップを使用しなくても同じ結果になります。



位置および位置コード

エラーが処理されるとき、ロボットの呼出し側にエラーをレポートしたり、エラーを記録できます。いずれの場合も、エラーを簡単に説明し、エラーが発生したステップの位置および位置コードを示すメッセージが含まれます。

エラーが発生したステップの位置とは、最初のステップからそのステップに達するまでに実行する必要があるステップ(繰返し番号を含む)のリストです。次のロボットを考えてみます。



Step Bの2回目の繰返しで、Step Cでエラーがレポートされた場合、その位置は「Step A - Step B[2] - Step C」と記述されます。位置には、ステップ名と繰返し番号がハイフンで区切られて含まれていることに注意してください。分岐点は省略されます。

位置コードは位置と似ていますが、各ステップの名前がそのステップの一意識別子に置換されるため、名前の衝突を回避できます。前述の位置の例の場合、位置コードは「{a-i1-a}」になります。Design Studioで位置コードを使用すると、エラーがレポートされたステップに直接移動できます(「編集」メニューの位置に移動を使用)。

位置および位置コードの繰返し番号は0に索引付けされるため、最初の繰返しは「{a-i0-a}」になることに注意してください。

スニペット

スニペットは、複数のロボットで再利用できるステップのグループです。スニペットは、ロボットとは別に、ファイル内で保守されます。あるロボット内でスニペットの内容が変更されると、同じスニペットを使用する他のロボット内でも自動的に更新されます。スニペットは、スニペット・ステップを使用してロボットに挿入され、インラインで編集されます。スニペットの内容は、ロボットに挿入されないかぎり編集できません。詳細は、次のビデオを視聴または読んでください。

ビデオによるスニペットの概念の紹介。

ロボット内のスニペット・ステップは、多くの点でグループ・ステップに似ています。ただし、グループ・ステップ内のステップはロボットの一部であるのに対し、スニペット・ステップ内のステップは別のファイルで保守され、同じプロジェクト内の他のロボットで再利用できます。ロボットが参照するスニペットがプロジェクト内に存在しない場合、ロボットは不完全で実行できません。

変換するステップを選択した後、選択からスニペットを作成アイコンを使用して、選択したグループ化可能なステップを簡単に再利用可能なスニペットに変換できます。単一のグループ・ステップのみを選択した場合は、グループへのスニペットの変換アイコンによってそのグループが再利用可能なスニペットに変換されます。スニペット・ステップを選択した後、グループへのスニペットの変換アイコンをクリックすると、スニペットをロボットに簡単に埋め込むことができます。

スニペットには一連の変数も定義でき、そのスニペットを使用するロボットの変数セットに含められます。

スニペットに説明を指定できます。これはスニペット・エディタで編集され、ロボット内でそのスニペットが出現するたびに表示されます。

変数およびタイプ

Design Studioには、変数とタイプの2つの重要な概念があります。変数を作成するときは、そのタイプを選択する必要があります。タイプには、複合タイプと単純タイプの2種類があります。複合タイプでは、一連の属性を定義します。これは、複合タイプの各変数に複数の(指定された)値があることを示します。通常、複合タイプの変数(例: Book)の各属性(例: title)は個別の変数として参照し、その値は完全修飾された属性名を使用して表します(例: Book.title)。複合タイプは、ニーズにあわせてDesign Studio内から作成できます。単純タイプでは属性を定義しませんが、単一値のタイプのみを表します。したがって、単純タイプの変数には単一値(例: テキスト文字列)が含まれ、その変数

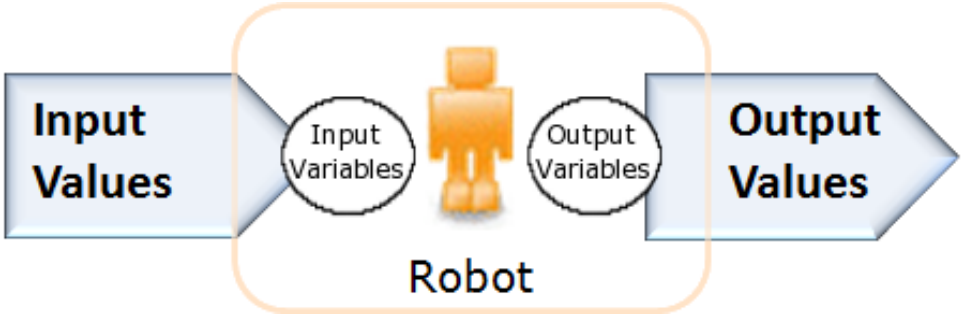
名(例: Username)によってのみ参照されます。単純タイプは組み込まれているため、編集はできず、作成することもできません。

すべての変数はデフォルトの初期値に関連付けることができ、その初期値はロボットが値を明示的に再割当するまで(実行時に値が抽出されて操作されるときに多く行われます)保持されます。ほとんどのロボットは、値を呼出し側に返したりデータベースに挿入するなどして、変数の値を出力します。また、ロボットは入力値を取ることができ、この入力値は、入力から値を受け取るとマークされた特定の変数に割り当てられます。これは、単に入力変数と呼ばれます。

複合タイプの変数と単純タイプの変数の重要な違いは、単純タイプの変数は入力変数として使用できず、その値を出力できないことです。ただし、これらは、一時データを抽出したり、グローバル・カウンタとして使用する場合などに便利です。通常、単純タイプの変数は一時変数としてロボット内部で表示されます。

複合タイプの変数の値は、様々な方法で出力できます。たとえば、Webサイトからニュースを抽出するロボットはニュース変数の値を出力し、各ニュース変数は「headline」、「bodyText」、「date」、「author」などの属性を持つ複合タイプで、出力される各ニュース値は指定された各属性の一意(可能な場合)の(サブ)値で構成されます。

入力変数を含むロボットの場合、指定の入力値が割り当てられた入力変数はロボットへの入力の一部として指定される必要があります。たとえば、http://amazon.comで書籍を注文するショッピング・ロボットは、ユーザーおよび書籍情報を含む入力値に依存します。これらは、ロボット内で、「User」および「BookInfo」タイプの「user」および「bookInfo」という2つの入力変数に割り当てられます。次の図に、ロボットが入力値を受け取り、出力値を生成する方法を示します。



この図は、ロボットの入出力を示しています。入力値が入力変数に割り当てられ、一部の変数の値が出力されます。複合タイプの変数のみ、入力から値を割り当てたり値を出力できます。

ライブラリおよびプロジェクト

ロボットおよびタイプはライブラリに編成されます。ライブラリとは、含まれているロボットの実行に必要なロボット定義、タイプ定義およびその他のファイルの集合です。ライブラリは、ロボットのデプロイメント単位として機能します。これは、RoboServerなどのランタイム環境にロボットを配布してデプロイするときに、ライブラリによってロボットや必要なファイルがまとめられることを意味します。

Design Studioで作業を行うときは少なくとも1つのロボット・プロジェクトで作業しますが、Design Studioではいつでも複数のロボット・プロジェクトを開くことができます。ロボット・プロジェクトの目的は、ロボット・ライブラリを開発することです。ロボット・プロジェクトには、指定のロボット・セットを開発中のロボット・ライブラリ、およびロボット・ライブラリでの作業に役立つその他のファイルが含まれます。ライブラリに配置されたファイルには、特別なライブラリ・プロトコルを使用してロボットからもアクセスできます。

したがって、ロボット・プロジェクトはロボットの開発時に作業する対象で、ロボット・ライブラリは作業を配布してデプロイするために使用します。

Design Studioの概要

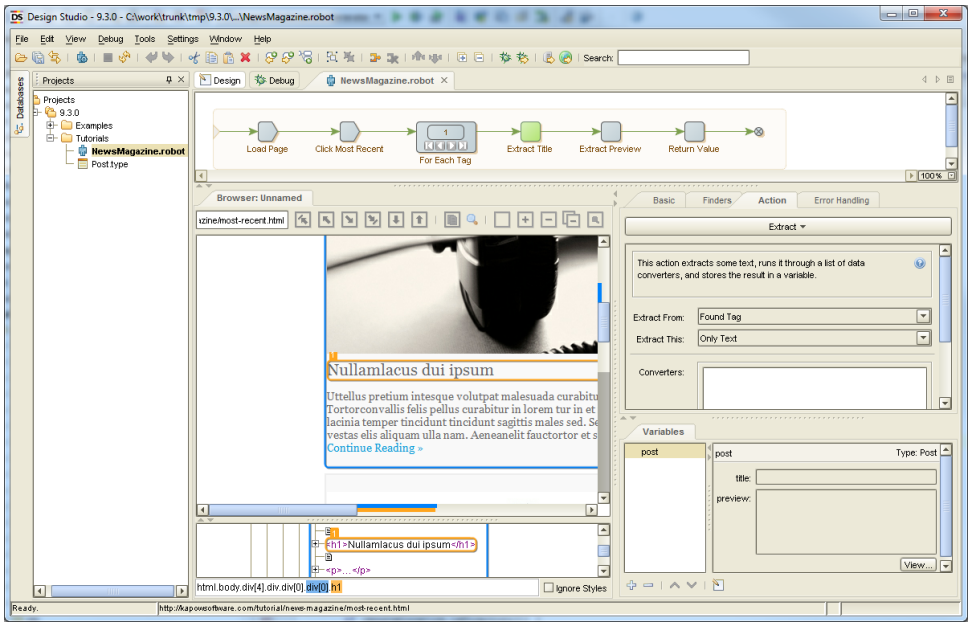
この項では、Design Studioの概要を説明します。ここでは、選択したメニューと機能、ロボット・エディタおよびタイプ・エディタを含む、Design Studioのユーザー・インタフェースについて説明

します。次に、ロボットのコアとなる構成要素であるステップ・アクションとデータ・コンバータについて説明します。最後に、パターンと式に関する項があります。

この項を読むときは、Design Studioを起動し、ツアーに従ってDesign Studioのユーザー・インタフェースを体験することをお薦めします。ただし、ツアーでは、ロボットを作成またはロードしていない場合の起動時に表示されるDesign Studioのユーザー・インタフェースを体験します。つまり、ユーザー・インタフェースが完全に空であるため、デバッグなどの多くの機能はあまり意味がありません。しかし、この項に続くチュートリアルを開始すると、作業中に多くの機能を体験する機会が十分にありますので心配しないでください。

Design Studioのユーザー・インタフェースのツアー

有効なライセンスを入力すると(ライセンス情報の入力方法の詳細は、「インストール・ガイド」を参照)、次に示すようなDesign Studioのメイン・ウィンドウが表示されます。

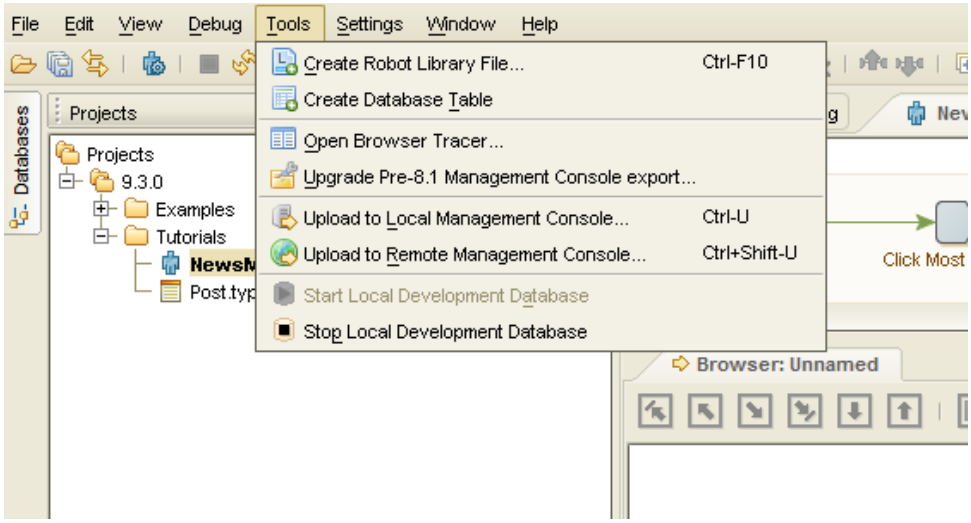


Design Studioのメイン・ウィンドウ

Design Studioの上部には、メニュー・バーとツール・バーが表示されます。左側はプロジェクト・ビューで、この右側がエディタ・ビューです。次の各項では、これらを詳細に説明します。

メニュー・バー

メニュー・バーは、Design Studioウィンドウの最上部にあります。



Design Studioメニュー・バーの例

使用可能なメニューとその項目は、エディタ・ビューで開いているファイルの種類によって異なります。ただし、次のメニューは、ファイルが開いていない場合でも常に使用可能です(一部の項目が無効な場合があります)。

- 「ファイル」メニューは標準的なファイル・メニューで、ファイルやプロジェクトなどを操作するための項目が含まれます。

- 「オプション」メニューを使用すると、Design Studioの一部のデフォルト設定を変更したり、Design Studioで使用するプロキシ・サーバーやデータベース接続を定義できます。

- 「ウィンドウ」メニューには、ユーザー・インタフェースのレイアウトを変更する項目(レイアウトのリセットなど)が含まれます。

- 「ヘルプ」メニューには、オンライン参照ヘルプ、ドキュメントおよびサポートへのアクセスに関するヘルプへのリンクが含まれます。

ロボットなどのファイルを開くと同時に、使用可能なメニューに「編集」メニューが追加されます。

- 「編集」メニューには、開いているファイルに対して実行できる幅広い編集アクションが含まれます。使用可能なアクションはファイルのタイプによって異なりますが、「元に戻す」とやり直しアクションは常に含まれます。

タイプまたはロボット・ファイルを開くと、さらにもう1つのメニューが使用可能になります。

- 「ツール」メニューには、データベース表の生成(タイプの場合)、ロボットのManagement Consoleへのデプロイメント(ロボットの場合)など、ファイルのタイプに関連する様々なタスクを実行するための項目が含まれます。

最後に、ロボット・ファイルを開いたときは、さらに3つのメニューが使用可能になる場合があります。

- ビューに対する様々なアクションを実行したり、デフォルトでは開いていない追加ビューを開くための項目が含まれる「ビュー」メニュー。

- デバッガに関連するアクションが含まれるデバッグ・メニュー(ロボットの現在位置でのデバッガの起動など)。

- デバッガのブレークポイントに関連するアクションが含まれるブレークポイント・メニュー(ブレークポイントの追加や削除など)。このメニューが使用可能になるのは、ロボット・エディタがデバッグ・モードの場合のみです。

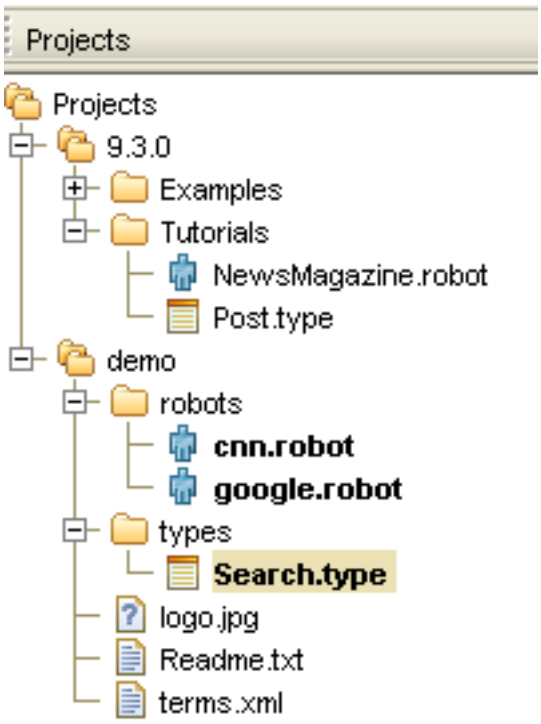
多くのメニュー項目には、キーボードを使用して同じアクションを実行できるようにキーボード・ショートカットがあります(たとえば、[Ctrl]-[D]は、ロボット・エディタで設計モードとデバッグ・モードを切り替えます)。キーボード・ショートカットは、メニューの項目名の右側に表示されます。

ツール・バー

Design Studioツール・バーの例

ツール・バーには、メニューを使用する場合と同じ多くのアクションを実行できるボタンが含まれます。使用可能なボタンも、エディタ・ビューでアクティブなエディタに応じて変わります。多くのボタンにはキーボード・ショートカットがあります(たとえば、[Ctrl]-[D]は、ロボット・エディタで設計モードとデバッグ・モードを切り替えます)。これを確認するには、特定のボタンの上にマウス・ポインタをしばらく重ねてツール・チップを表示します。特定のボタンにキーボード・ショートカットがある場合は、それがツール・チップに表示されます。

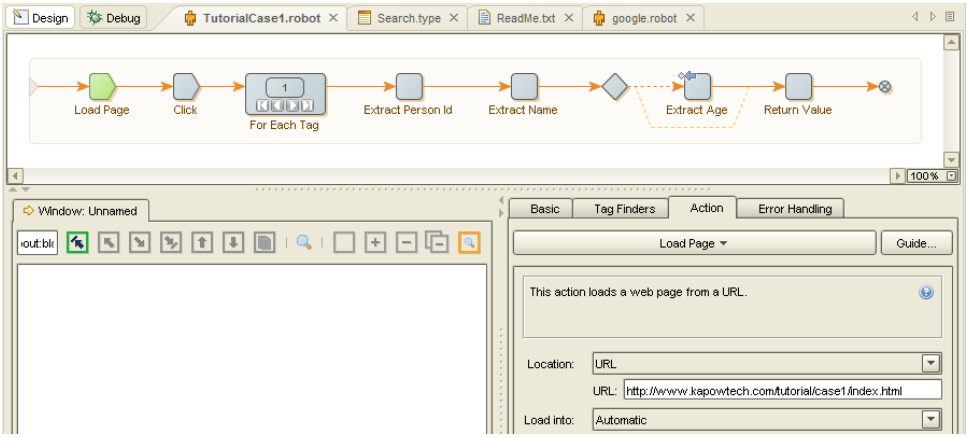
プロジェクト・ビュー



プロジェクト・ビュー

プロジェクト・ビューは、Design Studioのメイン・ウィンドウ内のツール・バー・アイコンの左下に表示されます。プロジェクト・ビューには展開/縮小可能なツリー構造が表示され、Design Studioで開いているロボット・プロジェクトが表示されます。このツリー内の「+」または「-」をクリックすると、対応するサブツリーを展開または縮小できます。プロジェクト・ビューには、開いている任意の数のロボット・プロジェクトを表示できます。プロジェクト・ビューにはポップアップ・メニューが用意されており、フォルダ内に新しいロボットを作成したり、以前に保存したロボットを開くなど、様々なアクションを実行できます。

エディタ・ビュー



エディタ・ビューで開いているロボットの例

エディタ・ビューは、ロボットおよびタイプを編集する場所です。複数のエディタを同時に開くことができますが、表示されるエディタは1つのみです。エディタはエディタ・ビューの上部にタブとし

て表示され、これらのタブをクリックして別のエディタに切り替えることができます。次の3種類のエディタがあります。

- 「ロボット・エディタ」では、ロボットを編集します。
- 「タイプ・エディタ」では、1つ以上の属性で構成される複合タイプを編集します。
- 「テキスト・エディタ」では、プレーン・テキスト・ファイルを編集します。

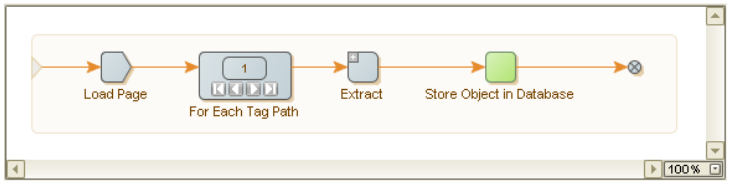
ロボット・エディタ

ロボット・エディタは、名前が示すように、ロボットを編集するときに使用するエディタです。ロボットを開くと、新しいロボット・エディタにロボットが開き、エディタ・ビューの新しいタブに配置されます。ロボット・エディタには設計とデバッグの2つのモードがあり、エディタは常に設計モードで開きます。エディタのモードを選択するには、ロボット・エディタの左隅にある2つのモード・ボタンのいずれかをクリックします。選択したモードに応じて、ビューが変更され、(ツール・バーやメニューなどの)使用可能なオプションが変わります。

モードごとに、ロボット・エディタのビューを構成する複数のサブビューがあります。設計モードの場合は、ロボットが表示されるロボット・ビュー、ロボットの現在のステップ前のページを表示するウィンドウ・ビュー、現在のステップの構成を表示するステップ・ビュー、ロボットで使用される変数の現在値を表示する変数ビューがあります。

次の項では、設計モードのビューについて説明します。デバッグ・モードについては、別の項で後述します。

ロボット・ビュー



ロボットが表示されたロボット・ビュー

ロボット・ビューは、ロボット・エディタの上部にあり、タブのすぐ下に表示されます。ロボット・ビューには、ロボット・プログラム、つまり、ロボットを構成するステップと接続が表示されます。ロボット・ビューでは、ロボット内をナビゲートし(ステップの選択など)、その構造を編集します(ステップの削除、移動または接続など)。次に、この方法について簡単に説明します。

現在のステップ

現在のステップ

ロボット・ビューには、現在のステップという概念があります。基本的な概念は、作成中の部分的なロボットを作成しながら実際に実行することです。現在のステップによってこの実行の位置がマークされ、Studioでこの状態がページ・ビューおよび変数ビューに表示されます。現在のステップは緑色でマークされます。ステップを左クリックすると、ロボットがそのステップまで実行され、可能な場合はそのステップが現在のステップになります。ロボットの実行中は左クリックしたステップが黄色で表示され、実行がそのステップまで進むと緑色になり、新しい現在のステップになります。クリックしたステップまで実行が進めなかった場合(HTMLページがロードできなかった場合など)、新しいステップまでの途中で到達した最後のステップで実行が停止し、このステップが新しい現在のステップになります。ロボットがすでに実行したステップをクリックすると実行は発生しませんが、新しいス

テップは即時に新しい現在のステップになります。ステップ・ビューで構成するのは、常に現在のステップです。

現在の実行パス

もう1つの概念は、現在の実行パスです。これは、ロボットの実行で現在のステップに達するまでに通過したパス、およびロボット内のある分岐の終了に達するまでロボットがこれから通過するパスです。現在の実行パスは、接続が濃い色でマークされます。接続をクリックするとその接続がパスに含まれ、現在の実行パスを変更できます。

選択

選択済ステップ

複数のステップまたは接続を選択するには、[Ctrl]キーを押しながらステップまたは接続を左クリックします。また、左マウス・ボタンを押しながらステップをドラッグしてマークすることもできます。現在選択されているステップや接続の選択を解除するには、ロボットの外部をクリックします。

ステップまたは接続が選択されると、それらにアクションを適用できます。たとえば、選択したステップの後に新しいステップを挿入するには、最初に、後に新しいステップを挿入するステップを選択し、次にツール・バーの  アイコンをクリックします。また、ステップまたは接続を右クリックすると、ポップアップ・メニューが表示されます。まだ選択されていないステップまたは接続でこの操作を行うと、そのステップまたは接続は自動的に選択されます。

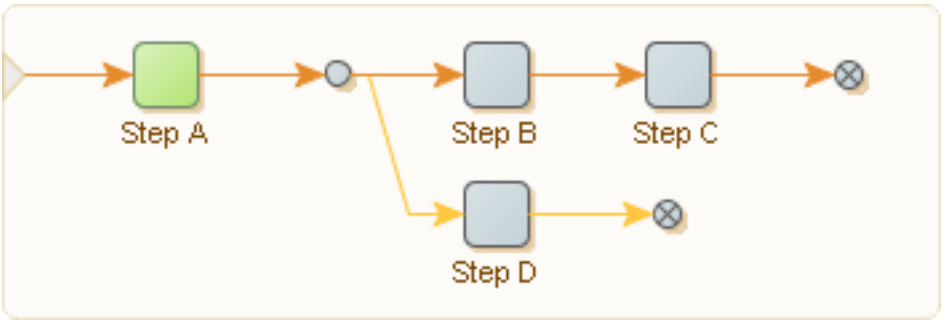
編集アクション

ロボット・エディタを使用すると、ステップおよび接続に対して広範囲にわたるアクションを実行できます。これには、コピー、貼付け、切取り、削除など標準的なエディタ・アクションが含まれ、ループの繰返しの変更など設計ビューでロボットの実行に影響を与えるアクションも含まれます。アクションは、現在のステップ(他のステップが選択されていない場合)、選択されたステップ(1つまたは複数)、または選択された接続(1つまたは複数)に対して実行できます。アクションを実行するには、選択した要素で対応するツール・バー・ボタンをクリックするか、ポップアップ・メニューを使用します。ほとんどのアクションにはキーボード・ショートカットもあります。これらは、ツール・バー・ボタンのツール・チップまたはポップアップ・メニュー項目に表示されます。

現在のステップ以外のステップを構成するには、[Ctrl]を押しながらクリックするか、または周囲の選択ボックスをドラッグしてそのステップを選択し、[F2]キーを押すか、またはポップアップ・メニューからステップの構成...を選択します。これによって、ステップ構成ダイアログが開き、ステップ・ビューと同じオプションが表示されます。

ステップのグループ化とグループ化解除

ここでは、ステップのグループ化とグループ化解除を行う2つの編集アクションについて詳細に説明します。ステップをグループ化するには、グループ化するステップを選択し、ツール・バーまたはステップのポップアップ・メニューからグループ化アクション()を使用します。選択内容によっては、グループ化できない場合があります。グループ・ステップには1つの入力接続と1つの出力接続が必要であるため、グループ化するステップを選択する際はこれも含める必要があります。これの唯一の例外は選択したステップに出力接続が含まれない場合で、選択内容はグループ化できますが、グループの最後に最上位の終了ステップが接続されるため、作成されたグループはグループ外の新しい終了ステップに接続されます。次のロボットの例について説明します。



ロボットの例

次に、このロボットでグループ化できるステップの例を示します。

- すべてのステップ
- 単一のアクション・ステップのみ(Step A、Step Bなど)
- 分岐点、Step B、Step C、およびStep Cの後の終了ステップ

次に、グループ化できないステップの例を示します。

- 分岐点とStep B (複数の出力接続)
- Step B、C、Dおよび2つの終了ステップ(複数の入力接続)

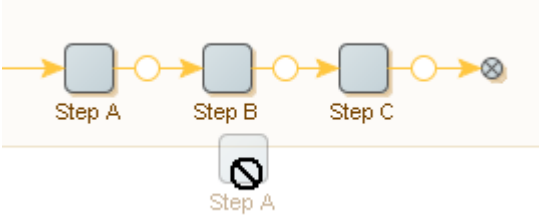
展開されたグループ・ステップを選択するには、[Ctrl]キーを押しながらグループ・ステップの端の近くをクリックするか、グループ・ステップを含めてドラッグします。

グループ・ステップ(1つまたは複数)をグループ化解除するには、グループ・ステップを選択し、ツール・バーまたはステップのポップアップ・メニューからグループ化解除アクション( )を使用します。グループ化とグループ化解除は逆のアクションであるため、選択したステップをグループした直後に再度グループ化解除しても、ロボットの構造は変わりません。

ロボット・ビューには、複数のグループを同時に展開または縮小するアクションも含まれています。ツール・バー・アクション と は、ロボット内のすべてのグループ・ステップを展開または縮小します。ステップのポップアップ・メニューに表示されるアクション と も動作は同じですが、選択対象はグループ・ステップに限定されます。

ドラッグ・アンド・ドロップ

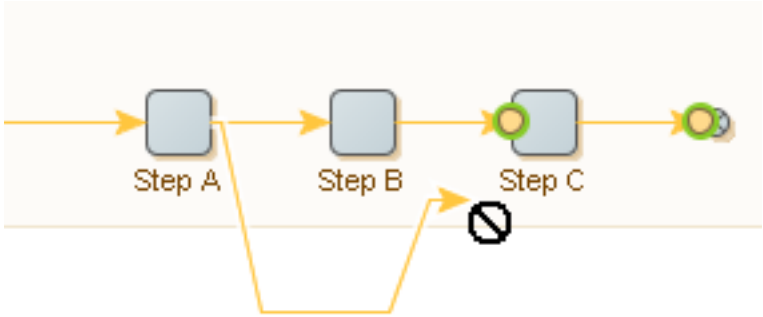
ロボットを編集するとき、アクションを使用する方法とは別に、マウスを使用して要素を直接操作する方法があります。ステップをクリックしてドラッグすることによって、ステップを直接移動できます。ステップをドラッグすると同時に、ドロップ可能な場所を示す特別なインジケータが表示されます。また、複数のステップを選択して、それらを一緒に移動することもできます。接続のエンドポイントを移動することもできます。これを行うには、最初に接続を選択します。次に、接続の終端にあるハンドルの1つにマウスを移動し、ハンドルをクリックして新しい位置に移動します。ハンドルをクリックすると同時に、端を接続可能な場所を示す特別なインジケータが表示されます。ドラッグ・アンド・ドロップ・アクションを中止するには、次に示すように、ロボットの外にマウスを移動してマウス・ボタンを放します。



ステップのドラッグ・アンド・ドロップ

新しい接続の追加

マウスを使用して、新しい接続を作成することもできます。ステップの終端近くにカーソルを置くと、インジケータ(緑の輪に囲まれたオレンジの円)が表示されます。インジケータをクリックすると、新しい矢印が表示されます。左マウス・ボタンを押しながらマウスを移動すると、マウスの移動に従って新しい接続が表示されます。新しいインジケータが表示されたら、いずれかのインジケータにマウスを移動し、そこで左マウス・ボタンを放して新しい接続のエンドポイントをドロップします。



新しい接続の追加

変更を元に戻す/やり直し

ロボットの編集時に実行したすべての操作は、元に戻したりやり直すことができます。したがって、ステップを間違った位置にドロップしたり、誤って接続を削除しても心配する必要はありません。いつでも、1回以上[Ctrl]-[Z]を押す(または アイコンをクリックする)と、実行した操作を元に戻すことができ、元に戻す操作をやり過ぎた場合は[Ctrl]-[Y]を押す(または アイコンをクリックする)とやり直すことができます。

ステップ検証

ロボットの編集中に、ロボット・ビューではステップが検証され、無効なステップは赤い下線が付きます。無効なステップにマウスを移動すると、ステップが無効な理由が表示されます。

ウィンドウ・ビュー

ウィンドウ・ビューは、ロボット・エディタ内のロボット・ビューの下に表示されます。ウィンドウ・ビューには、現在のロボット状態の部分、つまり、ロードされたページに関連するロボット状態の部分が表示されます。表示される状態は、現在のステップへの入力状態です。

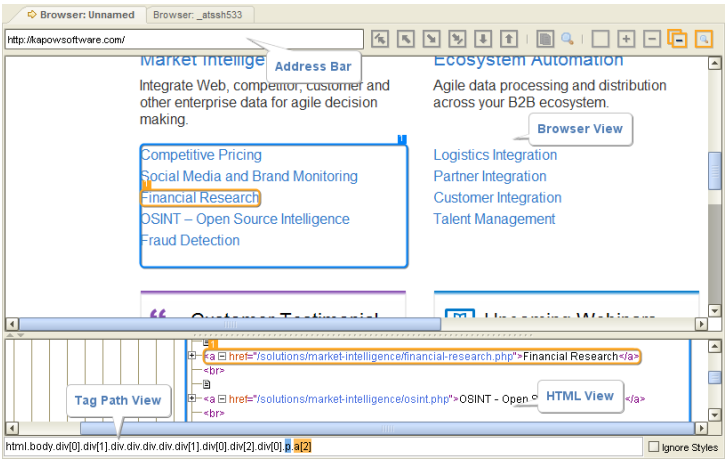
ウィンドウ・ビューには、現在のロボット状態のウィンドウのページ・ビューが表示されます。URLからロードされると、複数のウィンドウが開き、それぞれにページが含まれる場合があります。現在のウィンドウは矢印でマークされます。ウィンドウごとに、ページのタイプに応じてページ・ビューが複数のサブ・ビューに分割されます(たとえば、ロードされたページがHTMLページの場合、ページ・ビューにはこれを反映した複数のサブ・ビューが表示されます)。ページのタイプは、HTML、XML、JSON、Excelおよびバイナリの5タイプです。HTMLとバイナリは同じビューを使用し、その他のページ・タイプは独自の特殊なページ・ビューを使用します。次の各項では、これらのページ・ビューについて説明します。

現在のステップの状態のCookieビューを表示する場合は、「ビュー」メニューからCookieダイアログを開きます。Cookieを使用するWebページをロボットがロードすると、Cookieがこのリストに追加されます。

同様に、「ビュー」メニューから「認証」ダイアログを開き、現在の状態の認証を表示できます。

HTMLページ・ビュー

HTMLページ・ビューは、ウィンドウのページ・タイプがHTMLまたはバイナリの場合に使用されます。次に示すように、ビューは、アドレス・バー、タグ・パス・ビュー、ブラウザ・ビューおよびHTMLビューの複数のサブ・ビューに分割されています。



HTMLページ・ビュー

アドレス・バーには、ページ・ビューのドキュメントのURLが表示されます。(ブラウザと同様に)ここに新しいURLを入力でき、[Enter]を押すと、ロボット内の現在のステップの前にページのロード・ステップが新たに挿入されます。

タグ・パス・ビューには、ページのルート・タグから、選択されたタグまでのパスが表示されます。ブラウザ・ビューには、ブラウザに表示されるのと同じようにページが表示されます。HTMLビューには、JavaScriptが適用されて非同期データがフェッチされた後と同じように、ページのHTMLが表示されます。

ブラウザ・ビューでタグを選択するには、いずれかのビューで左クリックします。現在選択されているタグは、ブラウザ・ビューおよびHTMLビューには緑のボックスに囲まれて表示され、タグ・パス・ビューには緑の背景で表示されます。現在選択されているタグの内側を[Alt]を押しながらクリックすると、選択を1レベル上に移動(つまり、選択されているタグを含むタグを選択)できます。また、選択されているタグの内側を[Alt]と[Shift]を押しながらクリックすることもできます。これによって、クリックしたタグの内側に選択が1レベル下に移動されます。

アドレス・バーの右側にあるボタンを使用して、現在の選択を変更することもできます。  と  アイコンは、選択を1レベル上または下に移動します。  アイコンはページのルート・タグを選択し、  アイコンは選択されているタグ内の最も内側のタグを選択します。  と  アイコンは、選択されているタグの上または下のタグを選択します。  アイコンをクリックして、タグを検索することもできます。

ブラウザ・ビューには、現在のステップのタグ・ファインダによって検出されたタグも表示されます。これらのタグは検出されたタグと呼ばれ、ブラウザ・ビューおよびHTMLビューにはオレンジのボックスに囲まれて表示され、タグ・パス・ビューにはオレンジの背景で表示されます。タグ・ファインダを編集した場合は、  アイコンをクリックすると、新たに検出されたタグを表示できます。また、タグ・ファインダを構成するとき、現在選択されているタグのみを使用する場合は  アイコンをクリックし、現在選択されているタグおよび検出された他のタグを使用する場合

は  アイコンをクリックし、現在選択されているタグを使用しない場合は  アイコンをクリックし、タグをまったく使用しない場合は  アイコンをクリックします。

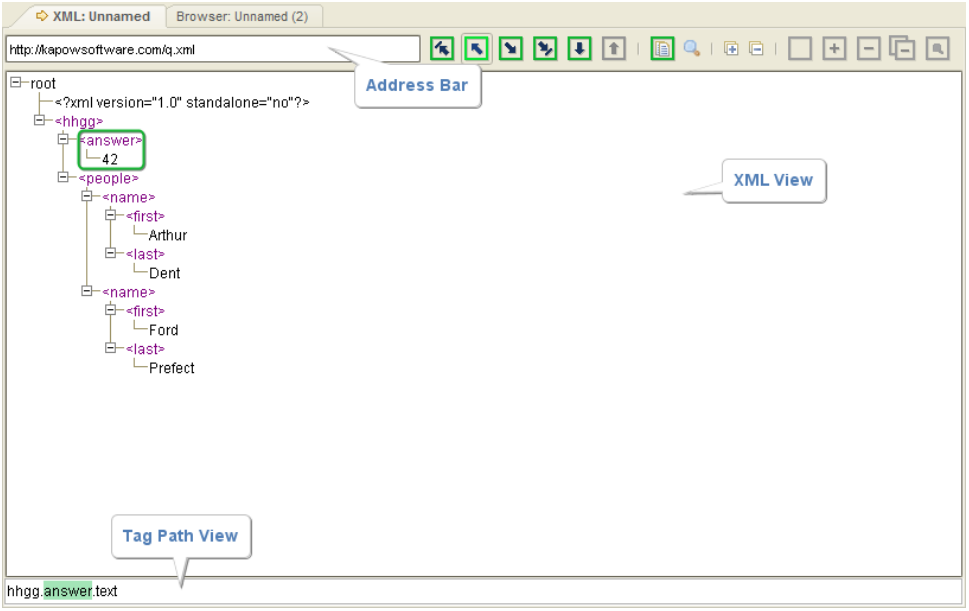
さらに、ページ・ビューには指定タグが表示されます。指定タグは、他のタグを検索するときに参照として使用されるマーカー・タグです。指定タグはステップ・アクションによって設定でき、たとえば、ループ・アクションで指定タグを使用してループの現在の繰返しの結果をマークします。指定タグは手動でも設定できます。指定タグは、ブラウザ・ビューおよびHTMLビューには青のボックスに囲まれて表示され、タグ・パス・ビューには青の背景で表示されます。

すべてのページ・ビューで、タグを右クリックするとポップアップ・メニューが開き、現在のステップを構成したり、新しいステップを挿入できます。これは非常に便利で、現在のステップを構成する最適な方法として使用できます。メニューから、このタグのみ使用またはこのタグを使用を選択して、クリックしたタグを検索するようにタグ・ファインダを構成できます。また、メニューから、テキストの入力などのアクションも選択できます。これによって、クリックしたタグに対して対応するアクション(この場合はテキストの入力)を使用するように、現在のステップを構成できます。ポップアップ・メニューで使用可能なオプションはクリックするタグによって異なり、通常は1つのオプションが太字テキストで表示されます。これはデフォルト・オプションで、(ポップアップ・メニューを使用せずに)ブラウザ・ビューでタグをダブルクリックして直接選択することもできます。たとえば、入力タグにテキストを入力するときは、ブラウザ・ビューで入力フィールドをダブルクリックします。これによってダイアログが起動し、このダイアログでテキストを入力してから「OK」ボタンをクリックすると、ロボット内の現在のステップの前に新しいステップが挿入され、入力したテキストを使用してテキストの入力アクションが構成されます。

HTMLビューからテキストまたはHTMLをコピーするには、対象のタグを右クリックし、表示されたメニューからオプションを選択します。

XMLページ・ビュー

XMLページ・ビューは、ウィンドウのページ・タイプがXMLの場合に使用されます。ビューはHTMLページ・ビューに似ていますが、ブラウザ・ビューはありません。次に示すように、アドレス・バー、タグ・パス・ビューおよびXMLビューの複数のサブ・ビューに分割されています。



XMLページ・ビュー

XMLページ・ビューには、URLからロードされたXMLドキュメント(ページのロード・ステップ・アクションを実行した結果、またはWebサービス呼出しからのレスポンスとして)、またはXML変数からロードされたXMLドキュメントが表示されます。ドキュメントがURLからロードされた場合は、読取り専用であるため変更できません。ドキュメントが変数からロードされた場合は、基本的にその変数のビューであるため、ドキュメントの変更は変数の値に反映されます。

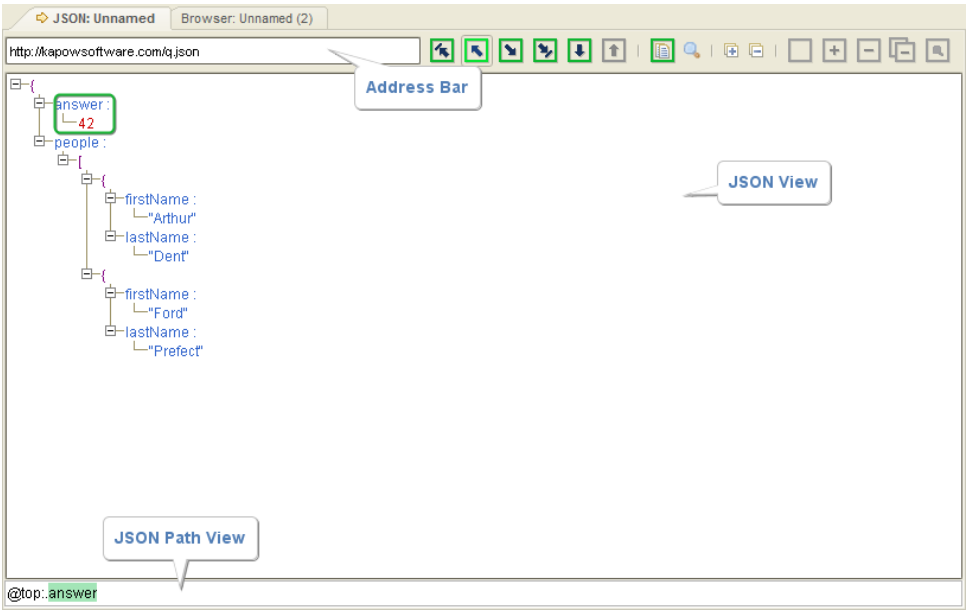
アドレス・バーには、ページ・ビューのドキュメントのURLが表示されます。(ブラウザと同様に)ここに新しいURLを入力でき、[Enter]を押すと、ロボット内の現在のステップの前にページのロード・ステップが新たに挿入されます。アドレス・バーは、ドキュメントがURLからロードされた場合のみ表示され、変数からロードされた場合は表示されません。

タグ・パス・ビューには、ページのルート・タグから、選択されたタグまでのパスが表示されます。XMLビューには、ページのXMLが表示されます。

その他のほぼすべての点でXMLページ・ビューはHTMLページ・ビューと同様に機能し、左クリックして選択したり、右クリックして新しいステップを構成および挿入でき、ツール・バー・ボタンの機能も同じです。右クリック・メニューで使用可能なオプションは、XMLに関連するオプションが選択されるため、オプションが表示されても有効にならない場合は、指定された選択箇所では使用できないことを意味します(たとえば、属性の抽出は、選択したXMLタグに1つ以上の属性がある場合のみ有効になります)。

JSONページ・ビュー

JSONページ・ビューは、ウィンドウのページ・タイプがJSONの場合に使用されます。このビューはXMLページ・ビューとほぼ同じですが、展開可能なツリーにはXMLではなくJSONが表示されます。次に示すように、アドレス・バー、JSONパス・ビューおよびJSONビューの複数のサブ・ビューに分割されています。



JSONページ・ビュー

JSONページ・ビューには、URLからロードされたJSON(ページのロード・ステップ・アクションを実行した結果、またはWebサービス呼出しからのレスポンスとして)、またはJSON変数からロードされたJSONが表示されます。JSONがURLからロードされた場合は、読取り専用であるため変更できません。JSONが変数からロードされた場合は、基本的にその変数のビューであるため、ビュー内のテキストの変更は変数の値に反映されます。

アドレス・バーには、ページ・ビューのJSONのURLが表示されます。(ブラウザと同様に)ここに新しいURLを入力でき、[Enter]を押すと、ロボット内の現在のステップの前にページのロード・ステップが新たに挿入されます。アドレス・バーは、JSONがURLからロードされた場合のみ表示され、変数からロードされた場合は表示されません。

JSONパス・ビューには、ページのルート・オブジェクトから、選択された項目までのパスが表示されます。これは、JavaScriptの値にアクセスする方法に似ています。JSONビューには、ページのJSONが表示されます。

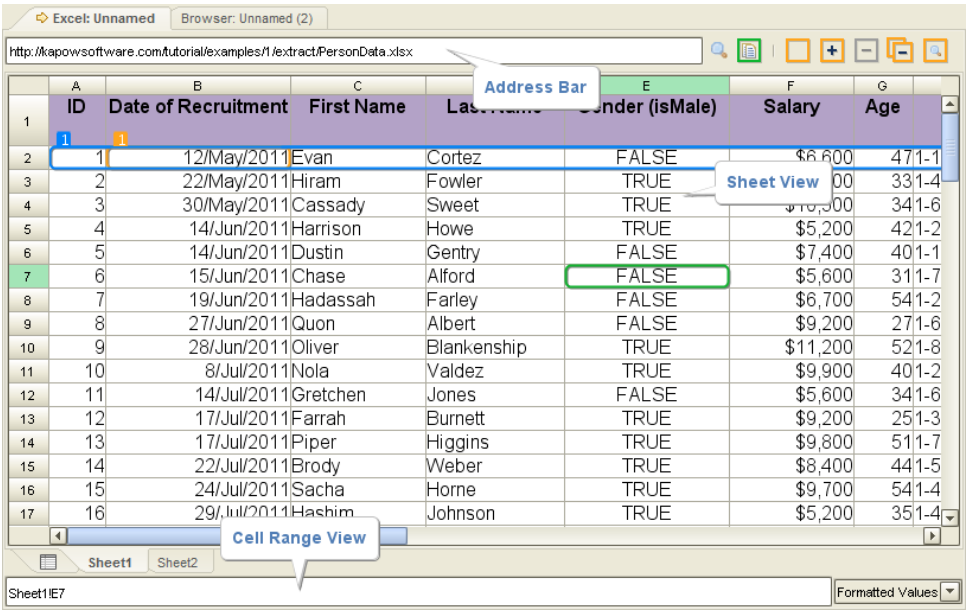
その他のほぼすべての点でJSONページ・ビューはXMLページ・ビューと同様に機能し、左クリックして選択したり、右クリックして新しいステップを構成および挿入でき、ツール・バー・ボタンの機能

も同じです。右クリック・メニューで使用可能なオプションは、JSONに関連するオプションが選択されるため、オプションが表示されても有効にならない場合は、指定された選択箇所では使用できないことを意味します(たとえば、プロパティ名の抽出は、選択箇所がプロパティ宣言(名前/値ペア)である場合のみ有効になります)。

スプレッドシート・ビュー

スプレッドシート・ビューは、ウィンドウにロードされたドキュメントがExcelドキュメントの場合に使用されます。

次に示すスプレッドシート・ビューは、アドレス・バー、シート・ビューおよびセル範囲ビューの複数のサブ・ビューに分割されています。



	A	B	C	D	E	F	G
1	ID	Date of Recruitment	First Name	Last Name	Gender (isMale)	Salary	Age
2	1	12/May/2011	Evan	Cortez	FALSE	$6,600	47 1-1
3	2	22/May/2011	Hiram	Fowler	TRUE	00	33 1-4
4	3	30/May/2011	Cassady	Sweet	TRUE	00	34 1-6
5	4	14/Jun/2011	Harrison	Howe	TRUE	$5,200	42 1-2
6	5	14/Jun/2011	Dustin	Gentry	FALSE	$7,400	40 1-1
7	6	15/Jun/2011	Chase	Alford	FALSE	$5,600	31 1-7
8	7	19/Jun/2011	Hadassah	Farley	FALSE	$6,700	54 1-2
9	8	27/Jun/2011	Quon	Albert	FALSE	$9,200	27 1-6
10	9	28/Jun/2011	Oliver	Blankenship	TRUE	$11,200	52 1-8
11	10	8/Jul/2011	Nola	Valdez	TRUE	$9,900	40 1-2
12	11	14/Jul/2011	Gretchen	Jones	FALSE	$5,600	34 1-6
13	12	17/Jul/2011	Farrah	Burnett	TRUE	$9,200	25 1-3
14	13	17/Jul/2011	Piper	Higgins	TRUE	$9,800	51 1-7
15	14	22/Jul/2011	Brody	Weber	TRUE	$8,400	44 1-5
16	15	24/Jul/2011	Sacha	Horne	TRUE	$9,700	54 1-4
17	16	29/Jul/2011	Hashim	Johnson	TRUE	$5,200	35 1-4

スプレッドシート・ビュー

スプレッドシート・ビューのアドレス・バーはHTMLページ・ビューのアドレス・バーと同じで、その機能も同じです。

シート・ビューには、スプレッドシートの内容が通常の方法で表示されます。これには、スプレッドシート・ドキュメントのシートごとに1つのタブが表示され、追加タブにはこのドキュメントのドキュメント・プロパティ(Excelの「ファイル」>「情報」に表示)も表示されます。このドキュメント・プロパティ・タブはシート・ビューの最初のタブで、  アイコンによって識別されます。ドキュメント・プロパティには、タイトル、著者名、件名、最終変更日などの詳細が含まれます。ドキュメント・プロパティ・タブのビューはドキュメントから通常のシートとして編成され、その内容はシートと同じ方法で抽出してループできます。

シート・ビューの要素は、様々な方法で選択できます。スプレッドシートの1つのセルを選択するには、シート・ビューでマウスを使用してそのセルをクリックします。左マウス・ボタンを押しながらマウスをドラッグすると、複数のセルを選択できます。現在の選択を行った後に、キーボードを使用してその選択を移動できます(たとえば、矢印キーを使用したり、[Home]、[End]、[Page Up]または[Page Down]のいずれかのキーを使用します)。

列ヘッダーまたは行ヘッダーをクリックすると列または行全体が選択され、シート・ビューの左上隅をクリックするとシート全体が選択されます。列、行またはシート全体が選択されているときは拡張可能な選択になり、列、行またはシートのサイズを変更すると、列、行またはシート全体が選択されたままで、列、行またはシートを選択したときの特定のサイズから変更されます。たとえば、(ループ・ステップを使用して)シートのすべての行を抽出するロボットを記述した場合、ロボットを作成した日のシートには42行が含まれ、翌日にシートの行数が42行から変更されていてもロボットは失敗しません。これに対して、表に対して「Sheet1!A1:I42」のような選択を使用した場合、シートに実際に含まれる行数(および列数)に関係なく、ロボットは常に42行(および9列)を抽出しようとします。

[Shift]キーを押しながらマウスまたはキーボードを使用して選択すると、現在の選択が拡張します。この拡張は、現在のリード選択および新たに追加するセルを含む範囲を選択して行います。現在のリード選択とは、現在の選択を作成したときに選択した最初のセルです。たとえば、マウスを使用してSheet1のセルC5からE7にドラッグすると、選択は「Sheet1!C5:E7」でリード選択セルはC5になり、E7からC5にドラッグすると、リード選択セルはE7になります。

セル範囲ビューには、現在の選択がExcelセル範囲として表示されます(例: Sheet1!C5:E7)。ビューに新しいセル範囲を入力して[Enter]を押すと、その範囲が新しい現在の選択になります。セル範囲の構文については、セル範囲に関する参照ドキュメントを参照してください。

シート・ビューには、現在のステップの範囲ファインダによって検出されたセル(1つまたは複数)も表示されます。これらのセルは検出されたセルと呼ばれ、シート・ビューにはオレンジのボックスに囲まれて表示されます。範囲ファインダを編集する場合は、  アイコンをクリックして、新たに検出されたセルを表示できます。また、範囲ファインダを構成するとき、現在選択されているセル(1つまたは複数)のみを使用する場合は  アイコンをクリックし、現在選択されているセルおよび検出された他のセルを使用する場合は  アイコンをクリックし、現在選択されているセルを使用しない場合は  アイコンをクリックし、セルをまったく使用しない場合は  アイコンをクリックします。

HTMLページ・ビューと同様に、  アイコンをクリックしてスプレッドシート・ビュー内の特定の内容を検索でき、検索によって検出されたセルが新しい現在の選択になります。また、  アイコンを使用して、現在の選択の内容をクリップボードにコピーできます。複数のセルが含まれる範囲をコピーした場合、クリップボードの内容はExcelに直接貼り付けることができ、各セルの値はExcelのセルに個別に配置されます。

ビュー・モード・セレクタは、スプレッドシート・ビューの右下隅にあるコンボ・ボックスです。ここでは、シート・ビューのセルに表示する内容を決定します。選択可能なオプションは次のとおりです。

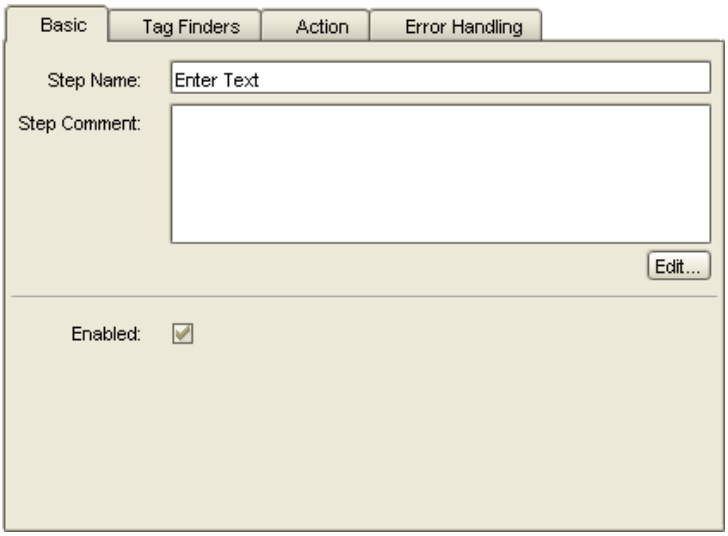
書式設定された値: セルにはExcelに表示される内容が含まれます(たとえば、書式設定された日付が表示されたり、セルの実際値より小数値が少ない数字が表示されます)。

プレーン値: セルの値が書式設定されていない場合、セルにはExcelに表示される実際値が含まれます(たとえば、端数処理していない小数値)。

式: セルに式が含まれる場合はその式が表示され、そうでない場合はプレーン値モードと同じ値が表示されます。

ステップ・ビュー

ステップ・ビューは、ページ・ビューの右側に表示されます。ステップ・ビューは仕切線によってページ・ビューと区切られているため、仕切線を左または右にドラッグして一方のビューの幅を広くすることができます。



アクション・ステップのステップ・ビュー

ステップ・ビューには、現在のステップの構成が表示されます。「基本」、ファインダ、「アクション」およびエラー処理のいずれかのタブをクリックして、ステップの様々なプロパティを表示および編集できます。ステップ・タイプ(試行ステップやグループ・ステップなど)によっては、一部または全部のプロパティが関係ない場合があり、その場合は対応するタブが表示されません。したがって、ステップ・ビューの実際の表示は現在のステップによって異なります。

「基本」タブには、ステップの名前、および添付されたコメントが表示されます。コメントが添付されているステップは、ロボット・ビューに名前が太字で表示されます。ステップにマウス・ポインタを置くと、コメントが反転テキストとして表示されます。

ファインダ・タブでは、ステップのファインダのリストを表示して構成できます。通常は、ページ・ビュー内で要素を右クリックしてファインダを構成します。詳細は、「タグ・ファインダを使用する方法」を参照してください。

ステップがアクション・ステップの場合は、「アクション」タブでステップのアクションを表示して構成できます。使用可能なアクションについては、後述の「ステップ・アクションおよびデータ・コンバータ」を参照してください。

ステップ・ウィンドウのエラー処理タブには、現在のステップのエラー処理方法が表示されます。詳細は、「エラーを処理する方法」を参照してください。

変数ビュー

変数ビューは、ステップ・ビューの下に表示されます。変数ビューは仕切線によってステップ・ビューと区切られているため、仕切線を上または下にドラッグして一方のビューの幅を広くすることができます。

変数ビューは、同様の仕切線によって左部分と右部分に分割されています。左部分には、リストに変数が表示されます。このリストから変数を選択すると、その変数の詳細がビューの右部分に表示されます。ビューには、ロボット実行の現在のステップにおける変数の値が表示され、これらの値は編集できません。

変数を追加または削除するには、変数のリストを右クリックします。これによって、リストに追加できる変数タイプのメニューが表示されます。このメニューから、選択した変数を削除することもできます。

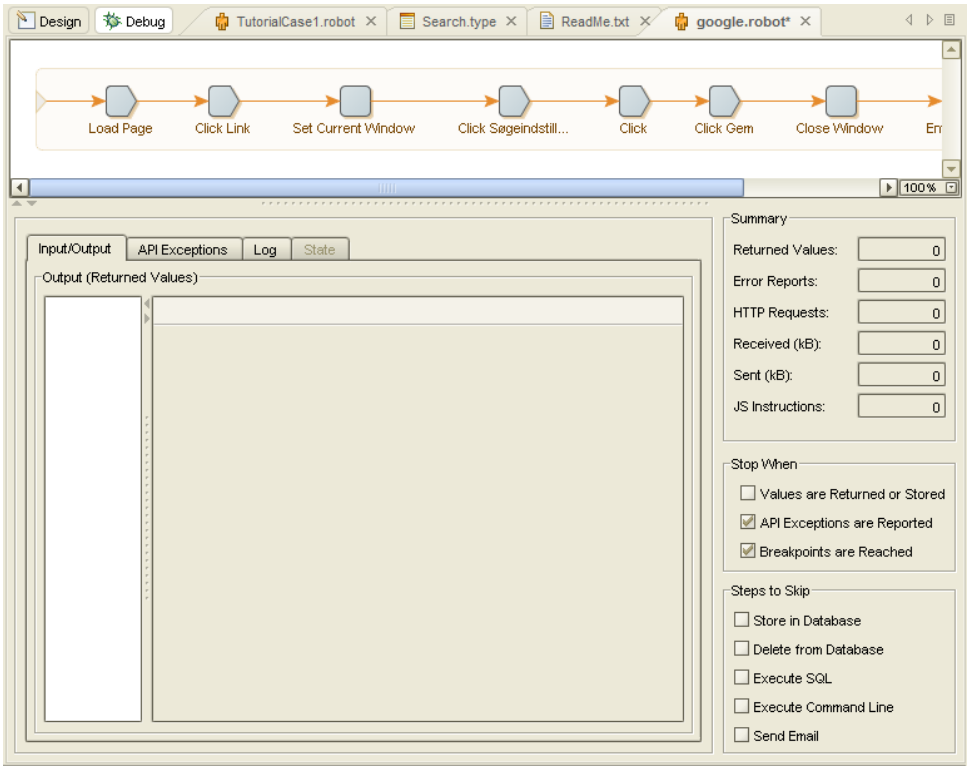
変数の初期値を編集するには、「編集」ボタンを押すか、または変数リストをダブルクリックします。これによって、変数ビューと外観がよく似ていて同様に機能するダイアログが開きます。重要な違いは、ダイアログにはステップが実行される前の変数の値が表示され、これらの値は編集できることです。

入力変数の初期値は、ロボットを記述してテストするときに使用されます。ロボットが本番で実行されると、入力変数は、ロボットを実行するアプリケーションによって決定された値に初期化されます(アプリケーションが値を決定できない場合、ロボットの実行は失敗します)。

変数の初期値は、ロボットの開始時、つまり最初のステップにおける値です。この値は、ロボットを記述してテストし、本番で実行するときに適用されます。

デバッグ・モード

ロボット・エディタには、ロボットをデバッグするための特別なモードがあります。設計モードとデバッグ・モードを切り替えるには、ロボット・エディタの左隅にある2つのモード・ボタンのいずれかをクリックします。Design Studioのメイン・ウィンドウのツール・バーにあるアイコンをクリックして、デバッグ・モードに切り替えることもできます。または、Design Studioの現在のステップからデバッグする場合は、アイコンをクリックできます。デバッグ・モードに切り替えると、ロボット・エディタの外観が次に示すように変更されます。



デバッグ・モード

デバッグ・モードでは、ロボット・エディタの上部に設計モードと同様のロボット・ビューが表示されます。ただし、デバッグ・モードでは、ロボットを実際にデバッグしているときのみ、ロボット・ビューに現在のステップが表示されます。この現在のステップは、設計モードのロボット・ビューに表示される現在のステップと必ずしも同じではありません。

ロボット・ビューの下には大きいパネルがあり、4つのサブパネル(メイン・パネルおよび「サマリー」、停止時点、スキップするステップの3パネル)に分割されています。

メイン・パネルには、デバッグ・プロセスの結果が4つのタブに分割されて表示されます。入力/出力タブには、使用されているすべての変数のリスト(ある場合)、およびデバッグ・プロセスで現在までに返されたすべての値のリストが表示されます。API例外タブには、デバッグ・プロセスで現在までにレポートされたAPI例外のリストが表示されます。「ログ」タブには、デバッグ・プロセスで現在までにログに書き込まれた内容が表示されます。(フォームのループ・アクションなど、特に実行に時間がかかるアクションでは、ステータス情報をこのログに書き込む場合があります。ステップ・エラーもログに書き込まれます(ステップがそのように構成されている場合))。デバッグ・プロセスが一時的に停止した場合は常に、「状態」タブに、現在のステップに入力されたロボット状態が表示

されます。「状態」タブには、7つのサブタブが含まれています。「変数」サブタブには、変数のリストが表示されます。「ウィンドウ」、Cookieおよび「認証」サブタブには、Design Studioの状態ビューと関連するダイアログと同様に、状態が表示されます。ローカル記憶域およびセッション記憶域サブタブには、ローカルに保持されているHTML5オブジェクトが表示されます。API例外サブタブには、現在のステップで生成されたAPI例外(ある場合)が含まれます。すべてのAPI例外(およびその一部に含まれるエラー)について、  「移動」ボタンをクリックすると、エラーが生成されたステップに直接移動できるため、エラーが生成されたステップがDesign Studioの現在のステップになります。

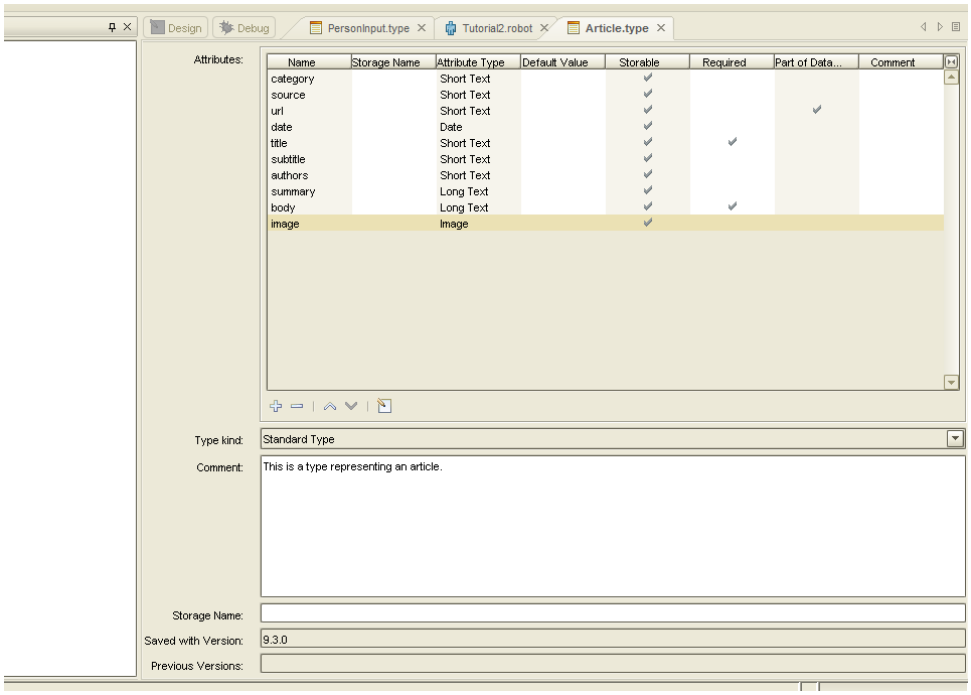
「サマリー」パネルには、デバッグ・プロセスで現在までに返された変数またはデータベースに書き込まれた変数、およびAPI例外が生成された変数の数の概要が表示されます。

停止時点パネルには、デバッグ・プロセスを一時的に停止する時点(通常に終了する場合以外)に関する条件を指定できます。

ロボットのデバッグの詳細は、「ロボットをデバッグする方法」を参照してください。

タイプ・エディタ

この項では、タイプ・エディタのユーザー・インタフェースを紹介して、タイプ・エディタの概要を説明します。次に、このエディタのビューを示します。



タイプ・エディタ

現在編集されているタイプは、メイン・ウィンドウで構成されます。特に、タイプの属性を構成できます。これは、属性表で行います。属性表の下にあるボタンを使用して、新しい属性の追加、属性の削除、順序の変更、および属性の構成ができます。

テキスト・エディタ

テキスト・エディタは、プレーン・テキスト・ファイル(.txt)用の単純なエディタです。これは、Readmeファイルなどの単純なテキスト・ファイルを編集する便利な方法としてのみ含まれています。エディタで開くことができる拡張子はtxt、java、jsp、js、log、html、xmlおよびcsvですが、エディタではこれらの拡張子が示すファイル・コンテンツの情報は使用されません。すべてのファイルは、プレーン・テキスト・ファイル(構文の強調なし)として扱われます。

一般的な編集

この項では、Design Studioでのロボットの編集に関連する一般的なヒントをいくつか示します。これについては、ロボット・ビューでロボットを編集するときにすでに説明しましたが、ここで示すヒントはそれよりも一般的です。また、適用されるのは、ステップ・ビューのロボット、タイプ・エディタのタイプ、またはテキスト・エディタのテキストに対して変更を加えたときです。

変更を元に戻す/やり直し

基本的に、Design Studioで何かを編集するときに行ったすべての操作は、[Ctrl]-[Z]を押すか、または  アイコンをクリックして元に戻すことができます。同様に、元に戻した変更は、[Ctrl]-[Y]を押すか、または  アイコンをクリックしてやり直すことができます。

切取り、コピーおよび貼付け

Design Studioでは、[Ctrl]-[X]、[Ctrl]-[C]および[Ctrl]-[V]のショートカットを使用して、ほとんどの項目(例: ステップ、データ・コンバータ、ファインダなど)の切取り、コピーおよび貼付けができます。ステップのファインダのリストなど、ほとんどのリストでは、ショートカット[Ctrl]-[Shift]-[C]を使用してすべての項目をコピーできます。

タイプ

タイプを使用する場合は、関連するすべてのプロパティを構成することが重要です。そうしないと、タイプが無効になるか、Design Studioで使用するときに正常に動作しません。

すべてのタイプには有効な名前が必要です。タイプ名は文字またはアンダースコアで始まり、文字、数字およびアンダースコアのみ使用する必要があります。同じプロジェクト内に2つのタイプがある場合は、同じ名前を使用できません。タイプの名前は、拡張子を除いたファイル名になります。たとえば、ファイル名が「ExampleType.type」のタイプは、Design Studioで「ExampleType」として使用できます。

また、すべてのタイプには、そのタイプの使用方法を示すタイプの種類が必要です。タイプの種類は、属性表の下にあるタイプの種類ドロップダウン・ボックスを使用して選択できます。通常、ここでは何も選択する必要はなく、レガシーのタイプの種類であるデータベース出力タイプが必要である場合を除き、タイプの種類は標準タイプが常に使用されます。詳細は、Design Studioに関する参照ドキュメントを参照してください。

属性の構成

タイプを有効にするには、タイプ内の属性も正しく追加および構成される必要があります。属性ごとに名前とタイプの両方を指定する必要があります。使用可能な属性タイプは次のとおりです。

属性タイプ	説明
整数	整数(例: 12)。可能な範囲は-9223372036854775808から9223372036854775807(両端を含む)です。
数値	数値(例: 12.345)。可能な範囲は$\pm 2.2 \times 10^{-308}$から$\pm 1.8 \times 10^{308}$で、15桁強の精度です。
ブール	ブール値で、trueまたはfalseのいずれかです。
文字	単一文字(例: A)。
短いテキスト	短いテキスト。1行のテキスト・フィールドに表示されます。
長いテキスト	長いテキスト。複数行のテキスト・フィールドに表示されます。
パスワード	パスワード。「パスワード」フィールドに表示され、パスワードの文字ではなくアスタリスクが表示されます。

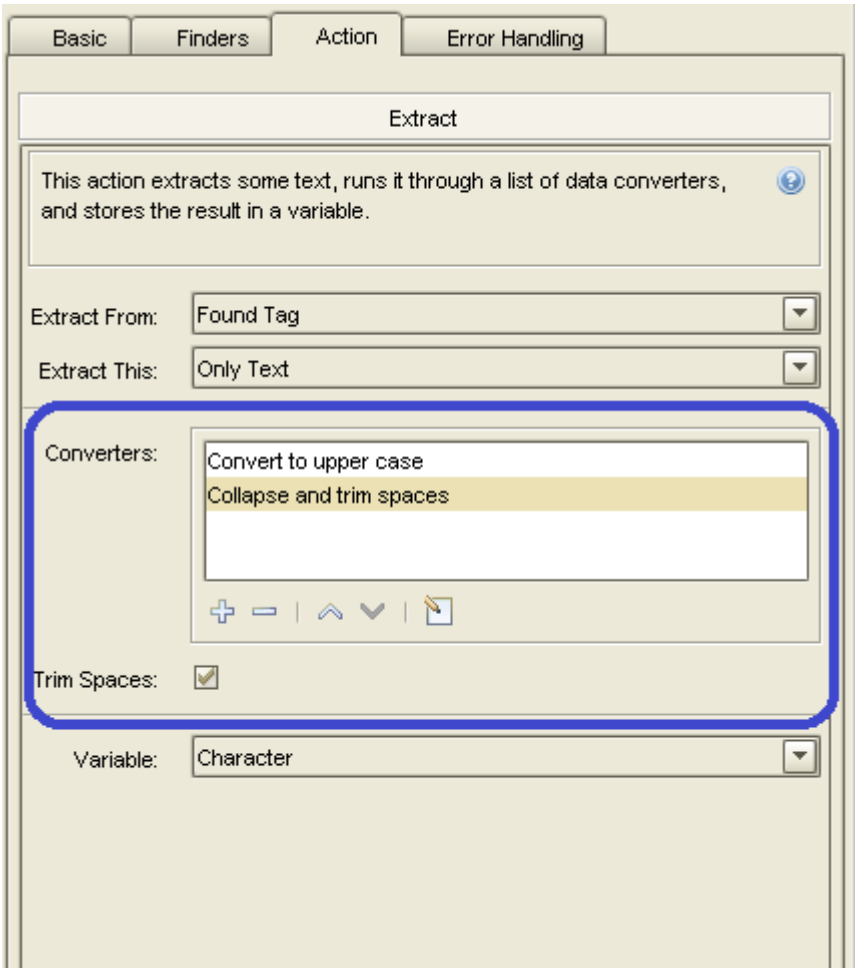
属性タイプ	説明
HTML	HTMLクリップ。クリップはブラウザ・ウィンドウでプレビューできることを除いて、長いテキストと同じです。
XML	XMLドキュメント。正しい形式のXMLドキュメントのみ許可されることを除いて、長いテキストと同じです。
日付	日付。日付のフォームはyyyy-mm-dd hh:mm:ssである必要があります(例: 1992-04-25 10:33:06.0)。
バイナリ	バイナリ・データ(つまり、連続するバイト)。
イメージ	イメージ。イメージはプレビューできることを除いて、バイナリ・データと同じです。
PDF	PDFドキュメント。PDFドキュメントはプレビューできることを除いて、バイナリ・データと同じです。
セッション	セッション(Cookie、認証などを含む)。
通貨	ISO-4217標準で定義された通貨コード(例: ユーロは「EUR」)。
国	ISO-3166標準で定義された国コード(例: ドイツは「DE」)。
言語	ISO-639標準で定義された言語コード(例: ドイツ語は「de」)。

属性には、これ以外に構成可能なプロパティがいくつかあります(たとえば、Design Studio内から属性を表示できるかどうか、タイプの変数値が格納される前に属性にデータが含まれる必要があるかどうかなど)。詳細は、Design Studioに関する参照ドキュメントを参照してください。

ステップ・アクションおよびデータ・コンバータ

この項では、Design Studioで使用可能なステップ・アクションおよびデータ・コンバータに関する一般情報を説明します。Design Studioでは、各アクションおよびデータ・コンバータとともに短い説明が表示され、アクションまたはデータ・コンバータの詳細情報を表示するには、説明に関連付けられている「詳細...」ボタンをクリックします。さらに、Design Studioの「ヘルプ」メニューから、すべてのステップ・アクションおよびデータ・コンバータに関するヘルプ・エントリを検索できます。

いくつかのアクション(抽出など)では、データ・コンバータのリストを介してテキスト・コンテンツを実行し、その結果を変数に格納する可能性があります。



スプレッドシート・ビュー

前述のように、データ・コンバータはテキストを処理します。たとえば、数字の抽出データ・コンバータは、ある書式の数字が含まれる入力テキストを受け取り、同じ数字が含まれるテキストを標準化された書式で出力します。データ・コンバータはテキストを入力として受け取り別のテキストを出力するため、あるデータ・コンバータの出力が次のデータ・コンバータの入力になるように、複数のデータ・コンバータをつなげることができます。最終出力は、データ・コンバータのリストにある最後のデータ・コンバータで出力されたテキストです。たとえば、データ・コンバータのリストが「大文字に変換」データ・コンバータの後に「空白を削除」データ・コンバータが続くように構成されており、リストへの入力テキストが**"R oboMa ker"**の場合、出力テキストは**"ROBOMAKER"**になります。

パターン

パターンについて理解していない場合は、パターンに関する紹介ビデオを視聴することをお薦めします。

パターンは、テキストを説明する正式な方法です。たとえば、テキスト"32"は、2つの数字が含まれるテキストと説明できます。ただし、"12"や"00"などのテキストも2つの数字が含まれるため、これらも2つの数字が含まれるテキストと説明できます。これは、テキストに2つの数字のみ含まれることを表す正式な方法のパターン「¥d¥d」(¥dは数字を表す記号)で表すことができます。これらのテキストはこのパターンに一致すると言えます。Design StudioのパターンはPerl5構文に準拠します。

パターンは、通常文字と特殊記号で構成されます。各特殊記号にはそれぞれ特別な意味があります。たとえば、特殊記号"."(ドット)は単一文字を意味し、"a"、"b"、"1"、"2"などの単一文字がすべて一致します。

次の表に、一般的に使用される特殊記号の概要を示します。使用可能なすべての特殊記号の概要は、パターンに関する参照ドキュメントを参照してください。

表10.1 一般的に使用されるパターン特殊記号

特殊記号	意味
.	単一文字(例: "a"、"1"、"/"、"?"、"."など)。
¥d	10進数字(例: "0"、"1"、...、"9")。
¥D	数字以外で"."と同じですが、"0"、"1"、...、"9"は除きます。
¥s	空白文字(例: " "、改行)。
¥S	空白以外の文字で"."と同じですが、空白(" "、改行など)は除きます。
¥w	単語(英数字)文字(例: "a"から"z"、"A"から"Z"、"0"から"9")。
¥W	単語(英数字)以外の文字で"."と同じですが、"a"から"z"、"A"から"Z"、"0"から"9"は除きます。

例: パターン".an"は長さが3文字で"an"で終わるすべてのテキストと一致します(例: "can"と"man"は一致しますが、"mcan"は一致しません)。

例: パターン"¥d¥d¥s¥d¥d"は、長さが5文字で、2つの数字で始まり、1つの空白が続き、2つの数字で終わるすべてのテキストと一致します(例: "01 23"と"72 13"は一致しますが、"01 2s"は一致しません)。

"."や"¥"などの特殊文字を通常文字として使用する場合は、その文字の前に"¥" (バックスラッシュ)を追加してエスケープできます。したがって、任意の単一文字ではなく"."文字と完全一致させる場合は、"¥."と記述する必要があります。

例: パターン"m¥.n¥¥o"は、テキスト"m.n¥o"とのみ一致します。

1つのパターンを複数のサブパターンに編成するには、カッコ"("と")"を使用します。

例: パターン"abc"は"(a)(bc)"に編成できます。

すべての単一文字はサブパターンとみなされます。

例: パターン"abc"の各単一文字"a"、"b"および"c"はサブパターンとみなされます。

サブパターンは、パターン演算子を適用するときに便利です。次の表に、使用可能なパターン演算子の概要を示します。

表10.2 パターン演算子

演算子	意味
?	先行するサブパターンまたは空のテキストと一致します。
*	先行するサブパターンの任意の回数の反復、または空のテキストと一致します。
+	先行するサブパターンの1回以上の反復と一致します。
{m}	先行するサブパターンのm回の反復と一致します。
{m,n}	先行するサブパターンのmからnの間(両端を含む)での反復と一致します。
{m,}	先行するサブパターンのm回以上の反復と一致します。
a|b	式aと一致するすべて、または式bと一致するすべてと一致します。

例: ".*"は、"Design Studio"、"1213"、"" (空のテキスト)など、すべてのテキストと一致します。

例: "(abc)*"はテキスト"abc"の任意の回数の反復と一致し、""、"abc"、"abcabc"および"abcabcabc"は一致しますが、"abca"は一致しません。

例: "(¥d¥d){1,2}"は2つまたは4つの数字のいずれかと一致し、"12"および"6789"は一致しますが、"123"は一致しません。

例: "(good)?bye"は"goodbye"および"bye"と一致します。

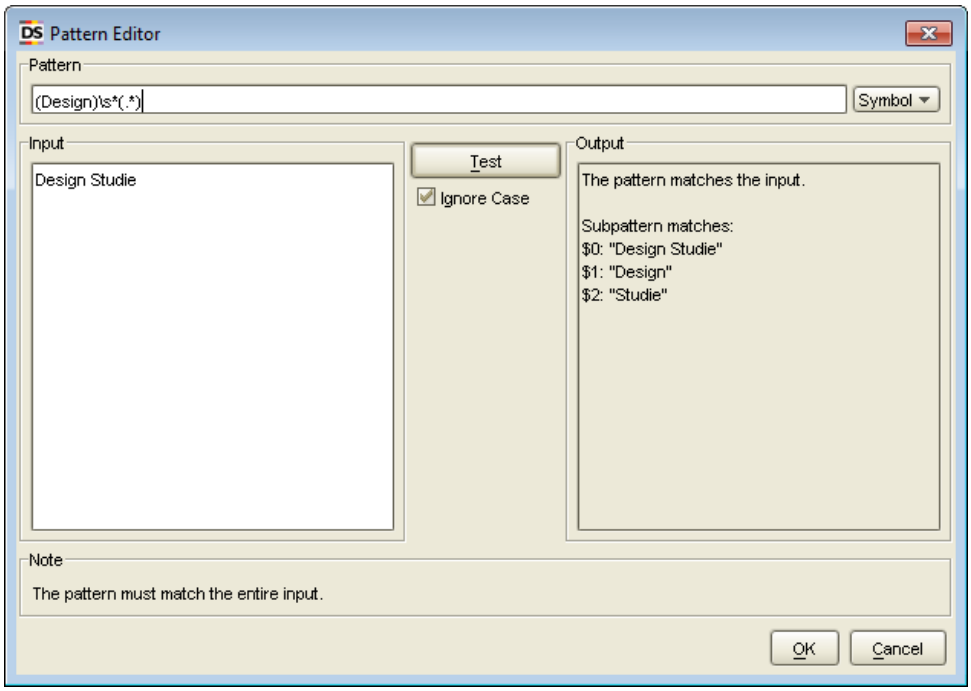
例: "(good)|(bye)"は"good"および"bye"と一致します。

他の特殊文字と同様に、パターン演算子に含まれる特殊文字は、その文字の前に"¥" (バックスラッシュ)を追加してエスケープできます。

サブパターンは、テキストから特定のテキスト部分を抽出する場合に役立ちます。カッコを使用してサブパターンを作成すると、そのサブパターンによって一致するテキストの部分を抽出できます。たとえば、パターン"abc (.*) def (.*) ghi"を考えてみます。このパターンには、カッコを使用して作成された2つのサブパターンがあります。このパターンをテキスト"abc 123 def 456 ghi"と照合すると、最初のサブパターンはテキスト"123"と一致し、2番目のサブパターンはテキスト"456"と一致します。式(「式」を参照)では、これらのサブパターン一致は"$1"および"$2"と記述して参照できます。たとえば、式"X" + $1 + "Y"+ $2 + "Z"では、"X123Y456Z"という結果が作成されます。これは、Design Studioの非常に重要な抽出テクニックです。

デフォルトで、反復パターン演算子(*、+、{...})は、可能なかぎり多くの先行パターンの反復と一致します。演算子の後に"?"を配置すると、可能なかぎり少ない反復と一致する演算子に変わります。たとえば、パターン".*(¥d¥d¥d).*"を考えてみます。このパターンをテキスト"abc 123 def 456 ghi"と照合すると、最初の*演算子は可能なかぎり多くの反復と一致するため、サブパターン"(¥d¥d¥d)"はテキスト内の2番目の数字("456")と一致します。*演算子の後に"?"を配置してパターンが".*?(¥d¥d¥d).*"になると、*?演算子は可能なかぎり少ない反復と一致するため、サブパターン"(¥d¥d¥d)"はテキスト内の最初の数字("123")と一致します。

ユーザー自身がパターンを体験することをお薦めします。これを行う最適な方法は、Design Studioを起動し、タグのテスト・アクションなどでパターンを入力できる場所を見つけることです。次に、パターン・フィールドの右側にある「編集...」ボタンをクリックして、次に示すようなパターン・エディタ・ウィンドウを開きます。



パターン・エディタ・ウィンドウ

パターン・エディタ・ウィンドウでは、パターンを入力し、「入力」パネルのテスト入力テキストと一致するかどうかをテストできます。現在の入力ロボット状態で特定のステップが実行されている場合、このウィンドウを開くと、Design Studioでは通常、テスト入力テキストはパターンに一致するテキストに設定されます。ただし、テスト入力テキストを自分で編集して、他の入力に対してパターンをテストすることもできます。パターンをテストするには、「テスト」ボタンをクリックします。一致の結果は「出力」パネルに表示されます。

「記号」ボタンは、パターンに特殊記号を入力するときに非常に役立ちます。これをクリックするとポップアップ・メニューが表示され、パターンに挿入する記号を選択できます。したがって、すべての特殊記号やその意味を覚える必要はありません。

パターンの詳細は、パターンに関する参照ドキュメントを参照してください。

式

通常、式はテキストとして評価されます。たとえば、次の式を考えてみます。

```
"The author of the book " + Book.title + " is " + Book.author + "."
```

変数Book.titleおよびBook.authorにそれぞれテキスト"Pet Sematary"および"Stephen King"が含まれる場合は、テキスト"The author of the book Pet Sematary is Stephen King."と評価されます。

式内で数値計算を行うこともできます。たとえば、変数Book.priceに書籍の価格が含まれる場合は、次の式を使用して、この価格に100を乗算できます。

```
Book.price * 100
```

次の表に、一般的に使用されるサブ式タイプの概要を示します。使用可能なすべてのサブ式タイプの概要は、式に関する参照ドキュメントを参照してください。

表10.3 一般的に使用されるサブ式タイプ

サブ式タイプ	表記	意味
テキスト定数	"text"または>>text<<	指定したテキストとして評価されます(例: "Stephen King"、>>Stephen King<<)
変数	variablename.attributename	指定した変数の値として評価されます(例: "Book.author"は"Stephen King"と評価されます)。
現在のURL	URL	現在のページのURLとして評価されます。
サブパターン一致	$n	関連付けられたパターン(ある場合)内のサブパターンnによって一致するテキストとして評価されます。たとえば、後に示すように、これは拡張抽出データ・コンバータで使用されます。$0は、パターン全体によって一致するテキストとして評価されます。
関数	func(args)	指定の関数に指定の引数を渡し、その結果をテキストに変換することによって、関数が評価されます。

テキスト定数を指定するには、引用符表記または>>text<<表記(例: "Stephen King"または>>Stephen King<<)のいずれかを使用できることに注意してください。引用符表記を使用するときに、テキスト内に引用符文字を含める場合は、2つの引用符文字を記述する必要があります。たとえば、テキストが"This is some "quoted" text"の場合は、"This is some ""quoted"" text"と記述します。>>text<<表記を使用する場合は、">>"と"<<"を除いて、テキスト内にあらゆるものを含めることができます。したがって、引用符も>>This is some "quoted" text<<のように直接記述できます。>>text<<表記は、HTMLなど多数の引用符文字が含まれる長いテキストの場合に便利です。

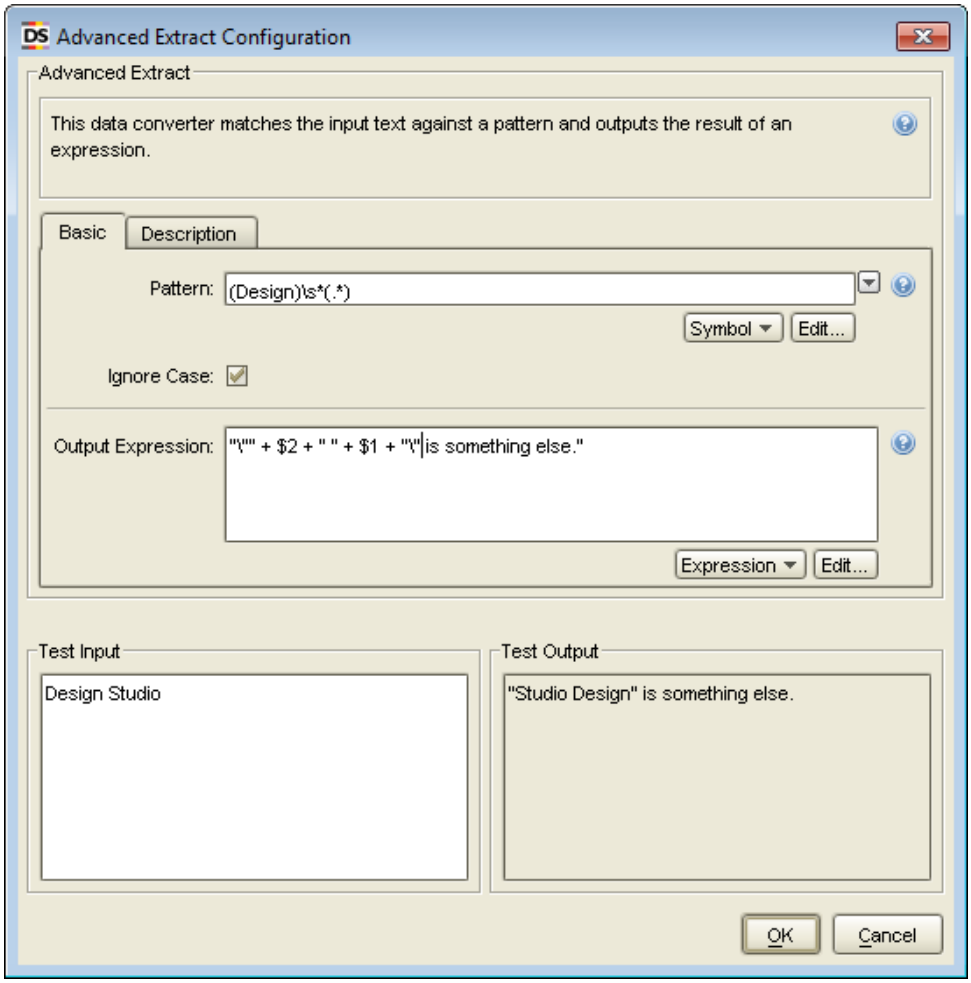
次の表に、式で一般的に使用される関数を示します。使用可能なすべての関数の概要は、式に関する参照ドキュメントを参照してください。

表10.4 一般的に使用されるサブ式関数

関数	意味
toLowerCase(arg)	引数を小文字に変換します。
round(arg)	引数を最も近い整数に四捨五入します。

例: 品目の古い価格が$99.95で、新しい価格が$89.95のとき、式"The discount is " + round((Item.oldPrice - Item.newPrice) / Item.oldPrice) + "%."は"The discount is 10%."として評価されます。

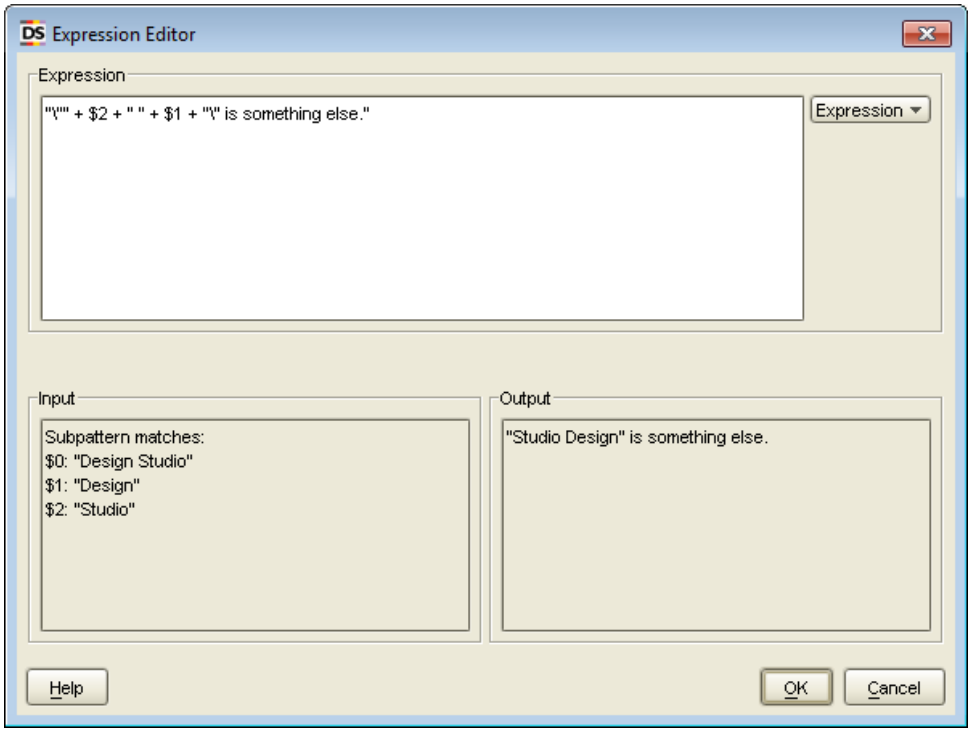
ユーザー自身が式を体験することをお薦めします。式を体験する最適な方法は、Design Studioを起動し、現在のステップに対して抽出アクションを選択して、拡張抽出データ・コンバータを追加することです。  アイコンをクリックして、データ・コンバータを構成します。これによって、次に示す拡張抽出構成ウィンドウが開きます。



拡張抽出構成ウィンドウ

この例では、$n表記を使用して、入力テキストの一部分を抽出しています。ユーザー自身の独自の入力テキストを左側のテキスト領域に、独自のパターンをパターン・プロパティに、独自の式を出力式プロパティに入力してみてください。式を入力しながら、右側の領域で結果を確認できます。

また、式フィールドの右側にある「編集...」ボタンをクリックしてみてください。これによって、次に示す式エディタ・ウィンドウが開きます。



式エディタ・ウィンドウ

式エディタ・ウィンドウでは、式を入力し、評価内容をテストできます。拡張抽出データ・コンバータと同様に、式がパターンに関連付けられている場合は、現在の入力テキストに対するパターンの照合結果が「入力」パネルに表示されます。パターンが一致するかどうかを確認し、一致する場合は、$n表記を使用して、式に一致するサブパターンを参照できます。Design Studioのすべての場所でテスト機能は使用できないことに注意してください。

「式」ボタンをクリックすると便利なポップアップ・メニューが開き、使用可能なサブ式タイプおよび関数を選択できます。したがって、これらすべてを覚える必要はありません。

式の詳細は、参照ドキュメントを参照してください。

プロジェクトおよびライブラリの使用

Design Studioで作業を行うときは常にプロジェクトを使用し、いつでも任意の数のプロジェクトを開くことができます。プロジェクトの目的は、ロボットおよびロボットで必要なファイルの集合が含まれるライブラリを開発することです。通常、ロボットの使用ごとに1つのプロジェクトを作成します(たとえば、社内でロボットを使用するアプリケーションごとに1つのプロジェクトを作成します)。2つのプロジェクトでファイルは共有できません(たとえば、1つのタイプは常に1つのプロジェクトに属しており、タイプの有効範囲はそのタイプが属するプロジェクトになります)。

プロジェクトとは、ファイル・システム内のいずれかの場所に配置されるフォルダです。プロジェクト・フォルダには任意の名前を指定できますが、次のサブフォルダを含める必要があります。

`Library` – このファイルには、プロジェクトのライブラリが含まれます。

`Library`フォルダには、ロボット・ファイル、タイプ・ファイル、およびロボットで使用される他のファイル(ロボット・ライブラリからロードされるファイルなど)をすべての格納する必要があります。**`Library`**フォルダ内のファイルは、必要に応じてサブフォルダを使用して、任意の方法で編成できます。

次の例は、ニュース・サイトからニュースを抽出し、株式サイトから株価を抽出するロボット・ライブラリを開発するプロジェクト用の、**`NewsAndStocksProject`**という名前のプロジェクト・フォルダを示しています。

表10.5 ニュース抽出用プロジェクトの例

```
NewsAndStockProject/
  Library/
    News/
      CNN.robot
      Reuters.robot
      News.type
   Stocks/
      Nasdaq.robot
      NYSE.robot
      Stocks.type
```

このように、プロジェクトにはロボット・ファイルとタイプ・ファイルが含まれる**Library**フォルダがあり、NewsおよびStocksサブフォルダに分割されています。

Design Studioを閉じても、開いていたプロジェクトやファイルは記憶されるため、次回にDesign Studioを開くとそれらが再度開きます。

現在のプロジェクト

Design Studioでは多くのプロジェクトを使用できますが、Kapow Katalystの他のアプリケーション(RoboServerなど)は常に特定のプロジェクトで機能し、そのプロジェクトは現在のプロジェクトと呼ばれます。Kapow Katalystをインストールすると、デフォルトのプロジェクトが作成され、現在のプロジェクトとして選択されます。Design Studioを最初に開いたとき、開いているプロジェクトはこの現在のプロジェクトのみです。Design Studioを閉じる前にすべてのプロジェクトを閉じた場合、次回にDesign Studioを開くと、選択されている現在のプロジェクトが開きます。

現在のプロジェクトの選択を変更するには、Settingsアプリケーションを開き、「プロジェクト」タブの現在のプロジェクト・フォルダ・プロパティに新しいプロパティ・フォルダへのパスを指定し、「OK」をクリックしてSettingsを閉じます。

ロボット・プロジェクトの操作

新規プロジェクトを作成するには、「ファイル」メニューから「新規プロジェクト...」を選択します。これによって、次に示すように「新規プロジェクト」ダイアログが開きます。ここで、新規プロジェクトの名前と場所を入力します。ダイアログで「終了」を押すと、指定した場所に新規プロジェクトが作成され、プロジェクト・フォルダの名前はプロジェクトに対して指定した名前になります。名前を**MyProject**、場所を**C:/KapowProjects**と入力した場合は、次のフォルダが作成されます。

c:/KapowProjects/MyProject c:/KapowProjects/MyProject/Library

既存のプロジェクトを開くには、「ファイル」メニューからプロジェクトを開く...を選択し、開いたプロジェクトを開くダイアログで既存のプロジェクトのプロジェクト・フォルダを選択します。

プロジェクトを再度閉じるには、プロジェクト・ビューでプロジェクトを右クリックし、ポップアップ・メニューから「閉じる」を選択します。「ファイル」メニューからすべてのプロジェクトを閉じることもできます。

ロボット・ライブラリ・ファイル

ランタイム環境(RoboServerなど)でロボット・ライブラリを配布およびデプロイする場合は、ロボット・ライブラリをロボット・ライブラリ・ファイルと呼ばれる単一ファイルに圧縮できます。これを実行するには、Design Studioの「ツール」メニューからロボット・ライブラリ・ファイルの作成を選択します。これによって、現在のエディタ内でファイルのロボット・ライブラリに含まれているすべてのファイルが圧縮され、選択した名前が付けられた単一ファイルとして保存されます。最新の変更内容をロボット・ライブラリ・ファイルに取り込むには、この操作を行う前に、開いている

すべてのファイル(ロボット、タイプなど)を保存する必要があることに注意してください。その後に、RoboServerに対してロボット・ライブラリ・ファイルを使用可能にし、ロボット・ライブラリからロボットを実行できます。この方法の詳細は、「RoboServerユーザーズ・ガイド」を参照してください。

前述のように、ロボット・ライブラリにはロボットで使用されるファイルを含めることができます。ロボットが属しているロボット・ライブラリのファイルから、ページをロードできます。これを行うには、**library**という名前の特別な非標準プロトコルを使用します。たとえば、**MyPage.html**というファイルがロボット・ライブラリ・フォルダ内の**MyFolder**フォルダに格納されている場合、次のURLを使用してこのファイルからロードできます。

```
library:/MyFolder/MyPage.html
```

これは、ロボット・ライブラリがフォルダとして表示されているか、またはロボット・ライブラリ・ファイルに圧縮されているかに関係なく機能します。

データベースとの相互作用

データベースとの相互作用が必要な場合が多くあります。次の各項では、Design Studioからこれを行う時期と方法について説明します。

データベース・マッピング

ロボットは様々なデータベース・アクセス・ステップ(データベースに格納など)を介してデータベースへのアクセスが必要になる場合があり、これを行うには、それらのステップに対して指定データベースへの参照を設定する必要があります。ロボットがRoboServerで正常に実行されるには、ロボットで使用される指定データベースにRoboServerからアクセスできる必要があります。

ただし、Design Studioでロボットを設計するときは、RoboServerからは使用できないローカル・データベースを使用すると便利な場合が多くあります。ロボットをデプロイする前に様々なデータベース・アクセス・ステップの指定データベースを変更するのではなく、Design Studioでは追加の抽象レイヤーであるデータベース・マッピングを使用してこの問題を克服できます。データベース・マッピングによって、ロボットのデータベース・アクセス・ステップの指定データベースをDesign Studioデータベースにマップします。ロボットがDesign Studio内から実行されるかぎり、データベース・アクセス・ステップの指定データベースは、マッピングによって指定されたDesign Studioデータベースにマップされます。したがって、Design Studioのユーザーは、ロボットを設計およびテストするときにローカル・データベースを使用でき、ロボットをデプロイする前にデータベース・アクセス・ステップの参照先指定データベースを変更する必要がありません。

また、データベース・マッピングを使用すると、Design Studioのユーザーは、別のデータベースを指し示すように再構成するだけで、ロボットが別のデータベースに値を格納するように簡単に設定できます。

通常、Design Studioでのデータベース・マッピングの使用方法は非常に単純ですが、いくつかの事項を理解しておくと役立ちます。

データベース・マッピングは、マップするデータベースに加えて、ユーザーがマッピングおよび参照先データベースを正しく構成するために役立つ様々な警告をDesign Studioで表示するかどうかを単に定義する小さい構成ファイルです。マッピングの名前は、構成ファイルのファイル名です。これは、ファイル名が「objectdb」のマッピングを作成した場合、ロボットではマッピングが指し示すデータベースに「objectdb」という名前でアクセスできることを意味します。

データベース・マッピングは、次のいくつかの方法で作成できます。

- 「ファイル」メニューから、新規データベース・マッピングを選択します。これによってウィザードが開き、マッピング・ファイルの名前の入力を求められ、マッピング構成オプションが表示されます。ウィザードが終了すると、選択したプロジェクトとフォルダにマッピングが作成され、ロボットで使用できるようになります。
- データベース・ビューで、プロジェクトに関連付けるデータベースを右クリックし、Add to Projectを選択して、データベースを追加するプロジェクトを選択します。これによって、データ

ベース・マッピングで使用する名前(マッピング・ファイル名、およびデータベースにアクセスするときの名前)の入力を求めるプロンプトが表示されます。名前が提示されていることに注意してください。これはデフォルトのデータベース名で、Design Studio以外の他のKapow Katalystアプリケーションでこのデータベースにアクセスする際に使用される名前です。

- データベース警告システムから行います。 Design Studio で、マッピングされていないデータベースを使用するロボットを開くと、このことを知らせる警告が表示され、ロボットで参照されるデータベースの名前を使用してマッピングを作成するかどうかを尋ねられます。これによって、ロボットを変更せずに、たとえば、他のデータベースを定義した他の開発者から送信されたロボットをすぐに実行できます。

タイプおよびデータベース

ロボットが変数の値をデータベースに出力する場合、これらの変数のタイプでは、値をデータベースに格納するときに使用されるキーの一部になる属性を定義する必要があります。値のデータベース・キーは、データベース・キーの一部としてマークされる属性のセキュアなハッシュとして計算されます。

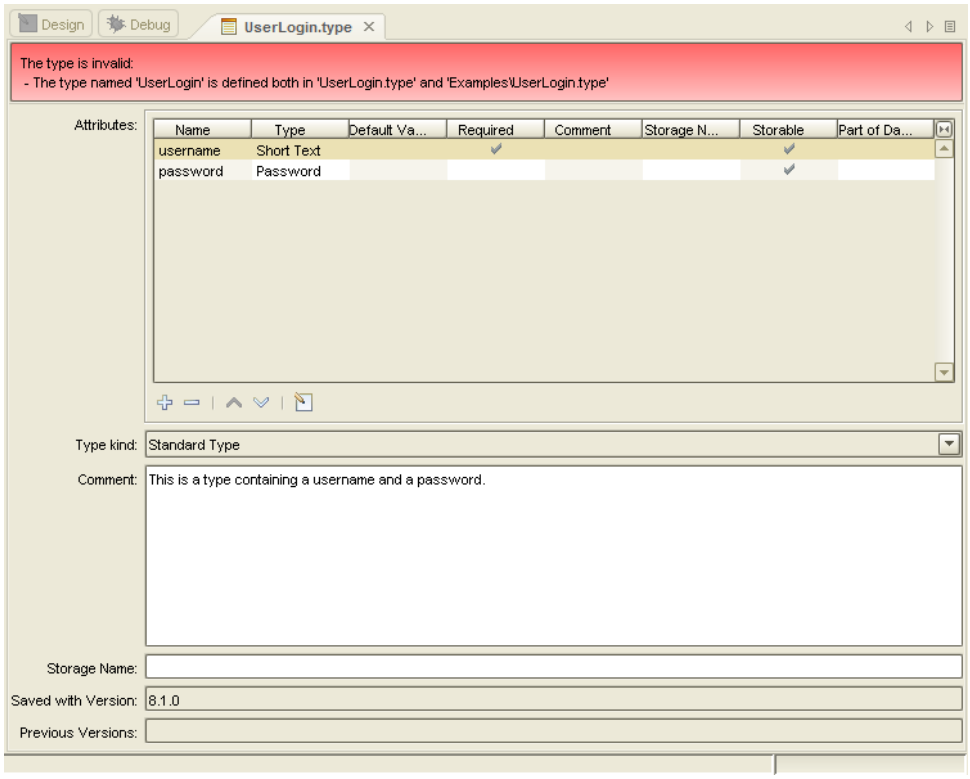
また、属性定義の中で、記憶域名も指定できます。これはオプションで、属性を格納するときに使用する別の名前です。

データベースに格納アクションを使用して値をデータベース記憶域に保存するときは、使用可能なデータベースに適切なデータベース表が存在している必要があります。これは、タイプの属性と一致する列が表に含まれることが要件であることを意味します。

Design Studioを使用して適切なデータベース表を設定する方法の詳細は、「データベース表の作成と削除」を参照してください。Design Studioでデータベース接続を設定する方法の詳細は、「Design Studioの設定」を参照してください。

データベース警告

データベース警告は、データベース・マッピング、ロボットおよび参照先データベースを正しく構成するのに役立ちます。警告システムによって、タイプ検証の問題、表の欠落、データベース・マッピングの欠落などの潜在的な問題が自動的に監視され、問題が発生した場合は、警告メッセージがステータス・バーに表示されます(次の図を参照)。



タイプ・エディタでの無効なタイプ

警告システムでは、ロボットで使用されているデータベース名も監視して、欠落しているマッピングの作成も支援します。また、システムではデータベースの浅い監視を実行しており、データベースを常にpingしてデータベースがオンラインかどうかを確認するのではなく、必要な場合に情報を更新します。これは、過剰な回数のデータベース問合せを防ぐために、システムで関連のデータベース表の表構造をキャッシュし、そのキャッシュを使用して警告を計算することを意味します。キャッシュの再作成は、それが必要になる事象が発生したことをDesign Studioが認識したときに実行されます。データベース表の外部での変更、またはデータベースの可用性は監視対象ではありません。ただし、データベース・キャッシュを再作成する対象が1つのデータベースかすべてのデータベースかを選択するオプションがあります。このオプションは、データベース・ビュー、および関連する警告から使用可能になります。たとえば、データベースのキャッシュを再作成するには、データベース・ビューでデータベースを右クリックし、「リフレッシュ」を選択します。

データベース表の作成と削除

抽出した変数値をデータベースに格納する場合は、照合表をデータベースに作成する必要があります。Design Studioでは、この表の作成を支援します。これを行うために、ユーザーが作成したタイプが検証され、新しいデータベース表の作成に適したSQLが生成されます。したがって、あるタイプの変数の値を格納するときは、そのタイプを表す表がデータベースに存在する必要があります。

「ツール」メニューからデータベース表の作成()を選択するとウィンドウが開き、データベースの名前、データベースのタイプ、および表を作成するタイプを選択できます。SQLの生成をクリックすると、表を作成するためのSQL文の候補が表示され、ユーザーはこれを変更したり、実行または保存できます。

表示されるSQLは推奨される候補であるため、必要な場合、ユーザーはニーズにあわせて文を変更できます。たとえば、データベース領域を節約するために短いテキスト属性の列タイプをVARCHAR(255)からVARCHAR(50)に変更したり、自動的に増分される主キーを追加できます。ただし、通常の状況では、表名や一部の列名の変更、または一部の列の削除は行わないでください。

注意: データベース表がすでに存在する場合、SQLを実行するとそのデータベース表はデータベースから削除されます(最初にDROP TABLE文があるため)。

データベースへのデータの格納

この項では、Kapow Katalystのデータベース記憶域の機能について説明します。

ObjectKey

タイプに対してデータベースに作成する表には、タイプの属性ごとに1つの列があり、さらに、ObjectKey、RobotName、ExecutionId、FirstExtracted、LastExtracted、ExtractedInLastRunおよびLastUpdatedの7つの事前定義フィールドが含まれます。最も重要なのはObjectKeyで、これは表の主キーです。

注意: 「ObjectKey」という名前は、Kapow Katalystで以前に使用されていた用語をそのまま使用しているためです。以前は、タイプおよび変数はオブジェクトと呼ばれていました。この詳細は、アップグレード・ガイドを参照してください。新しい用語に従うと、「ObjectKey」は「ValueKey」にする必要があります。ただし、名前を変更すると多くの下位互換性の問題が発生するため、古い名前をそのまま使用することになりました。

タイプのObjectKeyは、データベースに値を格納するときに、そのタイプの変数から抽出された値を一意に識別するために使用されます。したがって、そのタイプの値を一意に識別するものを把握する必要があります。車のリポジトリを作成する場合、各車を一意に識別するにはVIN番号(車両登録番号)で十分です。野球の試合結果を収集する場合、各試合を一意に識別するには年、チーム名、球場および日付が必要です。

タイプを作成するときに、ObjectKeyの計算方法を選択できます。これを行うには、新しい属性を作成するときにデータベース・キーの一部オプションを選択します。車の例の場合は、VIN番号がデー

タベース・キーの一部とマークされる唯一の属性で、野球の試合結果の例の場合は、年、チーム名、球場および日付の各属性がすべてデータベース・キーの一部としてマークされます。

ロボット開発者は、タイプに対して定義されているデフォルトのアルゴリズムを上書きする必要がある場合は、データベースに格納アクションでキーを直接指定することもできます。

データベース・キーの一部以外の属性は、キー以外のフィールドと呼ばれることがあります。たとえば、車には価格属性がありますが、価格が変更されても同じ車として識別されます。

データベースに格納

Kapow Katalystには、データベース内の値を管理するために、データベースに格納、データベースの検索、データベースからの削除の3つのアクションが用意されています。検索および削除アクションは通常のアクションですが、データベースに格納には値を格納する以外の機能もあります。

データベースに格納を使用すると、(あるタイプの)新しい値を表に挿入するか、または以前に格納された既存の値を更新できます。次に、正確な処理内容を示します。

1. ある変数の値を格納するときは、変数のタイプがデータベース・キーの一部とマークされている属性の変数の値に基づいてObjectKeyが計算され、ロボット開発者がアクションに対してキーを指定した場合は、そのキーがかわりに使用されます。

2. 計算されたキーを使用して、値がデータベース内にすでに存在するかどうかのチェックが行われます。

3. 値が存在しない場合は、(このObjectKeyで)新しい行がデータベースに挿入されます。

4. 値がすでに存在する場合は、値が更新され、つまりキー以外のすべての属性が(このObjectKeyで)表に書き込まれます。

事前定義フィールド

値が挿入されるたびに、7つの事前定義フィールドがすべて更新されます。更新時には、一部のフィールドのみ変更されます。次の表に、その概要を示します。

表10.6 7つの事前定義フィールドおよび変更時期

フィールド	説明	変更時期
ObjectKey	この値の主キー	挿入
RobotName	この値を格納したロボットの名前	挿入および更新
ExecutionId	この値を格納したロボット実行の実行ID	挿入および更新
FirstExtracted	最初に値が格納された時間。	挿入
LastExtracted	最後に値が格納された時間。	挿入および更新
LastUpdated	値が最後に更新された日付。	更新*
ExtractedInLastRun	最新の実行で値が抽出されたかどうか(yおよびnを使用)。	挿入および更新*

(ロボットでデータベースに格納を使用した)各ロボット実行の後に、以前にこのロボットによって収集されて今回の実行で格納されなかったすべての値について、ExtractedInLastRun setが「n」に設定され、LastUpdatedが現在に設定されて、最新の実行時に値がWebサイトで見つからなかったことが示されます。

注意: 値が前回の実行で見つかったが、キー以外のフィールドは変更されなかった場合、LastUpdatedは更新されません。ただし、値が前回の実行では見つからなかったが、それよりも前の実行では見つかっている場合、キー以外のフィールドが変更されていなくても、LastUpdatedは更新されます。これは、その値はサイトから削除されて、後で再度表示されたためです。

収集表

Kapow Katalystで作成される表は、ロボットがこれらの表にデータを収集するため、多くの場合、収集表と呼ばれます。

最後にロボットが実行されたときにWebサイトで使用可能な情報を検索するには、次のようなSQLを使用できます。

```
SELECT * FROM table WHERE ExtractedInLastRun = 'y'
```

ただし、ロボットがデータを表に格納している最中に、同時に表に対して問合せを実行すると、前回の実行によるデータと、実行中のロボットがその時点までに格納したデータが混在した結果が返されます。したがって、データを収集表から別の本番表セットにコピーして、安定したデータ・セットに対して問合せを実行することをお薦めします。

ロボットがデータベースにデータを格納するために使用するソリューションは多くありますが、そのほとんどは、次の3つのシナリオのいずれかに該当します。

表10.7 本番表への収集データのコピー

シナリオ	説明
Webサイトと一致するリポジトリ(小規模データ・セット)	Webサイト上の項目と1対1で一致するリポジトリを使用する考え方です。 これを実現する最も簡単な方法は本番表を使用することで、本番表は ロボットの実行が終了するたびに切り捨てられ(すべての行が削除され)、ExtractedInLastRunがyのすべてのレコードが収集表からこの表にコピーされます。これは、小規模データ・セットに適しています。
Webサイトと一致するリポジトリ(大規模データ・セット)	前述の説明と同様ですが、データ・セットが大きすぎて、毎回ロボットが実行された後にすべてのデータをコピーできないため、かわりに、各ロボット実行の後に、発生した変更に基づいて本番表を更新します。 この場合は、LastUpdatedフィールドが役立ちます。更新されたすべての値は、LastUpdatedフィールド値がロボットの開始時間より大きくなります。開始時間はデータベース・ロギング表から取得するか、または、ロボットで開始時間を任意の場所に格納させることができます。 削除された値は次のように検出できます: **SELECT * FROM table WHERE LastUpdated > 'StartTime' AND ExtractedInLastRun = 'n'** 新しい値は次のように検出できます: **SELECT * FROM table WHERE LastUpdated > 'StartTime' AND ExtractedInLastRun = 'y' AND FirstExtracted > 'StartTime'** 更新された値は次のように検出できます: **SELECT * FROM table WHERE LastUpdated > 'StartTime' AND ExtractedInLastRun = 'y' AND FirstExtracted < 'StartTime'** 次に、これに応じて本番表を更新します。
履歴データ	デフォルト設定では、値が最初に抽出された時期、および値が最後に更新された時期を確認できますが、ロボットの5回目、7回目および10回目の実行時に存在したが、6回目と8回目の実行時には存在しなかった値を確認することはできません。 このような場合、ロボット実行の後にすべてのデータを収集表から別の表にコピーする必要がありますが、新しい表ではObjectKeyを主キーにできません。かわりに、RUN_IDという追加の列を作成し、この列とObjectKeyを組み合せて複合主キーを作成します。RUN_IDが不要な場合は、自動的に増分する列を作成し、その列をセカンダリ表の主キーとして使用できます。各実行の前に、収集表が切り捨てられます。

すべての事前定義フィールドを本番表にコピーする必要はなく、本番表を更新するために必要なのはObjectKeyのみです。

同時実行に関する考慮事項

複数のロボットが同じタイプの値を同じデータベースに格納する場合は、考慮する事項がいくつかあります。

- 値が格納されるたびに、RobotName列が更新されます。2つのロボットが(ObjectKeyで識別される)同じ値を格納すると、2つのロボットの実行終了後には最後の値のみが表示されます。
- 2つのロボットが同じ値を完全に同時に格納すると、エラーが発生します。両方とも、値が表に存在しないことを検出して値を挿入しようとしますが、成功するのは1つのみです。実際には完全に同じ値であるため、ほとんどの場合、エラーは無視できます。
- 同じロボットを同時に2回実行し、そのロボットがデータをデータベースに格納すると、ExtractedInLastRun列の使用方法が崩れてしまいます。

  1回目のロボットの実行が終了すると、ロボットが格納しなかったすべての値についてExtractedInLastRunがnに更新されます。これには、2回目のロボットがその時点までに格納した値がすべて含まれます。その後、2回目のロボットが終了すると、1回目のロボットが格納したすべての値についてExtractedInLastRunがnに設定されるため、1回目の実行が完全に無効になります。

値の関係

記憶域システムには、値の間の関係を自動的に管理する方法はありません。「個人」タイプの値と「アドレス」タイプの値があり、それらをリンクする場合は、ユーザーがこのリンクを保守する必要があります。

リンクを作成する最も簡単な方法は、「個人」の値のObjectKeyを、この個人にリンクする「アドレス」の値の外部キーにすることです。

ObjectKeyがタイプから自動的に計算される場合は、アドレス値を格納する前に、ObjectKeyの計算アクションを使用して、キーを生成し、そのキーを各アドレス値に割り当てることができます。

格納される値の間に関係があるロボットを作成するときは、注意が必要です。「個人」の値を格納するときにエラーが発生した場合は、「アドレス」の値が格納されていないことを確認する必要があります。

ObjectKeyに関する注意

MySQL、OracleまたはSybaseを使用する場合は、ObjectKeyに関して次の点に注意する必要があります。

- Oracleでは、空の文字列がnullとして格納されます。
- Sybaseでは、空の文字列が" "(1つの空白の文字列)として格納されます。
- MySQLでは、タイムスタンプにミリ秒精度がありません。

これら3つのすべてのケースで、データがデータベースに格納されるときにデータが失われる可能性があります。ObjectKeyは指定の変数のデータに基づいてロボット内部で計算されるため、データベース・キーの一部としてマークされた属性でデータが失われた場合、後でデータベースから値をロードしてObjectKeyを再計算するとObjectKeyが異なります。

まとめ

ここまで、Design Studioの主要な概念が紹介され、Design Studioのユーザー・インタフェースのツアーを体験し、ロボット、タイプ、ステップ・アクションおよびデータ・コンバータの概念を把握して、パターンや式についても理解してきました。

ここでは、このDesign Studioに関するすべての知識を使用して、ロボットを作成します。ただし、最初に、一般的なロボットの構築方法、つまりロボットの構造の概要を理解する必要があります。

ロボットは人間の動作を模倣するため、人間がブラウザを使用してインターネット上のコンテンツを検索するときに行うのとほぼ同じ動作をロボットが行います。最初に、コンテンツの検索から始めます。検出すると、それを読んで処理します。同様に、ほとんどのロボットは、おおまかにナビゲーション部分と抽出部分の2つに分割できます。

ナビゲーションとは、コンテンツの場所に移動することです。ナビゲーションには、主に、ページのロードおよびフォームの発行が含まれます。Design Studioでナビゲートするとき、通常はクリック・アクションを使用して、WebページおよびWebページ間をナビゲートします。

抽出とは、正しいコンテンツを取得することです。抽出には、主に、ナビゲートしたWebページからコンテンツを選択、コピーおよび標準化することが含まれます。Design Studioで抽出するとき、通常は、タグのテスト・アクションを使用して対象外(不要)のコンテンツをスキップし、抽出アクションを使用してコンテンツを変数にコピーし、データ・コンバータを使用してコンテンツを標準化するため、必要な書式(日付と数字の正しい書式など)が得られます。抽出後に、データベースに格納または値を返すアクションによって値を出力します。

したがって、一般的なロボットは、Webサイト上の対象のコンテンツにナビゲートするために、ページのロードまたはクリック・アクションがそれぞれ含まれる1つ以上のステップで開始されます。次に、抽出アクションがそれぞれ含まれる1つ以上のステップを進み、抽出された値を格納または返すステップで終了します。

抽出するコンテンツは複数のページに存在するため、多くのロボットで、ナビゲーション部分と抽出部分は重複することに注意してください。これは、ユーザーが自分でコンテンツを検索するときに、複数のページを閲覧して必要なコンテンツを取得するのに似ています。

ほとんどのロボットには、これまで説明した以外のアクションが含まれています(たとえば、For Each Tagアクションは、外観が類似する複数のページをロードしたり、外観が類似する複数の表の行から値を抽出します)。ロボットには様々なタスクがあるため、そのニーズも異なります。このような理由から、Design Studioには多数のステップ・アクションとデータ・コンバータが含まれています。最も一般的に使用する基本的なステップ・アクションとデータ・コンバータを最初に理解してから、次の段階に進んでください。ほとんどのロボットは、少数のステップ・アクションとデータ・コンバータのみを使用して作成できることが多いようです。したがって、他のステップ・アクションやデータ・コンバータも使用する必要が生じるまでは、自分のお気に入りのステップ・アクションやデータ・コンバータを使用し続けてください。

適切な構成のロボットを記述する方法

ロボットはプログラムであるため、他のプログラミング言語でプログラムを記述する場合と同じ理由で、適切な構成のロボットを記述することが重要です。適切に構成されていないロボットを記述することは、章や目次がない本を書くのと同じです。適切な構成のロボットを記述することが重要である理由は、次のとおりです。

- ロボットをドキュメント化するのに役立ちます。
- ロボットの保守が簡単になります。
- ロボットの作成方法を簡単に見つけることができます。

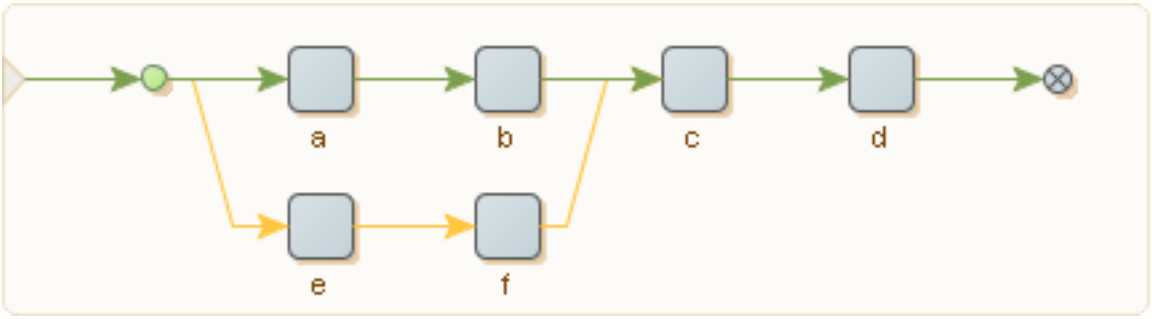
適切な構成のロボットを記述する別の利点は、Design Studioでロボットのロードが速くなることが多く、通常は編集時(ロボット・ビューの更新時など)の応答も速くなることです。

適切な構成のロボットを記述するために便利に使用できる主なツールは、スニペット・ステップとグループ・ステップの2つです。これら2タイプのステップは、ロボットの一部分を取り出し、わかりやすい名前を指定して、単一ステップにまとめる方法として使用されます。この方法では、ロボットの一部分の動作を詳細に記憶する必要がなく、ロボットの他の問題に集中できます。これは、他のプログラミング言語(メソッド、関数、プロシージャなど)の概念に非常によく似ています。

適切に定義されたタスクを実行するステップをまとめて非表示にするためにグループ・ステップを使用し、グループ・ステップにわかりやすい名前を付けることができます(サイトXへのログイン、レポート・エラーなど)。グループ・ステップには、グループ内のステップの動作を説明する、比較的短くわかりやすい名前を付けることが重要です。これができない場合は、適切に定義されたタスクを実行していないことが考えられます。グループ・ステップを導入すると、名前によってロボットの一部分の動作が説明されるため、ロボットをドキュメント化するのに役立ちます。

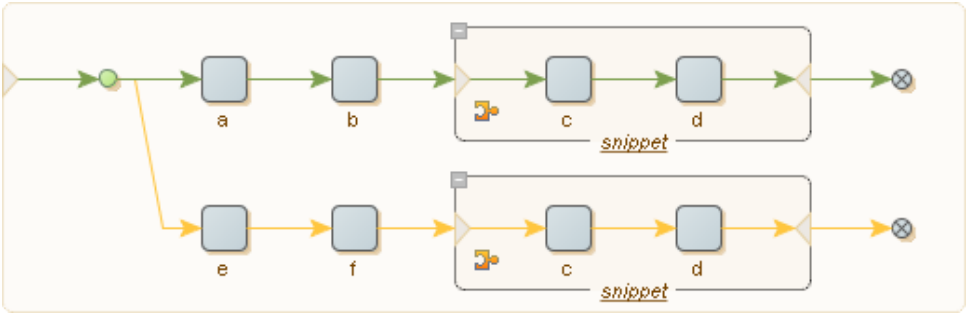
スニペットは主に複数のロボット間で機能を共有するために導入されますが、単一のロボット内でロボットを構成するために使用することもできます。ロボット内に複数の分岐で使用されるステップの集合が含まれる場合(たとえば、ロボットの異なる部分から出る端をステップの先頭に結合する場合)、このようなステップの共有のかわりに、ステップを含むスニペットを導入できます。

端を結合するかわりに、スニペットとグループを使用してロボットを構成する方法の単純な例を説明します。次のようなロボットがあるとします。



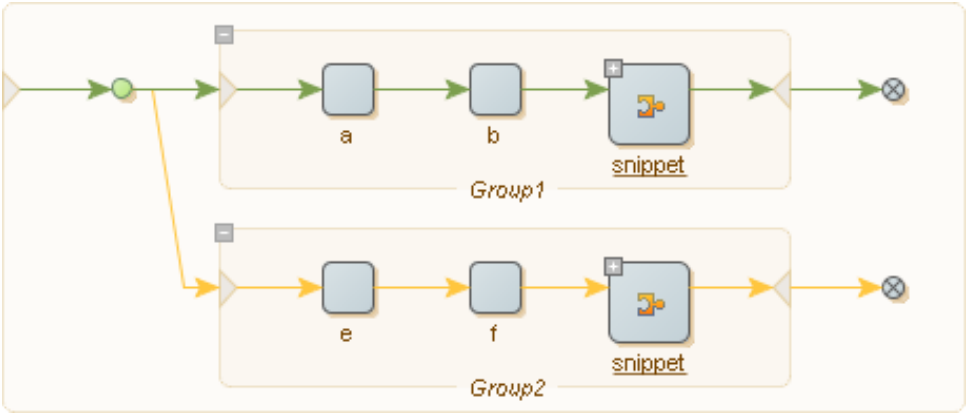
端の結合を使用したステップの共有

最後の2つのステップcとdは、ステップaとcから開始される2つの分岐によって共有されています。実際のロボットはこれよりも大規模で、このようにステップを共有する分岐が複数あり、関連するステップが離れている可能性があります。このため、ロボットの全体像を把握するのが非常に困難になる場合があります。適切に構成されたロボットを作成する最初のステップとして、ステップcとdを含むスニペット・ステップを紹介します。これは次のようになります。



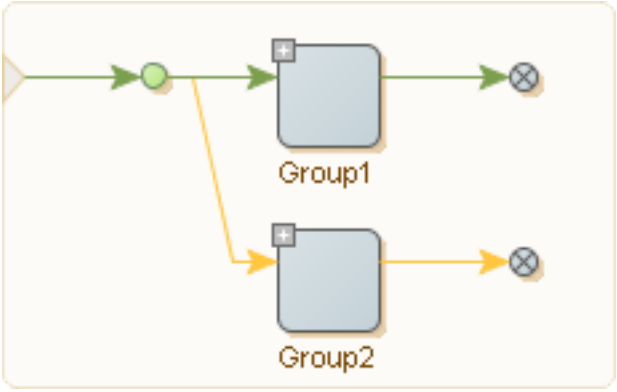
スニペットを使用したステップの共有

これで、スニペット・ステップ内のステップを編集しても、その変更が2つの分岐で共有されるようになります。さらに、2つの分岐をそれぞれグループ・ステップに組み込んでロボットを構成できます。



分岐ステップのグループ化

最後に、2つのグループ・ステップを閉じると、次のような単純なロボットになります。



再構成による最終結果

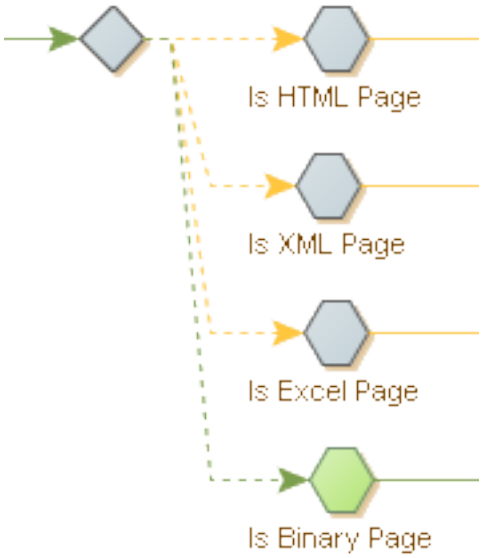
このように構成されたロボットは2つのタスクを実行し、1つはGroup1によって実行され、もう1つはGroup2によって実行されます。これら2つのグループにわかりやすい名前を指定することによって、ロボットは当初よりも論理的な構造になります。

これは非常に単純な例ですが、ロボットが特定のサイズを超えてロボット・ビュー全体に縦横に端が含まれると、ロボットの全体像がすぐに把握できなくなり、スパゲティ・コードと呼ばれるような状態になってしまいます。前述の方法でロボットを再構成すると、全体像を再び把握するのに役立ちます。

ページのタイプを判別する方法

HTMLページのリンクをクリックするなどしてページをロードしたとき、ロードしたページもHTMLページであることを確認できません。ロードしたページは、XMLページ、Excelページ、またはDesign Studioでサポートされていないコンテンツ・タイプのページである場合があります。ロードしたページのページ・タイプが、各ページ・タイプを個々に処理するための分岐を追加できるタイプかどうかをロボットでテストするには、どうすればよいでしょうか。これを行うには、ページ・タイプのテスト・ステップを使用します。

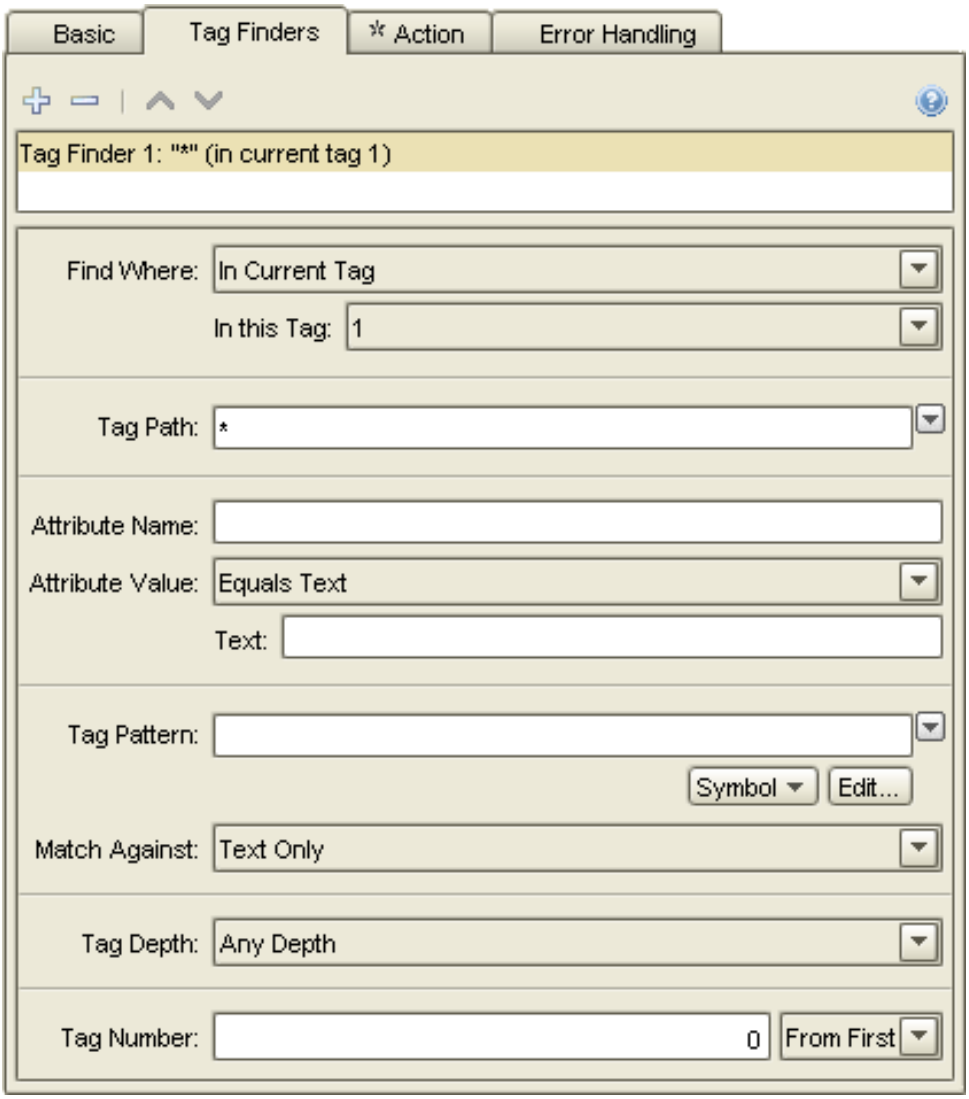
ページ・タイプのテスト・ステップは、現在のウィンドウ内のページのタイプをテストします(現在、ページ・タイプにはHTML、XML、Excelおよびバイナリの4種類があります)。次の図に、異なる4つのページ・タイプをテストするロボットの一部を示します。これは、1つの試行ステップおよびその後の4つの分岐で構成され、各分岐は各ページ・タイプをテストするように構成された、ページ・タイプのテスト・ステップで開始されています。ページ・タイプのテスト・ステップの名前は、ページ・タイプを示すように変更されています。



ページ・タイプをテストするステップ

タグ・ファインダを使用する方法

タグ・ファインダを使用して、ページにあるタグを検索します。タグ・ファインダはステップ内で使用するのが一般的で、タグ・ファインダを使用して、選択したアクションを適用するタグを検索します。次に示すように、現在のステップのタグ・ファインダ・リストは、ステップ・ビューのタグ・ファインダ・タブに表示されます。



ステップ・ビューのファインダ・タブ

タグ・パスの理解

タグ・ファインダを理解するには、タグ・パスの概念が重要です。タグ・パスは、ページ上でのタグの位置を表すコンパクトなテキストです。次のタグ・パスを考えてみます。

```
html.body.div.a
```

このタグ・パスは、<html>タグ内の<body>タグ内にある、 タグ内の<a>タグを指します。

1つのタグ・パスが同じページの複数のタグと一致する場合があります。たとえば、前述のタグ・パスは、3番目を除いて、このページのすべての<a>タグと一致します。

```
<html>
<body>
<div>
     <a href="url...">Link 1</a>
     <a href="url...">Link 2</a>
</div>
<p>
     <a href="url...">Link 3</a>
```

```
</p>
<div>
     <a href="url...">Link 4</a>
     <a href="url...">Link 5</a>
     <a href="url...">Link 6</a>
</div>
</body>
</html>
```

索引を使用すると、そのレベルで同じタイプのタグの中から特定のタグを示すことができます。次のタグ・パスを考えてみます。

```
html.body.div[1].a[0]
```

このタグ・パスは、<html>タグ内の<body>タグ内にある、2番目の タグ内の最初の<a>タグを指します。したがって、前述のページでこのタグ・パスに一致するのは、Link 4の<a>タグのみです。タグ・パスの索引は0から始まることに注意してください。前述の最初のタグ・パスのように、タグ・パスの特定タグに索引が指定されていない場合、パスはそのレベルで同じタイプのすべてのタグと一致します。索引が負数の場合、一致するタグは後方からカウントされ、最後の一致タグ(索引-1に対応)から始まります。次のタグ・パスを考えてみます。

```
html.body.div[-1].a[-2]
```

このタグ・パスは、<html>タグ内の<body>タグ内にある、最後の タグ内の最後から2番目の<a>タグを指します。したがって、前述のページでこのタグ・パスに一致するのは、Link 5の<a>タグのみです。

アスタリスク(*)を使用すると、任意のタイプの任意の数のタグを示すことができます。たとえば、次のタグ・パスを考えてみます。

```
html.*.table.*.a
```

これは、<table>タグ内の任意の場所にある<a>タグを示し、<table>タグ自体も<html>タグ内の任意の場所にあります。タグ・パスの前には暗黙的なアスタリスクが付いているため、ページ上のすべてのtableタグを示すには、"*.table"のかわりに単に"table"と記述できます。唯一の例外はピリオド(.)で始まるタグ・パスで、これはタグ・パスの前に暗黙的なアスタリスクがないことを意味するため、タグ・パスはページの最初のタグ(最上位レベル)から一致する必要があります。

アスタリスクを使用すると、長期にわたり変更される可能性がある重要でないタグ(レイアウト関連タグなど)を除外できるため、ページの変更に対してより強固なタグ・パスを作成できます。ただし、アスタリスクを使用すると、正しくないタグを誤って示すリスクも増加します。

次のタグ・パスに示すように、可能なタグを|で区切ったリストを指定できます。

```
html.*.p|div|td.a
```

このタグ・パスは、<html>タグ内の任意の場所にある<p>、 または<td>タグ内の<a>タグを指します。

タグ・パスでは、キーワード"text"を使用して、ページ上のテキストをその他のタグとして指します。技術的にテキストはタグではありませんが、タグ・パスではタグとして扱われて表示されます。たとえば、次のHTMLを考えてみます。

```
<html>
<body>
     <a href="url...">Link 1</a>
     <a href="url...">Link 2</a>
</body>
</html>
```

タグ・パス"html.body.a[1].text"は、テキスト"Link 2"を指します。

タグ・ファインダを使用する方法

次のプロパティを使用してタグ・ファインダを構成できます。

検索場所: このプロパティには、指定タグに関連するタグを検索する場所を指定できます。デフォルト値は、ページ内すべてで、指定タグを使用しないでタグを検索することを意味します。

タグ・パス: このプロパティには、前の項で説明したタグ・パスを指定できます。

属性名: このプロパティには、タグに特定の属性(位置合せなど)が必要であることを指定できます。

属性値: このプロパティには、タグに特定の値を持つ属性が必要であることを指定できます。「属性名」プロパティが設定されると、属性値がその特定の属性名にバインドされます。

- テキストと等しいは、属性値が指定のテキストと一致する必要があることを指定します。テキストは属性値全体と一致する必要があることに注意してください。
- テキストを含むは、属性値に指定のテキストが含まれる必要があることを指定します。
- パターンは、属性値がパターンと一致する必要があることを指定します。パターンは属性値全体と一致する必要があることに注意してください。

タグ・パターン: このプロパティには、タグ(タグ内のすべてのタグを含む)が一致する必要があるパターンを指定できます(例: .*.*Stock Quotes.*.*)。このプロパティはロボットのパフォーマンスに多大な影響を与える可能性があるため、このプロパティを使用する際は注意が必要です。これは、一致する1つのタグを検索するために、タグ・パターンがページ全体に複数回適用されるためです。これを回避する1つの方法は、一致対象プロパティでテキストのみを選択することです。

一致対象: このプロパティには、タグ・パターンがタグのテキストのみ、またはHTML全体が一致するように指定できます。デフォルトはテキストのみの一致で、通常はこれが処理が速いためです。

タグの深さ: このプロパティは、一致する複数のタグが一方の内部に含まれている場合に、どのタグを使用するかを決定します。デフォルト値はすべての深さで、一致するすべてのタグを受け入れます。最も外側のタグを選択すると最も外側のタグのみが受け入れられ、同様に、最も内側のタグを選択すると最も内側のタグのみが受け入れられます。

タグ番号: このプロパティは、複数のタグがタグ・パスおよび他の基準に一致する場合に、どのタグを使用するかを決定します。使用するタグの番号を、一致する最初のタグから順に、または最後のタグから逆順にカウントして指定します。

たとえば、"table"へのタグ・パスを設定するとき、タグ属性プロパティを"align=center"、タグ・パターン・プロパティを".*Business News.*"に設定すると、タグ・ファインダは、中央揃えされて、"Business News"というテキストを含む最初の<table>タグを検索します。

現在のステップのタグ・ファインダの構成

ロボット・エディタでは、現在のステップのタグ・ファインダをいくつかの方法で構成できます。最初の方法は、手動で構成する方法です。構成した後は(自動または手動)、ページ・ビューで  アイコンをクリックすると、タグ・ファインダによって検出されたタグを表示できます。

タグ・ファインダを構成する2番目の方法は、ページ・ビューでタグを選択し、  アイコンをクリックする方法です。これによって、単純モードでタグ・パスを使用して、選択したタグを検索するように、タグ・ファインダが構成されます。

3番目の方法は、ページ・ビューでタグを右クリックし、表示されるポップアップ・メニューからアクションを選択する方法です。メニューからタグの使用を選択すると、単純モードでタグ・パスを使用して、右クリックしたタグを検索するように、タグ・ファインダが構成されます。同様に、メニューから別のアクションを選択すると、対応するステップ・アクションが選択され、右クリックしたタグを検索するように、タグ・ファインダが構成されます。

タグ・ファインダを構成する4番目の方法は、新しいステップ・アクションを選択する方法です。アクションによっては、選択すると、そのアクションで通常使用するタグを検索するようにタグ・ファインダが構成されます。たとえば、フォームの発行アクションでは、1つのタグ・ファインダを使用し、タグ・パスを"form"に設定して、ページ内の最初の<form>タグを検索します。

フォームを発行する方法

フォームの発行は、ロボットの共通タスクです。たとえば、検索結果を取得して抽出するために検索フォームを発行したり、オーダー取引を行うためにオーダー・フォームを発行する必要がある場合があります。実際にフォームを発行する必要がない場合もありますが、単にフォーム発行を表すURLを作成したり、フォーム内の現在値を変更する場合があります。この項では、これらの方法について説明します。

単純なフォーム発行

Design Studioでフォームを発行する最も単純で推奨される方法は、一般的なブラウザでフォームを発行する方法と同じで、最初にフォームを入力し、次にフォーム発行ボタンをクリックします。

フォームに入力するには、次のアクションを使用できます。

- テキストの入力
- パスワードの入力
- オプションの選択
- 複数オプションの選択
- チェック・ボックスの設定
- ラジオ・ボタンの選択

フォームを発行するには、クリック・アクションを使用できます。

また、次のアクションを使用して、フィールド値(テキスト入力)のオプションまたはラジオ・ボタンをループできます。

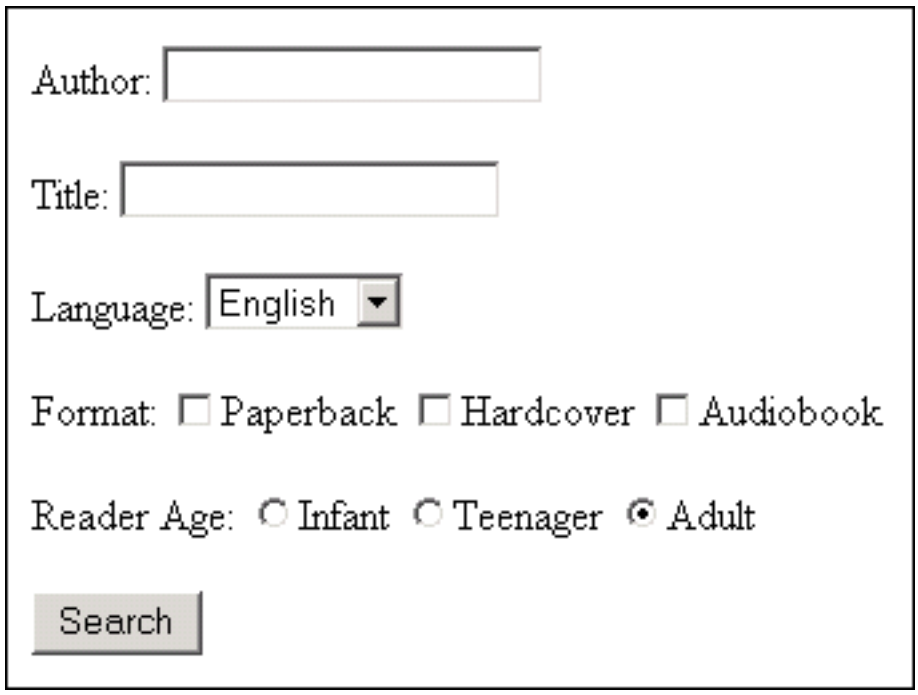
- Loop Field Values
- For Each Option
- For Each Radio Button

フォームの基本

この項では、フォームのいくつかの基本プロパティについて説明します。次の書籍検索フォームの例(最初はHTML、次はブラウザの表示)を考えてみます。

表10.8 書籍検索フォーム(HTML)

```
<html>
<body>
<form action="http://www.books.com/search.asp" method="get">
Author:
<input type="text" name="book_author">
<p>
Title:
<input type="text" name="book_title">
<p>
Language:
<select name="book_language">
<option value="lang_0" selected>English</option>
<option value="lang_1">French</option>
<option value="lang_2">German</option>
<option value="lang_3">Spanish</option>
</select>
<p>
Format:
<input type="checkbox" name="book_format" value="format_pb">Paperback
<input type="checkbox" name="book_format" value="format_hc">Hardcover
<input type="checkbox" name="book_format" value="format_ab">Audiobook
<p>
Reader Age:
<input type="radio" name="reader_age" value="age_inf">Infant
<input type="radio" name="reader_age" value="age_teen">Teenager
<input type="radio" name="reader_age" value="age_adult" checked>Adult
<p>
<input type="submit" value="Search">
</form>
</body>
</html>
```



書籍検索フォーム(ブラウザ)

フォームにはいくつかのフィールドが含まれます。たとえば、フォーム例の最初の<input>タグは"book_author"というフィールドを定義します。通常、フィールドの名前は、ユーザーに対してブラウザに表示される内容と異なることに注意してください。たとえば、"book_author"フィールドは、ブラウザには"book_author"ではなく"Author"という名前で表示されます。

1つのフィールドを複数のタグによって定義できます。たとえば、フォーム例の"book_format"フィールドは、3つの<input>タグによって定義されています。同じフィールド名を使用し、フィールド・タイプ(テキスト・フィールド、ラジオ・ボタン、チェック・ボックスなど)が同じ複数のタグによって、同じフィールドを定義します。

フィールドには、1つ以上の値を割り当てることができます。たとえば、"book_format"フィールドには、ペーパーバック形式を選択するための値"format_pb"を割り当てることができます。フィールド名と同様に、通常、フィールドに割り当てられる値は、ユーザーに対してブラウザに表示される内容と異なることに注意してください。たとえば、ペーパーバック形式を選択するとき、ユーザーには値"format_pb"ではなくテキスト"Paperback"が表示されます。フィールド・タイプに応じて、複数の値を同時に割り当てることが可能なフィールドがあります。たとえば、"book_format"はチェック・ボックス・フィールドであるため、ペーパーバック形式とハードカバー形式の両方を選択できるように、"book_format"フィールドに値"format_pb"と値"format_hc"の両方を割り当てることができます。

ほとんどのフィールドにはデフォルト値が設定されています。デフォルト値は、フォーム内のフィールドに最初から割り当てられている値です。たとえば、"book_language"フィールドは、"selected"属性によってデフォルト値が"lang_0"に設定されています。

フォームは、フィールドの現在値をWebサイトに送信することによって発行されます。1つ以上の現在値があるフィールドのみ送信されます。たとえば、フォーム例の"book_format"フィールドでチェック・ボックスが選択されていない場合、このフィールドについて送信される値はありません。

ブラウザでは、通常、ユーザーが発行ボタンをクリックするとフォームの発行が発生します。発行ボタンには、通常の発行ボタンとイメージの発行ボタンの2種類があります。通常の発行ボタンは<button>タグまたは<input>タグを使用して定義され、いずれの場合も"type"属性は"submit"に設定されます。通常の発行ボタンにフィールド名と値がある場合、ボタンがクリックされると、そのフィールドは指定された値とともに送信されます。

イメージの発行ボタンは<input>タグを使用して定義され、"type"属性は"image"に設定されます。イメージの発行ボタンでは"button name.x"と"button name.y"という2つのフィールドが定義さ

れ、button nameは<input>タグの"name"属性に含まれる名前です。<input>タグに"name"属性がない場合、フィールドの名前は"x"と"y"になります。イメージの発行ボタンがクリックされると、これら2つのフィールドには、マウスがクリックされたイメージの位置のx軸とy軸が割り当てられます。Webサイトによっては、ユーザーがクリックした場所に応じて動作が異なるイメージ・マップを作成するために、この方法を使用する場合があります。

一部のフォームではJavaScriptを使用します。たとえば、<form>タグに、フォームが発行される前に実行されるJavaScriptが含まれる"onsubmit"属性を含めることができます。同様に、<input>タグに、ユーザーがフィールドをクリックすると実行されるJavaScriptが含まれる"onclick"属性を含めることができます。ロボットは、このJavaScriptを自動的に実行します。

パフォーマンス上の理由から、フォームの発行時にJavaScriptを実行しないようにする場合があります。これを行うには、フォーム発行ステップでオプションのJavaScriptの実行の選択を解除する必要があります。

使用するステップ・アクション

すでに説明したように、フォームを発行する最も単純な方法は、前述の適切なアクションを使用してフォームを入力することです。

1回のフォーム発行では必要な結果が得られないときは、フォームのループが必要になる場合があります。書籍検索フォームの例を考えてみます。選択可能なすべての言語とすべての読者年齢について書籍を検索する場合、サイトではこのような全般的な検索ができないため、1回のフォーム発行では検索を行うことができません。かわりに、言語と読者年齢をループして、言語と読者年齢の組合せごとにフォームを発行する必要があります。これを行うには、フォームのループ・アクション(Loop Field Values、For Each OptionおよびFor Each Radio Button)を使用します。フォームのループ・アクションではフォームを発行しないため、フォームの発行ボタンの1つをクリックする後続のクリック・アクションで、フォームの発行を別に行う必要があります。フィールド値の組合せをループするには、フォームを発行するクリック・アクションの前に、ステップ後にフォームのループ・アクションを使用するいくつかのステップを配置します。

フォームの発行を表すURLを作成する場合は、フォームの発行ボタンでURLの抽出アクションを使用します。

フォームのループ・アクションの使用

フォームのループ・アクションには、Loop Field Values、For Each OptionおよびFor Each Radio Buttonの3つがあり、それぞれ、テキスト入力(typeが"text"のINPUT要素およびTEXTAREA要素)、オプション(SELECT要素)およびラジオ・ボタン(typeが"radio"のINPUT要素)の3種類のフォーム・コントロールに対応しています。これらのループの使用方法については、次のビデオを視聴または読んでください。

フォームでのループの使用に関するビデオ

フォームをループするには、ループするフォーム・コントロールおよびその順序(これによって出力値が生成される順序が決定します)を決定する必要があります。これを行った後に、対応するフォーム・アクションを含むそれぞれのステップを挿入します。これを行うには、ページ・ビューでコントロールを右クリックし、ポップアップ・メニューから「フォーム」|<フォーム・アクション>を選択します。<フォーム・アクション>は該当するアクションで、たとえば、コントロールがテキスト入力コントロールの場合は、「フォーム」|Loop Field Valuesを選択します。

フォームのループ・アクションが実行されるたびに、HTLMページのフォーム・コントロール要素の値が変更されます。これは、ユーザーがブラウザで手動で実行した操作に対応し、フォーム・コントロールにJavaScriptイベントが添付されている場合は、このイベントが起動してJavaScriptが実行されます。場合によっては、このJavaScriptによってフォームが変更される場合があります(SELECT要素のオプションの変更など)。その場合は、ロボットで必要なときに正しいオプションが使用可能になるように、コントロールをループする正しい順序を慎重に選択する必要があります。通常は、ブラウザで手動で実行する場合の順序に従うと適切に機能します。

フォームのループ・アクションのすべてのステップをロボットに挿入した後は、フォームの発行ボタンの1つをクリックするクリック・アクションを含むステップを追加する必要があります。

ファイルのアップロード

ファイルのアップロードが可能なファイル・フィールドを含むフォームがあります。ファイル・フィールドは、次の例のように、typeがfileの<input>タグによって定義されます。

```
<INPUT type="file" name="attachedFile">
```

ファイルの選択アクションでは、次のように、ファイル・フィールドを使用してファイルをアップロードする方法が2つあります。

最初の方法は、ファイル・システムからファイルをアップロードする方法です。これを行うには、ドロップダウン・ボックスからローカル・ファイル・システムのファイルを選択し、ファイル名を入力します。フォームを発行すると、指定したファイルがファイル・システムからロードされ、フォーム発行の一部としてアップロードされます。

ファイル名は、ドライブ名(ある場合)およびファイルへのディレクトリ・パスを含めた絶対ファイル名である必要があります。

ファイルをアップロードする2番目の方法は最も一般的で、ファイル・システムからファイルをロードするのではなく、アップロードするファイル・コンテンツを指定する方法です。これを行うには、ドロップダウン・ボックスから変数に含まれるファイルを選択します。次に、ファイル・コンテンツという名前のドロップダウン・ボックスから、ファイル・コンテンツを保持する変数を選択できます。通常は、以前にターゲットの抽出アクションを使用してファイルをダウンロードしたバイナリ変数、または以前に抽出したテキストを含む変数からコンテンツを取得します。

オプションで、ファイルのコンテンツ・タイプとファイル名を指定できます。コンテンツ・タイプはコンテンツのMIMEタイプである必要があり、オプションで後にcharsetを指定できます。事前定義されたコンテンツ・タイプの1つを属性から取得して使用するか、またはカスタムのコンテンツ・タイプを指定できます。たとえば、イメージのコンテンツ・タイプは次のようになります。

```
image/gif
```

プレーン・テキストの場合は次のようになります。

```
text/plain; charset=iso-8859-1
```

ターゲットの抽出を使用してファイルをダウンロードするとき、ダウンロードしたデータのコンテンツ・タイプとファイル名を他の変数に格納できることに注意してください。この情報は、ファイルの選択アクションを使用してファイルをアップロードするときに使用できます。

ページ・ビューでのポップアップ・メニューの使用

ページ・ビューでは、フォームの発行アクションおよびフォームのループ・アクションを選択して構成するためのショートカットとして、ポップアップ・メニューを使用できます。現在のステップでフォームの発行アクションまたはフォームのループ・アクションを選択するには、ページ・ビューで<form>タグの内部を右クリックし、ポップアップ・メニューの「フォーム」サブメニューから、フォームの発行の使用またはフォームのループの使用を選択します。

現在のステップにフォームの発行アクションまたはフォームのループ・アクションが含まれている場合、値をフィールドに割り当てるには、ページ・ビューでそのフィールドを右クリックし、「フォーム」サブメニューからフィールドへの割当の追加を選択します。ダイアログが表示され、割り当てる値を選択できます。

現在のステップにフォームのループ・アクションが含まれている場合、フィールドをループするには、そのフィールドを右クリックし、「フォーム」サブメニューからフィールドのループの追加を

選択します。ダイアログが表示され、値を含むループ対象の1フィールド・フィールド・グループをフィールドに対して構成できます。

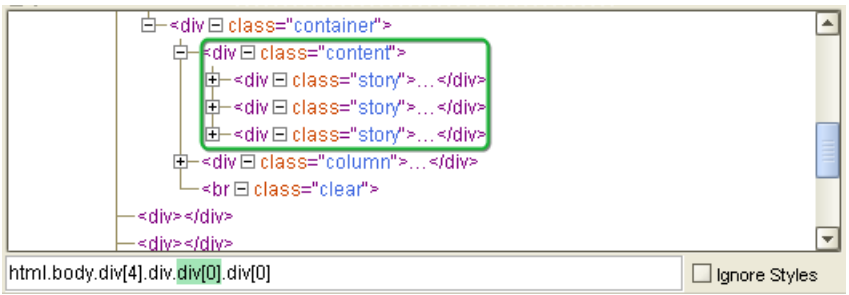
また、現在のステップにフォームの発行アクションまたはフォームのループ・アクションが含まれている場合、発行ボタンを選択するには、ページ・ビューでその発行ボタンを右クリックし、「フォーム」サブメニューから発行ボタンの選択を選択します。

ページ上のタグをループする方法

ロボットは、多くの場合、各要素でアクションを実行するためにページ上の要素をループする必要があります。たとえば、検索の各結果や表の各行から特定のプロパティを抽出する場合があります。この項では、この方法について説明します。複数の様々なタイプのループ・ステップを使用すると、同じ状況でも多くの方法で処理できることに留意してください。この項では、2つのケースについて説明します。

同じクラスのタグのループ

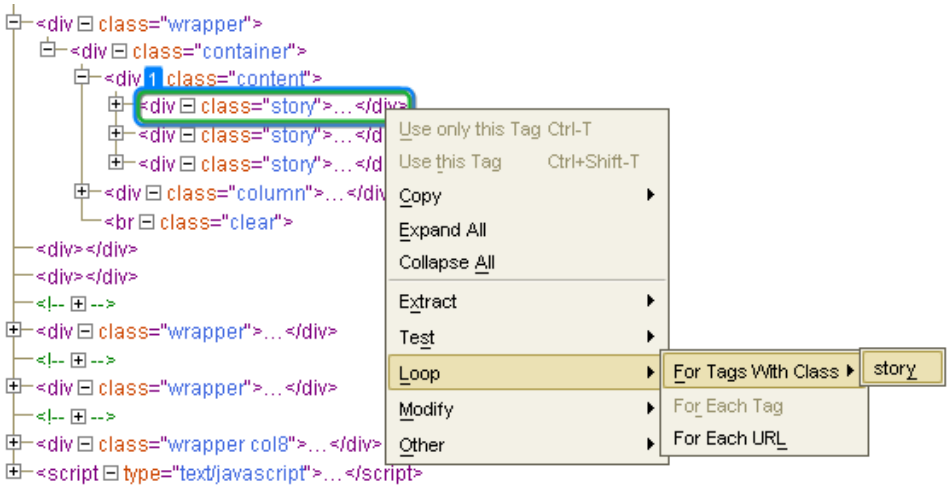
ループを設定するには、2つの方法があります。最初の方法はクラス属性を共有するすべてのタグをループする方法で、実行可能な場合は最も簡単です。



各div要素に属性class="story"があります。

同じクラスのタグをループできるかどうかを判断するには、HTMLビューで要素を検索します。前述のケースでは、属性**class="story"**の3つのdivタグをループできます。

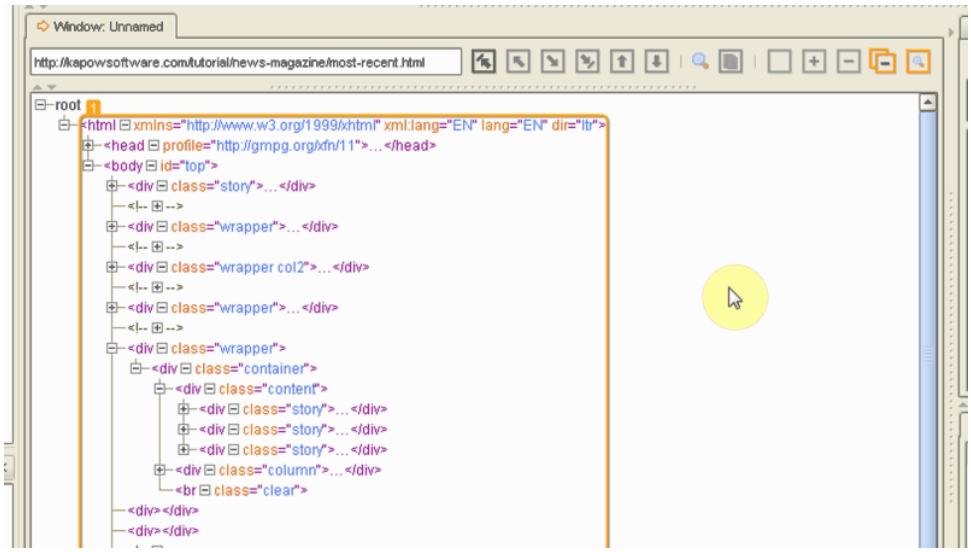
ループを作成する最も簡単な方法は、次に示すように、最初のタグを右クリックし、ループ、For Tags with Classの順に選択します。



右クリックして、class="story"のすべてのタグをループするステップを挿入します。

これによって、ページ上の指定クラスの要素がすべてループされるFor Each Tag Pathステップがロボットに作成されます。ループ・ステップ上の矢印を使用して、正しいタグがループに含まれていることを確認してください。指定クラスを使用するタグの中に、ループに含める必要がないタグがページに存在する場合があります。多くの場合、これらは簡単な修正でループから除外できます。For

Each Tag Pathステップをクリックし、HTMLビューを調べます。次に示すように、For Each Tag Pathでは、検出されたタグとしてページ全体を自動的に含めています。



For Each Tag Pathステップでループするタグを制限します。ページの最上部近くにある、class="story"のタグに注意してください。これは意図的にループから除外されます。

ただし、検出されたタグを変更することによって、別の指定タグ内のタグのみをループするようにロボットに強制できます。

ループが正常に作成された後、For Each Tag Pathステップの後に追加されたステップがループの繰返しごとに繰り返されます。

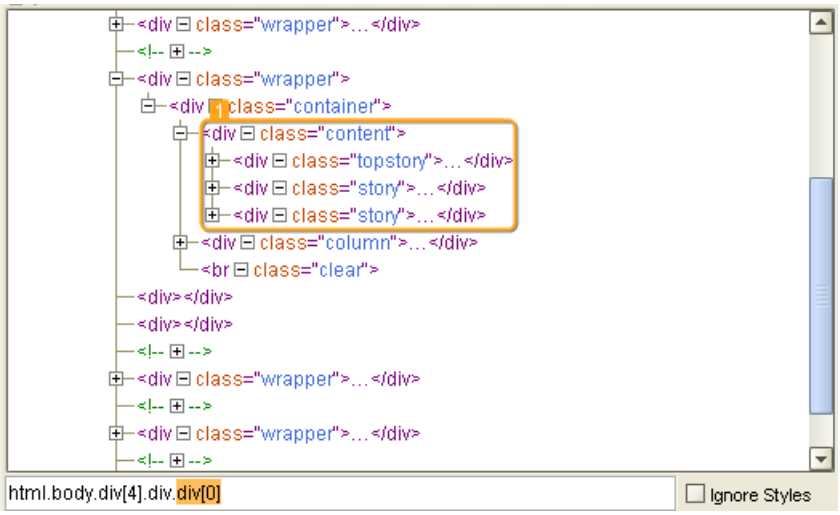
ループ・ステップの後のステップが繰返しごとに実行されます。

前述のロボット・ビューの例では、ループの繰返しごとに、ロボットは**title**と**preview**の2つのテキストを抽出してその値を返します。

異なるクラスのタグのループ

同じクラスのタグをループする方法は一般的なシナリオですが、最も簡単な方法というわけではありません。多くの場合、クラスが必ずしも同じではないタグをループする必要があります。ここでは、別のシナリオを説明します。

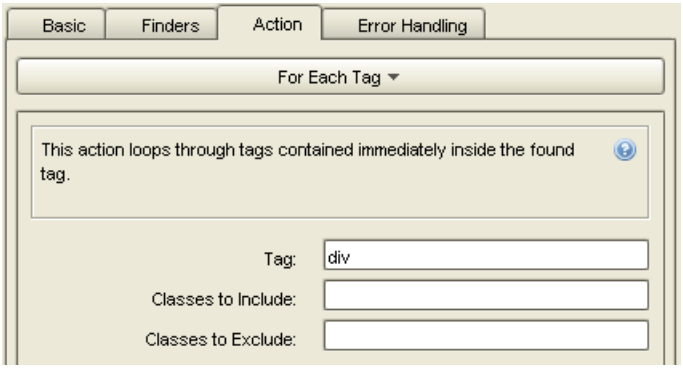
For Tags With Classが失敗するほとんどのケースで、For Each Tagステップは非常に有効です。For Each Tagステップは、検出されたタグのすぐ内側にある、すべてのタイプのタグをループします。ただし、右クリックして挿入する以外に、さらに少しの構成が必要です。次に、その使用方法を示します。



For Each Tagステップを使用して、検出されたタグのすぐ内側にある、指定タイプの各タグをループします。

ここでは、検出されたタグに3つのdivタグが含まれていますが、すべてのタグが同じクラスというわけではありません。これは、For Each Tagステップで処理可能なシナリオです。

最初に、  を使用して空のステップが挿入され、次に示すように、ステップ・ビューでステップに対してFor Each Tagアクションが選択されます。



For Each Tagステップは、1タイプのタグのみループするように構成されます。

次に、検出されたタグがページ・ビューから選択され、最後に、ループするタグのタイプがステップ・アクション・ビューから選択されます。前述の場合、これはdivタグです。

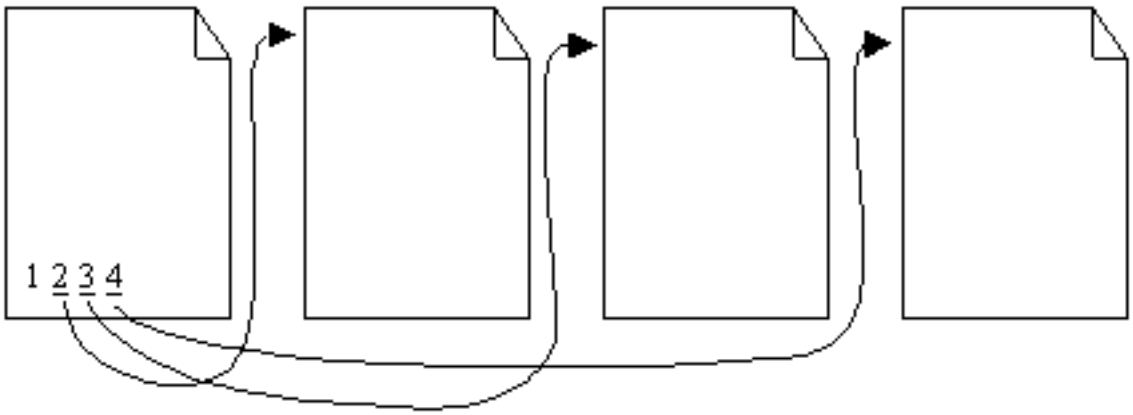
最後に、For Each Tagステップの後にステップが追加されます。ループの繰返しごとに、これらのステップが繰り返されます。

HTMLページをループする方法

多くの場合、ロボットは複数のページをループする必要があります。検索リクエストの結果を表示する多くのWebサイトでは、たとえば、各ページに検索の結果が20ずつ含まれる複数のページをループします。検索結果を取得するには、複数のページをループして、一度に1ページを処理する必要があります。この項では、この方法について説明します。

最初のページが他の全ページにリンクする場合

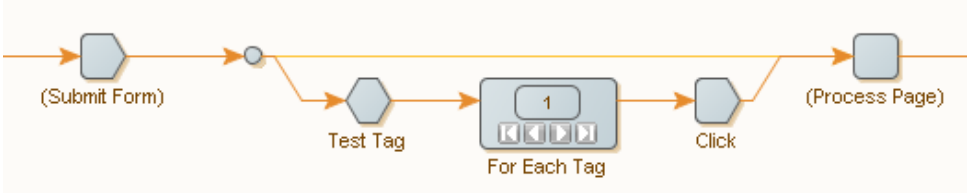
すべてのページをリンクする一般的な方法は2つあります。次に、最初の方法を示します。



最初のページが他の全ページにリンク

ここでは、最初のページに、他の全ページへの直接リンクが含まれています。つまり、最初のページから対応するリンクに従って任意のページに直接移動できます。最初のページに、最初のページへのリンクが含まれる場合もあります。

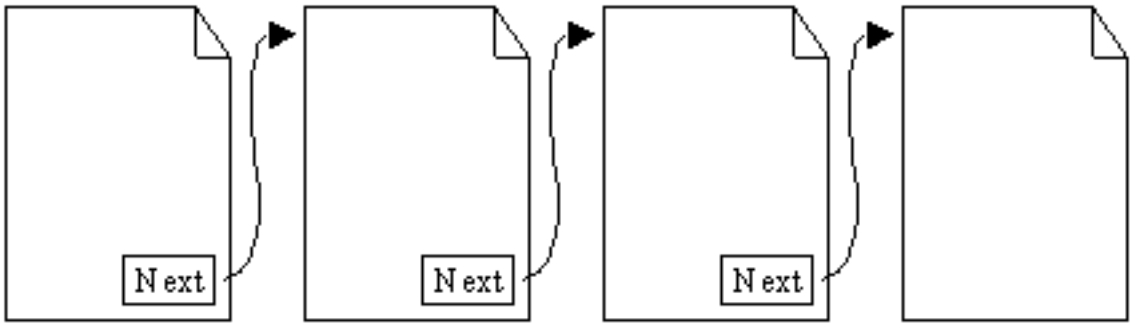
次のロボットの抜粋に示すように、このようなページはFor Each Tagステップを使用して簡単にループできます。



ここでは、"(Submit Form)"という名前のステップで表されている検索リクエストから、結果ページをループします。最初の結果ページは直接処理されるため、フォーム発行ステップから、ページを処理するステップ("(Process Page)"という名前のステップで表されている)が直接接続されています。残りのページは、フォーム発行ステップからの2番目の分岐で、For Each Tagアクションによってループされます。最初に、タグのテスト・ステップで、実際に複数のページがあることをチェックします。ある場合は、ページへのリンクを含むタグをループし、クリック・アクションを使用して各ページをロードし、ページの処理を続行します。最初のページに、最初のページへのリンクが含まれる場合は、最初のページが2回処理されないために、この最初のリンクをスキップするようにFor Each Tagアクションを構成する必要があります。

各ページが次のページにリンクする場合

次に、すべてのページをリンクするもう1つの一般的な方法を示します。



各ページが次のページにリンク

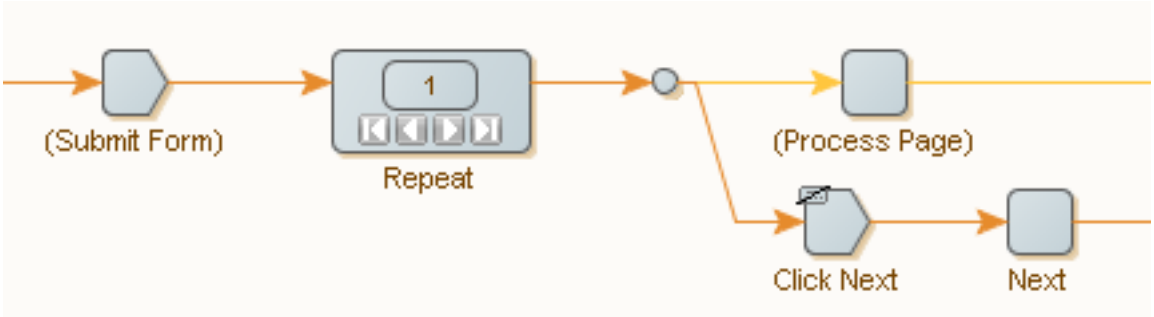
ここでは、リンク(通常は"Next"などの名前のフォーム・ボタン)を使用して、各ページが次のページにリンクしています。

このようなページをループするには、繰返しアクションを使用します。繰返しアクションは、「次」という名前の別のアクションによって提供されるページをループします。この方法については、次のビデオを視聴または読んでください。

Repeat-Nextループの使用方法をデモするビデオ

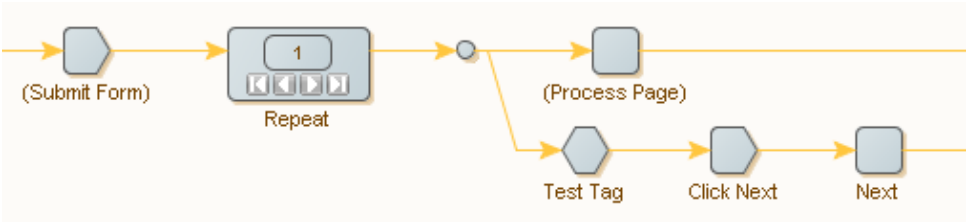
原則として、繰返しアクションには、最初のページが入力として渡される必要があります。次に、ページをループして、繰返しごとにページを出力します。繰返しごとに現在のページを処理でき、「次」アクションを使用して繰返しアクションに次のページを渡す必要があります。繰返しアクションに新しいページが渡されないと、次の繰返しは発生せず、ループは終了します。

次のロボットの抜粋を例として示します。



前述のように、"(Submit Form)"という名前のステップで表されている検索リクエストから、結果ページをループします。フォーム発行ステップで最初の結果ページが出力され、このページを繰返しアクションに渡します。繰返しアクションからの最初の分岐で、現在のページを処理します。2番目の分岐で、リンクをクリックして次のページをロードします。「次」アクションでページを繰返しアクションに戻し、次の繰返しでページを出力します。最後のページに達すると、クリック・アクションでエラーが生成されます。したがって、クリック・ステップはループを終了するように構成されます。クリック・ステップでは、エラー処理タブで処理プロパティをループの中断に設定してこれを行います。この詳細は、「エラーを処理する方法」を参照してください。

次のロボットの抜粋は、最後のページを処理する別の方法を示しています。



最後のページに達したことを検出するには、2番目の分岐でタグのテスト・アクションを使用します。タグのテスト・アクションでは、テキスト「次」が含まれる<a>タグを検索するなどして、ページに次のページへのリンクが含まれていることをチェックします。ページにそのようなリンクが含まれている場合は、このページをロードして「次」アクションに渡します。最後のページに達すると、タグのテスト・アクションは2番目の分岐への実行を停止し、新しいページが繰返しアクションに渡されないため、ループが終了します。

次のページへのリンクの検出は、慎重に構成する必要があることに注意してください。最初のページ、後続のページおよび最後のページの間でページのレイアウトが少し変わるため、次のページへのリンクではなく、同じページ上の前のページへのリンクを検出する誤りがよくあります。また、最後のページを確実に検出できない誤りもよくあります。適切に機能するには、ステップのタグ・ファインダを慎重に構成する必要があります(「タグ・ファインダを使用する方法」を参照)。

Design Studioでロボットを使用するとき、Design Studioは繰返しアクションの繰返しを前後に必ずしも正しく進めるわけではありません。Design Studioが正しく機能しているかどうか不明な場合は、「リフレッシュ」をクリックして更新します。

HTMLからコンテンツを抽出する方法

Design Studioには、HTMLページのタグからコンテンツを抽出するための6つのステップ・アクションがあります。

- 抽出アクションを使用して、タグ(オプションでHTMLタグを含む)からテキスト・コンテンツを抽出します。

- URLの抽出アクションを使用して、URLが含まれるタグ属性からURLを抽出し、そのURLを絶対URLにします。

- タグ属性の抽出アクションを使用して、タグ属性の値を抽出します。

- ターゲットの抽出アクションを使用して、イメージやPDFファイルなどのバイナリ・データを抽出しますが、すべての種類のバイナリ・データを処理します。

- フォーム・パラメータの抽出アクションを使用して、検出されたタグ内のフォームURLからフォーム・パラメータを抽出し、その値を変数に格納します。

- 選択したオプションの抽出アクションを使用して、選択したオプションを<select>タグから抽出し、それを変数に格納します。

抽出されたコンテンツは再書式化(または標準化)が必要な場合が多く、抽出およびタグ属性の抽出アクションでは、データ・コンバータのリストを構成することによって再書式化ができます。

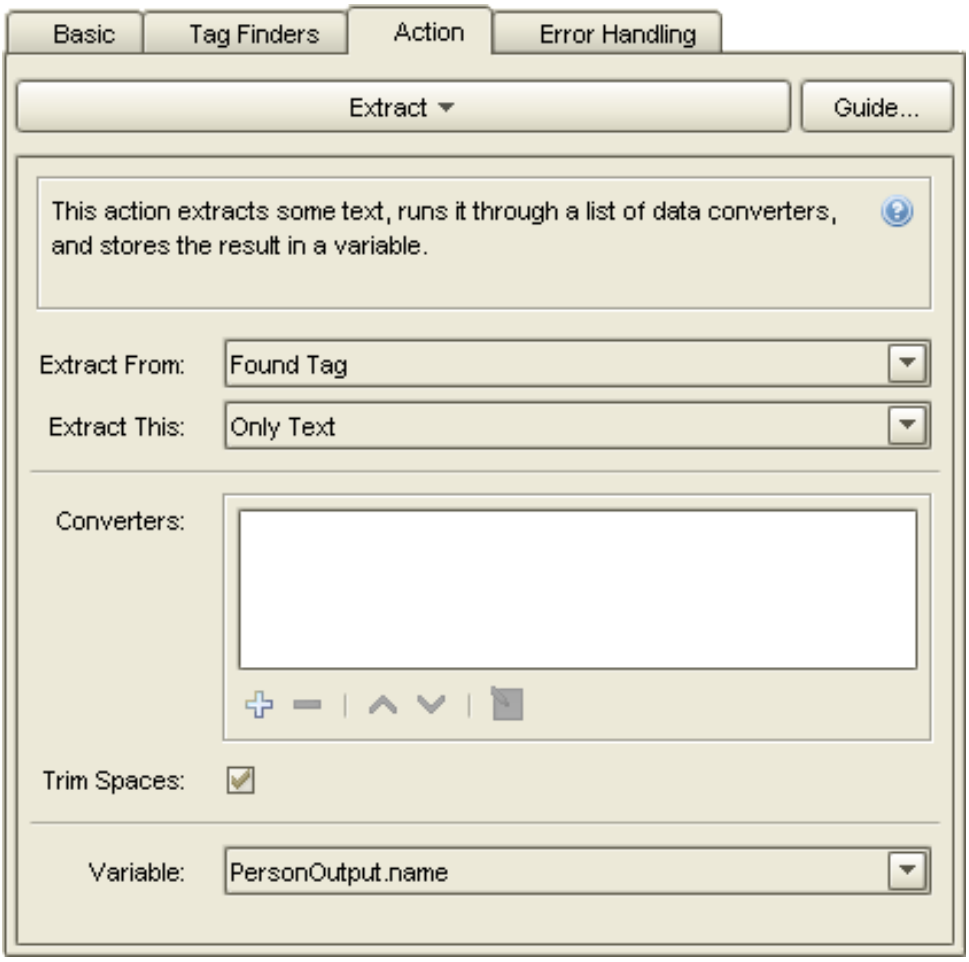
また、様々なバイナリ・データ形式(PDF、Flashなど)からデータを抽出するための2つのアクションもあります。これらは、データを抽出して、構造化されたフォームでデータが含まれるHTMLページを作成し、ロボットがそのデータにアクセスできる点で、前述のものとは異なります。ただし、これらのアクションは実際のデータ抽出前の初期ステップで使用され、作成されたHTMLをループして、そこからテキストを抽出できます。

- PDFからのテキストの抽出アクションを使用して、選択した属性にバイナリ・データとして含まれるテキストをPDFドキュメントから抽出します。

- Flashから抽出アクションを使用して、検出されたタグ内のFlashオブジェクトからデータを抽出します。

テキストの抽出

抽出アクションは、テキストを抽出するために使用します。



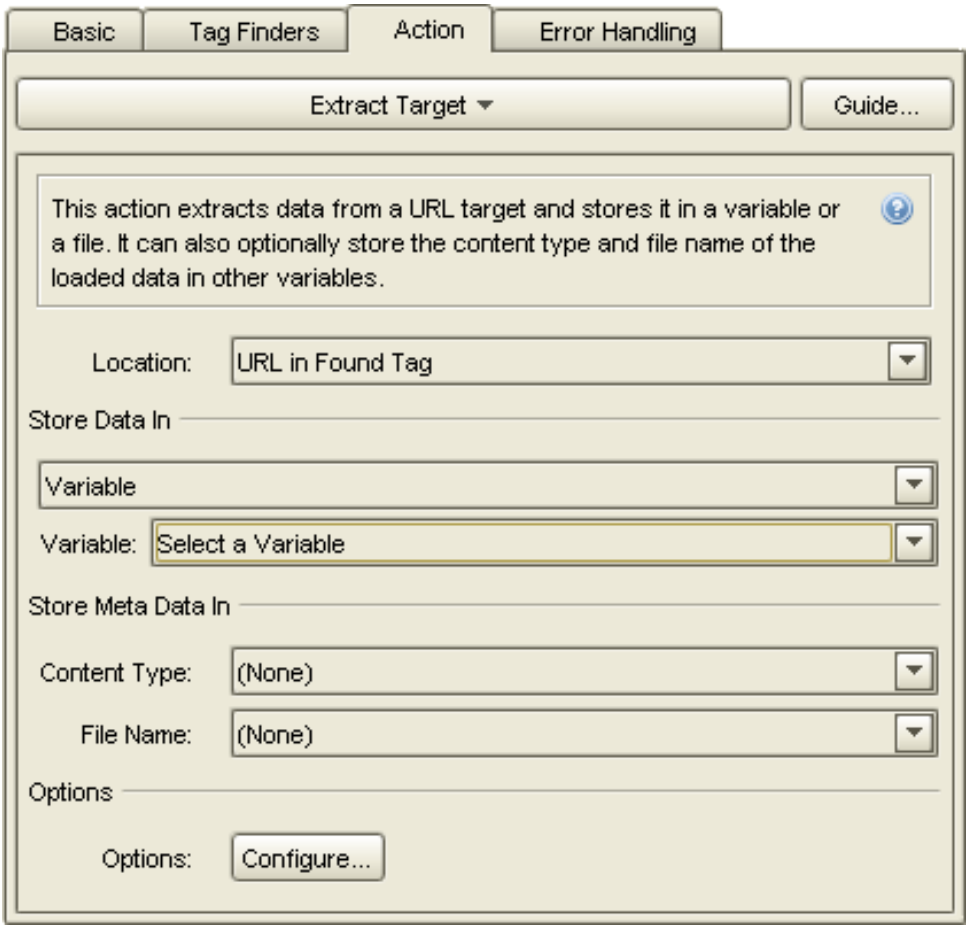
抽出アクション

製品名や価格のように、短いテキストの場合はテキストのみとして抽出します。これは、単にタグ間のテキストを抽出します。

セクションや見出しなどの長いテキストをプレーン・テキストとして抽出し、そのテキストをブラウザでの表示に近い方法で表示する場合は、テキストを構造化されたテキストとして抽出する必要があります。見出しを囲む大カッコなど特別なマークアップが必要な場合は、構造化されたテキストを使用すると基本的にこれがサポートされます。構造化されたテキストではマークアップ要件を実現できない場合は、高度に構造化されたテキストを使用すると、HTMLタグから独自のマークアップへのマッピングを設定できます。

バイナリ・データの抽出

バイナリ・データはターゲットの抽出アクションを使用して抽出し、URLからデータをロードして、変数に格納するか、またはファイルに直接格納します。



ターゲットの抽出アクション

通常、バイナリ変数はロードされたデータを格納するために使用されます。バイナリ変数の選択可能なタイプは、「バイナリ」、「イメージ」、PDFおよび「セッション」です。「イメージ」、PDFおよび「セッション」タイプはデータをプレビューできることを除いて、これらはすべて同等です。

ページ・ビューでのポップアップ・メニューの使用

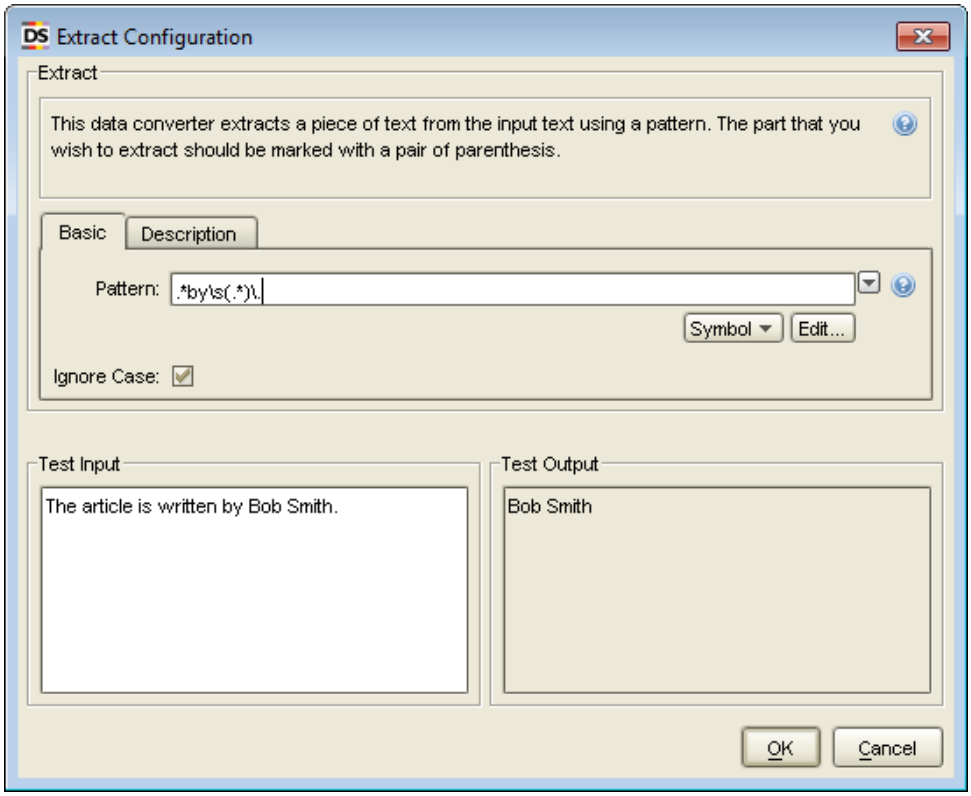
ページ・ビューでは、抽出ステップ・アクションを選択して構成するためのショートカットとして、ポップアップ・メニューを使用できます。抽出元のテキストやタグ、またはロード元のリンクを右クリックして、表示されるポップアップ・メニューの抽出メニューから該当するオプションを選択します。

共通タスクの実行

この項では、理解する必要がある共通の抽出タスクについて説明します。

テキストの一部のみを抽出

タグ内のテキストの一部のみを抽出する場合は、タグのテキストに対してパターンを使用できます。たとえば、"The article is written by Bob Smith."というテキストから"Bob Smith"という名前を抽出するとします。これを行うには、抽出データ・コンバータ(抽出ステップ・アクションと混同しないでください)を使用して、次のように構成します。



抽出データ・コンバータの使用

原則として、サブパターンと照合して抽出するテキストをカッコで囲み、テキスト全体が一致するようにパターン・プロパティを構成します。この場合、使用されているパターンは".*by¥s(.*)¥."で、"by"とピリオドの間のテキストがサブパターンと照合されることを意味します。パターンの詳細は、「パターン」を参照してください。

コンテンツの変換

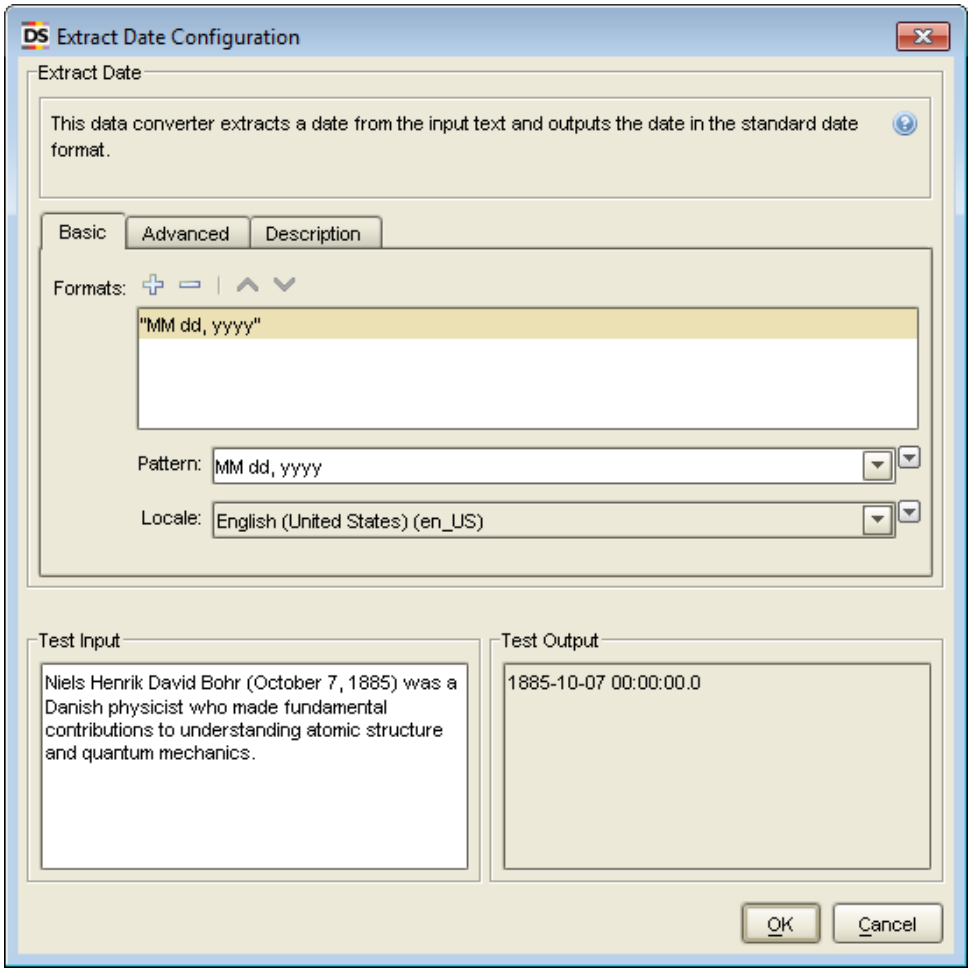
あるテキストを別のテキストに置換する場合など、コンテンツを標準化するときは変換が使用されます。たとえば、国コードを自然言語表記に標準化する場合です("US"を"United States"に標準化するなど)。プレーン・テキストの変換の場合は、リストを使用した変換データ・コンバータを使用します。パターンまたは式に基づく変換の場合は、If Thenデータ・コンバータを使用します。

数字の抽出および書式設定

コンテンツから数字を抽出する場合は、常に数字の抽出データ・コンバータを使用する必要があります。数字をさらに書式設定する場合は、数字の書式設定データ・コンバータを使用します。多くの場合、コンテンツから数字を抽出する場合は数字の抽出データ・コンバータを追加します。数字の抽出データ・コンバータによって抽出(出力)されたテキストを再書式設定するには、数字の書式設定データ・コンバータを追加してさらに書式設定します。

テキストからの日付の抽出

日付の抽出は、数字の抽出と同じ方法で実行します。日付の抽出データ・コンバータを使用して、任意のテキストから日付を抽出します。日付の抽出では、パターンを使用して日付を抽出します。パターンは必ずしもテキスト全体と一致する必要はなく、日付のみ一致する必要があります。抽出された日付は標準的な日付書式に変換され、日付の書式設定データ・コンバータを使用してなんらかの書式設定ができます。



日付の抽出データ・コンバータの使用

次の2つのビデオでは、テキストから日付を抽出する方法を説明します。

単純な日付抽出に関するビデオ・チュートリアル

複雑な日付抽出に関するビデオ・チュートリアル

検出されたタグ内のタグのサブセットのみを抽出

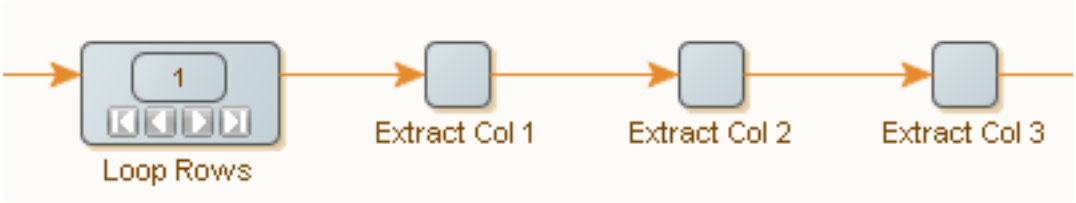
単一のタグではなく、ある範囲のタグから抽出する場合があります。抽出アクションでは、範囲内の最初のタグと最後のタグを指定することによってタグの範囲を指定します。

たとえば、記事の本文テキストを抽出する場合、本文テキストはそれぞれ独自のタグ内にある複数のセクションで構成され、記事のタイトルや著者に関する情報は他のタグ内に含まれます。記事のタイトルや著者を除いて本文テキストのみを抽出するには、抽出アクションを使用してテキストを抽出し、本文部分のタグの範囲のみが抽出されるようにアクションを構成します。

HTML表からコンテンツを抽出する方法

対象のコンテンツが表内に存在する場合があります。ただし、HTML表は、コンテンツと構造の両方で非常に不規則であることが多いです。次に説明するように、Design Studioはこのような不規則性に対処するように設計されています。(この項で説明するテクニックは、表のコンテンツや構造の不規則性への対処に限定されません。これらを使用して、あらゆる種類のタグの不規則性に対処できます。)

対象のコンテンツが含まれる表がコンテンツと構造の両方で完全に規則的である場合、コンテンツは「HTMLからコンテンツを抽出する方法」で説明したように抽出できます。通常、ロボットは次のようになります。



最初のステップにはFor Each Tagアクションが含まれ、<table>タグの<tbody>タグ内にある<tr>タグをループします。その後に、それぞれ表の行のセル(列単位)からコンテンツを抽出するいくつかのステップが続きます。

コンテンツの不規則性

同じ表の列内でも、セルのコンテンツの書式が異なる場合があります。たとえば、空の場合、"Bob"(名)が含まれる場合、"Bob Smith"(名と姓)が含まれる場合などがあります。

コンテンツの不規則性に対処する最も簡単な方法の1つは、値の抽出を行うステップでIf Thenデータ・コンバータを使用することです。書式の各バリエーションに一致するように、"If"および"Else If"プロパティを構成します。次に、対応する"Then"プロパティで、一致するサブパターンを抽出します。

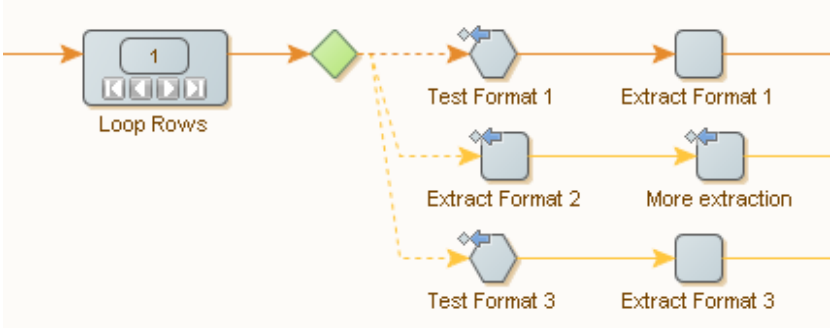
ただし、"Bob Smith"の場合は2つの値(名と姓)が含まれるため、名を抽出するステップと姓を抽出するステップの2つを作成する必要があります。これは、抽出アクションで抽出できるのは1つの値のみであるためです。2つのステップにはそれぞれ抽出アクションとともにIf Thenデータ・コンバータが含まれるため、最初のステップで名を抽出し(ある場合)、2番目のステップで姓を抽出します(ある場合)。

構造の不規則性

表の行に含まれるセルの数が異なる場合があります。このような不規則性に対処する一般的な方法は、表の各行の書式をテストすることです。たとえば、特定数のセルが含まれる行のみ、または特定のテキストが含まれる行のみを対象にするとします。

これを行うには、表の行をループするFor Each Tagステップの後に、試行ステップを追加します。試行ステップの分岐ごとに1つの書式を処理します。このため、各分岐は、たとえば、(パターンとして記述された)書式と一致するすべての行を受け入れるタグのテスト・アクションなど、次の代替の試行エラー処理を行う条件ステップで開始します。条件ステップの後には、条件アクションによって書式が受け入れられた場合に実行される1つ以上の抽出ステップが続きます。書式の抽出を試行して、失敗した場合は次を試行するように、条件ステップと抽出を組み合せることも可能です。

次のロボットは、両方のアプローチを使用しています。2番目の書式の抽出は、2ステップのプロセスであることに注意してください。次の代替の試行エラー処理が両方のステップに設定されているため、2つのステップのいずれかが失敗すると3番目の分岐が試行されます。これは、2番目の分岐の複雑な条件を表しています。



このロボットが実行されると、いずれかの分岐が成功するまで各分岐が順に実行されます。これは、前の分岐が失敗したことは後の分岐で認識されるため、後の分岐の条件で前の分岐の条件を繰り返す必要はないことを意味します。

条件ステップの後の分岐は、必ずしも分離している必要はないことに注意してください。複数の分岐で抽出ステップを共有している場合は、異なるステップの後で分岐をマージできます。

ロボットでローカル・ファイルを使用する方法

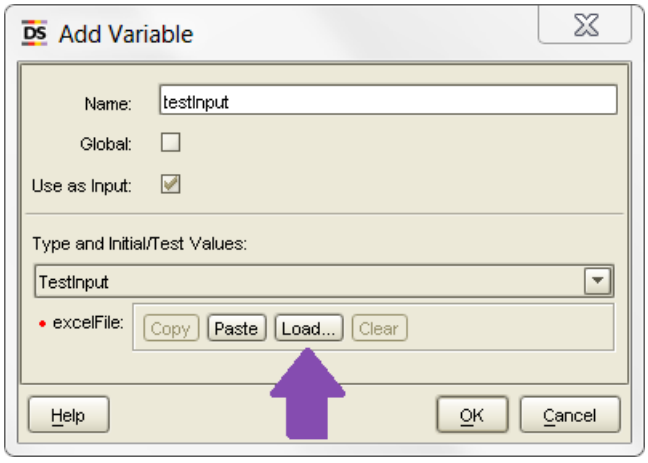
ロボットは、HTML、Excel、CSVおよび規則的なテキスト・ファイルを含む多くのタイプのファイルをロードできます。このため、ロボットは様々なソースからデータを抽出できます。

- ロボットは、HTML、XML、ExcelおよびJSONの各ファイル・タイプをネイティブにロードできます。
- それ以外のプレーン・テキスト、CSVおよびPDFの各ファイル・タイプはロードできますが、ロボットで処理する前にHTMLに変換されます。

一般的に、各種ファイル・タイプをロードする手順は2種類あります。ファイルがインターネット上に存在する場合、ページのロード・アクションを使用してファイルのURLを指定するか、またはクリック・アクションを使用してファイルへのリンクをクリックしてファイルをロードします。これによって、ページ・ビューにファイルが自動的にロードされます。これに対して、ファイルがユーザーのシステム上に存在する場合は、次の方法でファイルをロードすると、ロボットがManagement ConsoleにアップロードされてスケジュールされたりKappletに追加されるときにそのファイルも使用可能になるため有益です。

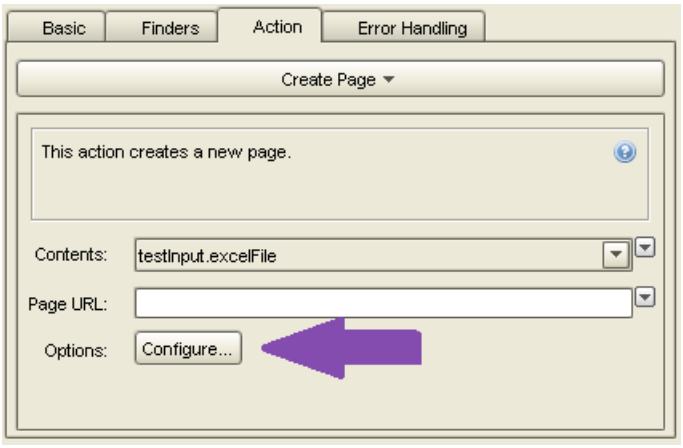
PDF以外のすべてのファイル・タイプは、次の方法でロードされます。

最初に、バイナリ・タイプの変数をロボットに追加します。(PDFやHTMLなど他の変数タイプも使用できますが、バイナリ・タイプほど柔軟性がなく、ユーザー入力ができません。)



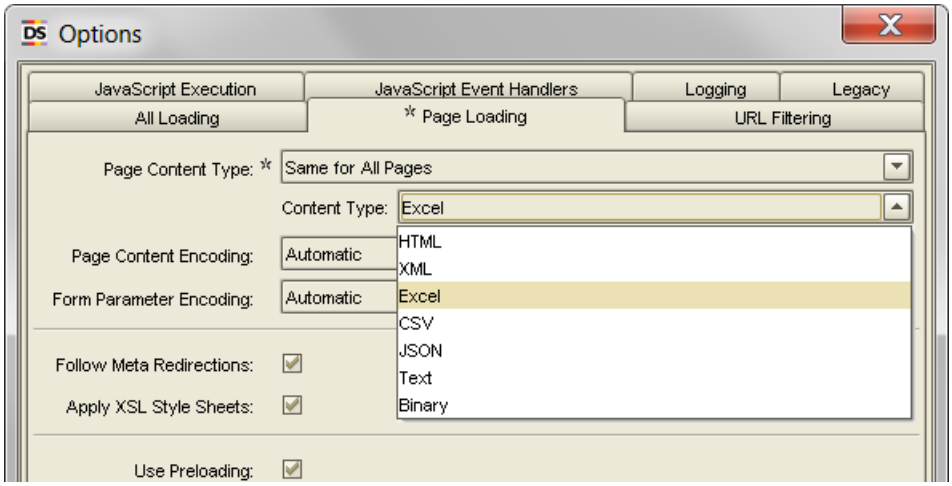
バイナリ・タイプの属性を持つ変数がロボットに追加されます。これは入力変数として定義され、ユーザーはKappletおよびスケジュールに他のファイルを入力できます。Design Studioでのテストのために、「ロード...」ボタンを使用してExcelファイルが属性にロードされます。

変数を追加するときは、変数が入力変数である必要があるかどうかを確認してください。入力として使用を選択するか選択しないかは、ロボットがManagement ConsoleのKappletでスケジュールまたは使用される場合のみ重要です。入力変数はユーザーが定義できるため、ロボットが実行されるたびにファイルを交換できます。これに対して、ロボットが実行されるたびにファイルが同一である必要がある場合、入力変数を使用する必要はありません。最後に、「ロード...」ボタンを使用してテスト・ファイルをロードします(入力として使用が選択されていない場合は、これが最後のファイルになります)。



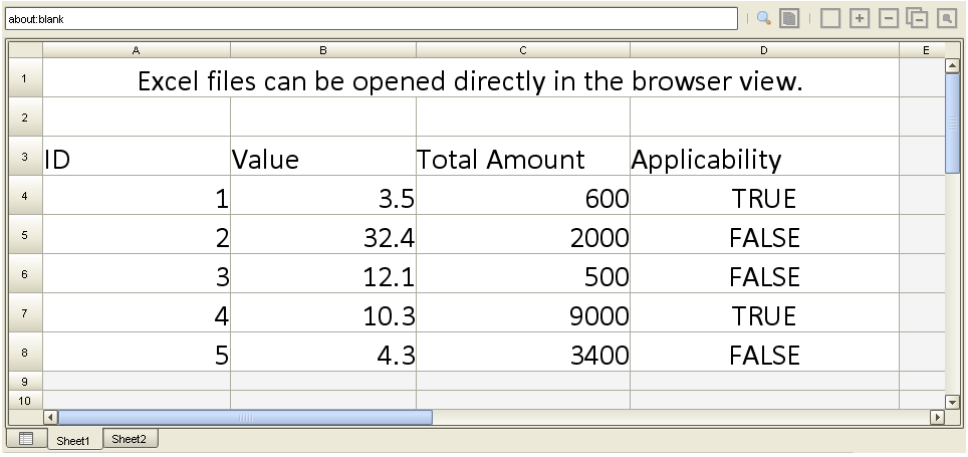
バイナリ・タイプの変数にファイルがロードされた後に、ページの作成を使用してページ・ビューにファイルをロードします。ステップは、機能する前に、正しいタイプのファイルをロードするように構成する必要があります。

バイナリ変数からファイル・コンテンツをロードするには、ページの作成ステップが使用されます。「コンテンツ」フィールドでは、値セレクタが変数に設定され、バイナリ・タイプの変数が選択されています。その後に、ステップは正しいタイプのコンテンツをロードするように構成されます。



ページの作成ステップは、バイナリ変数から正しいタイプのコンテンツをロードするように構成されます。この場合は、Excelが選択されています。

ステップ構成のページのロード・タブで、ページ・コンテンツ・タイプがすべてのページで同一に設定され、「コンテンツ・タイプ」はロードするファイルと同じタイプが選択されています。

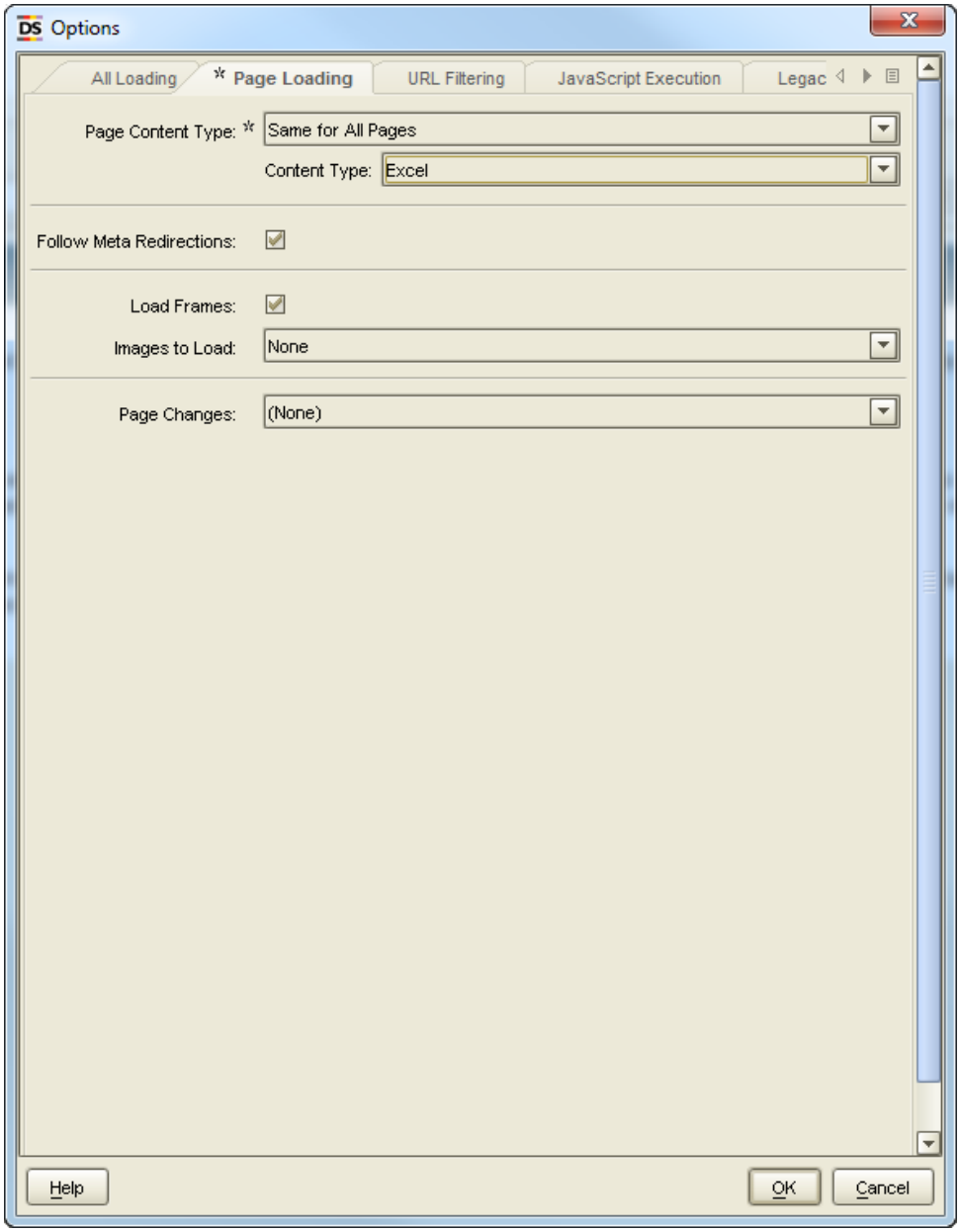


ページの作成ステップによって、ページ・ビューにファイルがロードされます。

前述のステップに従い、ページの作成ステップによってページ・ビューにファイルがロードされます。PDFファイルをロードする場合は、参照の「PDFからの抽出」を参照してください。スケジュール用の入力ファイルを使用する場合は、「単一ロボットの作成」を参照してください。

変数からExcelページをロードする方法

Design Studioでページをロードする最も一般的な方法はページのロード・ステップを使用することですが、ロボットはExcelドキュメントをバイナリ属性の入力として受け取る場合があり、このドキュメントをロボットにロードすると、ロボットでそのデータをループして抽出できます。これは可能ですが、少しの構成が必要であるため、次にその方法について説明します。最初に、ページの作成ステップをロボットに挿入し、バイナリ属性からコンテンツを取得するようにロボットを構成しますが、これが機能するには、属性内のデータが実際にはExcelドキュメントであり、どの種類のバイナリ・データでもないことをステップに対して指示する必要があります。これを行うには、ページの作成ステップで「オプション」構成を開きます。次に示すように、「オプション」ダイアログのページのロード・タブで、ページ・コンテンツ・タイプをすべてのページで同一に設定し、「コンテンツ・タイプ」は「Excel」を選択します。



「コンテンツ・タイプ」を「Excel」に設定

Excelからコンテンツを抽出する方法

Design Studioには、スプレッドシートからコンテンツを抽出するための3つのステップがあります。

- セルの抽出ステップを使用して、検出された範囲からテキスト・コンテンツを抽出します。
- シート名の抽出ステップを使用して、検出された範囲のシートのシート名を抽出します。
- HTMLとして抽出ステップを使用して、スプレッドシートの検出された範囲を、範囲内のセルが表に含まれているHTMLページとして変数に抽出します。

セルの抽出ステップおよびHTMLとして抽出ステップでは、セルから抽出する内容を指定できます。これは、抽出内容オプションの値によって管理されます。ここでの選択は、スプレッドシート・ビューのビュー・モードと同じです。選択可能なオプションは次のとおりです。

書式設定された値: 抽出される値はExcelに表示される内容で、たとえば、書式設定された日付や数字の値が抽出される場合、その数字はセルの実際値より小数値が少なくなります。

プレーン値: セルの値が書式設定されていない場合、抽出される値はExcelに表示される実際値です(たとえば、端数処理していない小数値)。

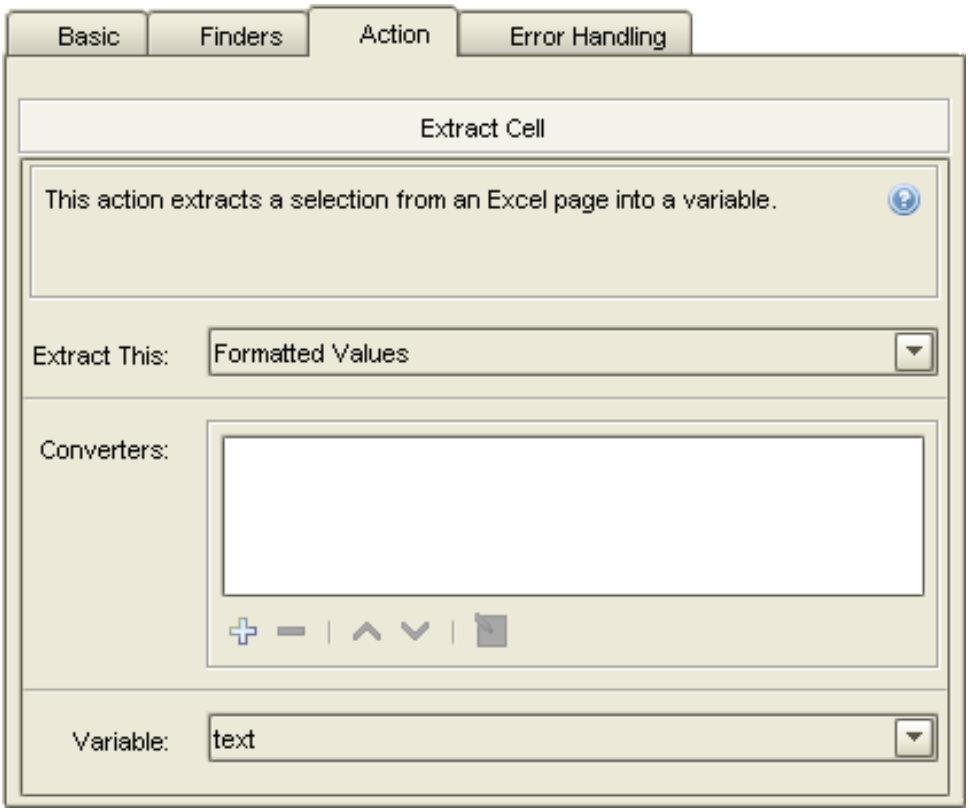
式: セルに式が含まれる場合はその式が抽出され、そうでない場合はプレーン値オプションと同じ値が抽出されます。

スプレッドシート・ビューを右クリックしてステップを作成すると、抽出内容の値は、選択したビュー・モードの値に設定されます。つまり、ビュー・モードを「式」に設定し、ページ・ビューを右クリックしてポップアップ・メニューから抽出>テキストの抽出(テキスト変数へ)を選択すると、セルの抽出アクション・ステップの抽出内容オプションは「式」に設定されます。

抽出されたコンテンツは再書式化(または標準化)が必要な場合が多く、セルの抽出アクションでは、データ・コンバータのリストを構成することによって再書式化ができます。

セルからの値の抽出

セルの抽出ステップを使用して、1つのセルまたはセルの範囲のコンテンツを変数に抽出します。



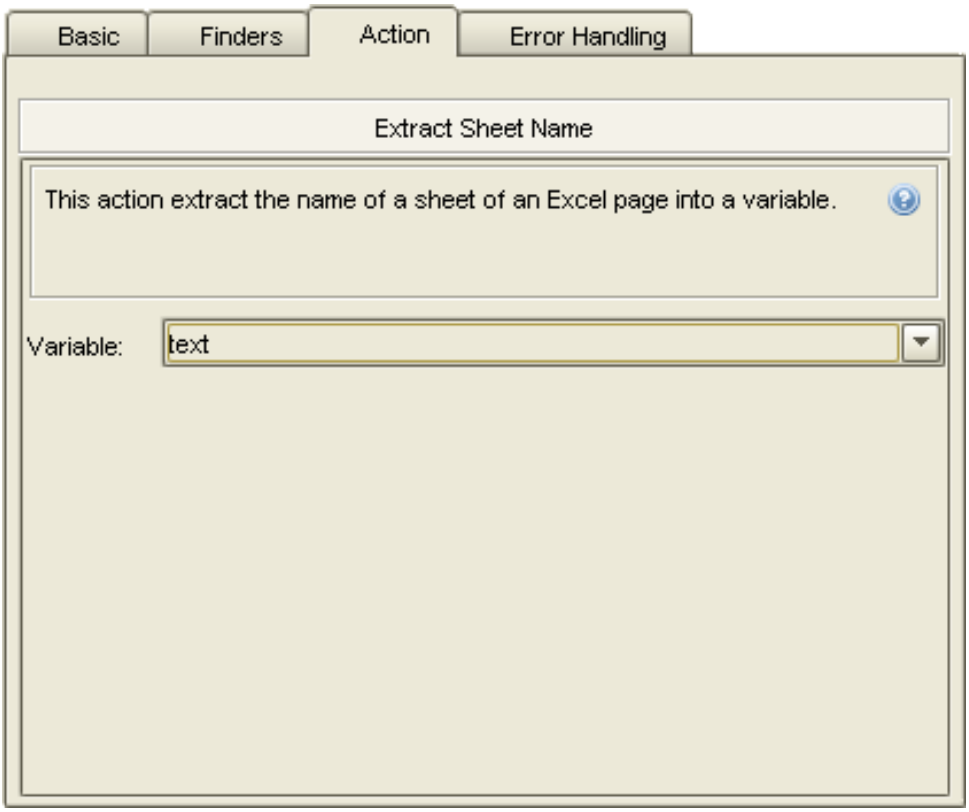
セルの抽出ステップの「アクション」タブ

検出された範囲が1つのセルの場合は、このセルの値が抽出されます。検出された範囲に複数のセルが含まれている場合、それらのセルの値は、セルがタブで区切られ行が改行で区切られたテキストとして抽出されます。いずれの場合も、抽出されて変数に格納される値は、抽出される値にコンバータを適用して作成されます。

セルから抽出された値は、基本的に、抽出内容オプションの値に対応する、Excel内のセルのコンテンツです。空白のセルの場合、値は空の文字列になり、セルがマージされたセルの一部の場合(たとえば、セルのマージによってExcelで作成されたC4:D6)、セルがマージされたセルの左上部のセルC4である場合を除き、抽出された値は空白になります。

シート名の抽出

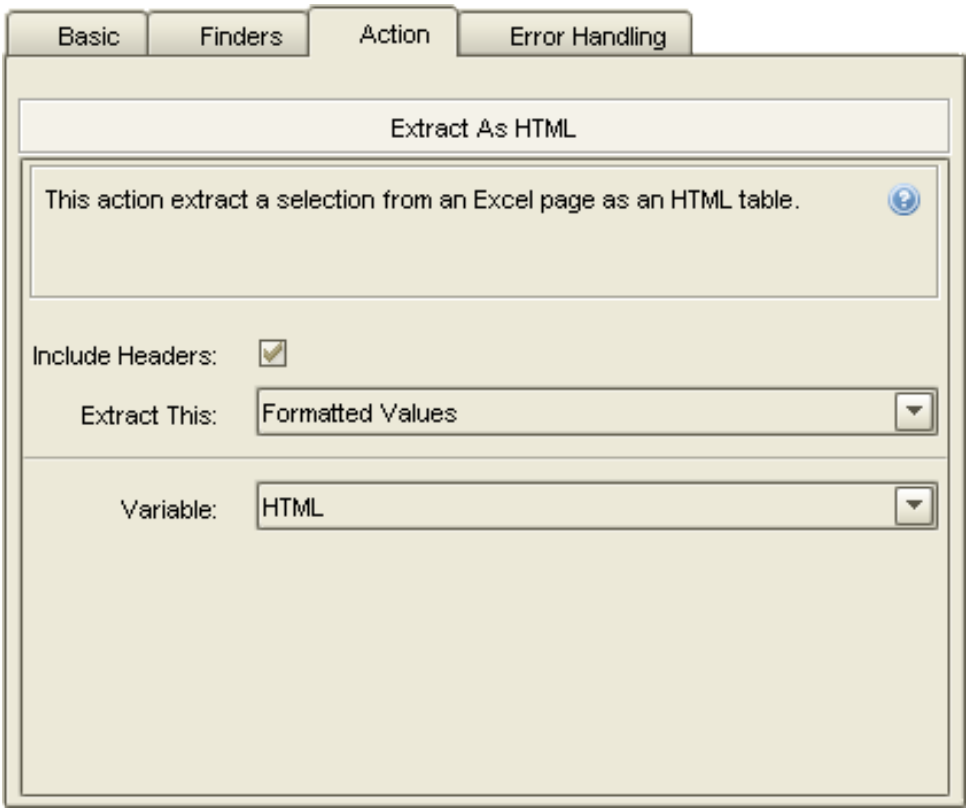
シート名の抽出ステップを使用して、シートの名前を抽出します。これは、すべてのシートをループするときに値のテスト・ステップと組み合せて指定の名前のシートをスキップする場合、または複合タイプの変数の属性にシート名を抽出して戻り値に含める場合に役立ちます。



シート名の抽出ステップの「アクション」タブ

HTMLとして抽出

HTMLとして抽出ステップを使用して、スプレッドシート・ドキュメントの一部をHTMLソース・コードとして抽出し、構造化されたテキスト変数(HTMLタイプなど)に格納します。抽出されたコードには、headerタグ内で抽出された範囲(例: Sheet1:A1:H17)が含まれるため、シートの名前がコードに含まれます。検出された範囲のセルは、生成されたコードで表に配置されます。このステップはスプレッドシートの一部のHTMLバージョンを取得する(たとえば、これがロボットに返されてブラウザに表示される)ことが主な目的ですが、このステップをロボットで使用すると、ページの作成ステップを使用して抽出されたコードを含むHTMLページを作成することもできます。HTMLとして抽出ステップを使用してスプレッドシートをHTMLページに変換し、そのコンテンツにアクセスすることは、ロボットのパフォーマンスが低下する可能性があるためお薦めできません。



HTMLとして抽出ステップの「アクション」タブ

ヘッダーを含むオプションは、抽出される表に列ヘッダーと行ヘッダーを含めるかどうかを指定します。次のイメージは、ヘッダーを含むオプションがtrueに設定された場合の抽出されたHTMLを示しています。検出された範囲は'My Sheet'!B1:E10で、B1からE10のセルが表に抽出されていますが、表にはBからEの列のヘッダー、および1から10の行ヘッダーも含まれています。

'My Sheet'!B1:E10

	B	C	D	E
1	Signup Date	First Name	Last Name	Gender (isMale)
2	8/23/13	Evan	Cortez	FALSE
3	11/6/11	Hiram	Fowler	TRUE
4	10/5/13	Cassady	Sweet	TRUE
5	11/9/12	Harrison	Howe	TRUE
6	9/16/12	Dustin	Gentry	FALSE
7	6/18/13	Chase	Alford	FALSE
8	7/8/11	Hadassah	Farley	FALSE
9	2/12/14	Quon	Albert	FALSE
10	6/15/12	Oliver	Blankenship	TRUE

抽出されたHTMLの例

Excelのセル・タイプをテストする方法

Excelページのセルのコンテンツをテストする場合は、最初にセルのコンテンツを抽出し、値のテスト・ステップを使用して実際のテストを実行します。これは、他のページ・タイプ(HTMLなど)で行う場合と基本的に違いはありませんが、セルのセル・タイプを判別する場合は簡単ではなく、たとえば、セルが空白か、または空のテキストが含まれているかを判別する方法がないため、セルのコン

テンツを抽出した後にそのコンテンツに対してテストを実行する場合もあります。Design Studioには、このようなテストを実行するためのセル・タイプのテスト・ステップが含まれています。

テストできるセル・タイプは6種類あり、これらはISTEXT、ISNUMBERなどの関数を使用してExcelでテストできるタイプに直接対応しています。

空白: これに対応するExcel関数は ISBLANKです .

テキスト: これに対応するExcel関数は ISTEXTです .

数値: これに対応するExcel関数は ISNUMBERです。このタイプには日付も含まれます(Excelでは数値として表示されるため)。

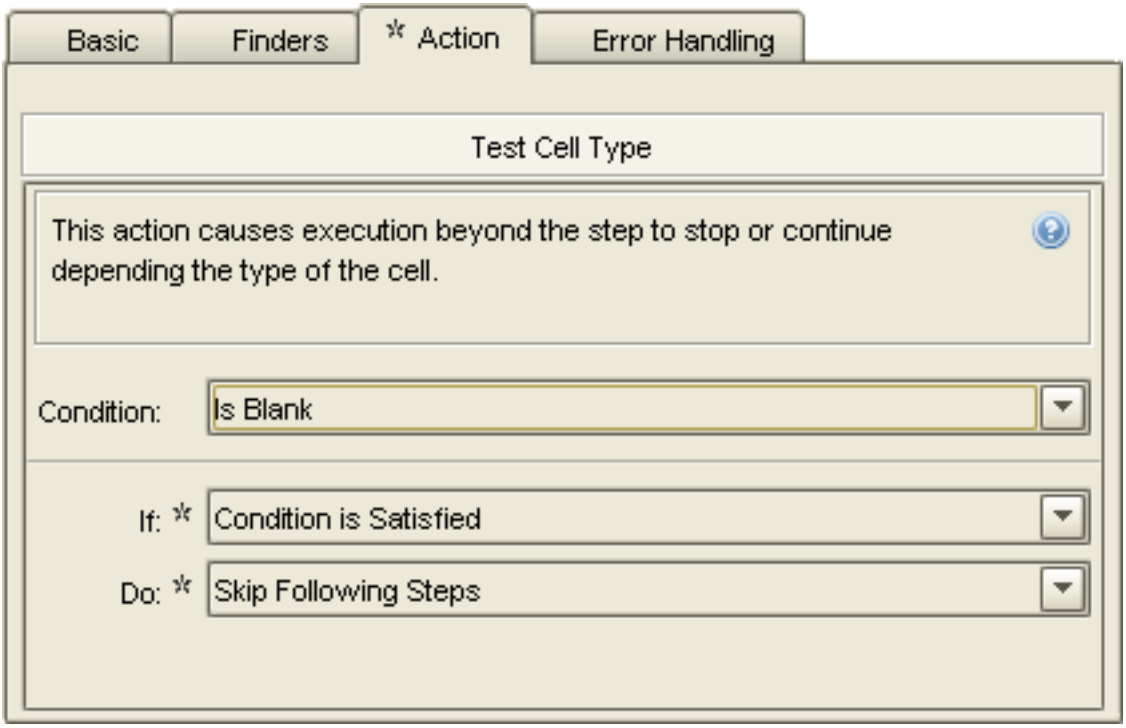
論理: これに対応するExcel関数はISLOGICALです。これに対応するのはのDesign Studioのブール・タイプです。

エラー: これに対応するExcel関数はISERRORです。

式: これに対応するExcel関数はISFORMULAです。

セル・タイプのテストは、他のテスト・ステップと同様に機能します。これは、検出された範囲内のセル・タイプが指定のタイプと一致するかをテストし、その結果に基づいて、分岐に従って続行するか、後続のステップをスキップするかを判断します。このステップの詳細は、セル・タイプのテスト・ステップに関する参照ドキュメントを参照してください。

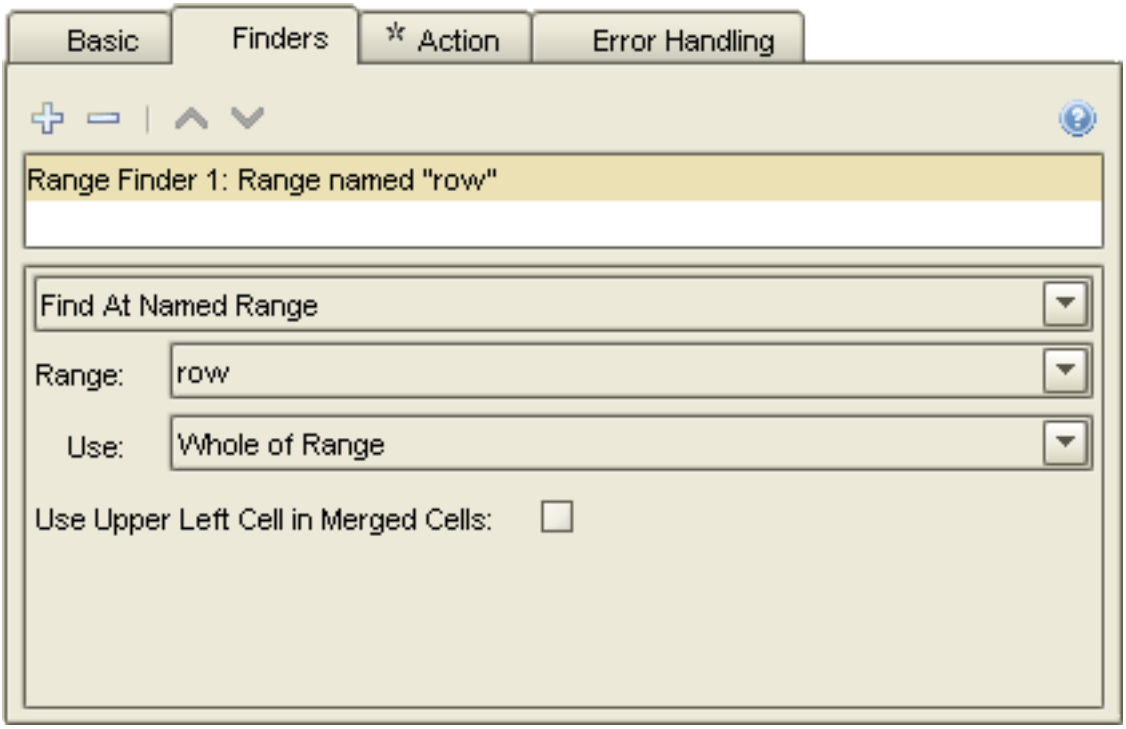
セル・タイプのテスト・ステップの重要なプロパティは、多くのステップのタイプを同時にテストできることです。その一例として、行全体が空白かどうかをテストする方法を考えてみます。これは、構造的に同一の複数の表が空白行で区切られて含まれているドキュメントをループする場合に役立ちます。次の図に、このステップを構成する方法を示します。この例では、検出された範囲内のセルがすべて空白の場合、ステップの後の分岐がスキップされます。



空白のセルをテストする例

次の図に、行全体を検索するように範囲ファインダを構成する方法を示します。この場合、範囲は"row"という名前で、この名前は、セル・タイプのテスト・ステップの前に実行されて行をルー

プするExcelでのループ・ステップで設定されています。使用プロパティでWhole of Rangeを選択して、結果が行全体になるように指定しています。



行全体を選択する範囲ファインダ

Excelでループする方法

ExcelでのループはHTMLでのループと多くの点で似ていますが、HTMLでループする方法は非常に多様であるのに対して、Excelは構造が単純であるためExcelでのループはかなり単純です。基本的に、ドキュメント内のすべてのシートをループしたり、検出された範囲の行、列またはセルをループすることによってシートのセルをループできます。Excelでループするには、Excelでのループ・ステップを使用します。このステップには、最初の索引、増分など、HTMLでループするステップと共通する多くのオプションがあります(詳細は、参照ドキュメントを参照)。

次のビデオでは、1つのシート内の表をループし、各行のセルを複合タイプの変数に抽出して、値を返すステップを使用してセルを返す方法を説明します。

Excelでのループに関するビデオ・チュートリアル

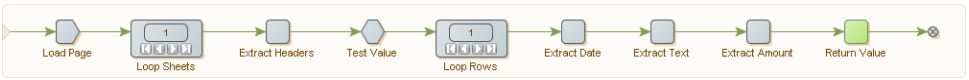
シートおよび行のループ

通常、Excelでのループは、前のビデオで説明したより少し複雑です。Excelドキュメントでのループで多く発生するシナリオは、ドキュメントに、同じタイプのデータが格納された表を含む複数のシートが含まれる場合です(年間の月ごとにアカウント情報が含まれるシートなど)。この場合、ロボットでは、最初にシートをループし、次に各シートの行をループします。また、ドキュメントに、他のシートとは異なるタイプのデータが格納されたシート(シートが空白の場合など)が含まれる状況も処理できる必要があります。次のイメージは、このようなロボットの構造を示しています。

Excelでのシートおよび行のループ

このロボットの最初のステップは、URLからExcelドキュメントをロードするページのロード・ステップで、次にドキュメントのすべてのシートをループするExcelでのループ・ステップが続きます。前のビデオで説明した方法と同じ方法で、ロボットは最初のループ・ステップの繰返しごとに、シート

の各行をループするExcelでのループ・ステップを実行します。行をループするステップのエラー処理プロパティ処理は次の繰返しに設定されており、これは、ステップの範囲ファインダが表のサイズ範囲の一致に失敗した場合に、次の繰返しに進むことを意味します。この簡略化されたエラー処理では、シートが空白であるような単純な状況を処理しますが、データのタイプが完全に異なる表がシートに含まれる状況は処理できません。一般的に、シートの一部を抽出するステップを挿入し、その後にその構造をテストするステップが続く必要があります。その一例として、列ヘッダーを抽出し、それらが指定の構造かどうかをテストします。次のイメージは、このように拡張されたロボットを示しています。「Extract Headers」という名前のセルの抽出ステップでシートの最初の行を変数に抽出し、値のテスト・ステップにはこの値をテストする条件が指定されています。ロボットは、値が一致した場合は次のステップ(行のループ・ステップ)を実行し、一致しなかった場合は後続のステップをスキップします(値のテスト・ステップの処理プロパティが後続ステップのスキップに設定されているため)。



テストを含むExcelでのシートおよび行のループ

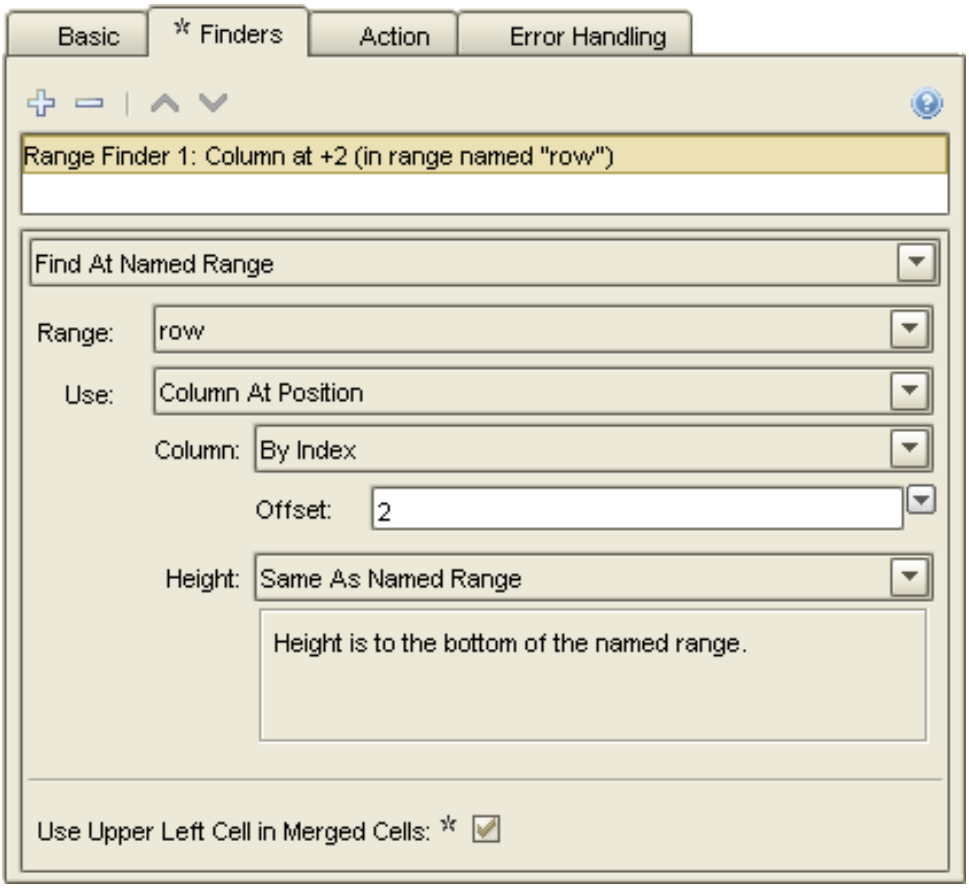
ループおよびマージされたセル

次に説明するシナリオは、ループするExcelページにマージされたセルが含まれる場合です。Excelのマージされたセルとは、1つのセルにマージされて1つとして表示される、複数の隣接するセルです。マージされたセルのコンテンツは左上のセルに格納され、その他のセルはすべて空白になります。このようなマージされたセルを含む表をループすると、問題が発生する場合があります。これを示す簡単な例について説明します。次のシートには生徒のテスト結果が表示されており、一部の生徒はテストを受けておらず、マージされたセルを使用して2つのテストが表示されている場合があります。

	A	B	C	D	E
1	Test Results				
2		**Test1**	**Test2**	**Test3**	
3	**Alice**	12	7	9	
4	**Bob**	Missed		4	
5	**Jane**	11	8	7	
6	**John**	12	Missed		
7	**Zach**	10	Missed	7	
8					

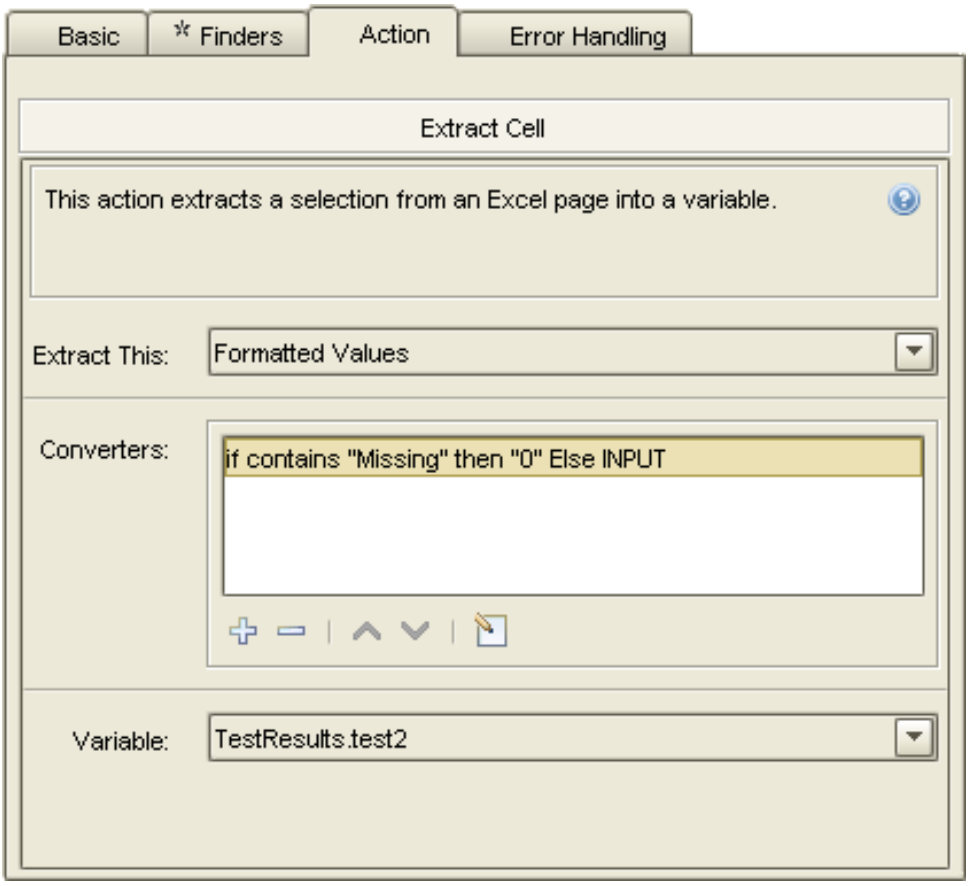
マージされたセルを含むシートのループ

シートの行をループして生徒のテスト結果を抽出すると、テキスト"Missed"は数字ではないため、生徒がテストを受けていない場合は結果を正しく抽出できません。これを解決するには、これに対するテストを挿入し、失敗した結果には値0を格納します。この方法は、値"Missed"が含まれるセルC4には有効ですが、コンテンツが空白値のC4では失敗します。セルが空白かどうかのテストをさらに追加するのではなく、マージされたセル内でセルを検索し、マージされたセルの左上のセルを返す単純なオプションがすべての範囲ファインダにあります。次のイメージに、このように範囲ファインダを構成する方法を示します。



範囲ファインダ

これで、セルの抽出では、テキスト"Missed"をテストし、存在する場合は0を使用し、そうでない場合は抽出した値を使用します。これは、次に示すようにIf Thenデータ・コンバータを使用して実行できます。



セルの抽出アクション

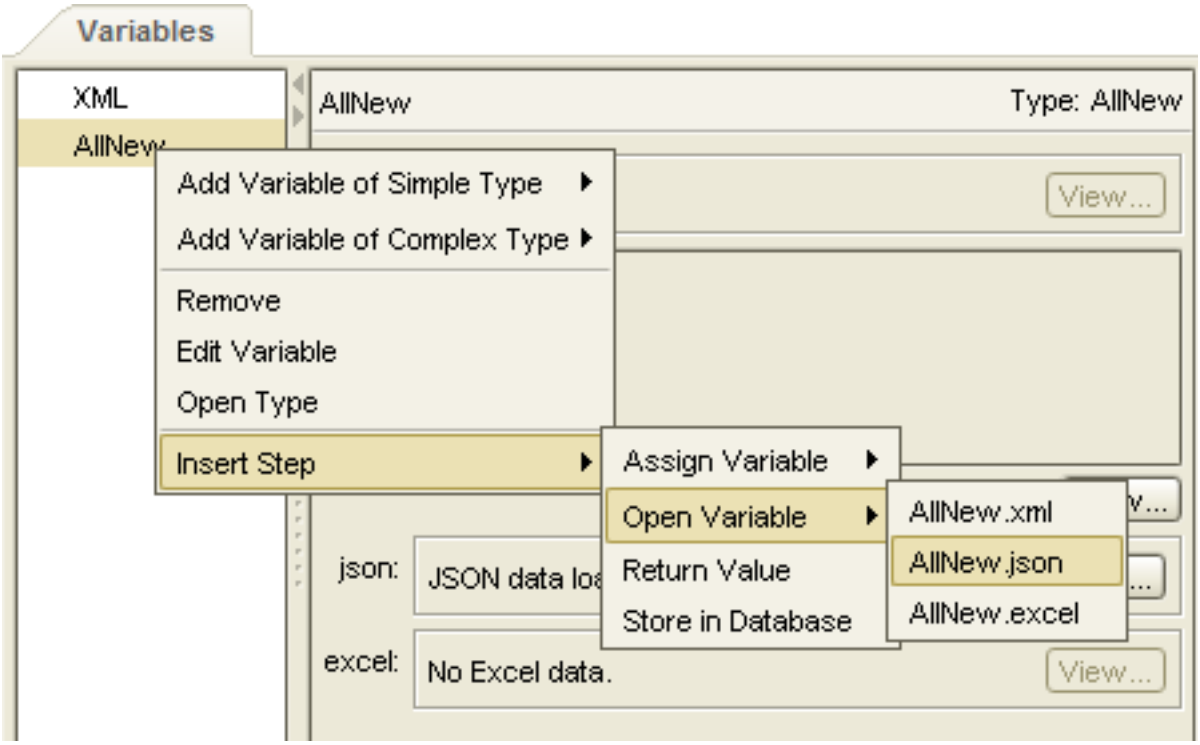
ウィンドウ・ビューで変数を使用する方法

ウィンドウ・ビューには、ロードされたHTMLページやJSONドキュメントなど、現在のロボット状態の部分が表示されますが、特定の単純タイプ(XML、JSONおよびExcel)の変数または属性もウィンドウ・ビューのタブに表示される場合があります。ウィンドウ・ビューに変数が表示されているときは、他のドキュメントがウィンドウ・ビューにロードされたときと同じ方法で変数を操作でき、たとえば、ビューから変数を抽出してテストし、ループして、ほとんどの場合に変数を変更できます。

たとえば、入力としてXMLを受け取り、出力としてもXMLを返すWebサービスを呼び出すことができます。次に、XML変数(変数が表示されるウィンドウのコンテンツで機能するステップ・アクションを使用して変更)を使用する入力XMLを作成し、必要なフォームに設定して、Webサービス・ステップ・アクションに入力として渡します。このWebサービス・ステップ・アクションによって別のXML変数にレスポンスを格納し、変数をループしてデータを抽出できます。

変数を開く

ウィンドウで変数(または属性)を使用するには、最初に新しいウィンドウで変数を開く必要があります。これを行うには、変数を開くステップ・アクションを使用します。これを実行する最も簡単な方法は、変数ビューで変数を右クリックして、メニュー・オプションのステップの挿入>変数を開くを選択することです。次の図に、複合タイプのJSON変数の属性を開く方法を示します。



変数ビューからの変数を開くステップ・アクションの挿入

メニューから変数を開くオプションを選択すると、ロボットの現在のステップの前に変数を開くステップが挿入され、このステップが実行されると、新しいウィンドウが開いて変数のコンテンツが表示されます。この点では、変数を開くステップ・アクションはページのロード・ステップ・アクションと動作がよく似ています。変数がすでに開いている場合、新しいウィンドウは開きませんが、変数が表示されているウィンドウが新しい現在のウィンドウになります。この点では、変数を開くステップ・アクションはページのロード・ステップ・アクションと動作が異なり、現在のウィンドウの設定ステップ・アクションによく似ています。このステップ・アクションは変数を開くと呼ばれますが、変数の属性に対しても機能します(ウィンドウに開くことができるタイプの場合)。

変数(または属性)が開くと、URLからロードされたドキュメント(XMLドキュメントなど)と同様に、その変数に対して作業できます。ビューを右クリックすると、変数を操作するステップ・アクションを挿入でき、ロード(開く)元が変数かURLかに関係なく、挿入ステップは現在のウィンドウに対して機能します。唯一の違いは、URLからロードされたドキュメントは変更できないことです(変更不可とみなされます)。ドキュメントを変更する場合は、最初にドキュメントを変数に抽出してから変更する必要があります。

変数の変更

現在は、XML変数およびJSON変数のみ変更できます。これら2タイプの変数には、変数の変更で使用できる広範囲の専用ステップ・アクションがあります(たとえば、属性の設定は既存の属性の値を設定したり新しい属性をXMLタグに追加し、プロパティ名の設定ステップ・アクションはJSONオブジェクトのプロパティの名前を変更します)。また、変数の割当など、変数を直接操作するステップ・アクションを使用して変数を変更でき、この場合は、ビューにもこれらの変更が反映されます。

現在のウィンドウからXML変数を変更するステップ・アクションは、次のとおりです。

- タグの設定
- コンテンツの設定
- テキストの設定
- タグ名の設定

· 属性の設定

· コンテンツの挿入

· タグの削除

· コンテンツの削除

· 属性の削除

現在のウィンドウからJSON変数を変更するステップ・アクションは、次のとおりです。

· JSONの設定

· プロパティ名の設定

· JSONの挿入

· JSONの削除

XMLまたはJSONタイプの属性が現在のウィンドウに表示されているとき、これらのステップ・アクションを挿入するためのメニュー・オプションはウィンドウのコンテキスト・メニュー(右クリック・メニュー)から使用でき、指定タイプに関連するメニュー・オプションのみが表示されます。一部が無効になっている場合がありますが、これは、ビュー内の現在の選択とは関連がないことを示します(たとえば、選択したタグに属性がない場合、属性の削除は無効です)。

JSONを使用する方法

JSON (JavaScript Object Notation)は軽量なデータ交換形式で、**{ "x" : 5 , "y" : 7 }**などのJavaScriptリテラル表記に似ています。JSONはテキスト形式ですが、ロボットでは、XMLの表現方法に似た構造的な方法で表現および表示されます。JSONは、HTML、XMLおよびExcelと同様に、独自のページ・タイプを持つ独自のデータ形式として扱われます。以前のバージョンのDesign StudioのようにXMLには変換されません。つまり、ページ・タイプのテスト・ステップ・アクションは、現在のウィンドウのコンテンツがJSONかどうかをテストします。ウィンドウ・ビューにロードされるJSONは、URL、または単純タイプのJSONの変数/属性からロードされたJSONであることが可能です。ウィンドウ・ビュー(JSON変数をこのビューで開く場合)および変数ビュー(JSON変数の値をここから表示する場合)の両方にJSONを表示する専用ビューがあり、JSONでのみ機能する専用ステップ・アクションがあります。

次に、JSONテキストの例を示します。

```
{ "answer" : 42, "people" : [ {
"firstName" : "Arthur",  "lastName" : "Dent" },
{ "firstName" : "Ford", "lastName" : "Prefect" } ]
}
```

次の項では、JSONの構造を説明し、JSONテキストの各部を説明するために使用する用語を定義します。

JSON 用語

JSONテキストは、オブジェクト(例: **{ "a" : 5 }**)または配列(例: **[1, 2, 3]**)のいずれかです。JSON値はJSONテキストまたはJSON単純タイプのいずれかで、JSON単純タイプはJSONリテラル、数値または文字列のいずれかです。JSONリテラルは、**false**、**null**または**true**のいずれかです。リテラルの**false**と**true**はブールと呼ばれます。数値は、整数または浮動小数点数のいずれかです。数

値の精度やサイズに制限はありませんが、別の表現に変換されると同時にその表現の制限が適用されます。たとえば、整数が整数変数に抽出された場合、値の範囲は-2^{63}から$2^{63}-1$で、この範囲外の場合は抽出ステップでエラーが発生します。JSON文字列は前後を二重引用符(")で囲み、Unicode文字を含めることができます(″、¥または制御文字を除き、これらの文字は¥″、¥¥、¥rのように¥を使用してエスケープできます)。JSON形式については、「RFC 4627」を参照してください。

次の図に、JSONの構文の概要を示します。

表10.9 JSONの構文

```
JSONテキスト = JSONオブジェクト | JSON配列 JSONオブジェクト
= {} | { プロパティ } JSON配列
= [] | [ 項目 ] プロパティ
= プロパティ(単数), プロパティ(複数) プロパティ
= 文字列 : JSON値 項目
= JSON値, 項目 JSON値
= JSONテキスト | 文字列 | 数値 |
false | null | true 文字列
= "" | " 文字
" 文字 = 文字(単数) 文字(複数) 文字
= ″、¥または制御文字を除くUnicode文字 |
\" | \\ | \/ | \b | \f | \n | \r | \t | \u
4桁の16進数 数値
= CまたはJava数値とよく似ている数値(詳細は「RFC 4627を参照)。
```

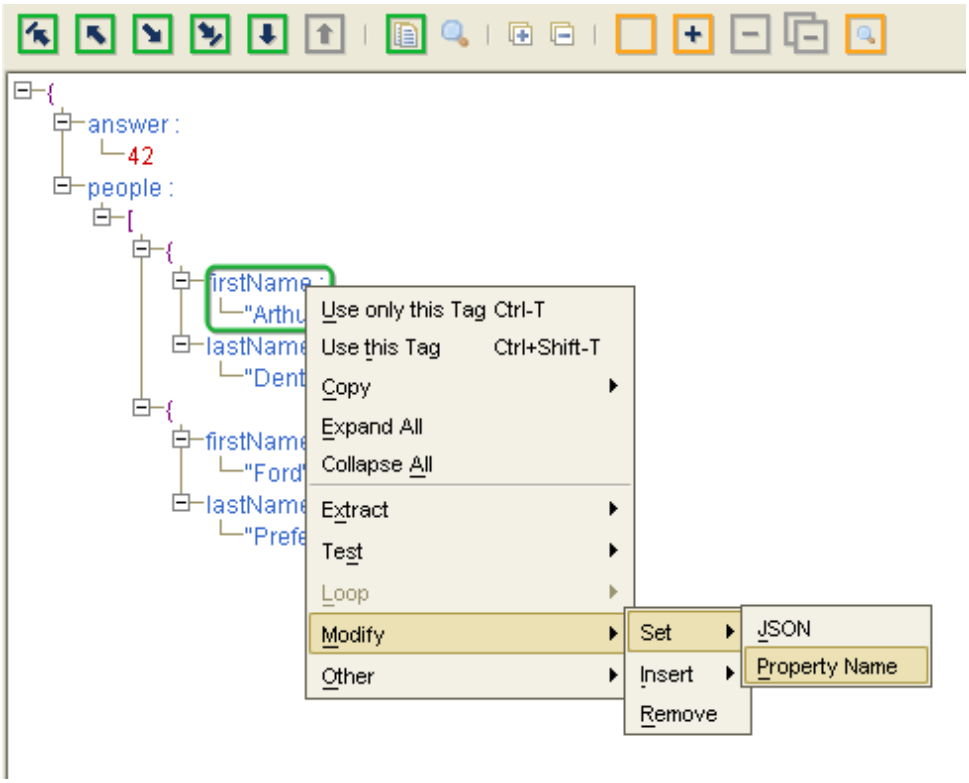
JSON MIMEタイプ

JSONテキストのMIMEメディア・タイプは**application/json**であるため、厳密に言うと、このMIMEタイプに対して必ずしもすべてのJSON値が有効ではありません。JSONを受け入れたり返すサービスの実装者はJSON値を自由に受け入れたり返すことができるため、JSONの処理に対してより自由なアプローチを取ることによって、JSON変数にJSON値を含めることができ、JSONビューにこのような値を表示できます。

データがURLからロードされ、MIMEタイプが**application/json**の場合、ロードされたJSONはJSONページ・ビューに表示されます。これ以外の場合は、ページのロード・ステップでページ・コンテンツ・タイプを「JSON」に設定して、実際にデータがJSONを表すように指定できます。この方法は、MIMEタイプが使用可能なソースからデータがロードされていない場合にも使用できます(ページの作成ステップ・アクションなど)。

JSONで機能するステップ・アクション

JSONのみで機能するステップ・アクションがいくつかあるため、現在のウィンドウに表示されるデータはJSONである必要があります(JSONがXMLに変換されていた古いレガシー形式ではJSONでない必要があります)。これらのステップ・アクションは、ステップ・ビューの「アクション」タブにあるステップ・アクション・セレクタで「JSON」というステップ・アクション・カテゴリに含まれています。ただし、現在のウィンドウにJSONが含まれる場合、これらを選択する最も簡単な方法はウィンドウ・ビューでコンテキスト・メニュー(右クリック・メニュー)を使用することです。次のイメージは、この例を示しています。



ウィンドウ・ビューでのJSONテキスト上のコンテキスト・メニュー

JSON値から抽出するステップ・アクションは次の2つです。

- JSONの抽出。このステップ・アクションは常にJSON値を抽出します(たとえば、ビュー内でプロパティが選択されている場合、抽出されるのはプロパティの値です)。これは、HTMLやXMLから抽出する抽出ステップと多くの点で似ていますが、マークアップとテキストに区別がないなど、データ形式が単純であるため、より単純です。

- プロパティ名の抽出。このステップ・アクションはプロパティの名前を抽出します。

JSONテキストをループするステップ・アクションは次の2つです。

- For Each Property。このステップ・アクションは、JSONオブジェクトの各プロパティをループします。

- For Each Item。このステップ・アクションは、JSON配列の各JSON値をループします。

これらのステップ・アクションは両方とも、繰返しごとに対象のJSON値の一部を指定JSON（指定タグと同様)として設定します。繰返し中に変数の値が変更されると、繰り返された値が変更されて繰返しが失敗する(繰り返されたリストから項目が削除されるなど)ため、変数に対する繰返しはグローバルにできません。

JSONを変更するステップ・アクションは次の4つです(JSONが変数内にある場合のみ)。

- JSONの設定。これは、JSON値の選択された部分を新しいJSON値に置換します。

- プロパティ名の設定。これは、選択されたプロパティのプロパティ名を新しい名前に設定します。

- JSONの挿入。これは、JSONオブジェクトに新しいプロパティを挿入するか、またはJSON配列に新しい項目(JSON値)を挿入します。新しいプロパティまたは項目を挿入する位置(先頭、末尾など)についてはいくつかのオプションがあります。ステップ・アクションの完全なリストは、参照ドキュメントを参照してください。

- JSONの削除。これは、JSON値の選択した部分を削除します(たとえば、JSONオブジェクトからプロパティを削除したり、JSON配列から項目を削除します)。

最後に、JSONで機能するステップ・アクションがさらに2つあります。

- JSONのテスト。このステップ・アクションは、JSON値のタイプ(オブジェクト、配列、文字列など)をテストします。

- 指定JSONの設定。このステップ・アクションは、他のタイプの日付に対応するステップ・アクション(指定タグの設定、指定範囲の設定など)に似ています。これは、JSON値の一部への指定参照を定義するため、後続のステップでJSON値の他の部分を検索するときに参照として使用できます。ビュー内では青いボックスとして表示されます。

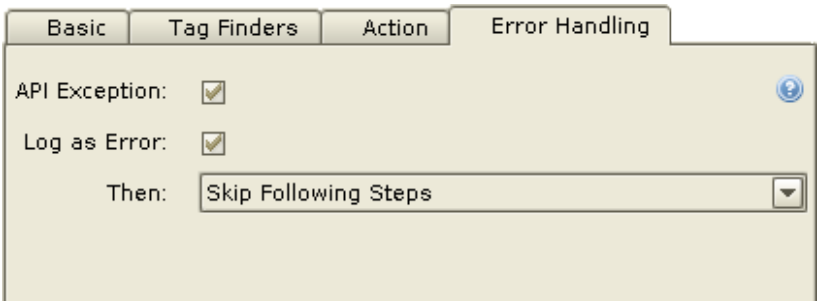
エラーを処理する方法

ロボットのステップでは、実行時にエラーが生成される場合があります。これは、たとえば、タグ・ファインダが対象のタグを検出できない場合、またはステップ・アクションでエラーが生成された場合に発生します。テスト・ステップは、テストが失敗するとエラーが発生したのと同様に機能するように構成することもできます。ロボットのデフォルト動作では、エラーを即時にレポートして記録し、失敗したステップより後のステップの実行を省略します。ただし、ロボットのステップのエラー処理プロパティを構成すると、この動作を変更できます。たとえば、エラーが生成されたステップをロボットでスキップしたり、別の分岐を試行するようにできます。この項では、この方法について説明します。

始める前に、ここで説明するエラー処理動作は、ロボットのランタイム実行(つまり、RoboServerまたはデバッグ・モードでの実行)に適用され、Design Studioの設計モードでの実行には適用されないことに注意してください。通常、設計モードでは、エラーは即時にレポートされ、後続のステップの実行は中止されます。ステップがエラー発生時に「無視して続行」するように構成されている場合は例外で、Design Studioはエラーを無視して、ランタイム実行時と同様に次のステップを実行します。

API例外およびロギング

ステップ・ビューのエラー処理タブには、エラーをレポートするかどうか、およびその方法を選択する2つのチェック・ボックスがあります。



API例外チェック・ボックスは、ロボットの呼出し側にエラーをレポートするかどうかを指定します。これは、ロボットがクライアントによってAPIの1つを介して実行され、RoboServerで実行されるときに最も有用です。この場合、エラーはAPIを介して呼出し側に**RobotErrorResponse**として送信され、これによって呼出し側では、少なくともデフォルトの**RQLHandler**を使用するときに例外が発生します。ロボットが他の方法で実行される場合の詳細は、参照ドキュメントを参照してください。

エラーとして記録チェック・ボックスは、エラーを記録するかどうかを指定します。ロボットがDesign Studioで実行されているか、またはRoboServerで実行されているかに応じて、異なる方法でロギングが発生します(参照ドキュメントを参照)。

チェック・ボックスは選択または選択を解除できますが、明示的にデフォルト以外の値に設定されたことを示すアスタリスク(*)でマーク付けされる場合があることに注意してください。これは「デフォルトからの変更の表示」で説明しており、アスタリスクを削除してデフォルト値に戻す方法も説

明しています。デフォルト値が適用される(つまり、アスタリスクが表示されていない)場合、エラーの処理方法に応じてデフォルトが異なります。

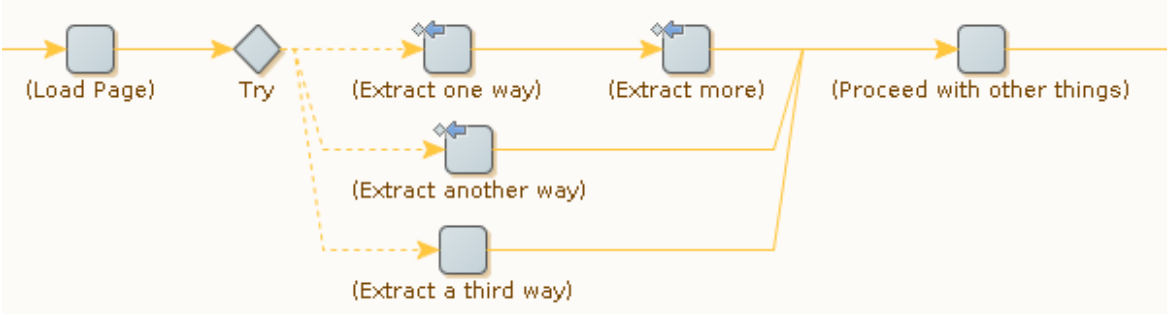
エラー後に実行を続行する方法

処理プロパティは、エラーが発生した後にロボット実行を続行する方法と時期を定義します。選択可能なオプションと例については、後続の項で説明します。詳細は、参照ドキュメントを参照してください。

複数の代替の試行

この例では、目標を達成するために、エラー処理を使用して複数の代替方法を選択する方法を示します。これは「条件およびエラー処理」の説明に関連しており、このトピックにはチュートリアル・ビデオも含まれています。

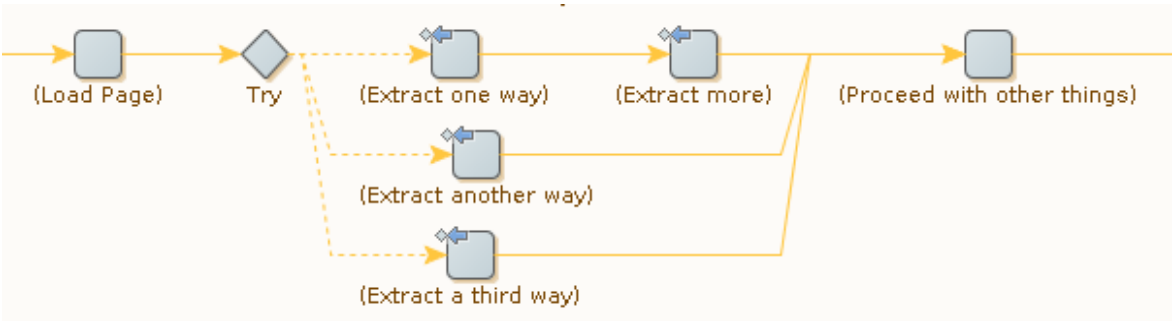
Webページの各部分は構造やレイアウトが異なりますが、常に3つのケースのいずれかに該当するとします。各ケースに、抽出対象の情報があります。このため、一度に1つのケースについて抽出を試行します。失敗すると、次のケースが試行され、3つ目のケースまで試行されて成功するとします。



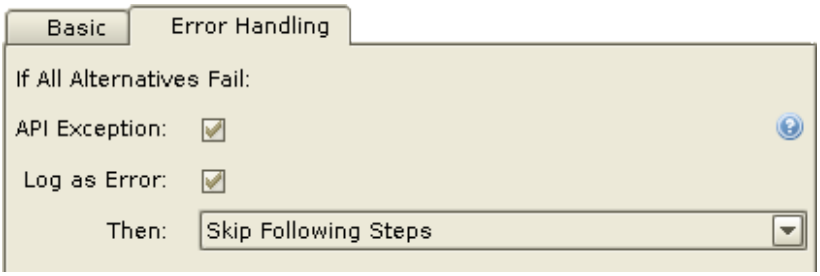
抽出ステップの(次の代替の試行)アイコンは、抽出が失敗した場合は試行ステップからの次の分岐が実行される(試行ステップからの分岐が成功すると、次の分岐は実行されない)ことを意味します。したがって、抽出ステップでは、Webページからの抽出を行い、それが実行できない場合は次のアプローチを試行するという、2つの処理を同時に行っています。最初の分岐にある2つのステップのいずれかが失敗すると、2番目の分岐が実行されることに注意してください。これは、分岐の「成功条件」をステップの組合せによって表す方法の例です。

このアプローチは、抽出の3番目の方法が万全である場合に最適です(たとえば、Webページからデータを実際に抽出するのではなく、デフォルト値の固定セットを適用するなど)。3番目の分岐が最初の2つの分岐と同様にWebページにアクセスする場合、3番目の分岐が成功すると仮定するのは賢明ではありません。Webサイトが大幅に変更されたために、ロボットが次回に実行されたときに3つの方法がいずれも成功しなくても、ロボットは妥当な方法でこれに対処できる必要があります。

これに対処する最も簡単な方法は、呼出し側に問題をレポートして記録し、抽出および後続のすべての処理を中止することです。これを行うには、最初の2つの分岐と同様に、3番目の分岐で処理が失敗したかどうかを試行ステップに通知するようにします。



これで、3つの分岐がすべて失敗すると、試行ステップは問題をレポートして記録します。これは、試行ステップでのエラー処理のデフォルト構成です。



(試行ステップの場合、後続ステップのスキップは、レポートおよび記録以外の追加アクションは実行しないことを意味します。)

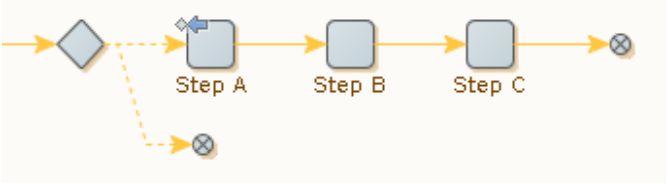
あるいは、問題を前の試行ステップに伝播して処理するように試行ステップを設定できます。この方法については、「Try-Catch」の項で説明します。

一般的なケース用のショートカット

試行ステップおよび次の代替の試行エラー処理(前の項を参照)は非常に柔軟なツールです。これらを適切に使用すると、多くの方法でエラーを処理できます。この項では、2つの単純で一般的なケースを説明します。実際、これらのケースは一般的で、特別なエラー処理オプションでもサポートされています(これについても説明します)。

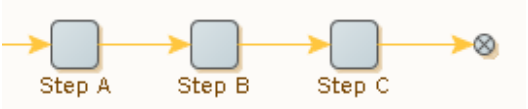
後続ステップのスキップ

多くの場合、ロボットはWebページ上のオプション要素を処理できる必要があります。つまり、要素が存在する場合は処理する必要があります(たとえば、データを抽出する必要があります)が、存在しない場合はこれらの要素の処理を単にスキップできます。存在しない状態はエラーではなく、予期される状況です。ロボットでは、これを次のように表すことができます。Step Aは要素が存在するかどうかをテストし(対象の抽出を試行するなど)、Step BとCは、Step Aの成功に基づいてさらに処理を進めます。

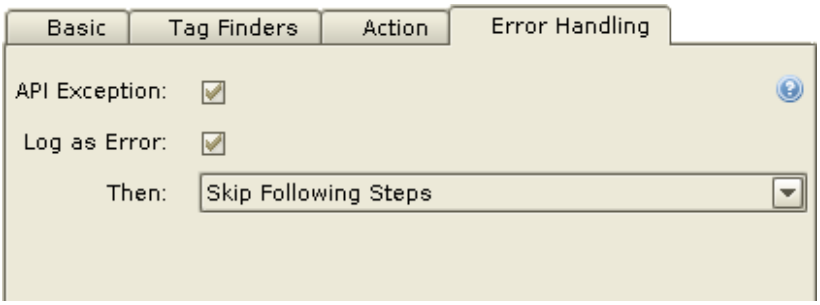


Step Aが成功しない場合は(Webページ上に要素が存在しないため)、その次の代替の試行()エラー処理が試行ステップ(この例では名前が付いていません)に通知を送信します。これによって2番目の空の分岐が実行され、試行ステップで開始された分岐全体の実行が終了します。したがって、Step Aが成功しない場合、Step BとCは実行されません。

この状況は頻繁に発生するため、特別なエラー処理オプションの後続ステップのスキップがショートカットとして導入されました。この例は次のように簡略化できます。



Step Aのエラー処理は次のように構成されます。これは、すべて新しいステップの場合のデフォルト構成です。

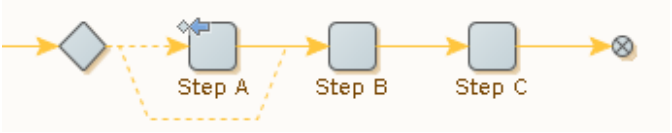


厳密に言うと、前述の試行ステップと完全に同じ動作を行うには、API例外およびエラーとして記録チェック・ボックスは選択を解除する必要があります。これは、エラー処理を行うこの2つの方法で、これらのチェック・ボックスのデフォルト値が異なるためです。

Step BがStep Aと同様の場合は(つまり、Step Bにも次の代替の試行エラー処理がある場合)、この同じショートカットが使用できることに注意してください。

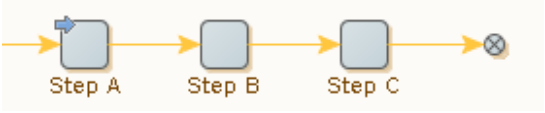
無視して続行

ある条件が満たされるとアクション(抽出など)を実行し、そうでない場合は単純にスキップできる場合があります。後続のステップは結果に基づきません(または、結果に対する適切なデフォルトを事前に設定します)。これは次のように表すことができます。

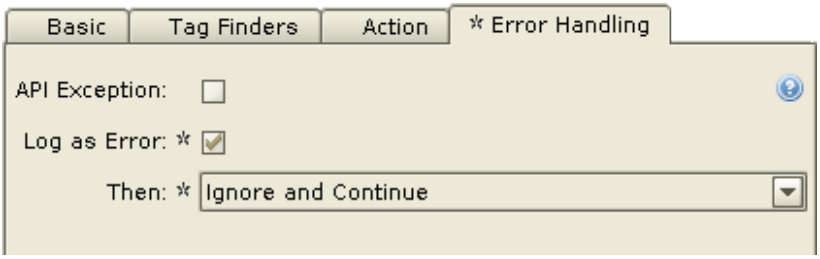


Step Aが成功しない場合は、その次の代替の試行()エラー処理によって、(名前が付いていない)試行ステップからの2番目の空の分岐が実行されます。この後に、Step Bでは、Step Aに入力されたのと同じロボット状態で実行が続行されます。したがって、Step Aは事実上スキップされます。

これは、Step Aでエラー処理オプションの無視して続行()を使用すると、試行ステップを使用しなくても実現できます。



無視される場合でも、その状況を記録しておくことが可能です。これを行うには、Step Aのエラー処理を次のように構成します。



試行ステップを使用する方法でも同じ処理を実行できます。

場所ターゲット

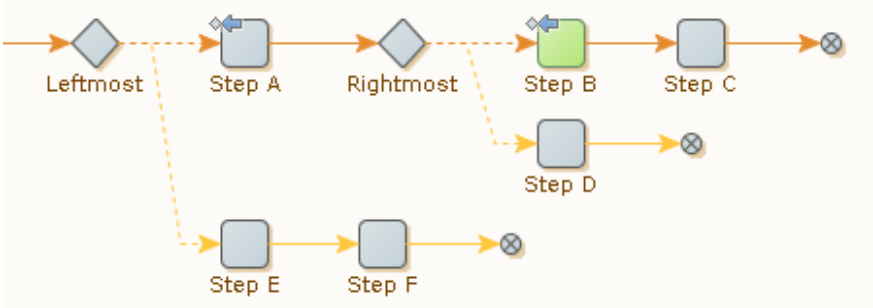
本質的に、次の代替の試行エラー処理は試行ステップを参照します。この試行ステップは、現在の実行パス上で前のステップであり、つまり、現在のステップまでに実行されたステップの1つである必要があります。そうでないと、次の代替の試行の「次の代替」が意味をなしません。

ここで興味深いのは、現在の実行パスに複数の試行ステップがあるとどうなるかということです。その場合、ロボットは各試行ステップに対応する「現在の分岐」を実行しているため、各試行ステップ

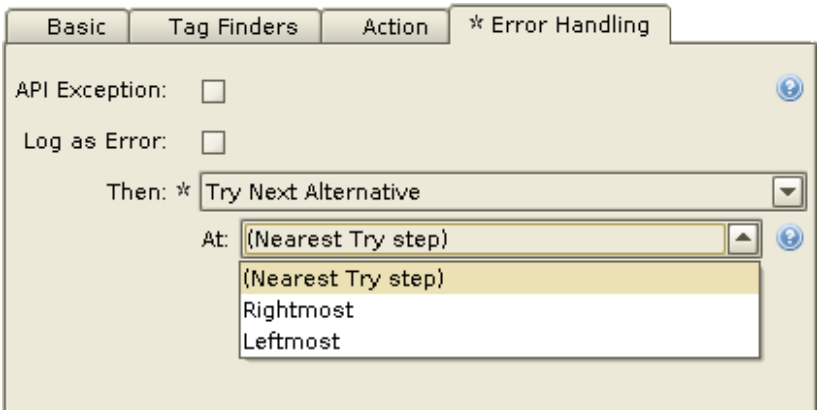
の「次の分岐」が異なります。たとえば、次のロボットのStep Bでエラーが発生した場合、次の代替の試行は次のどちらの場所で実行を続行するでしょうか。

・ 最も右の試行ステップにある次の分岐(実行はStep Cをスキップし、Step Dで続行されます)

・ 最も左の試行ステップにある次の分岐(実行はStep CとDをスキップし、Step Eで続行されます)



正解はいずれも選択可能で、Step Bのエラー処理構成は次のようになります。



デフォルトは最も近い試行ステップですが、実行パス上の他の試行ステップも明示的に選択できます。試行ステップは名前によって参照しますが、同じ名前の試行ステップが複数ある場合は、同じ名前の最も近い(最も右)試行ステップが選択されます。「Try-Catch」で説明するように、これは有効に使用できます。

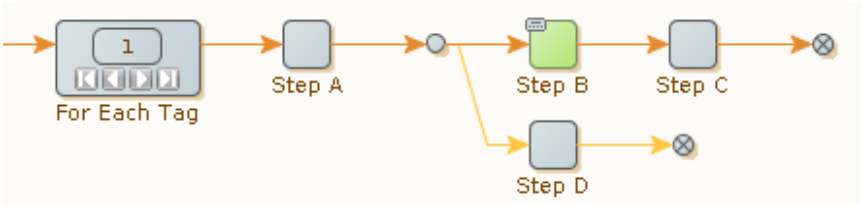
ここでは、次の代替の試行エラー処理のみに関連付けて場所ターゲットを説明していますが、同様の参照は次の繰返しおよびループの中断エラー処理オプションでも使用できます(後の項を参照)。その場合は、試行ステップではなく、ループ・ステップへの参照になります。

ループ

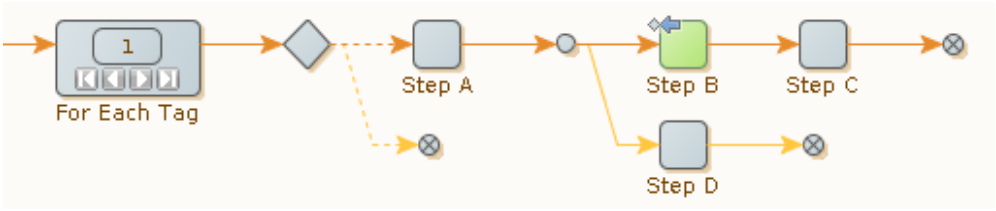
エラーが発生したりテストが失敗したときに、ループの現在の繰返し、またはループ全体の実行を中止することが適切な対応である場合があります。これは、2つの特別なエラー処理オプションを使用してサポートされています。

次の繰返し

次のロボットには、Step Bに次の繰返しエラー処理があります。これは、このステップの実行中にエラーが発生すると、ループの現在の繰返しの実行が停止されることを意味します。したがって、Step CとStep Dは両方とも実行されず、かわりに、ループ・ステップが繰り返す次のタグを反映したロボット状態で、Step Aで実行が続行されます。



このエラー処理オプションは一種のショートカットで、次の代替の試行および試行ステップを次の方法で使用した場合と同じ結果になります。



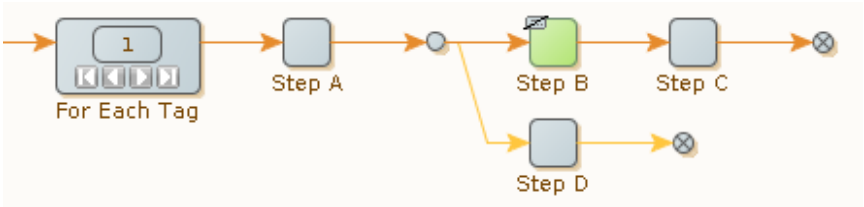
この変換を行う場合、他の試行ステップが干渉する可能性があるため、通常は場所ターゲットの使用が必要です。

ロボットで、ステップ後に複数のループ・ステップが含まれる場合、次の繰返しに進むステップを選択できます。これについては、「場所ターゲット」で説明しています。

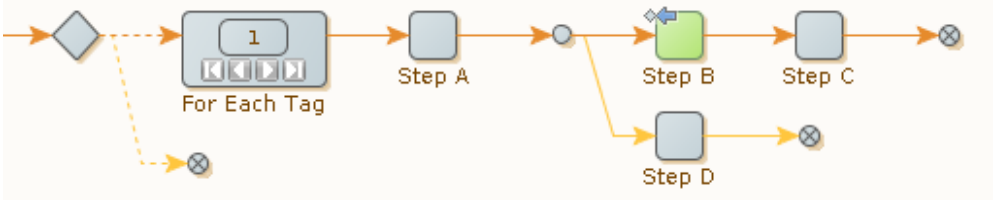
次の繰返しは、Repeat-Nextループでは機能しません。この2つのケースでは、ブラウザ状態に対して「次」という語の意味が大きく異なります。

ループの中断

次の繰返しを使用してループの1回の繰返しを完了するのではなく、次のようにループの中断を使用してループ全体を中止できます。



このエラー処理オプションも一種のショートカットです。次のロボットも同じ結果になります。



次の繰返しと異なり、ループの中断はRepeat-Nextループで機能します。

Try-Catch

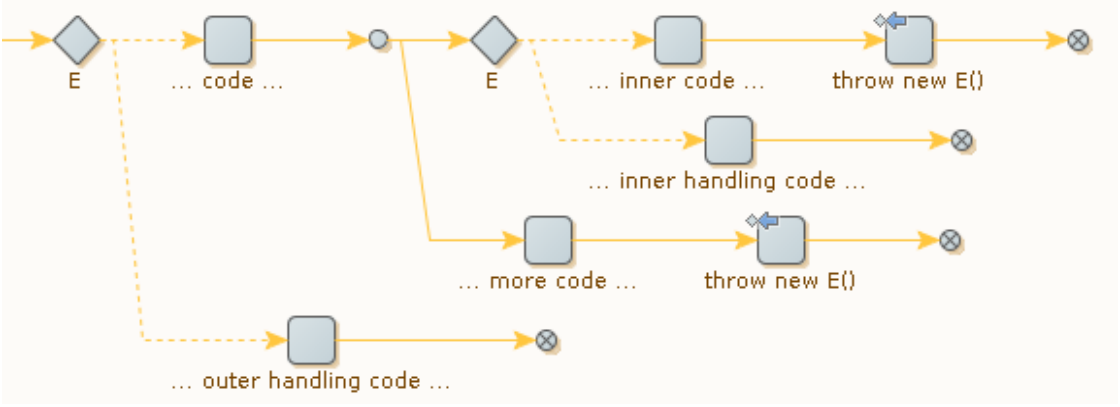
ターゲットの試行ステップへの明示的な場所参照を指定して次の代替の試行エラー処理を使用する場合、そのステップは名前によって識別されます。多くの場合、ターゲット・ステップとその名前を適切に区別することは重要ではありませんが、JavaやC#のtry-catch構造と同様の例外処理機能を提供するために利用できます。

このようなプログラミング言語では、"try"と"catch"間のコードのセクションに特別なエラー処理が含まれます。このセクション内で(名前付きの例外をスローすることによって)特定のエラーが示されると、同様に名前が付けられた"catch"に続くコード部分が実行されます。Try-catch構造はネストすることができ、名前付きの例外は常に、名前が一致して最も内側に囲まれた"catch"によって処理されます。次に例を示します。

```
try {   ... code ...
try {     ... inner code ...      throw new E();
// caught by innermost "catch"   }
```

```
catch (E e) {     ... inner handling code ...   }   ... more code ...
throw new E();
// caught by outermost "catch" }
catch (E e) {   ... outer handling code ... }
```

ロボットでは、試行ステップを使用して同様の処理を実行できます。特定の名前がある試行ステップへの場所参照は、現在の実行パスに従って、その名前の最も近い前のステップを意味します。同じ実行パス上であっても、複数の試行ステップに同じ名前を使用できます。したがって、各try-catch構造は、同じ名前を持つ試行ステップを使用して、例外としてモデル化できます。試行ステップには2つの分岐があり、1つは"try"構造のコード部分、もう1つは"catch"構造のコード部分用です。



Java/C#構文とDesign Studio用語は次のように対応します。

Java/C#構文	Design Studioで使用するもの
try { ...code... }	試行ステップの最初の分岐(ステップは**code**に対応します)
例外の名前	試行ステップの名前
throw new E() **try**のコード内	"Eでの次の代替の試行"を使用したエラーの処理
catch E { ...code... }	"E"という名前の試行ステップの2番目の分岐(ステップは**code**に対応します)

したがって、エラー処理で試行ステップを使用するときは、処理対象のエラー状態の後に試行ステップを指定するのが基本的な考え方です。次の利点があります。

- 各試行ステップの目的を明確にするのに役立ちます。
- (ロボットのより左側にある試行ステップを使用して)通常レベルでエラーが処理されるとき、(同じ名前が付いた2番目の試行ステップを使用して)特別な処理を簡単に実行できる場合があります。

ロボット・ビューにおけるステップのエラー処理の表示

ロボット内のステップに特別なエラー処理が設定されている場合、ロボット・ビューではデフォルトでステップに小さい記号でマークされます。この記号はステップに定義されているエラー処理の種類によって異なり、ユーザーはステップにカスタム・エラー処理が定義されていることを視覚的に識別できます。ユーザーがステップにカスタム・エラー処理のマーク付けを希望しない場合は、「オプション」メニューにあるDesign Studioの設定ダイアログで無効にできます(「ロボット・エディタ」の項を参照)。

入力変数を使用するロボットを記述する方法

入力変数を取るロボットは要件がかなり厳密であることが多いため、おそらくDesign Studioで記述できる最上級のロボットと考えられます。通常、これは速度と高い信頼性が求められます。(これら

の要件はすべてのロボットに該当しますが、通常の抽出ロボットはバッチで実行されるのに対して、入力変数を取るロボットはリアルタイムで実行されるのが一般的です。また、通常は、値が誤って抽出されることより、銀行振替が誤って入力されることの方が問題は大きいです。)

次に、速度と信頼性の要件について順に説明します。

速度の要件は、最適化によって達成できます。Design Studioでは、ナビゲーション(クリックなどのアクション)を含むステップ・アクションが合計実行時間の大部分(80%以上)を消費します。したがって、可能なかぎり、不要なナビゲーションを回避する必要があります。不要なナビゲーションを減らす方法の1つは、関連する情報に直接リンクすることです。たとえば、フロント・ページからログイン・フォームにナビゲートするのではなく、ログイン・フォームを直接ロードします。セッションやCookieのため、このような直接のナビゲーションが不可能な場合がありますが、多くの場合は可能です。

信頼性の要件には、タスクをスムースに実行するか、またはそれができない場合にレポートするようにロボットを記述することが含まれます。これは、思ったほど簡単ではありません。実際、ロボットが成功したかどうかを把握できない状況があります。たとえば、ロボットがオーダーを発行し、Webサイトでレスポンスが送信されない場合、何が発生したのでしょうか。(通常のブラウザでユーザーがするようなことを行った可能性があります。)ただし、可能なかぎり、そのような不確実性が最小限になるようにロボットを記述する必要があります。具体的には、潜在的なエラーを検出するために、ロボットが不確実要素(Webサイトなど)と相互作用するたびに、その相互作用を完全に分析する必要があります。エラーが検出された場合、通常は、ロボットを起動したアプリケーションにエラーの原因を通知する戻り値の形式でレポートされる必要があります。エラーが検出されない場合、ロボットは次のアクションに進むことができます。詳細は、「ロボットをより堅牢にする方法」を参照してください。

入力変数を取るロボットを記述するとき、多かれ少なかれ、他の種類のロボットと同様のステップ・アクションやデータ・コンバータを使用します。ただし、入力変数を取るロボットに特に役立つステップ・アクションやデータ・コンバータがあります。次の項目については、参照ドキュメントを確認する必要があります。

- 変数の変換アクション: Webサイトから抽出した変数値を変換したり、入力変数値をフォームに挿入する前に変換します。
- 変数のテスト・アクション: 1つ以上の条件("price < 5000"など)に従って、入力変数値などの変数値をテストします。
- 変数の取得データ・コンバータ: 後続の処理、またはフォーム入力フィールドへの挿入のために変数値をフェッチします。
- リストを使用した変換データ・コンバータ: コンテンツを変換します。このデータ・コンバータは、Webサイト・フォームに挿入するために入力変数属性値を標準化する場合、またはWebサイトから抽出したコンテンツを標準化する場合に役立ちます。

スニペットを作成および再利用する方法

スニペットは、次の3つの方法で作成できます。

- 選択したステップからスニペットを作成: 1つ以上のステップ(グループ化可能なステップで、単一のグループ・ステップでないことが必要)を選択し、選択からスニペットを作成アクション  を選択します。これによって、新しいスニペットの名前を要求するプロンプトが表示され、選択したステップを含んだ指定の名前のスニペットが作成され、選択したステップのかわりに新しいスニペット・ステップが挿入されます。
- グループ・ステップをスニペット・ステップに変換してスニペットを作成: グループ・ステップを選択し、スニペットへのグループの変換アクション  を選択します。これによって、新しいスニペットの名前を要求するプロンプトが表示され、グループのステップを含んだ指定の名前のスニ

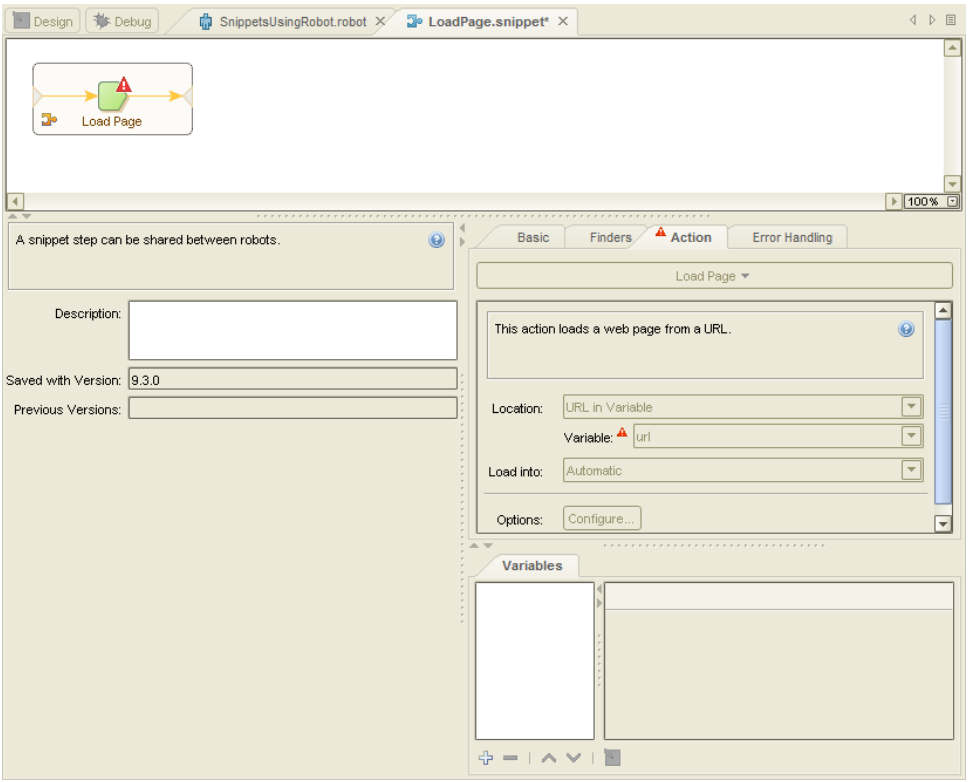
ペットが作成され、選択したグループ・ステップのかわりに新しいスニペット・ステップが挿入されます。

- 「ファイル」メニューの新規スニペットを選択してスニペットを作成。ウィザードが表示され、スニペットの名前を指定するように要求するプロンプトが表示されます。スニペットが作成された後に、空のスニペットがプロジェクトに表示され、そのスニペットのエディタが開きます。このエディタ内ではスニペットのコンテンツ(スニペット内のステップ)は編集できず、説明および参照先の変数リストのみ編集できることに注意してください。

変数およびスニペット

ロボット内のすべての場所のステップと同様に、スニペット内のステップでも変数を使用できます。スニペットのステップは、常にロボットの内部で編集されます。このコンテキストでは、ロボットに定義された変数をスニペットで使用できます。別のロボットでスニペットを再利用するには、そのスニペットを使用する各ロボットで、スニペット内のステップで使用する変数を定義する必要があります。

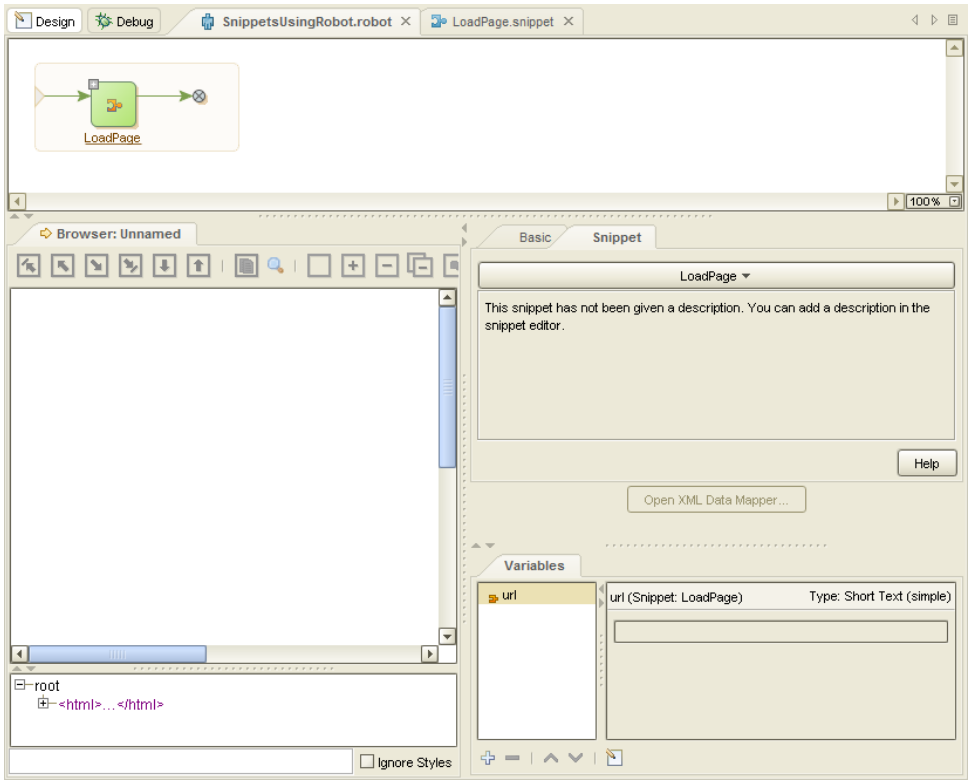
スニペットには独自の変数を定義できます。独自のエディタにスニペットを開き、変数をスニペットに定義します。スニペットが編集されたロボットに変数が存在し、その変数を使用するステップがスニペットにすでに含まれている場合、そのステップは赤いフラグでマークされます。



スニペット自体に定義されていない変数を使用する、独自のエディタ内のスニペット。

右下隅にあるアクティブな変数エディタ(ロボットと完全に同じ)に注意してください。

スニペットで変数を定義した場合、ロボットでそのスニペットを使用すると、スニペット変数がロボットの変数セットに自動的に追加されます。



変数'url'が定義されたスニペットを使用するロボット。

スニペットからインポートされた変数が変数リストでマークされているのがわかります。

ロボットには、使用するスニペットで定義された変数と同じ名前の変数定義を含めることができません。そうする場合は、変数タイプが一致する必要があります。ロボットからスニペットを削除すると、そのスニペットからインポートされた変数も削除されます。

スニペットの適切な使用

プロジェクトでスニペットを使用するときは、いくつかの点を考慮する必要があります。

- スニペットを使用する各ロボットにステップを設定するのではなく、スニペット内部でステップを実行する必要があるデフォルトでないロボット構成を、スニペットのステップに配置することをお薦めします。
- スニペットをロボットに挿入するときは、スニペットに定義する変数の名前が、ロボットに定義する変数と競合しないように注意してください。Design Studioでは、スニペットに定義する変数が、ロボットに定義する変数と同じ名前である状況に対応できます。スニペットが別のコンテキストで機能するために変数が必要な場合は、主にスニペットで変数を定義することをお薦めします。これによって、スニペットの再利用がより簡単になります。
- スニペットの記述でスニペットが機能するために必要な指定タグまたはウィンドウ(あるいはその両方)に関して、コンテキストをドキュメント化することをお薦めします。
- スニペットには、他のスニペットを参照するスニペット・ステップを含めることができます。ただし、最終的にスニペットにスニペットが含まれる循環参照は使用できません。その場合は、Design Studioでエラーがレポートされます。したがって、スニペット内部にスニペットを含める場合は注意してください。

ロボットをより堅牢にする方法

Webサイトは予告なしに変更される場合が多くあります。注意しないと、このような変更によってロボットがタスクの実行に失敗することがあります。堅牢性とは、ロボットがWebサイトの変更にどれ

だけ対応しているかを示すために使用される用語です。ロボットがより変更に対応して正しく機能するほど、ロボットはより堅牢になります。

ただし、堅牢性にはかなりの犠牲が必要です。堅牢なロボットを記述することは、信頼性の低いロボットを記述するよりも困難で時間がかかります。(すべてのプログラミング言語でプログラムを記述するときも同様です。)これには、対象のWebサイトの分析、および様々な状況での応答方法の把握(登録フォームが誤って記入されている場合など)が含まれます。ある意味では、堅牢なロボットの記述にはWebサイト・ロジックのリバース・エンジニアリングが含まれ、通常、これを実行する唯一の方法は検索を通して行うことです。

堅牢さを実現するには2種類のアプローチがあり、それぞれ目的が異なります。

- 可能なかぎり成功。

- 完全でない場合は失敗。

それぞれのアプローチについて順に説明します。

可能なかぎり成功は、ニュース・タイプ変数を抽出するロボットの場合、可能なかぎり多くのニュース項目を抽出することを意味します。Design Studioでは、条件アクションの試行ステップおよびデータ・コンバータを使用して、異なるレイアウト、情報の欠落、および書式設定が異なるコンテンツに対処します。

完全でない場合は失敗は、オーダー発行ロボットの場合、フィールドに正しく入力する方法が不明なとき、またはオーダー結果ページが正しいレイアウトと一致しないときにロボットが即時に失敗することを意味します。この意味での失敗は、API例外の生成を意味しません。かわりに、エラーおよび失敗発生を説明する専用の値をロボットが返すことを意味します。入力変数を取るロボットは、可能なかぎり成功ではなく、失敗することを選択するほうが多いです。Design Studioでは、専用のエラー・タイプ変数、エラー処理および条件アクションを使用して、予期しない状況を検出して処理します。

ロボットをより堅牢にするために使用できるDesign Studioテクニックの詳細は、「HTMLからコンテンツを抽出する方法」、「HTML表からコンテンツを抽出する方法」、「エラーを処理する方法」および「タグ・ファインダを使用する方法」の各項を参照してください。

セッションを再利用する方法

セッションとはWebサイトの参照の結果で、実行中に取得されたページ、ページURL、Cookieおよび認証で構成されます。ただし、必要な情報に簡単にアクセスできるセッションを取得するには、ログインなどいくつかのナビゲーション・ステップが必要な場合があります。

ロボットを頻繁に実行し、応答時間を非常に短くする必要がある場合は、ロボットに適切なセッションを取得するために必要な時間が足りなくなる可能性があります。ただし、セッションを1回取得し、それをロボット間で共有すると、ロボットの実行時間を大幅に短縮できる場合があります。これは、セッション・プールを使用して、セッションを再利用する方法です。

セッションの再利用で使用するのは、次の2つのステップ・アクションです。

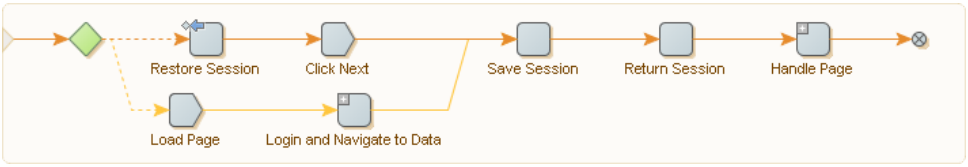
1. セッションの保存アクション: セッション・プールまたは変数にセッションを保存します。

2. セッションの復元アクション: セッション・プールまたは変数からセッションを復元します。

セッション・プールからセッションを復元するには、セッションを識別する必要があります。セッションは、サイト名、ユーザー名およびパスワードによって識別されます。

例について説明します。Webサイトにログインし、データを収集して返すロボットがあると仮定します。ただし、収集するデータは、(次のページへのリンクなどを使用して)リンクされた多数のページに分散しています。ロボットの最初の起動ではサイトにログインして最初のページのデータを返し、

後続の起動ごとにデータの新しいチャンク(次のページ)を返す必要があります。したがって、ロボットの起動ごとにログイン・ユーザーのセッションを共有しますが、返されたデータの量も記憶する必要があります。このロボットは次のようになります。

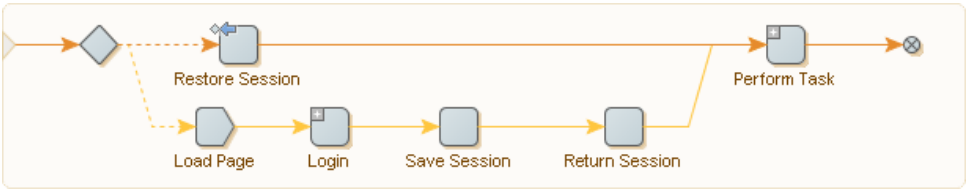


ロボットは呼び出されると、最初に入力変数からセッションの復元を試み、セッションが存在する場合は、そのセッションが使用されて、次のステップで次のページへのリンクがクリックされて新しいページのデータが取得されます。セッションがロボットに渡されない場合、そのステップは失敗して2番目の代替(ログインを行い、サイト上でデータを検出できる関連ページにナビゲートする)が実行されます。

ロボット実行が2つの代替分岐の1つを通過すると、セッションの保存ステップに達します。これによってセッションが保存されるため、ロボットが次回に呼び出されるときに使用できます。ただし、これを可能にするには、ロボットの呼出し側にセッションを返す必要があります。これは通常の値を返すステップであるセッションを返すステップで処理され、セッションが含まれる変数の値を返します。つまり、変数は属性タイプ「セッション」の属性を持つタイプで、セッションの保存ステップによってセッションが変数に格納されます。最後に、ロボットがデータの最後に達すると(つまり、ページに次のページへのリンクが存在しない)、次のセットをクリックでエラーが生成されます。エラー処理を後続ステップのスキップに設定しているため、ロボットではこれが無視されますが、API例外にチェック・マークを付ける場合は呼出し側は例外を受け取ります。たとえば、ロボットがJavaから呼び出される場合、これを使用してデータの最後に達したことを認識できます。

セッションが保存された後、ロボットの残りのステップでページからデータが抽出されます(たとえば、表をループして各行の値を返します)。

複数のロボットが同じサイトにログインして異なるタスクを実行する場合は、セッション・プールの最適化も使用できます。ログイン・プロセスで時間がかかったり、単純なタスクを実行するロボットが多くの回数呼び出される場合は、ログイン後にセッションを保存して再利用することをお薦めします。次の図に、セッション再利用ロボットのブループリントを示します。有効なセッションがないロボットがログインを実行して、他のロボットが再利用できるように保存されたセッションを返す場合があります。通常、ロボットは使用可能なセッションに依存しませんが、セッションを取得するために常にフォールバックを提供します。



Design Studioでは、セッション・プール(使用する場合)の内部動作について少し把握しておくことが重要です。これは、Design Studioでのロボットの実行が、ロボット実行の自然なフローで制御されておらず、ユーザーとの相互作用によって制御されているためです。最初に、セッションの保存アクションを含むステップを実行して、セッションを保存します。これを行うには、セッションの保存ステップに続くステップを選択します。この後に、格納されたセッションをセッションの復元アクションで取得できます。 アイコンをクリックしてキャッシュを更新したり、別のロボットをロードした後でも、保存されたセッションはセッション・プールに残ります。現在は、Design Studioを再起動する以外に、セッション・プールからセッションを削除する方法はありません。

既存のタイプを変更する方法

あるタイプの変数を使用してロボットを記述した後にタイプを変更する必要がある場合は、慎重に行う必要があります。誤った作業を行うと、ロボットの実行が停止する場合があります。

既存のロボットの変数ですでに使用されているタイプに対して次の変更を実行する場合は、注意が必要です。変更を行うと、その変数を使用できなくなります(ただし、ロボットは変数を使用しなくてもロードできます)。

- タイプの名前の変更。

- タイプの削除。

- タイプの属性の削除または名前変更(ロボットで、そのタイプの1つ以上の変数によって、デフォルト以外の値がその属性に割り当てられている場合)。

- 属性の属性タイプの変更(ロボットで、そのタイプの1つ以上の変数によって、新しい属性タイプと互換性がない属性に値が割り当てられている場合)。

ロボットが開いているときに前述のいずれかの変更を行うと、ロボット・エディタの上部にある赤いステータス・バーに、問題を説明するテキストが表示されます。また、ステータス・バーには、問題がある変数を削除してロボットを再ロードするためにクリックできるボタンも表示されます。もちろん、タイプを適切に変更して問題を解決することもできます。これを行った場合は、ロボットに戻って作業を続行できます。

タイプに対する次の変更は、そのタイプの変数が削除されずに(削除する場合は再ロードが必要)ロボットに自動的に継承されますが、ロボットが実行されたときにエラーが生成される場合があります(このエラーは後で修正できます)。

- 属性の名前の変更。

- 属性の「必須」プロパティをfalseからtrueに変更。

- 「必須」プロパティがtrueに設定された新しい属性の追加。

- 変数で値が割り当てられた属性の削除または名前変更。

- 変数で値が割り当てられた属性の属性タイプの変更。

次の変更は、既存のロボットに影響を与えずに常に実行できます。

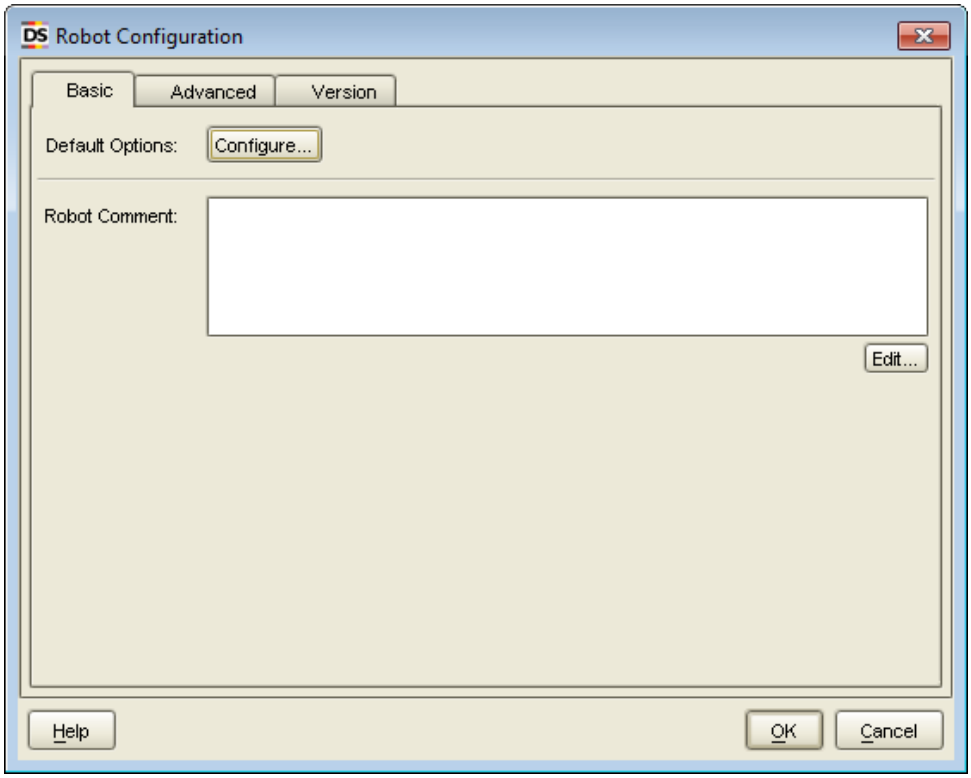
- 属性の「必須」プロパティをtrueからfalseに変更。

- コメントの変更(場所に関係なく)。

- 「必須」プロパティがfalseに設定された新しい属性の追加。

ロボットを構成する方法

ロボットには複数のプロパティがあり、Design Studioのツール・バーにある  アイコンをクリックするか、Design Studioのメイン・ウィンドウの「ファイル」メニューにあるロボットの構成を選択すると構成できます。これによって、ロボット構成ウィンドウが表示されます。



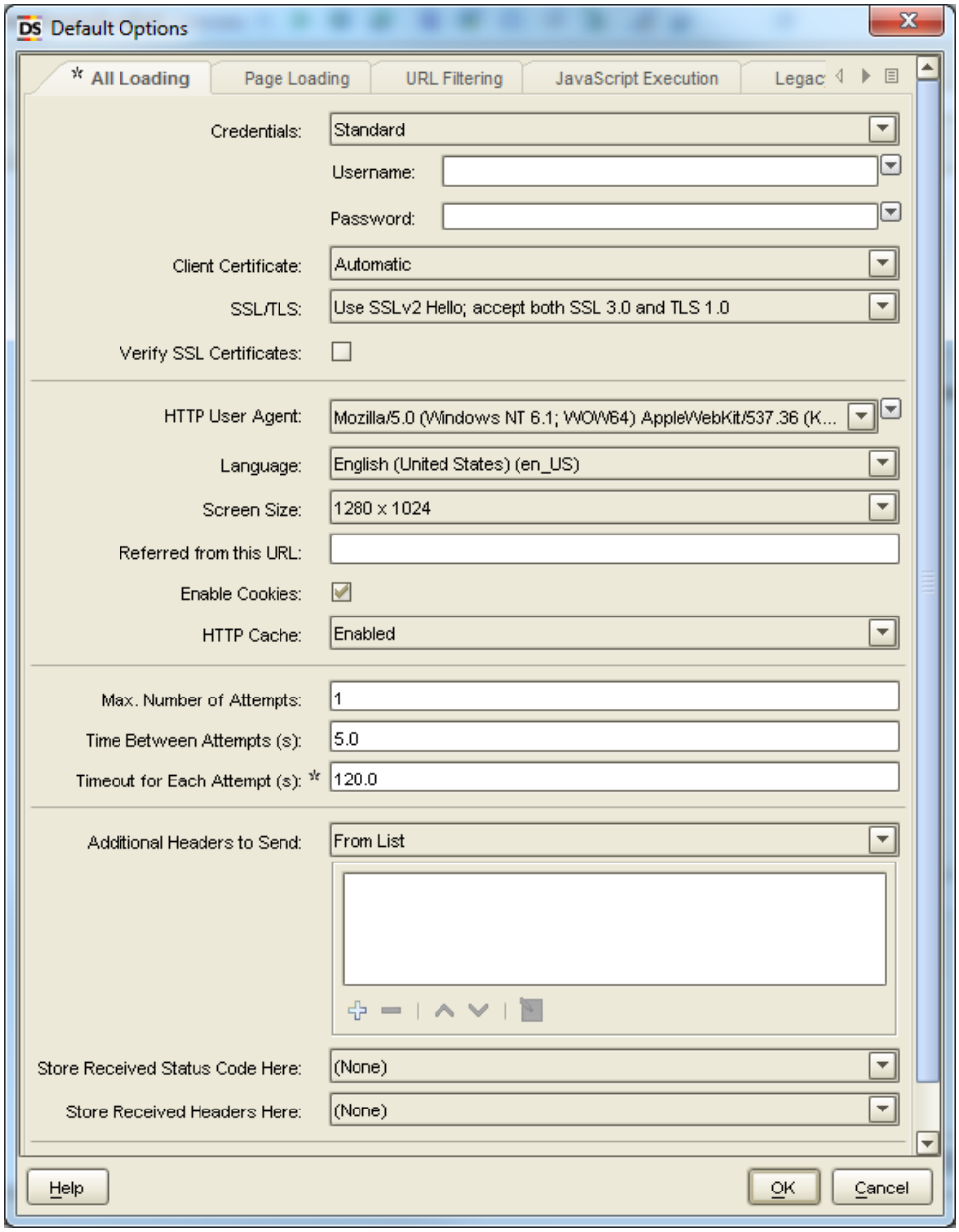
ロボット構成ウィンドウ

「基本」タブでは、このロボットのすべてのステップ・アクションに適用されるデフォルトのオプションを構成できます。ステップ・アクションは、必要に応じてこれらのグローバル・オプションを上書きできます。ロボットに対するコメントを入力することもできます。これは、ロボットの編集時の考慮点など、ロボットの仕組みをドキュメント化する場合に便利です。

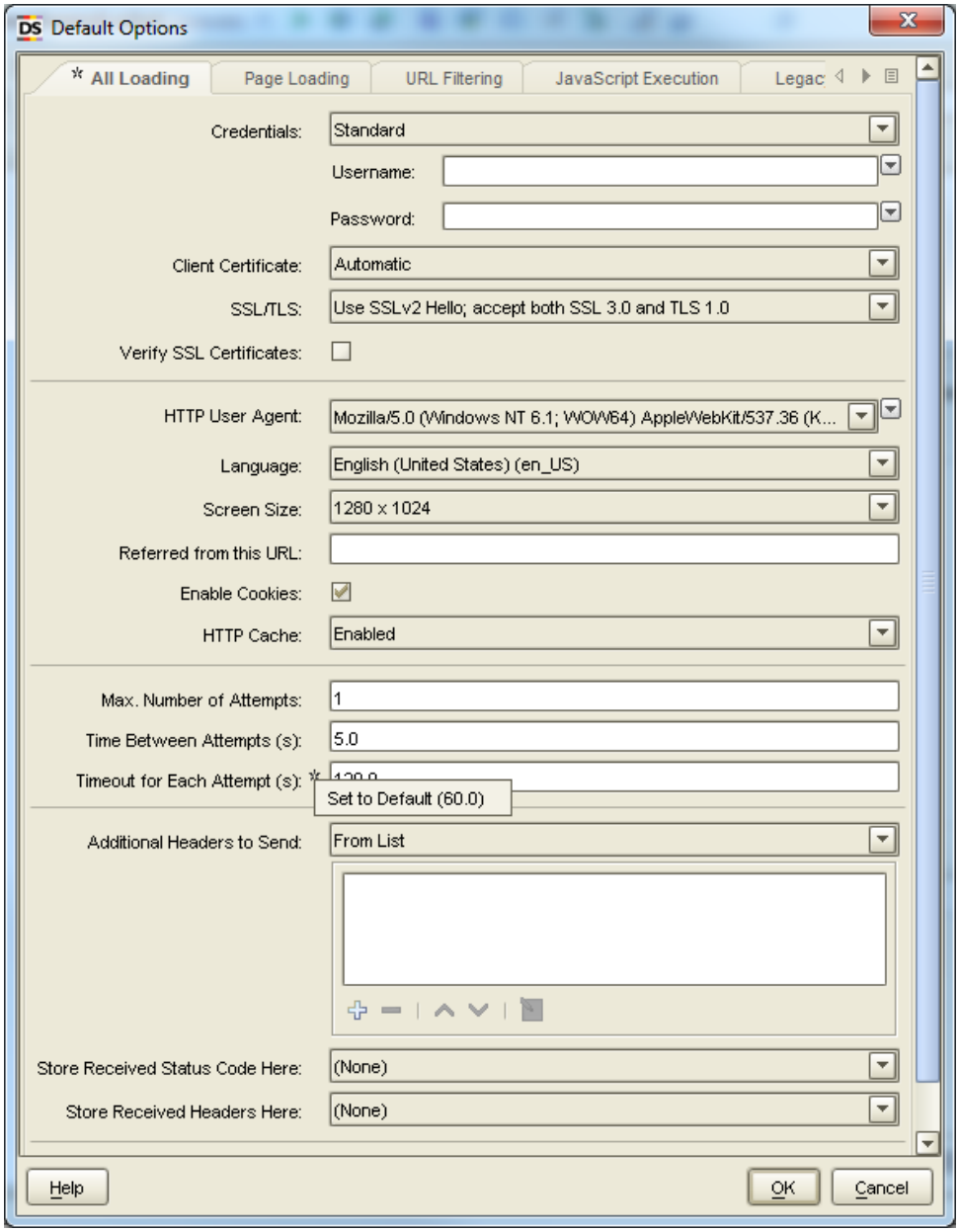
「拡張」タブでは、この特定のロボットが実行するページおよびデータのロードのすべてに対して使用されるオプションのプロキシ・サーバーを指定できます。このプロパティの使用が必要なのはまれである可能性があります。通常は、Design Studioの設定で、1つ以上のプロキシ・サーバーを指定することをお薦めします。詳細は、「プロキシ・サーバー」を参照してください。特定のロボットに指定されたプロキシ・サーバーは、その他の方法で指定されたプロキシ・サーバーを上書きします。さらに、プロキシ・サーバーは、ロボットの実行時にプロキシの変更アクションを使用して変更できます。

デフォルトからの変更の表示

多くの場合、ロボットの特定の部分の構成が非標準な方法(ページのロード・ステップに、デフォルト値の「60.0」より長いタイムアウトが設定されているなど)で設定されているかどうかを確認するのは困難な可能性があります。デフォルトから変更されたプロパティの場所に到達するには、ダイアログを開いて各タブをクリックする必要があります。次に、表示されたすべてのプロパティを調査して、これらの値のいずれかが非標準であるかどうかを確認する必要があります。また、この操作を可能にするには、すべてのプロパティのデフォルト値を記憶しておく必要があります。そのため、Design Studioには、変更の表示機能を導入しています。正しく定義されたデフォルト値があるプロパティでは、その値が変更されると、プロパティの名前の横にアスタリスク*のマークが付きます。次の図に、オプション・ダイアログでこれを示します。



タブにもアスタリスクがあることに注意してください。これは、タブ内の一部のプロパティが変更されていることを意味します。そのプロパティ名またはアスタリスクを右クリックすると、ポップアップ・メニューを使用してその値をデフォルトの値にリセットできます。この操作のショート・カットとして、プロパティ名またはアスタリスクをダブルクリックすると、プロパティの値がデフォルトの値に設定されます。また可能な場合、ポップアップ・メニューにはデフォルト値が表示されます。さらに、タブのアスタリスクには結合されたポップアップ・メニューがあり、これを使用すると、タブのすべてのプロパティをデフォルト値にリセットできます。次の図に、オプション・ダイアログの各試行のタイムアウト・プロパティでこれを示します。



変更の表示の機能が他の場所とは多少異なる場所があり、それが、ステップのオプションを構成する際に使用するオプション・ダイアログです。通常、オプションは2つの場所で構成できます。

・ ロボット構成

・ これらのオプションに依存する可能性があるステップ

両方の場所で、オプション・ダイアログを開くボタンをクリックしてオプションを構成しますが、2つの場所のデフォルト値は異なります。ロボット構成からダイアログを開く場合、デフォルトは固定のアプリケーションのデフォルト、つまりDesign Studioアプリケーションが提供するデフォルト値です。ステップ構成からダイアログを開く場合、デフォルトはロボット構成で定義された値(ロボットのデフォルト)です。つまり、ステップは、そのステップに対するオプション値が明示的に変更されないかぎり、ロボット構成からオプション値を継承します。たとえば、ロボット構成でオプションのCookieの有効化の選択が解除されている場合、このオプションに依存するすべてのステップでもこの設定が使用されます(ステップでこのオプションを明示的に変更しないかぎり)。ステップのオプションのアスタリスクは、ロボット構成で定義済の値とは異なる値をステップで使用することを示しますが、必ずしもアプリケーションのデフォルトと異なることを示しているわけではありません。

ステップのオプション・ダイアログとロボット構成のオプション・ダイアログのアスタリスクは、別の点でも意味が異なります。ステップのオプション・ダイアログのオプションの場合、アスタリスクは、特定のオプションの値が意図的に固定値(対応するロボット構成の値と必ずしも異なるわけではありません)に設定され、対応するロボット構成の値の変更によって影響されないことを意味します。たとえば、最初に、ステップの各試行のタイムアウト・プロパティを「120」に設定し、ロボット構成の対応する値が「60」の場合は、このオプションの横にアスタリスクが表示されます。ロボット構成の値が次に「120」に変更されて2つの値が実際には同じ値になっても、ステップの値にはアスタリスクのマークが付いたままで、ロボットの値が再度「120」から変更された場合も、このステップの値は「120」のまま変更されません。ただし、このステップの値をポップアップ・メニューを使用するか、ダブルクリックしてデフォルトの値(ロボット構成の値)に戻した場合、このステップの値にはロボット構成の値が使用されて、その後はこの値に加えられた変更に従います。

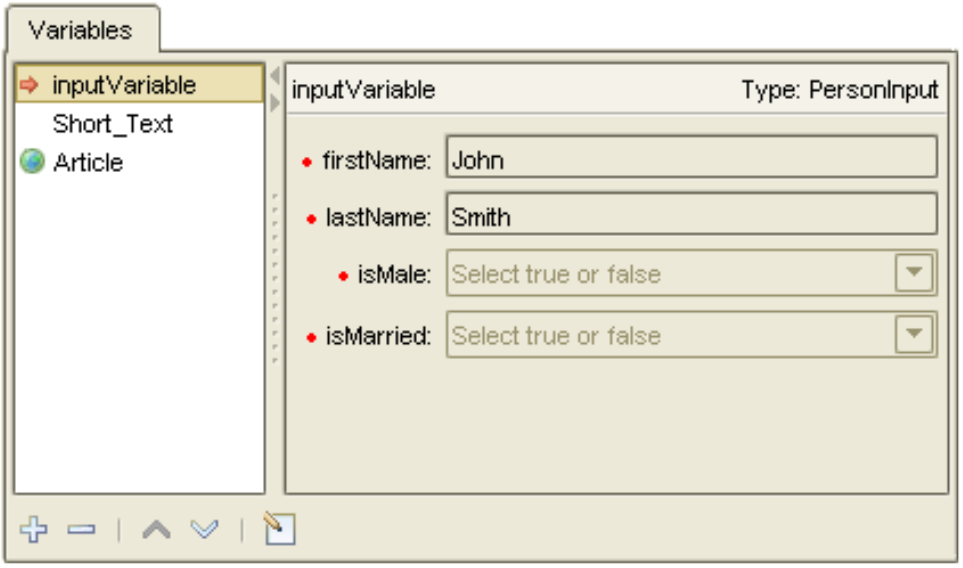
ステップのエラー処理の構成には、デフォルトの値が他の構成の選択に依存する別の状況が存在します。

変数を構成する方法

新しいロボットを作成するときは、通常、その変数の構成から開始します。変数は、(ある時点で変数の初期値の変更が必要になった場合など)ロボットの存続期間中はいつでも再構成できます。

ロボットの変数は、ロボット・エディタのステップ・ビューの下にある変数ビューで構成します。指定する変数は、ロボット状態の一部となり、ロボットの最初のステップへの入力として渡されます。

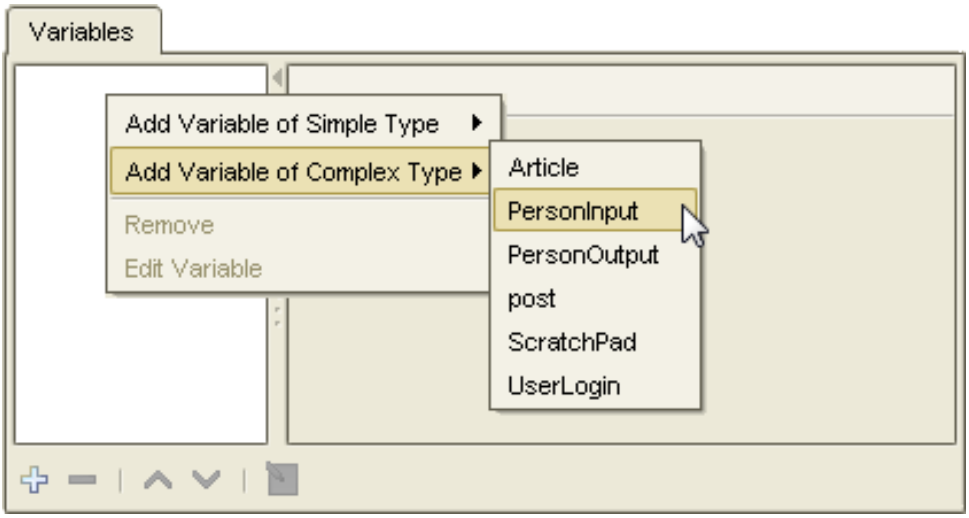
変数ビューには、選択した変数の詳細とともに変数のリストが表示されます。アイコンが横に表示されている変数は、入力変数として設定可能です。また、アイコンが横に表示されている変数は、グローバルとして設定可能です(変数の構成方法は次を参照)。次に示す変数ビューには、1つの入力変数、1つの通常の変数および1つのグローバル変数の3つの変数が含まれています。



変数ビュー

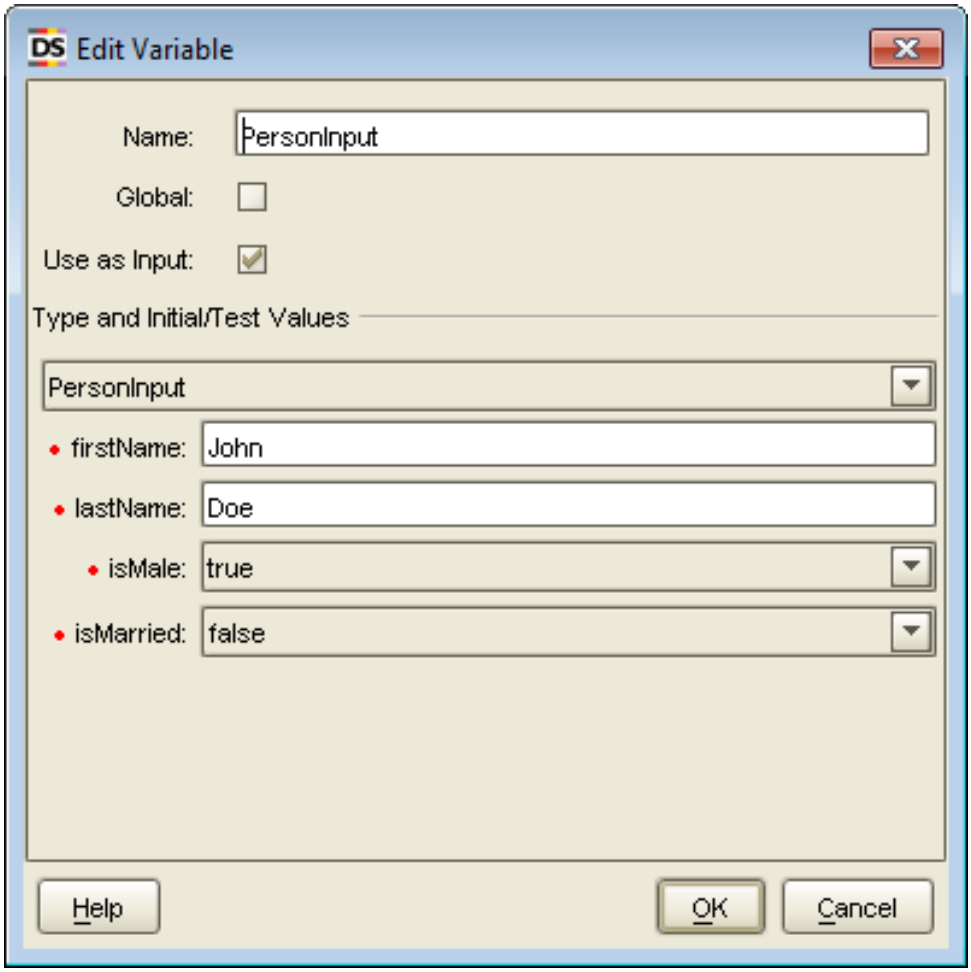
変数ビューには、現在のステップの変数の値が表示されます。これらの値はロボットの実行結果であるため、直接変更することはできません。ただし、変数を追加または削除することはできます。

新しい変数を追加するには2つ方法があります。変数ビューの下部にあるボタンをクリックするか、変数ビューを右クリックして、次の図に示すように、作成する変数のタイプを選択します。



新しい変数の追加

変数の追加は、変数構成ダイアログを使用して実行します。タイプを先に選択する右クリックの方法を使用して変数を追加する場合は、事前に選択済のタイプが含まれたダイアログが開きます。変数構成ダイアログが次の図に示すように表示されます。これは、既存の変数を構成する場合にも表示されるダイアログで、既存の変数をダブルクリックするか、  ボタンをクリックすると表示されます。



変数構成ダイアログ

このダイアログには、複数のオプションがあります。最初に、変数の名前を選択できます。この名前は、空白を含めることができないなど、一定の標準に準拠する必要があります。「OK」ボタンのクリック時に、名前が無効な場合はダイアログに示されるため、続行するには名前を変更する必要があ

ります(または「取消」をクリックします)。2番目に、変数のタイプを選択する必要があります。タイプおよび変数との接続については、ここを参照してください。タイプを選択すると、そのタイプの属性に応じて複数の入力フィールドが表示されます。これらのフィールドを使用して、変数に初期値を提供できます。変数に手動で名前を付ける必要はありません。名前を入力せずに「OK」をクリックすると、ダイアログによって、タイプの名前から名前を生成するかどうかを尋ねられます。

注意: 変数構成ダイアログでは初期値を編集できます。つまり、このダイアログは、変数ビューとして現在の値を表示しているわけではありません。指定する値は、実行の開始時に変数が保持する値です。

このダイアログの2つの残りのオプションは、「グローバル」および入力として使用の2つのチェック・ボックスです。これらを使用して、変数をロボットへの入力として使用するかどうか、またはグローバルとして使用するかどうか(あるいはその両方か)の重要なプロパティを構成します。

変数を入力として使用する場合は、RoboServerでのロボットの実行時にロボットにその変数の値を提供できるようになります。これは、変数を入力として選択すると、このダイアログで変数の属性として入力した値がテスト入力とみなされ、使用されるのはDesign Studioでこのロボットを使用するときのみであることを意味します。RoboServerでこのロボットを実行する場合、この入力値は、ロボットを実行するクライアントが提供する値によって上書き(置換)されます。単純タイプの変数は一時変数としてロボット内部で使用されるため、このタイプの変数は入力として使用できないことに注意してください。入力変数の使用の詳細は、ここを参照してください。

「グローバル」チェック・ボックスは変数をグローバルとしてマークし、これによって、この変数値をロボットの実行全体で保持します。これは、値をループの繰返しおよび分岐にわたって保持しない通常の変数とは異なります。

グローバル変数は、カウンタの作成方法を提供したり、繰返しおよび分岐全体に対してその他の種類の計算を実行します。また、グローバル変数は、カンマ区切りの値で構成されたテキストを累計するなど、繰返しまたは分岐にわたるデータの累計に対して使用できます。

注意: Design Studioでは、グローバル変数の値は、現在のステップに到達するために実行した特定のステップに依存します。ステップを正しい順序で実行するように注意しないと、値が実際にロボットが実行されたときと異なることになります。

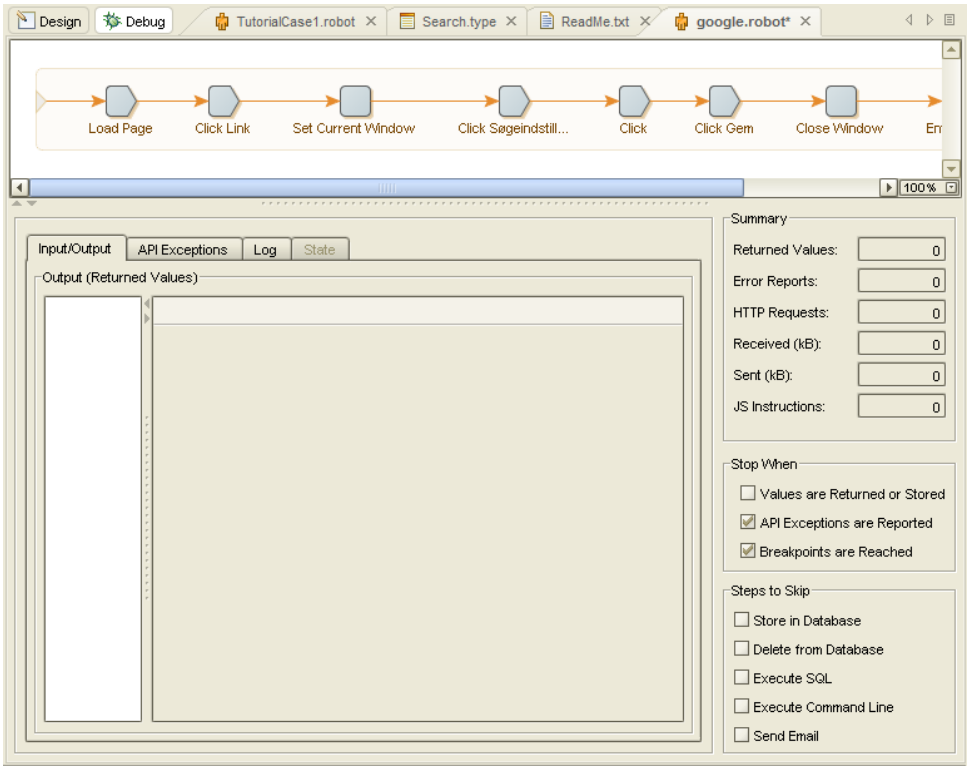
変数を削除するには、変数を単に右クリックして「削除」を選択します。あるいは、変数を選択して、リストの下部にある ボタンをクリックします。

ロボットをデバッグする方法

この項では、Design Studioに組み込まれているデバッグ・モードを使用してロボットをデバッグする方法ついて詳細に説明します。デバッグ・モードでは、RoboServerでの実行と同様にロボットを実行できます。これによって、ロボットが期待どおりに実行されるかを確認できます。

基本的なデバッグ

デバッグ・モードに切り替えるには、Design Studioで アイコンをクリックするか、デバッグ・ボタンをクリックします。ロボットのデバッグを開始するには、 アイコンをクリックします。



ロボット・エディタのデバッグ・モード

デバッグ・モードでのロボットの実行時には、ロボット・ビューに現在の場所を表示できます。メイン・パネルには、実行結果を表示することもできます。入力/出力タブの「入力」パネルには、入力変数が表示され(ある場合)、「出力」パネルには、実行で現在までに返されたすべての値が表示されます。ロボットに入力変数がない場合、「入力」パネルは表示されません。API例外タブでは、実行で現在までに生成されたすべてのAPI例外を確認できます。「ログ」タブでは、実行で現在までにログに書き込まれた内容を確認できます。「状態」タブでは、ロボットの状態を確認できます(ある場合)。また、メイン・パネルの右側にある「サマリー」パネルでは、返された値数、生成されたAPI例外数、HTTPリクエスト数の統計、データの送受信量、および実行済のJavaScript指示数を含む、実行のサマリーを確認できます。

デバッグ・モードでの実行は、Design Studioの設計モードでの実行とは無関係であることを理解することが重要です。したがって、設計モードでの現在のステップおよび現在のロボット状態とは無関係に、デバッグ・モードには、独自の現在のステップおよび独自の現在のロボット状態があります。デバッグ・モードでは、現在のステップは、デバッグ・プロセスでまもなく実行されるか、または実行中のステップであり、現在のロボット状態は、そのステップに対する入力です。

デバッグは、  アイコンをクリックしていつでも停止できます。特定のイベントが発生したときにデバッグを停止することもできます。これは、停止時点パネルで行います。ここで、デバッグを停止するのは、値が返されたときか、API例外がレポートされたときか、ブレークポイントに到達したときかを選択できます。ロボットの実行の完了時には、デバッグは必ず停止します。

デバッグが停止すると、ロボット・エディタの下部にあるステータス・バーで停止理由を確認できます。ロボットの実行の完了前にデバッグが停止した場合は、「状態」タブで現在のロボットの状態を確認できます。「変数」、「ウィンドウ」、Cookieおよび「認証」サブタブには、Design Studioの状態ビューと同様に、ロボット状態が表示されます。API例外サブタブには、API例外がレポートされたために実行が停止した場合にAPI例外が表示されます。

ロボットの実行の完了前にデバッグが停止した場合は、  アイコンをクリックしてデバッグを再開できます。  アイコンをクリックして、デバッグを再起動することもできます。これによって、現在のデバッグ・プロセスが中止されて、デバッガがロボットの最初から新しいデバッグを開始する準

備が整います。現在のロボットがDesign Studioで別のロボットによって変更または置換された場合も、デバッガが自動的に再起動されます。

ロボットに入力変数がある場合は、それらの変数の入力値を「入力」パネルで編集でき、[Enter]を押すと、デバッグが新しい入力値を使用して再起動します。デバッグの実行中は入力値を編集できないため、入力値を変更する場合は、最初にデバッグを再起動する必要があります。

設計モードの現在場所からのデバッグ

Design Studioで  アイコンをクリックすると、設計モードの現在場所からデバッグを開始できます。これによって、ロボット・エディタがデバッグ・モードに切り替わり、設計モードの現在場所に可能なかぎり対応する場所に対してデバッグが実行されます。デバッグは、この位置に到達すると停止します。  アイコンをクリックすると、この場所からデバッグを続行できます。

この機能は、特定の分岐や特定のループの繰返しアクションのようなロボットの一部をデバッグする場合に便利です。

Design Studioでロボットのデバッグ中に  アイコンを押すと、その場所に対するデバッグの実行の前にデバッグを再起動する必要があり、これを許可するかを尋ねられます。

デバッグの場所から設計モードに戻る

デバッグがロボットのある場所で停止した場合は、デバッグ・モードで  アイコンをクリックして、設計モードの同じ場所に切り替えることができます。これによって、その場所を設計モードで詳細に調査して、その場所の周囲のステップを変更したり、ロボットの他の部分を変更できます。

設計モードに切り替えて、値が返された場所に移動することもできます。これを実行するには、入力/出力タブの「出力」パネルで値を選択して、タブの右下隅にある「移動」ボタンをクリックします。これは、値が適切に抽出されないためにその理由を調査する場合に便利です。

設計モードに切り替えて、API例外がレポートされた場所(つまり、エラーの発生場所)に移動することもできます。API例外タブまたは「状態」タブのAPI例外サブタブでAPI例外を表示したときに、位置コードの右側にある「移動」ボタンをクリックすると、API例外が生成された場所に移動できます。特定のエラーの右側にある「移動」ボタンをクリックして、エラーの発生場所に移動することもできます。これは、エラーの理由を検索して問題を修正する場合に便利です。

設計モードで目的を完了した場合は、デバッグを再開できます。ロボットを変更していない場合は、  アイコンを単にクリックできます。ロボットを変更した場合はデバッグが自動的に再起動されるため、直接再開できません。かわりに、  アイコンをクリックすると、設計モードの現在場所から新しいデバッグ・セッションを開始できます。

ブレークポイントの使用

デバッグ中にロボットの特定のステップでDesign Studioを停止するには、そのステップにブレークポイントを設定します。これを実行する最も簡単な方法は、ロボット・ビューで目的のステップを右クリックし、ポップアップ・メニューのブレークポイントの切替えを選択します。このステップにブレークポイントが小さな  アイコンで示されます。

停止時点パネルでブレークポイントで停止しないことを選択しないかぎり、デバッグ中にブレークポイントに到達したDesign Studioは停止します。デバッグを再開するには、  アイコンをクリックします。

ブレークポイントをステップから削除するには、ブレークポイントを右クリックして、ブレークポイントの切替えを選択します。1つ以上のステップを選択する場合は、ブレークポイントの削除を選択

すると、それらのステップのすべてのブレークポイントを削除できます。  アイコンをクリックして、ロボット内のすべてのブレークポイントを削除することもできます。

シングルステップ

デバッグ・モードでは、一度に1つのステップを実行できます。これは、シングルステップと呼ばれます。これは、実行を厳密に調査する場合に便利です。

シングルステップを実行できるのは、Design Studioが新しいデバッグを開始するとき、またはデバッグ中に停止したときです。次のステップを実行するには、  アイコンをクリックします。そのステップの実行が完了すると、実行は停止します。次に  アイコンを再度クリックすると、次のステップが実行されます(その後も同様)。任意のステップで  アイコンをクリックすると、通常の実行を再開することもできます。

ステップイン

グループ・ステップを使用して複数のステップを含むグループを作成した場合は、その他のステップでブレークポイントを作成するときと同様に、デバッグ時にそのグループ・ステップにブレークポイントを作成できます。ただし、グループが縮小されている場合にシングルステップを使用すると、グループがシングルステップとして処理され、グループ化されている各ステップで停止しません。グループが展開されている場合は、シングルステップの使用時にグループの各ステップで停止します。グループが縮小されている場合は、  アイコンのかわりに  を使用すると、デバッガがそのグループにステップインし、各ステップで停止します。グループから移動して、デバッガをこのグループ・ステップの後のステップに進める場合は、  アイコンを使用すると、ステップ・アウトできます。

Design Studioの設定

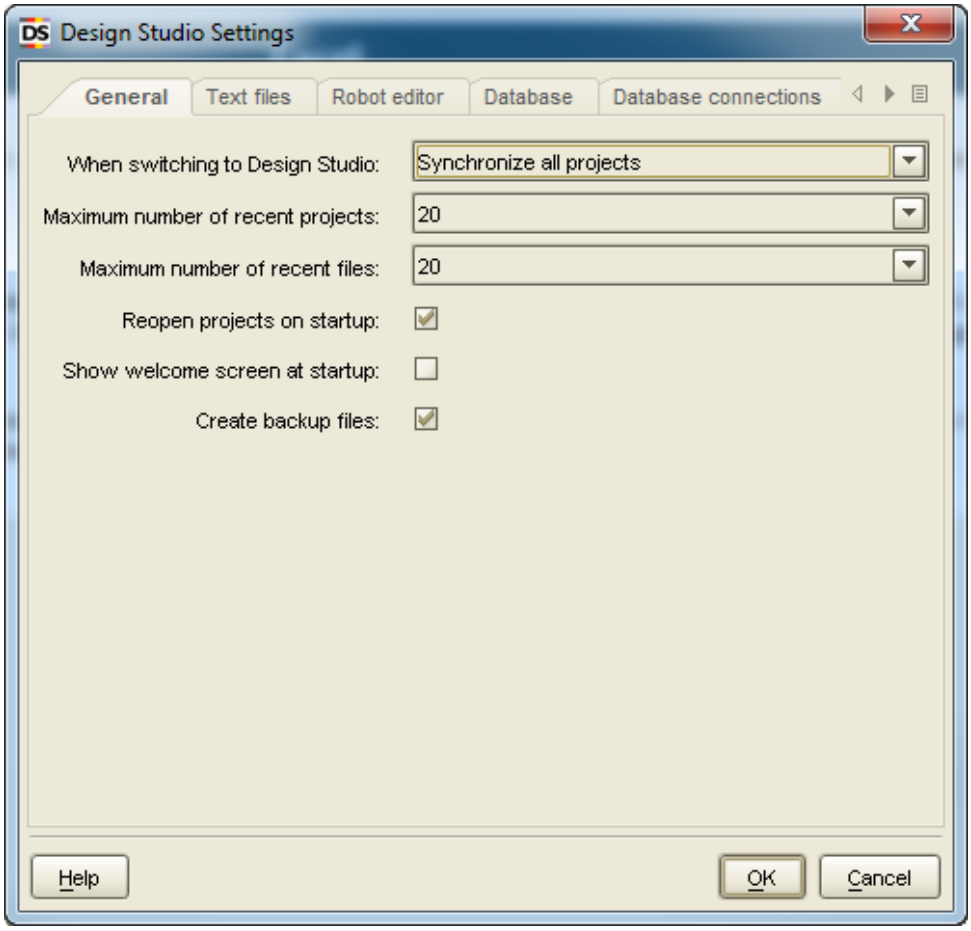
Design Studioの設定は、メニュー・バーの「オプション」にあるDesign Studioの設定ダイアログで変更できます。設定は、次にリストされている各セクションに分かれます。

- 「一般」設定
- テキスト・ファイル
- ロボット・エディタ
- データベース
- データベース接続
- プロキシ・サーバー
- 証明書
- 不具合レポート
- レガシー

このダイアログの異なるタブに表示されるこの各セクションについては、このユーザーズ・ガイドの次の各項で詳細に説明します。

一般

設定ダイアログの「一般」タブは、デフォルトではこのダイアログを開いたときに表示されます。このタブでは、Design Studioの一般的な設定の一部を変更できます。この項では、このタブに表示される各フィールドについて説明します。



Design Studioの「一般」設定

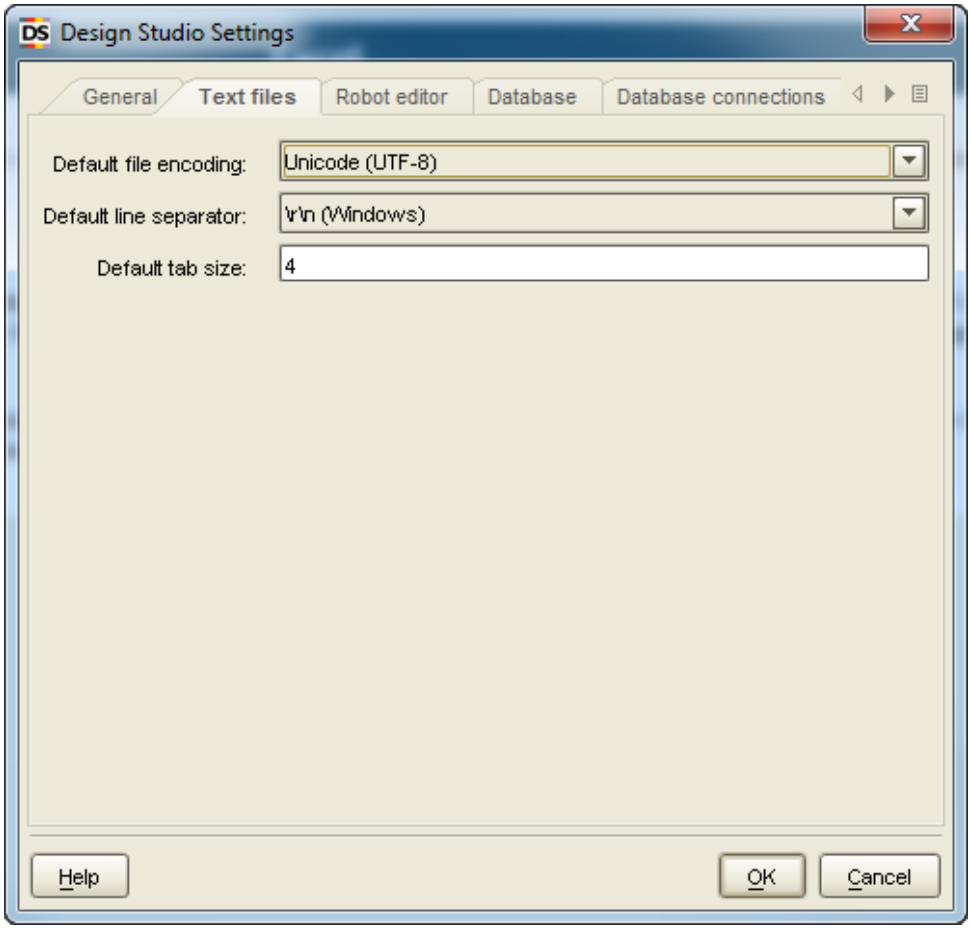
各フィールドには、デフォルト値が事前定義されています。ユーザーは、それぞれのデフォルト値をそのまま使用できますが、個々のニーズを満たすように変更することもできます。次の表では、各フィールドについて詳細に説明します。

表10.10 オプションの説明

オプション	説明
Design Studioへの切替時	Design Studioに切り替えたときに実行される内容を指定します。
最新プロジェクトの最大数	Design Studioでは、最新プロジェクトのローカル履歴を保持します。このオプションを使用すると、履歴に保持する最新プロジェクトの最大数を指定できます。最新プロジェクトは、「ファイル」->最新プロジェクトでアクセスできます。
最新ファイルの最大数	Design Studioでは、最新ファイルのローカル履歴を保持します。このオプションを使用すると、履歴に保持する最新ファイルの最大数を指定できます。最新ファイルは、「ファイル」->「最近使用したファイル」でアクセスできます。
起動時にプロジェクトを再度開く	このチェック・ボックスを選択すると、Design Studioを閉じるときに開いていたプロジェクトがDesign Studioの起動時に再度開かれます。
起動時にようこそ画面を表示	このチェック・ボックスを選択すると、Design Studioを起動するたびにようこそ画面が表示されます。
バックアップ・ファイルの作成	このチェック・ボックスを選択すると、保存済ファイルが変更されたときにDesign Studioによってバックアップ・ファイルが作成されます。バックアップ・ファイルは、末尾に~が付加されて表されます。

テキスト・ファイル

設定ダイアログの「テキスト・ファイル」タブでは、Design Studioのテキスト・ファイルの設定を変更できます。



Design Studioの「テキスト・ファイル」

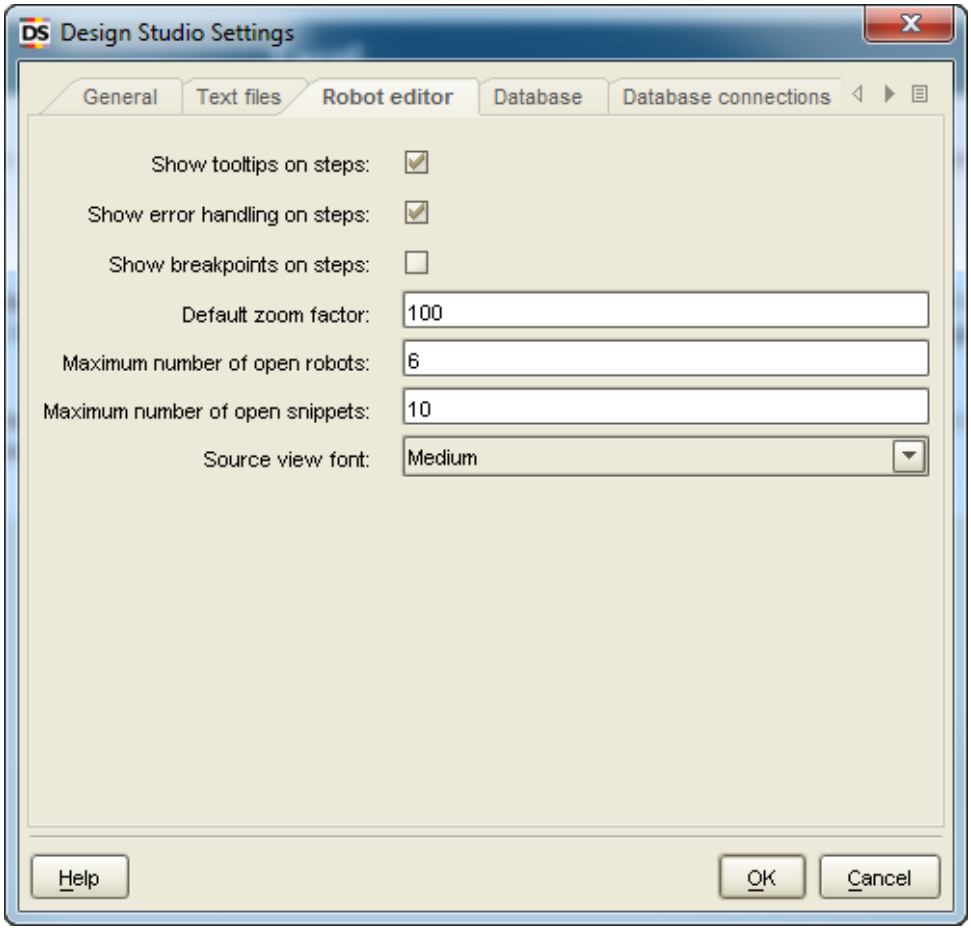
テキスト・ファイルに対するデフォルト設定は事前定義されていますが、必要に応じて変更できます。次の表では、各フィールドについて詳細に説明します。

表10.11 オプションの説明

オプション	説明
デフォルトのファイルのエンコーディング	テキスト・ファイルのデフォルトのエンコーディングを指定します。
デフォルトの行セパレータ	テキスト・ファイルのデフォルトの行セパレータを指定します。
デフォルトのタブ・サイズ	テキスト・ファイルのデフォルトのタブ・サイズを指定します。

ロボット・エディタ

Design Studioでは、複数のロボット・エディタ設定を変更できます。



Design Studioのロボット・エディタ設定

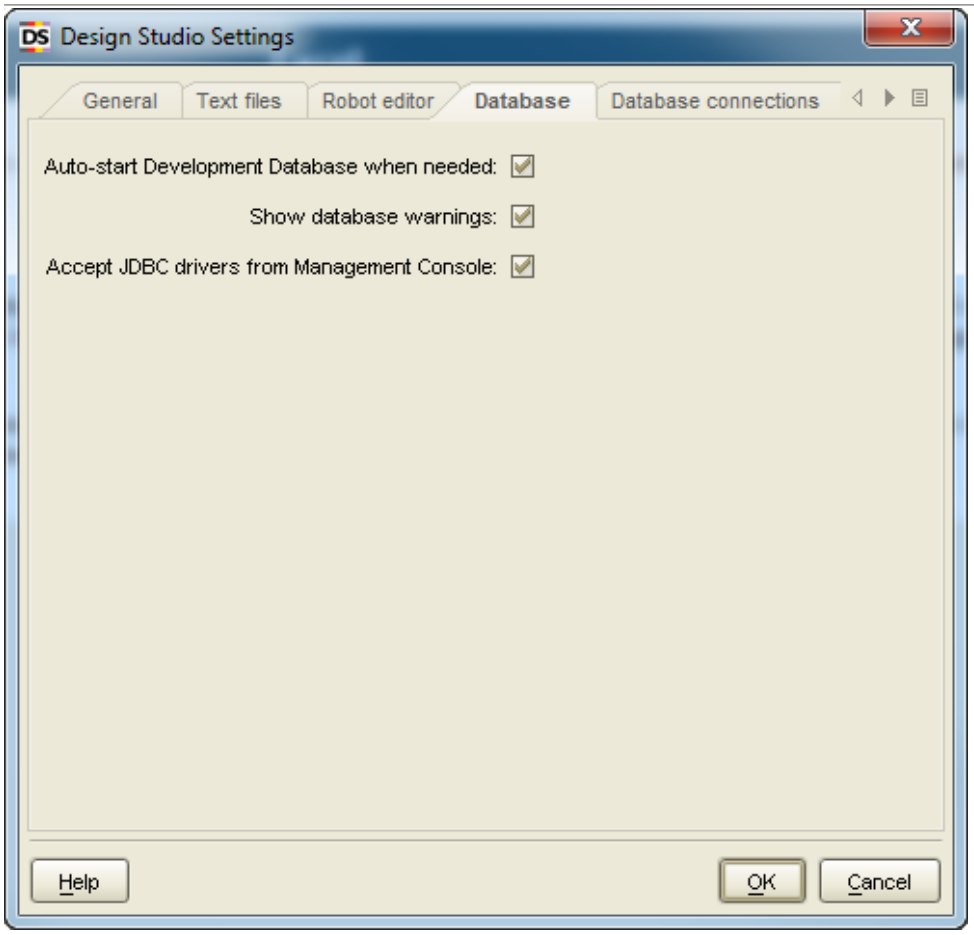
ロボット・エディタ設定には、デフォルトの値が事前定義されています。次の表では、各フィールドについて詳細に説明します。

表10.12 オプションの説明

オプション	説明
ステップでのツールチップの表示	このチェック・ボックスを選択すると、ロボットのステップに対するツールチップが表示されます。ステップのツールチップを無効にするには、このボックスの選択を解除します。
ステップでのエラー処理の表示	カスタム・エラー処理が指定されたロボットのステップは、記号でマークされて、カスタム・エラー処理がこのステップに適用されていることが視覚的に明確になります。
ステップでのブレークポイントの表示	ブレークポイントがロボットのステップに設定されている場合は、ステップが記号でマークされて、デバッガで実行する場合に実行がこのステップで停止することが視覚的に明確になります。
デフォルトのズーム・ファクタ	ロボット・エディタでロボットを開いたときに適用されるズーム・ファクタ。開いているロボットのズーム・ファクタを変更するには、ロボット・エディタの右下隅で手動で変更できます。
開くロボットの最大数	エディタに開くロボットの最大数。ロボットを最大数開いている場合に、さらにロボットを開こうとすると、閉じるロボットを選択するように求められます。
開くスニペットの最大数	エディタに開くスニペットの最大数。スニペットを最大数開いている場合に、さらにスニペットを開こうとすると、閉じるスニペットを選択するように求められます。

オプション	説明
ソース・ビューのフォント	ソース・ビュー(Design Studioの下部セクション)でのテキストのフォント・サイズを指定します。

データベース



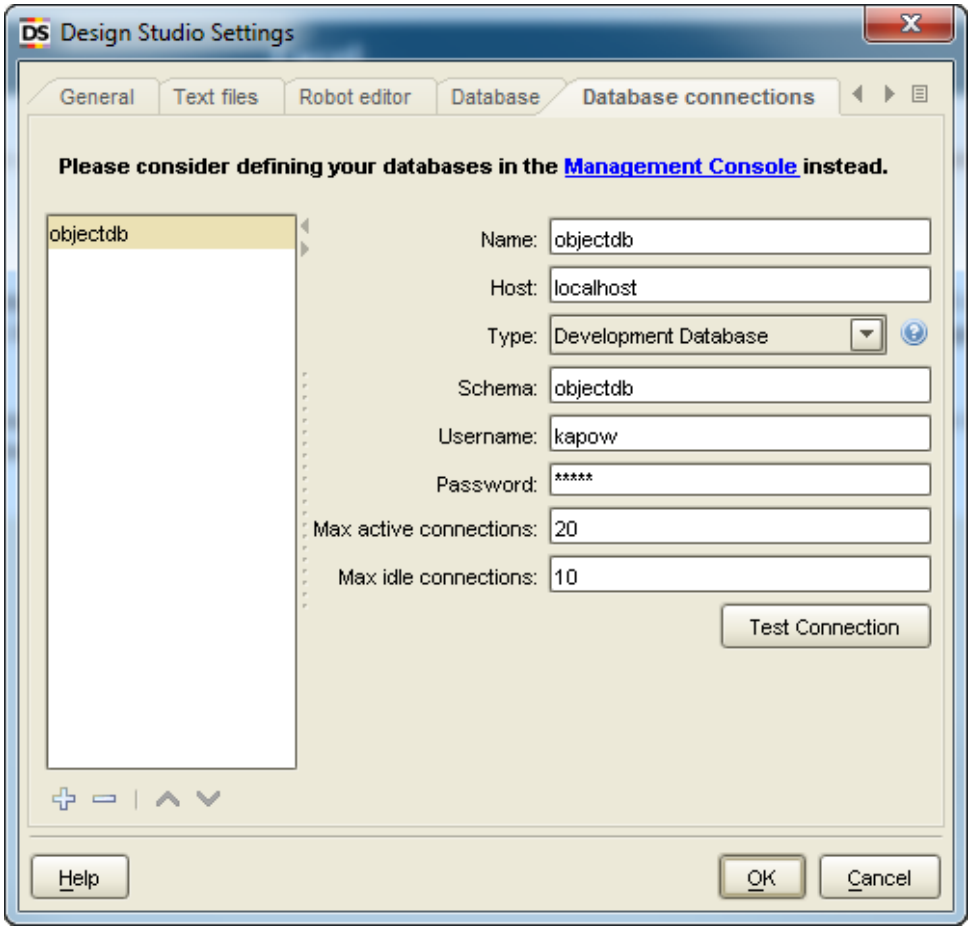
Design Studioの「データベース」の設定

表10.13 オプションの説明

オプション	説明
必要時に開発データベースを自動起動	このオプションを選択すると、開発データベースへのマッピングが存在する場合に開発データベースが自動的に起動します。
データベース警告の表示	このオプションを選択すると、表の欠落などのデータベース警告がロボット・エディタの上部に表示されます。
Management ConsoleからのJDBCドライバの受入れ	このオプションを選択すると、JDBCドライバがManagement ConsoleからDesign Studioに配布されます。このオプションを無効にする必要があるのはまれです。

データベース接続

この項では、Design Studioでのデータベースの作成方法について説明します。Design Studioで作成したデータベースはDesign Studioでのみ使用できることに注意してください。Design Studioとサーバーの両方でデータベースを使用可能にするには、Management Consoleでデータベースを構成する必要があります。



Design Studioでのデータベース接続の構成

ウィンドウの左側には、作成済の接続がリストされます。リストの下部にあるボタンを使用すると、新しい接続の作成、接続の削除または接続順序の変更を実行できます。現在選択している接続をウィンドウの右側で構成します。「名前」、「ホスト」、「タイプ」およびスキーマの各フィールドは必須で指定する必要があります。

様々なデータベース・タイプがManagement Consoleで定義されて、起動時にDesign Studioに自動的に配布されます。したがって、新しいデータベース・タイプは、Management Consoleで作成する必要があります。

次に、データベース・フィールドについて詳細に説明します。

表10.14 オプションの説明

オプション	必須	説明
名前	はい	Kapow Katalystでデータベースを一意に識別する名前。この名前はデータベースの内部参照に使用されて、英数字およびアンダースコアのみ使用できます。
ホスト	はい	データベース・サーバーのホスト名。これは、IPアドレスまたは完全修飾されたドメイン名(myhost.kapowtech.comなど)です。
タイプ	はい	データベース・タイプ(Oracleなど)。様々なタイプのデータベースがManagement Consoleで構成されて、Design Studioの起動時に自動的に提供されます。
スキーマ	はい	データベース・スキーマ(またはカタログ)の名前。
ユーザー名	いいえ	データベースのユーザー名。

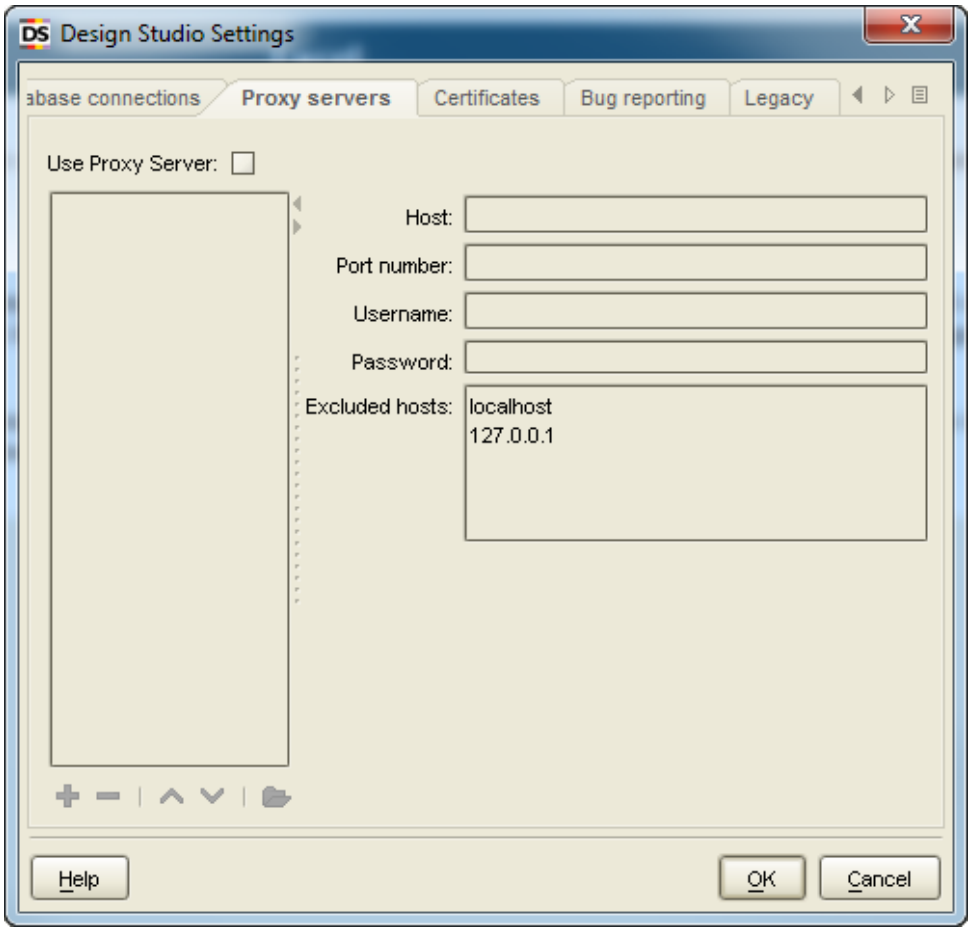
オプション	必須	説明
パスワード	いいえ	データベースのパスワード。
最大アクティブ接続数	はい	Kapow Katalyst (RoboServerまたはDesign Studio)がデータベースに対して作成する同時接続の最大数。接続は接続プールで管理され、これによって新しい接続の作成前に既存の接続が再利用されます。
最大アイドル接続数	はい	許可されるアイドル接続の最大数。接続の作成がこの数を超えると、過度の負荷のために、不要な接続が自動的に閉じられます。

現在の接続をテストするには、「接続のテスト」ボタンを押します。注意: これはデータベースに対する接続のテストのみを行い、そのデータベースに対して適切な権限があることはテストされません。

Oracleへの接続: Oracleデータベースを使用している場合は、「ユーザー名」フィールドにユーザー名およびロールを入力する必要があります。たとえば、ユーザー名が「sys」で、ロールが「sysdba」の場合は、「ユーザー名」フィールドに「sys as sysdba」と入力する必要があります。

プロキシ・サーバー

Design Studioの設定では、Design Studioで使用可能なプロキシ・サーバーを複数指定できます。



Design Studioのプロキシ・サーバーの構成

Design Studioの設定のプロキシ・サーバー・タブで、次のプロパティを使用してプロキシ・サーバーを指定します。

表10.15 オプションの説明

オプション	説明
プロキシ・サーバーの使用	プロキシ・サーバーの使用を有効にする場合に、このチェック・ボックスを選択します。
ホスト	プロキシ・サーバーのホスト名。これは、IPアドレスまたは完全修飾されたドメイン名(myproxy.kapowtech.comなど)です。
ポート番号	プロキシ・サーバーのポート番号。デフォルトのプロキシ・サーバー・ポートの8080を使用する場合は、空白のままにします。
ユーザー名	プロキシ・サーバーでログインする必要がある場合に使用するユーザー名。
パスワード	プロキシ・サーバーでログインする必要がある場合に使用するパスワード。
除外ホスト	ここでは、プロキシ・サーバーで使用しないホスト名のリストを指定できます。1行ごとに1つのホスト名を指定する必要があります。各ホスト名は、IPアドレスまたは完全修飾されたドメイン名(www.kapowtech.comなど)です。

プロキシ・サーバーのリストは、このリストの下部にあるファイルを開くボタンを使用してファイルからインポートすることもできます。このファイルでは、任意の数のプロキシ・サーバー定義を保持できますが、各定義は次の形式に準拠している必要があります。

```
proxyName.proxyServerUse = true proxyName.proxyServerHost
 = host name or IP address proxyName.proxyServerPort
 = port number proxyName.proxyServerUserName
 = user name proxyName.proxyServerPassword
 = password proxyName.proxyServerExcludedHostNames = list of hosts
```

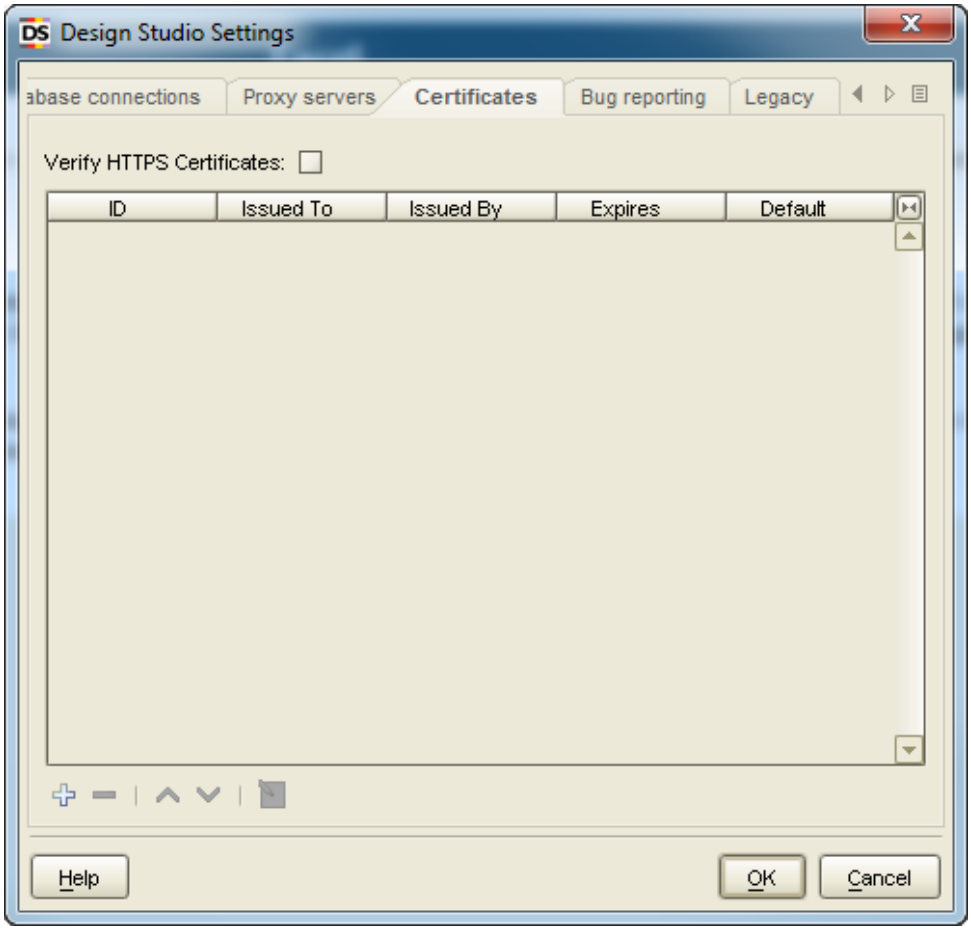
ここで、***proxyName***は、特定のプロキシ・サーバーを識別するために選択された名前です。各プロキシ・サーバーには、それぞれ一意の***proxyName***が指定されている必要があります。

複数のプロキシ・サーバーを指定すると、ロボットの実行のたびに新しいプロキシ・サーバーが選択されます。

個々のロボットに対してプロキシ・サーバーを指定することもできます。これは、Design Studioのロボット構成ウィンドウでロボットを構成するときに行います。このようなプロキシ・サーバーは、ここに指定したプロキシ・サーバーを上書きします。詳細は、「Design Studioユーザーズ・ガイド」を参照してください。さらに、プロキシ・サーバーは、ロボットの実行時にプロキシの変更アクションを使用して変更できます。

証明書

ロボットは、アクセス先のWebサーバーのアイデンティティを(HTTPS経由で)検証する必要がある場合があります。このような検証は、フィッシング攻撃を検出するために、通常のブラウザによって常に(およびわからない方法で)行われています。ただし、目的のWebサイト専用に作成されたロボットはそれらのWebサイトのみにアクセスするため、ロボットによる情報の収集時には通常、検証は必要ありません。したがって、検証はデフォルトでは有効化されていません。



Design Studioの証明書

検証はブラウザと同じ方法で行われます。Webサーバーの証明書は、ブラウザで構成可能なものと同じ信頼できるHTTPS証明書のインストール済セットに基づいて確認されます。

表10.16 オプションの説明

オプション	説明
HTTPS証明書の検証	ロボットがHTTPS経由でWebサイトにアクセスする場合は、サイトの証明書を検証します(このオプションが選択されている場合)。検証は、信頼できる証明書の2つのセットである、ルート証明書のセットと追加のサーバー証明書のセットに基づいて行われます。
HTTPSクライアント証明書	ロボットで使用できるクライアント証明書のリスト。証明書は、リストの下にあるボタンを使用して追加/削除できます。

ルート証明書は、ブラウザとともにインストールされる場合と同様に、Design Studioとともにインストールされることに注意してください。これらは、アプリケーション・データ・フォルダの**Certificates/Root**フォルダにあります。

一部のHTTPSサイトでは、デフォルトで組み込まれていない認証局を使用している場合があります。この場合、Design Studioでこれらのサイトからロードするには、適切な証明書をインストールする必要があります。多くの場合、これらは、アプリケーション・データ・フォルダの**Certificates/Server**フォルダにインストールされています。

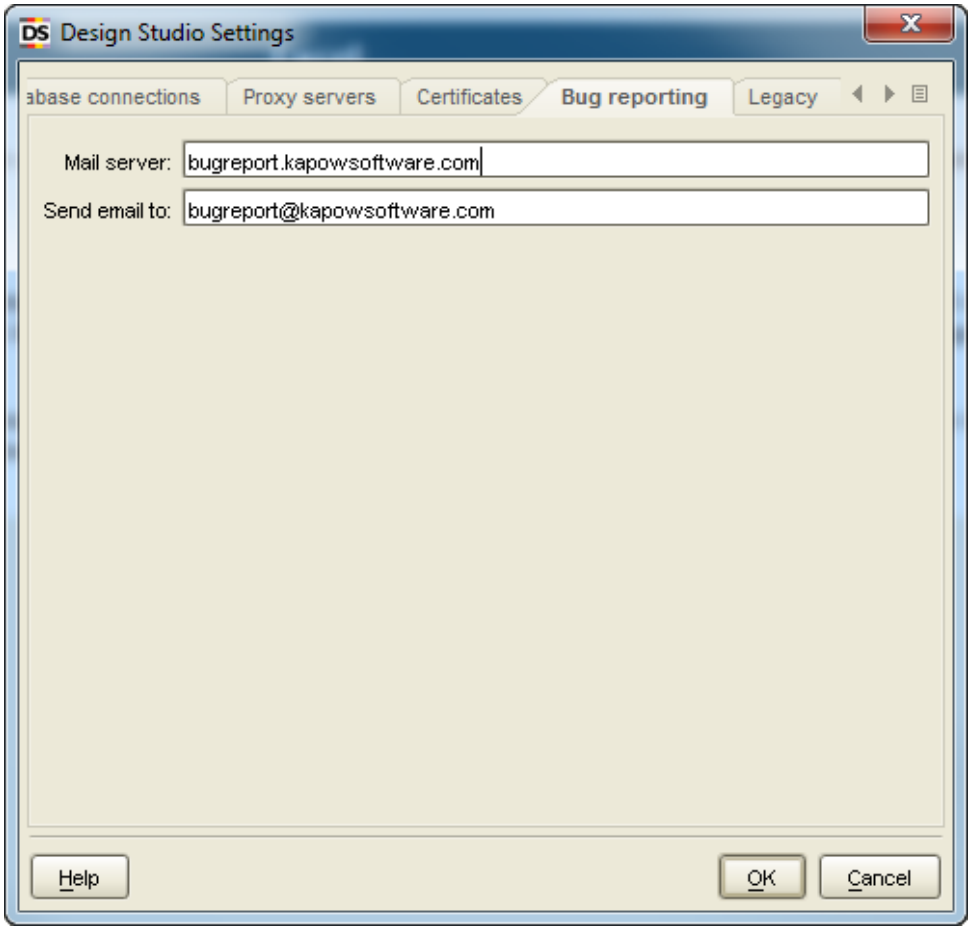
実際には、HTTPSサイトの処理が目的の場合、証明書をルート証明書のセットとサーバー証明書のセットのいずれに追加するかは問題ではありません。

証明書をインストールするには、PKCS#7証明書チェーン、Netscape証明書チェーンまたはDERでエンコードされた証明書として証明書を取得する必要があります。証明書をインストールするには、前述

の2つのフォルダのいずれかに証明書をコピーします。証明書を格納するファイルの名前は問題ではありません。

不具合レポート

Design Studioには、操作時に発生した内部エラーを処理する組込みメカニズムがあります。Design Studioで内部エラーが発生したとき、そのエラーの詳細が示された不具合レポートを送信できます。また、「ヘルプ」メニューから不具合レポート・アクションを選択すると、独自のイニシアティブに関する不具合レポートを送信できます。



Design Studioの不具合レポート

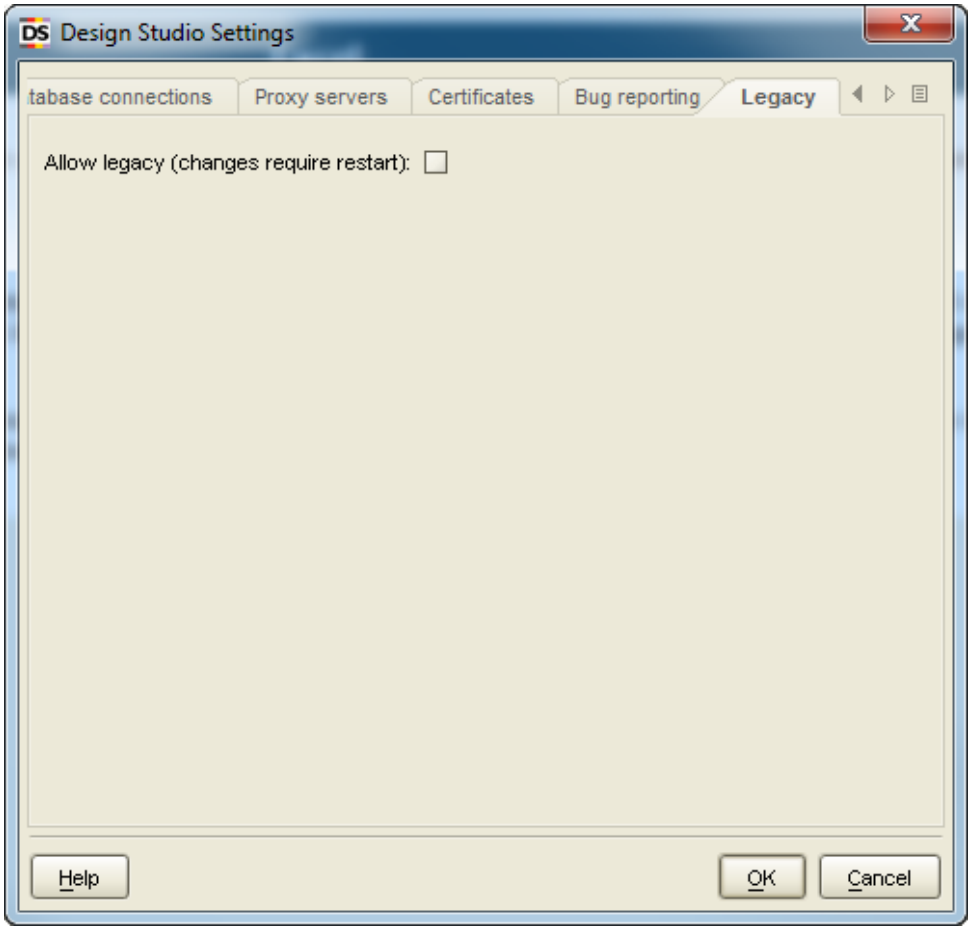
このタブのデフォルト設定は通常、絶対に必要な場合を除き、変更しないことをお薦めします。

表10.17 オプションの説明

オプション	説明
メール・サーバー	不具合レポートの送信時に使用するメール・サーバーを指定します。
電子メールの送信先	不具合レポートの送信先電子メール・アドレス。

レガシー

レガシー・タブでは、Design Studioのレガシー機能に関する設定を変更できます。



Design Studioのレガシー設定

レガシー機能を使用できるのは、Design Studioの以前のバージョンでこの機能を使用して、Design Studioを継続して正常に使用するためにこの機能に依存しているユーザーのみです。

表10.18 オプションの説明

オプション	説明
レガシー・プロパティの表示	環境のようなレガシー機能を表示するかどうかを指定します(バージョン7.2より前のDesign Studioのバージョンでこれらの機能を使用していたユーザーのみが使用可能)。このオプションを変更した場合は、Design Studioを再起動する必要があることに注意してください。

第11章 Management Consoleユーザーズ・ガイド

Management Consoleは、Kapow Katalystプラットフォーム・サーバーの監視および管理をポイント・アンド・クリックで実行できるWebベースのインタフェースです。

Management Consoleでは、次のことができます。

· リポジトリを使用したコラボレーションおよび共有が可能です。

· ユーザー・ロールと権限を管理して、ソリューションを集中的に管理します。

· リポジトリからロボットをスケジュールします。

· 抽出したデータを参照したり、MS Excelにエクスポートします。

· 本番結果とエラーに関する詳細ログにアクセスします。

· RoboServerの状態やリソースの使用状況をグラフィカルなダッシュボードで監視します。

· 複数のRoboServerのクラスタを構成します。

· Design Studioからリポジトリにロボットをデプロイします。

· Kappletsを実行します。

このユーザーズ・ガイドでは、Management Consoleで使用される概念およびユーザー・インタフェースについて説明します。詳細に入る前に、「Management Console初心者用チュートリアル」を参照できます。

Management Console構造の概要

Management Consoleは、5つの機能領域に分かれています。

ダッシュボード: ダッシュボードには、Management Consoleの概要が表示されます。この情報は、ポートレットを使用して提示され、RoboServer、スケジュールおよびロボットの状態が表示されます。

Kapplets: Kappletは、プログラミングなしにロボットを実行できるWebベースのユーザー・インタフェースです。

スケジュール: スケジュールは通常、1つ以上のロボットを、事前に計画済のある時点で実行したり、繰り返して実行するためのプランです。スケジュールではロボット自体は実行されず、ロボットの実行時期に関するプランが提供されるのみです(実行は、ロボットを構成済サーバーに渡すことによって行われます)。

リポジトリ: ロボット、タイプ定義およびリソースをDesign StudioからManagement Consoleリポジトリにアップロードしたり、Management ConsoleのWebインタフェースを使用して手動でアップロードできます。アップロードされたロボットは、スケジュールの一部としてまたはクライアント・コードを使用して実行できます(クライアント・コードは、KapowのJavaまたはC# APIを使用してロボットを実行します)。APIを使用して、リポジトリをプログラムで問い合せたり、更新することもできます。

データ: データ・ビューでは、ロボットがデータベースに格納したデータを表示できます。データ・ビューでは、このデータをExcelまたはXMLにエクスポートすることもできます。

ログ: ここでは、スケジュールの実行履歴を表示できます。データベース・ロギングが有効な場合は、すべてのロボットの実行詳細が含まれたRoboServerログを表示することもできます。

管理: 「管理」セクションでは、Management Consoleの設定を構成できます。RoboServerのクラスタとその設定を管理したり、プロジェクトおよび権限を管理することもできます。また、ここではライセンスを構成したり、バックアップを作成/復元します。

注意: 高可用性などの一部の機能は、ライセンス・キーに応じて使用できない場合があります。

Management Consoleの起動

Management Consoleコンポーネントは、RoboServerのオプション部分です。Management Consoleは、RoboServerを起動してManagement Consoleとして機能するように指示することで起動しますが、RoboServer機能も引き続き実行できます。

Management Console機能を備えたRoboServerは、「スタート」メニューのManagement Consoleの起動項目から起動できます(Windowsの場合)。LinuxおよびUnixバリアントでは、コマンドラインを使用できます。

```
RoboServer -MC
```

このコマンドでは、RoboServerはManagement Consoleとしてのみ起動するため、ロボットは実行できません。Management ConsoleのWebインタフェースにアクセスするには、Settingsで構成されているポートに接続します(「埋込みManagement Consoleの構成」を参照)。この方法でManagement Consoleを起動する場合は、Control CenterまたはShutDownRoboServerプログラムを介して再起動またはシャットダウンできません。

Management Consoleには、次のコマンドを使用して起動することもできます。

```
RoboServer -p 50000 -MC
```

これによって、ポート50000のソケットをリスニングするRoboServerが起動され、構成されたポートのWebインタフェースを介してManagement Console機能が提供されます(このポートはSettingsで構成します(「埋込みManagement Consoleの構成」を参照))。

起動方法にかかわらず、Management ConsoleがRoboServerおよびDesign Studioインスタンスのライセンス・サーバーとして機能するには、最初にライセンス・キーをWebインタフェースに入力する必要があります。

本番シナリオでは、Management Consoleを備えたRoboServerを起動しますが、このRoboServerはクラスタに追加しないことをお薦めします。この方法によって、RoboServerによるすべてのリソースの使用が原因でManagement Consoleのリソースが不足するのを防ぎます。Management Consoleを実行するRoboServerの設定後は、パラメータの**-p 50000**を使用して必要な数のRoboServerを起動できます。「クラスタ」タブのRoboServerクラスタに追加するのは、これらのRoboServerのみです。

RoboServerの起動方法の詳細は、「RoboServerの起動」を参照してください。

Management Consoleは、スタンドアロンWebアプリケーションとしてインストールすることもできます。スタンドアロン・モードのデプロイメントでは、プロジェクトベースの権限などの追加機能を使用できます。

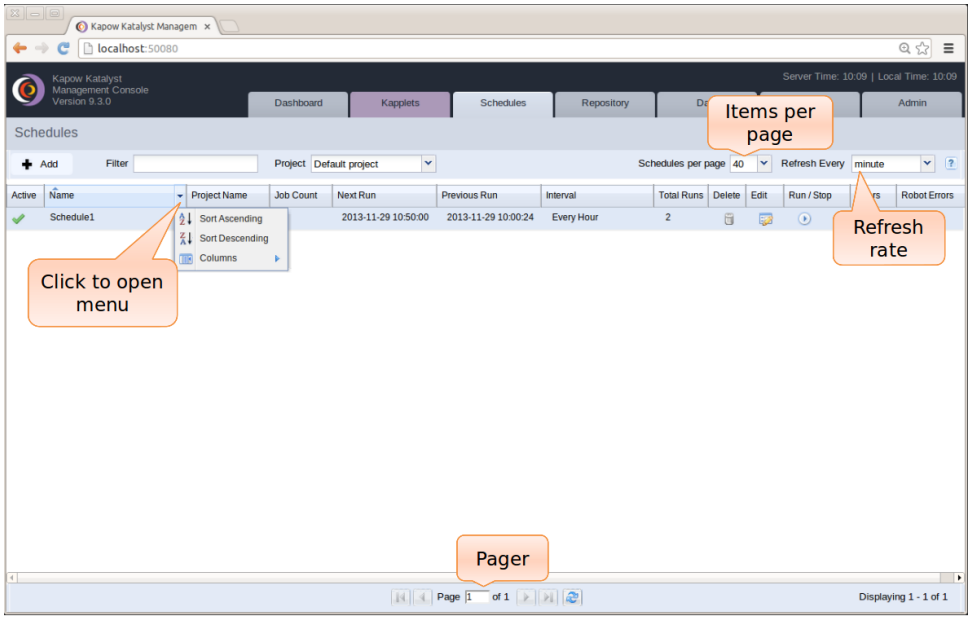
Management Consoleユーザー・インタフェース

Management ConsoleのWebベースのインタフェースは、同一ネットワーク上にある任意のマシンからのアクセスを容易にします。**hostname**というマシンにインストールした場合、必要なのはブラウザに次の内容を指定することのみです。

```
http://hostname:50080
```

ポート番号の**50080**は、「埋込みManagement Consoleの構成」の説明に従って構成できます。

Management Consoleのユーザー・インタフェースのナビゲートは、次に示すように、主にウィンドウの上部にあるタブを使用して実行します。



Management Consoleのメイン・ウィンドウ

Management Consoleのほとんどのサブセクションには、表内に特定タイプの項目が表示されます。表示する項目が多数ある場合は、複数のページに分割されます。前述のイメージに表示されている、1ページ当たりのスケジュール設定では、同時に表示する項目数(この場合は「スケジュール」)を選択できます。この数字は、ブラウザ・ウィンドウの高さに自動的に適用されないことに注意してください。したがって、表の下に空白があっても最後の項目を表示しているとはかぎりません。1ページ当たりの設定がウィンドウの高さと比較して少なすぎる可能性もあります。

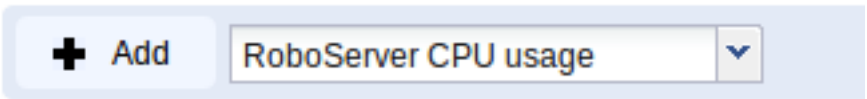
表の下部に表示される個々のページ間をナビゲートできるため、同一表内を上下に移動できます。 ボタンを使用すると、表示がリフレッシュされます。

この表ではソートがサポートされています。列の見出しをクリックすると、その列でソートされます。同じ列の見出しを再度クリックすると、ソート順序が元に戻ります。ソートは、個別の各ページに対してではなく、表全体に対して実行されます。したがって、複数のページがある場合は、ソート順序を変更すると、完全に異なる行のセットが表示される可能性があります。

各タブの内容については、次の各項で説明します。

ダッシュボード

ダッシュボードには、システムの様々な領域のステータスを表示する複数のポートレットが含まれています。ダッシュボードに初めてアクセスした際には4つのポートレットが含まれており、ダッシュボードに追加のポートレットを追加するには、ドロップダウンからポートレットを選択して「追加」ボタンをクリッします。



ダッシュボードへのポートレットの追加

大型画面を使用している場合は、ポートレットを2列ではなく3列で表示するように選択できます。各ポートレットには、4つのアイコンのボタン  を備えたツール・バーが含まれています。最初のボタンはポートレットを最小化し、2番目はポートレット・データをリフレッシュし、3

番目はポートレットに関する情報を表示し、4番目はポートレットを閉じます(再度追加できます)。ダッシュボードに対してポートレットを追加、削除または再配置すると、そのレイアウトがブラウザ内のCookieに格納されるため、次回そのダッシュボードにアクセスしたときに以前のレイアウトが復元されます。

ダッシュボードには、次のポートレットが含まれています。

エラーの多いロボット別 - 最新100の実行
: グラフには、エラーの多い10のロボット(抽出された値と比較)に対する最新100の実行が表示されます。グラフ上の地点をクリックすると、Management Consoleによってログ・ビューに切り替わり、この特定の実行に関する実行情報が表示されます。Y軸が対数であることに注意してください。このグラフが機能するには、少なくとも1つのクラスタでデータベース・ロギングを有効にしている必要があります。

過去24時間のサマリー
: Management Consoleの各プロジェクトのサマリーが表示されます。ビューでは、各スケジュールの最新20の実行を表示できます。状態バーには、各プロジェクトおよびスケジュールのステータスの概要が表示されます。状態バーには、赤、オレンジ、黄および緑の色が付きます。各色は、スケジュールの実行のステータスを示しています。

  赤　少なくとも1つのスケジュールにエラーが発生したことを示しています。スケジュール・エラーは、ロボットの実行時に発生したロボット以外のエラーです。

  オレンジ　このスケジュールで実行されたロボットの少なくとも1つでエラーが発生したことを示しています(スケジュールのエラーはありません)。

  黄　スケジュールはすでに実行済のために無視されたことを示しています。

  緑　スケジュールが正常に実行されて、スケジュール・エラーもロボット・エラーもありませんでした。

  状態バーの各色のサイズは、所定のステータスでのスケジュールの実行回数に関連します。

RoboServerのメモリー使用率
: 「クラスタ」タブにリストされている各RoboServerの一定期間(過去55時間)におけるメモリーの使用が表示されます。

RoboServerのCPU使用率
: RoboServerプロセスのCPU使用率が表示されます。複数のRoboServerが存在する場合は、それぞれにグラフが表示されます。

RoboServerのKCU使用率
: 各RoboServerのKCU使用率が表示されます。

RoboServerの待機時間
: KCUが使用可能になるまでにRoboServerが待機した時間。待機時間がある場合は、KCUを増やすライセンスを取得すると、ロボットのパフォーマンスを向上できます。

ロード済バイト数合計
: 「クラスタ」タブにリストされている各RoboServerにロードされたバイト数の合計(過去55時間)が表示されます。グラフは午前0時にリセットされます。

実行済ロボット数合計
: 「クラスタ」タブにリストされている各RoboServerで実行されたロボット数(過去55時間)が表示されます。グラフは午前0時にリセットされます。

同時ロボット
: 「クラスタ」タブにリストされている各RoboServerに対する同時ロボット数が表示されます。グラフは午前0時にリセットされます。

HTTPリクエスト数合計	「クラスタ」タブにリストされている各RoboServerでロードされたHTTPリクエスト数の合計が表示されます。グラフは午前0時にリセットされます。
拒否されたクリップ・セッション	すでに**-maxClippingSessions**のアクティブ・セッションがあるためにクリップ・セッションの作成を拒否した、各RoboServerの回数が表示されます。グラフは午前0時にリセットされます。このグラフが重要になるのは、クリッピング・ロボットを実行している場合のみです。すべてのサーバーがクリップ・セッションの作成を拒否した場合、新しいクリップは開始できないため、**-maxClippingSessions**コマンドライン・パラメータを増やす必要がある可能性があります(RoboServerに使用可能なメモリーが十分にある場合)。RoboServerパラメータについては、「RoboServerの起動」を参照してください。
プルーニング済クリップ・セッションの再起動	ユーザーによるクリップの終了前にクリップ・セッションが破棄された回数が表示されます。クリップ・セッションは、タイムアウト(ロボットで構成)によって、またはRoboServerが再起動されると破棄されます。クリップ・セッションが再起動すると、クリップは、以前の場所に関係なく最初のページに表示されます。再起動セッションが多数表示される場合は、ロボットのタイムアウトおよび-maxClippingSessions (「RoboServerの起動」を参照)もその数値を増やすことを検討してください。グラフは午前0時にリセットされます。

Kapplets

Kappletユーザーズ・ガイドは、ここを参照してください。

スケジュール

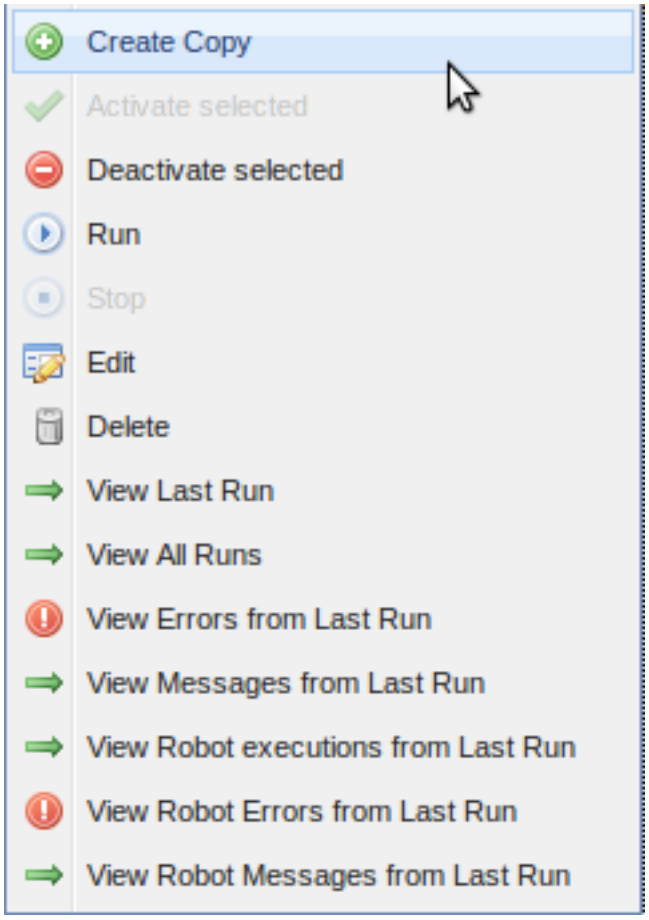
「スケジュール」サブセクションでは、このManagement Consoleのスケジュールを管理できます。スケジュールでは、ロボットおよびロボットの実行プランの選択肢が示されます。スケジュールの実行は、選択したロボットを(並列で、または順次に)実行し、オプションで実行前または実行後スクリプトまたはロボットを実行することを意味します。Fスケジュールのリストには、スケジュールごとに次の情報が表示されます。情報は各列に表示され、一部の列はデフォルトでは非表示ですが、列の見出しの下矢印をクリックして目的の列を選択すると表示できます。

表11.1 スケジュール情報

列	説明
アクティブ	計画済のスケジュールを実際に実行するかどうか。いくつかの理由でスケジュールを非アクティブにする場合があります。一部の例を次に示します。 · スケジュールで実行される機能が現在不要なため。 · ロボットにエラーが検出されたため、これらのエラーを修正してからスケジュールを実行する必要があるため。 · スケジュールの実行を毎回手動でトリガーする必要があるため。これは、一部のロボットおよびスケジュールでは適切な場合があります(準備またはクリーン・アップのタスクなど)。
名前	スケジュールの名前。
プロジェクト名	スケジュールが属するプロジェクトの名前(「すべて」プロジェクトが選択されている場合に便利)

列	説明
ロボット件数	ロボット合計とアクティブなロボットの組合せ。すべてのロボットがアクティブな場合は、単にアクティブなロボット数がリストされ、3つの内の2つがアクティブな場合は、**2 (3)**とリストされます。
ロボット合計(デフォルトでは非表示)	スケジュール内のロボットの合計数。(ロボットを表示するには、後続の説明に従ってスケジュールを編集します。)
アクティブなロボット(デフォルトでは非表示)	スケジュール内のアクティブなロボット数。(ロボットを表示するには、後続の説明に従ってスケジュールを編集します。)
次の実行	スケジュールで計画されている次の実行時間。
前の実行	スケジュールが最後に実行された時間。
間隔	スケジュールでの2つの連続した実行間で計画済の間隔。
実行合計	スケジュールの実行時間数。
作成者(デフォルトでは非表示)	このスケジュールを作成したユーザーの名前。
変更者(デフォルトでは非表示)	このスケジュールを最後に変更したユーザーの名前。
オブジェクトDB(デフォルトでは非表示)	(オブジェクトDBはレガシー機能で、バージョン7.2より前にコレクション・ロボットで使用されていました。) このスケジュールのロボットによって抽出された値を格納するデータベースの名前。値が入力されると、データベース記憶域環境がロボットとともにRoboServerに渡されます。
RoboManager DB(デフォルトでは非表示)	(RoboManager DBはレガシー機能で、バージョン7.2より前にコレクション・ロボットで使用されていました。) このスケジュールを使用して実行されたロボットに関するログ情報データを格納するデータベースの名前。値が入力されると、データベース・ロボット情報環境およびデータベース・メッセージ環境がロボットとともにRoboServerに渡されます。
クラスタ(デフォルトでは非表示)	スケジュールが実行されるクラスタ。
削除	スケジュールを削除する場合に、このボタンをクリックします。
編集	スケジュールを編集する場合に、このボタンをクリックします。これは、特定のスケジュールの詳細を表示する場合にも便利です。行をダブルクリックして、スケジュールを編集することもできます。
実行/停止	スケジュールを手動で実行する場合にクリックします。これは、非アクティブのスケジュールに対して特に便利です。 スケジュールがすでに実行中の場合は、そのスケジュールが停止されます。つまり、そのスケジュールで実行中のすべてのロボットが可能かぎり迅速に停止されます。該当するすべてのロボットが停止されるまで、スケジュールは「実行中」と表示されます。
エラー	スケジュールが最後に実行された際のスケジュール・エラー数。スケジュール・エラーには、ロボット・エラーは含まれません。スケジュール・エラーは、削除されたクラスタや前後処理のエラーなど、スケジュールが実行されないエラーです。
ロボット・エラー	このスケジュールによって実行されたロボットで発生したロボット・エラー数。

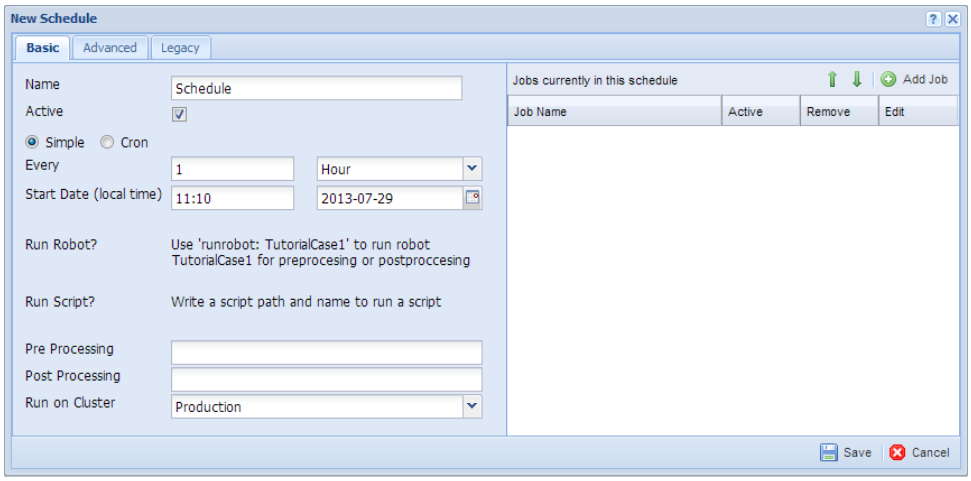
スケジュールを右クリックすると、次のポップアップ・メニューを表示できます。



スケジュールのコンテキスト・メニュー(右クリック)

左上隅にある「追加」ボタンをクリックすると、新しいスケジュールを作成するためのダイアログが開きます。現在のプロジェクトの選択が「すべて」の場合は、「追加」ボタンのクリック時に、実際のプロジェクトを選択してから新しいスケジュールを追加する必要があります。

「編集」列にある  ボタンをクリックしたり、既存スケジュールの行内の任意の場所をダブルクリックすると、編集ダイアログが表示されるため、スケジュールを変更したり、選択したスケジュールに関するすべての詳細を表示する際に便利です。



新しいスケジュールを作成するダイアログ
このダイアログには、「基本」、「拡張」およびレガシーの3つのタブが含まれています。「基本」タブには、通常のスケジュールの設定に必要なすべてが含まれています。「拡張」タブでは、ラン

タイム制約を構成できます。レガシー・タブにはいくつかの廃止オプションが含まれており、これによって、古いコレクション・ロボットを実行する場合の下位互換性が提供されます。

スケジュールに対して次の情報を構成できます。

表11.2 新規スケジュール/スケジュールの編集ダイアログの各フィールド

フィールド	説明
名前	スケジュールの名前。
アクティブ	アクティブなスケジュールには、チェック・マークが付きます。
簡易/Cron	スケジュールの時間のプランを定義する2つの異なる方法から選択する際に使用します。
間隔	(「簡易」スケジュールの場合のみ使用可能。) スケジュールでの2つの連続した実行間で必要な時間間隔。これは、1分や3時間などの整数の単位で入力します。
パターン	(Cronスケジュールの場合のみ使用可能。) スケジュールの実行時期を定義するパターン。形式の詳細は、「Cronスケジュール」を参照してください。
前処理	スケジュールの一部として、他のロボットの実行前に実行されるスクリプトまたはロボットの名前。 · スクリプトを実行するには、最初にRoboServer Settingsで、ファイル・システム・アクセスおよびコマンドライン・アクセスの許可を有効にする必要があります。これらの設定は、インストール・フォルダにあるか、またはWindowsの「スタート」メニューから検出できます。スクリプト・ファイルは、**.cmd**ファイル(Windowsの場合)または**.s**hファイル(Linuxの場合)です。フィールドには、**c:\scripts\truncatedb.cmd**のようにファイルの絶対位置を指定する必要があります。これを使用する場合、プロジェクトのインポート時またはバックアップの復元時には、スクリプトを忘れずにコピーする必要があります。 · ロボットの場合は、**runrobot: TutorialCase1** などを使用して、リポジトリからTutorialCase1ロボットを実行します。
後処理	スケジュールの一部として、他のロボットの実行後に実行されるスクリプトまたはロボットの名前。詳細は、前述の「前処理」を参照してください。
クラスタでの実行	このスケジュールを実行するクラスタの名前。
ロボット(右側)	後続の表を参照してください。
最大ランタイム(「拡張」タブ)	スケジュールの各ロボットの最大実行時間を選択します。ロボットの実行がこの時間を経過すると、サーバーは停止してエラーがログに記録されます。
最大抽出値(「拡張」タブ)	各ロボットが出力できる値の最大数を選択します。ロボットによる出力がこの値を超えると、サーバーは停止してエラーがログに記録されます。
ロボットの順次実行(「拡張」タブ)	選択すると、ロボットは「基本」タブにリストされている順序で実行されます。
RoboManagerデータベースの使用(レガシー・タブ)	このスケジュールを使用して実行されたロボットに関する履歴データを収集する場合に選択(およびデータベースを指定)します。詳細は、前述の表を参照してください。
オブジェクト・データベースの使用(レガシー・タブ)	このスケジュールのロボットによって抽出されたオブジェクトを格納する場合に選択(およびデータベースを指定)します。詳細は、前述の表を参照してください。

フィールド	説明
電子メール通知の使用（レガシー・タブ）	ロボットの失敗時に電子メールを受信する場合に選択します。スケジュール内の複数のロボットが失敗した場合は、スケジュールの実行のたびに各ロボットに対して1つの電子メールを受信します。これによって、電子メール・メッセージ環境がロボットとともにRoboServerに渡されます。 電子メール通知が機能するのは、「オプション」タブでSMTPサーバーを構成して、次のフィールドに必要な電子メール・アドレスを入力している場合です。
電子メール・アドレス（レガシー・タブ）	通知の送信先電子メール・アドレスのカンマ区切りのリスト。

右側には、スケジュールのトリガー時に実行されるジョブのリストが表示されます。

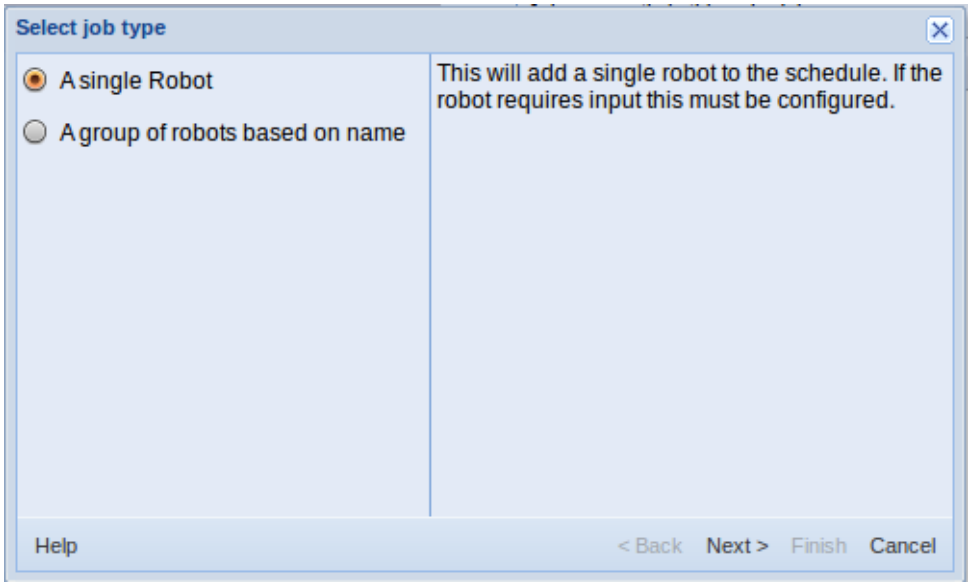
表11.3 スケジュールのロボット情報

列	説明
ジョブ名	ジョブの表示名。これは、ジョブの作成時に選択します。
アクティブ	スケジュールの実行時に実際にジョブを実行するかどうか。スケジュール全体を非アクティブにするときと同じ理由で、スケジュール内の単一ジョブを非アクティブにする場合があります。
削除	このスケジュールからジョブを削除する場合に、このボタンをクリックします。これによって、ジョブが参照していたロボットは削除されません。
編集	ジョブを編集する場合に、ここをクリックします。

スケジュールにジョブを追加するには、右上隅にある  ジョブの追加ボタンをクリックします。これによって、ジョブの作成手順を表示するウィザードが開きます。

ジョブの追加

 ジョブの追加をクリックすると、ジョブの作成をガイドするウィザードが開きます。次に最初のステップを示します。



ジョブ・タイプの選択

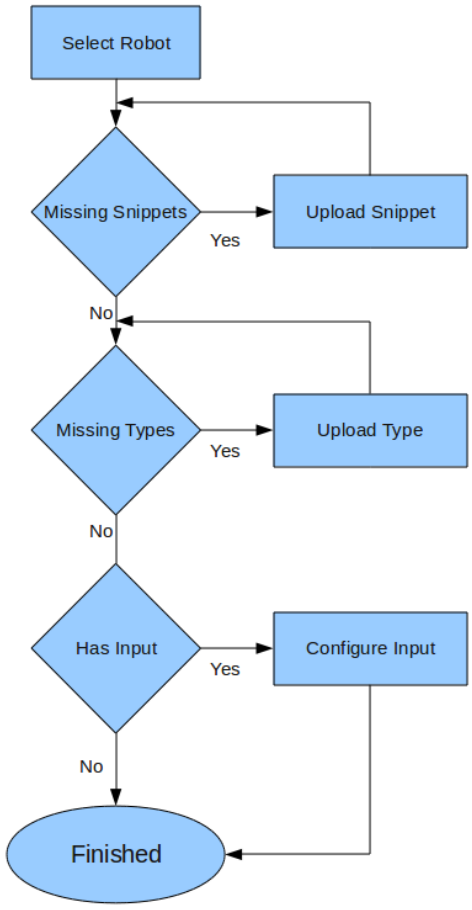
表11.4 ジョブ・タイプ

ジョブ・タイプ	説明
単一ロボット	これによって、単一のロボットを実行するジョブが追加されます。入力をロボットに渡す必要がある場合は、このオプションを選択する必要があります。
複数ロボット	これによって、名前が特定の基準に一致する複数のロボットを実行するジョブが追加されます。基準は、名前の開始文字、名前に次を含む、および名前がパターンと一致です。

ジョブ・タイプを選択すると、ウィザードによって、2つの異なる手順のいずれかが表示されます。

単一ロボットの作成

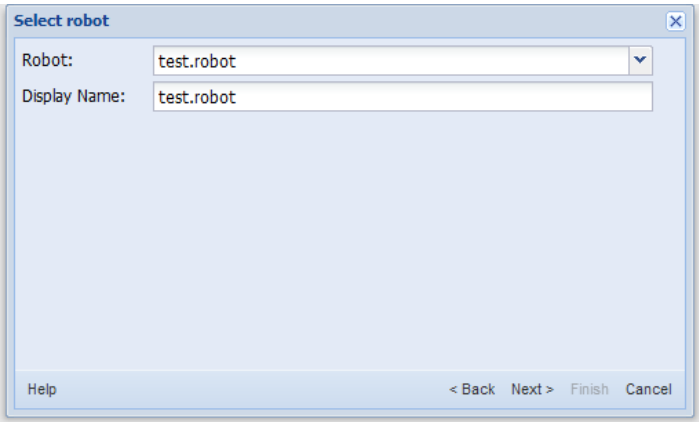
単一ロボットの追加ウィザードには、選択するロボットに基づいて1から4までのステップが含まれています。ウィザードは、次にリストされているフローに従います。4つの長方形は4つの可能性のあるステップを表します。



単一ロボットの選択のフロー

ロボットの選択: ドロップダウンを使用してロボットを選択します。

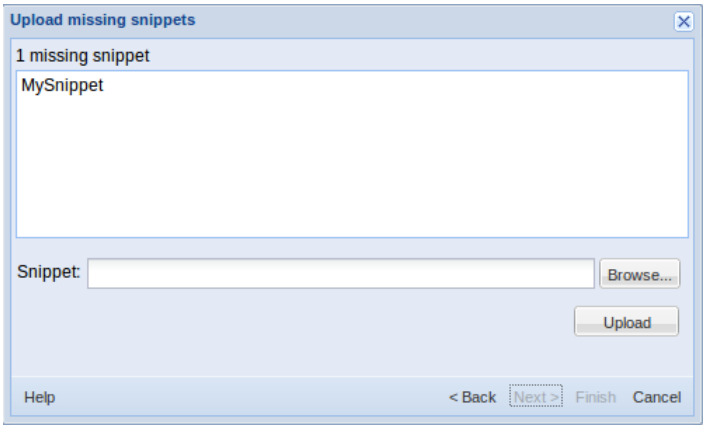
ロボットで使用されるすべてのスニペットおよびタイプがすでにアップロードされていて、ロボットに入力変数がない場合は、「終了」をクリックし、それ以外の場合は「次」をクリックする必要があります。



単一ロボットの選択

欠落しているスニペットのアップロード:

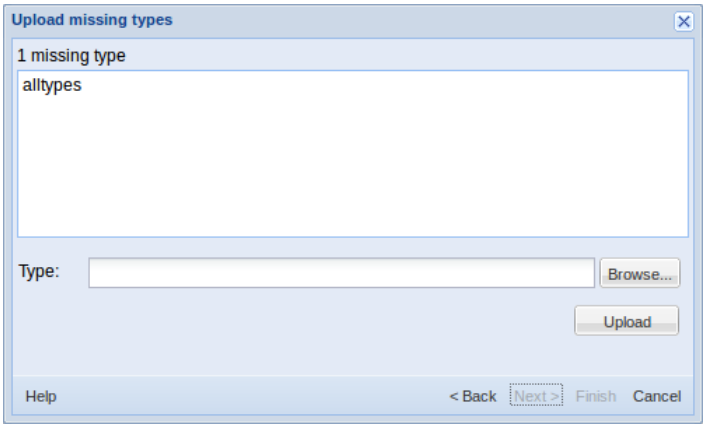
選択したロボットでManagement Consoleのリポジトリにアップロード済でないスニペットを使用する場合は、ここでそれらをアップロードする必要があります。



欠落しているスニペットのアップロード

欠落しているタイプのアップロード:

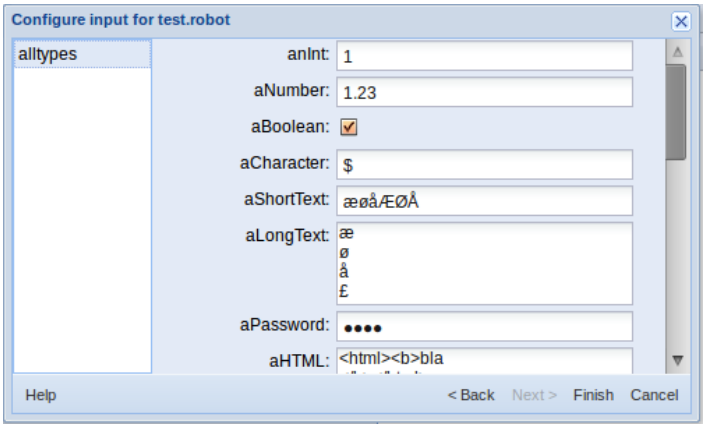
選択したロボットでManagement Consoleのリポジトリにアップロード済でないタイプを使用する場合は、ここでそれらをアップロードする必要があります。



欠落しているタイプのアップロード

入力の構成:

ここでは、このスケジュールの一部として実行するロボットに提供する必要がある入力を構成します。属性がバイナリ・タイプである場合は、ドロップダウンを使用してすでにアップロードされているリソースを選択するか、「アップロード」ボタンをクリックして属性をアップロードします。属性が必要な場合は、赤い下線が付きます。必須フィールドのいずれかが欠落している場合は、ウィザードを終了できません。



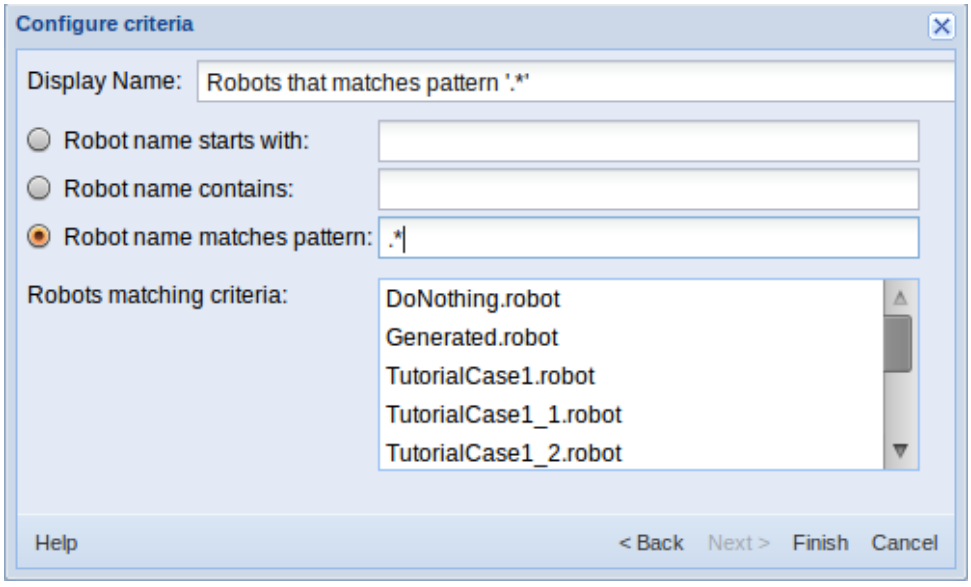
入力の構成

名前に基づいたロボットのグループの追加

名前が特定の基準に一致するすべてのロボットを実行する、単一ジョブを追加できます。基準はスケジュールの実行時に評価されるため、ジョブの作成後にアップロードされたロボットも、構成済の基準に一致する場合はすべて含まれます。

基準の評価は実行時のため、スケジュールの実行ログには、欠落しているスニペット/タイプなどのエラーがエラーとしてログに記録されます。基準に一致したために入力変数を使用するロボットが実行される場合も同様です。

名前の基準に基づいてグループを作成するウィザードには、単一のステップが含まれています。



名前に基づいてロボットを選択するジョブの作成

ラジオ・ボタンを使用して基準タイプを選択するか、またはテキスト・フィールドのいずれかへの入力を開始します。下部のリストには、選択した基準に一致するロボットが表示されます。「表示名」フィールドは、編集しないかぎり基準の選択に応じて変化しますが、編集を開始すると、自動命名が無効になります。一致するロボットは後でアップロードできるため、構成済の基準に一致するロボットがない場合も「終了」をクリックできます。

多数のロボットがある場合は、選択した基準に一致するロボットのリストをリフレッシュするのに時間を要する場合があります。

注意: このジョブ・タイプを、ジョブを順次に実行するように構成されたスケジュールに追加すると、基準に一致する複数のロボットのグループ内の順序を制御できません(ただし、順次実行されます)。

「スケジュール」タブの右上隅にある「ヘルプ」ボタンを使用すると、Management Consoleユーザーズ・ガイドがブラウザに開きます。

代替スケジュール作成

「ロボット」タブが表示されている場合も、スケジュールを作成できます。これを実行するには、任意の数のロボットを選択して右クリックし、コンテキスト・メニューからスケジュールの作成を選択します。これによって、ロボットが追加されている新規スケジュール・ダイアログが開きます。

リポジトリ

Management Consoleでは、ロボット、タイプ、スニペット、リソースおよびOAuth資格証明のリポジトリを保持します。このセクションは、このリポジトリを管理するのに役立ちます。

ロボット

このサブセクションは、リポジトリ内のロボットをプロジェクトごとに管理するのに役立ちます。スケジュールでロボットを実行できるようにするには、ロボットをリポジトリにアップロードする必要があります。ロボットをアップロードすると、そのロボットがリポジトリにコピーされます。したがって、後でDesign Studioでそのロボットに変更を加えた場合は、そのロボットを再度アップロードする必要があります(これは、Design Studio内から簡単に実行できます)。これを実行しても、スケジュールにすでに関連付けられているロボットはスケジュールから削除されません。かわりに、これらのスケジュールでは、次回の実行時に新しいバージョンのロボットが使用されます。

各ロボットは1つのプロジェクトに属します。「ロボット」タブの上部でプロジェクトを選択すると、そのプロジェクトのロボットを表示できます。

リポジトリでは、特定の1つのプロジェクト内に同じ名前の2つの異なるロボットを保持できません。これらは同じロボットとみなされ、最後にアップロードしたロボットによって以前のロボットが上書きされます。ただし、2つの異なるプロジェクトには、同じ名前のロボットを含めることができます。

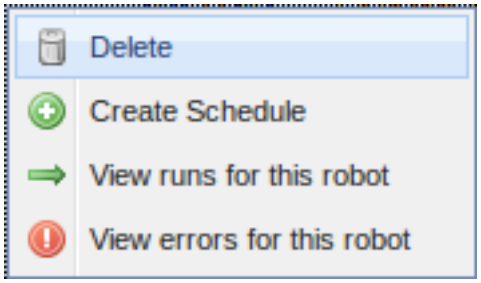
ロボットは表に表示され、デフォルトでは各ページに40のロボットが表示されます。情報は複数の列で構成されます。

表11.5 ロボットの情報

列	説明
名前	ロボットの名前。ロボットが使用するタイプまたはスニペットがリポジトリに存在しない場合は、その名前が赤でマークされます。
プロジェクト名	ロボットが属するプロジェクトの名前(「すべて」プロジェクトが表示されている場合に便利)
バージョン	ロボットの編集時に最後に使用されたKapow Katalystバージョン。
サイズ	ロボットのサイズ(バイト数)。
スケジュール	ロボットを実行するスケジュールの名前。
削除	リポジトリからロボットを削除する場合に、このボタンをクリックします。ロボットの実行に使用されたスケジュールから、そのロボットが自動的に削除されます。ファイル・システムにそのロボットのコピーが存在しない場合、そのロボットは完全に失われます。
入力タイプ	ロボットの入力変数として使用されるタイプ。ロボットを実行するには、これらの各タイプに対応するタイプ・ファイルが存在している必要があります。
返されるタイプ	ロボットによって返される値のタイプ。APIを介したロボットの実行時には、これらのタイプの値が返される可能性があります。ロボットを実行するには、これらの各タイプに対応するタイプ・ファイルが存在している必要があります。

列	説明
格納されるタイプ	ロボットによってデータベースに格納される値のタイプ。ロボットを実行するには、これらの各タイプに対応するタイプ・ファイルが存在している必要があります。
使用されるスニペット	このロボットによって使用されるスニペットの名前。ロボットがスニペットAを使用し、スニペットAがスニペットBを使用する場合は、スニペットAのみがここにリストされます。
作成者(デフォルトでは非表示)	このロボットを最初にアップロードしたユーザーのユーザー名。この機能を使用できるのは、スタンドアロンのManagement Consoleを実行している場合のみです。
変更者(デフォルトでは非表示)	このロボットを最後に変更したユーザーのユーザー名。この機能を使用できるのは、スタンドアロンのKapow Katalystを実行している場合のみです。
最終変更	ロボットの最新の変更日。
今すぐ実行	RoboServerでロボットの即時実行を開始する場合に、このボタンをクリックします。この機能は、入力を取得するロボットでは使用できません。
API	RoboServerでロボットを実行する際にJavaまたはC#コードの例を表示する場合に、このボタンをクリックします。
REST	これによって、ロボットをRESTサービスとして起動できるウィンドウが開きます。
ダウンロード	リポジトリからロボットのコピーをダウンロードしてファイル・システムに保存する場合に、このボタンをクリックします。

ロボットを右クリックすると、次のポップアップ・メニューが表示されます。



「削除」およびスケジュールの作成オプションを使用できるのは、複数のロボットを選択したときです。複数のロボットを選択してスケジュールを作成する場合は、選択したすべてのロボットが追加された新規スケジュール・ダイアログが開きます。この方法で追加したロボットのいずれかに入力が必要な場合は、後で追加する必要があります。

左上隅にあるロボットの追加ボタンをクリックすると、新しいロボットをアップロードするためのダイアログが開きます。既存のロボットと同じ名前のロボットをアップロードすると、既存のロボットが置換されますが、ロボットを実行するスケジュールは変更されません。新しいロボットをアップロードしても、スケジュールには自動的に追加されないため、自分自身で追加する必要があります。

ロボットをアップロードする別の方法は、Design Studioのアップロード機能の1つを使用することです。これは、Design Studioでは必要なタイプおよびスニペットもアップロードすることを除いては同じ方法で機能します。Management Consoleのリポジトリに複数のプロジェクトが含まれている場合は、ロボットのアップロード先のプロジェクトを選択するように要求するプロンプトが表示されます。

ロボットの実行

Management Consoleにアップロードしたロボットは、4つの異なる方法で実行できます。ほとんどの場合、ロボットはスケジュールの一部として、またはKappletとして実行しますが、Java/.Net APIを介して、またはRESTfulサービスとしてプログラムで実行することもできます。

API

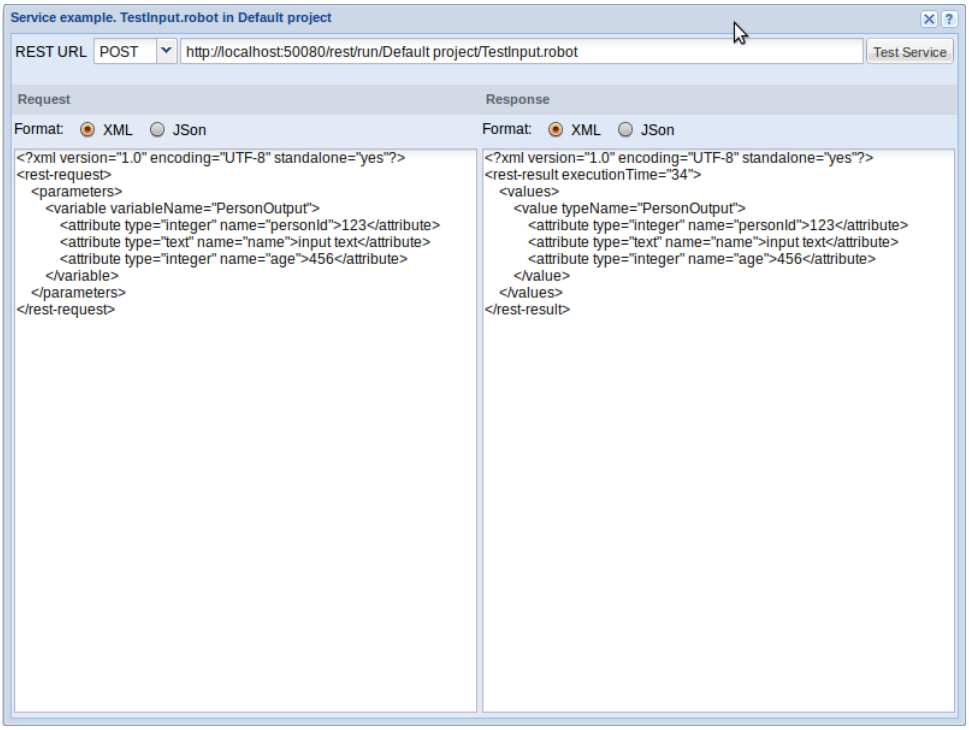
「ロボット」タブには、APIという列があります。この列をクリックすると、コード生成ウィンドウが表示されます。これによって、Javaまたは.Netのテンプレート・コードが生成されます。

APIを使用したロボットの実行を開始する前に、関連するプログラマーズ・ガイドの14章「Javaプログラマーズ・ガイド」および15章「.NETプログラマーズ・ガイド」を参照して、APIの仕組みを理解することをお薦めします。

REST

8.3の新機能として、ロボットをRESTfulサービスとして実行できます。これによって、プログラミング言語からロボットを起動したり、JavaScriptを使用してブラウザからロボットを直接起動できます。

「ロボット」タブには、RESTという列があります。この列内をクリックすると、ロボットをサービスとしてテストできるウィンドウが表示されます。



サービス・ウィンドウ

サービス・ウィンドウの左側では、リクエストを作成できます。次に、右上隅にある**Test Service**ボタンをクリックすると、ロボットを実行できます。結果がウィンドウの右側に表示されます。

「形式」ボタンを使用すると、テスト時にリクエストおよびレスポンスの形式を構成できますが、コードからサービスを呼び出す場合、形式はAcceptおよびContent-Type HTTPヘッダーによって制御されます。Content-Typeヘッダーではリクエスト形式が、Acceptヘッダーでは目的のレスポンス形式が指定されます。

入力が必要なロボットは、POSTを使用して起動する必要があります。入力なしのロボットは、GETまたはPOSTを使用して起動できます。

RESTサービスは、REST Webサービスの呼出しアクションを使用してロボットから簡単に起動できます。

プロジェクトまたはロボットの名前にASCII以外の文字が含まれている場合は、URLが適切にエンコード(UTF-8のURLエンコーディング)されていることを確認する必要がありますこれはロボットで自動的に実行されますが、コードからサービスを呼び出す場合、URLのエンコーディングは開発者の責任で行います。

注意: サービスとして実行されたロボットは、最初のAPI例外生成によって停止します。これは、ロボットがAPI例外を生成したかどうかに関係なく実行を続行するスケジュール済ロボットと異なる点です。RESTサービスは、Googleの検索や文章の翻訳のように期間の短いサービスと考える必要があります。

サービスとして実行される各ロボットでは、リクエスト・スレッドが使用されます。Management ConsoleがRoboServerに埋め込まれて実行されている場合は、最大40のリクエスト・スレッドがあります。ユーザーによるManagement Consoleへのアクセス、Design Studioからのアップロード、およびリポジトリAPIなど、すべてのタイプのHTTPリクエストに対してこれらの40スレッドが使用されます。実行する同時RESTサービス数を増やす必要がある場合は、リクエスト・スレッド数を制御できるように、スタンドアロン・バージョンのManagement ConsoleをTomcatにインストールする必要があります。

タイプ

このサブセクションには、Management Consoleのリポジトリにアップロードされているタイプがリストされます。「タイプ」タブの上部でプロジェクトを選択して、そのプロジェクトのタイプを表示します。

スケジュールが実行されると、リンクされているロボットがRoboServerで実行されます。多くのロボットには、入力または出力(あるいはその両方)の定義としてタイプが必要です。これらのタイプはリポジトリ内のロボットと同じプロジェクトで使用できる必要があり、使用できない場合は、ロボットの実行が失敗します。

タイプをリポジトリにアップロードすると、そのタイプがリポジトリにコピーされます。したがって、後でDesign Studioでそのタイプに変更を加えた場合は、そのタイプを再度アップロードする必要があります。各タイプ名は(各プロジェクト内で)一意である必要があるため、タイプを再度アップロードすると、前のバージョンが上書きされます。

同じ名前の複数のタイプは、別々のプロジェクトに配置する場合のみアップロードできます。

タイプごとに次の情報が表示されます。

表11.6 タイプの情報

列	説明
名前	タイプの名前。
サイズ	タイプのサイズ(バイト数)。
プロジェクト名	タイプが属するプロジェクトが表示されます(「すべて」プロジェクトが表示されている場合に便利)
削除	リポジトリからタイプを削除する場合に、このボタンをクリックします。ファイル・システムにそのタイプのコピーが存在しない場合、そのタイプは完全に失われます。
最終変更	このタイプの最後に変更に対するタイムスタンプ。
ダウンロード	リポジトリからタイプのコピーをダウンロードしてファイル・システムに保存する場合に、このボタンをクリックします。
作成者(デフォルトでは非表示)	このタイプを最初にアップロードしたユーザーのユーザー名。この機能を使用できるのは、LDAPを使用してスタンドアロンのKapow Katalystを実行している場合のみです。
変更者(デフォルトでは非表示)	このタイプを最後に変更したユーザーのユーザー名。この機能を使用できるのは、LDAPを使用してスタンドアロンのKapow Katalystを実行している場合のみです。

左上隅にあるタイプの追加ボタンをクリックすると、新しいタイプをアップロードするためのダイアログが開きます。タイプは、Design Studioによって暗黙的にアップロードされる場合があることに注意してください。これが実行されるのは、Design Studioを使用してこのタイプを使用するロボッ

トをアップロードした場合です。Design Studioではロボットとタイプ間の依存関係が認識されるため、ロボットとともに必要なタイプが常にアップロードされます。

「削除」列をクリックすると、削除を確認するプロンプトが表示されます。そのタイプがロボットまたはスニペットによって使用されている場合、確認メッセージには使用件数が含まれます。ロボットまたはスニペットによって使用されているタイプを削除すると、そのロボット(またはそのスニペットを使用する複数のロボット)は実行できなくなります。

スニペット

このサブセクションには、Management Consoleのリポジトリにアップロードされているスニペットがリストされます。スニペット・タブの上部でプロジェクトを選択して、そのプロジェクトのスニペットを表示します。

スケジュールが実行されると、リンクされているロボットがRoboServerで実行されます。一部のロボットはスニペットを使用しますが、これらのスニペットはリポジトリ内のロボットと同じプロジェクトで使用できる必要があり、使用できない場合は、ロボットの実行が失敗します。

スニペットをリポジトリにアップロードすると、そのスニペットがリポジトリにコピーされます。したがって、後でDesign Studioでそのスニペットに変更を加えた場合は、そのスニペットを再度アップロードする必要があります。各スニペット名は(各プロジェクト内で)一意である必要があるため、スニペットを再度アップロードすると、前のバージョンが上書きされます。

同じ名前の複数のスニペットは、別々のプロジェクトに配置する場合のみアップロードできます。

スニペットごとに次の情報が表示されます。

表11.7 スニペットの情報

列	説明
名前	スニペットの名前。
プロジェクト名	スニペットが属するプロジェクトが表示されます(「すべて」プロジェクトが表示されている場合に便利)
入力タイプ	スニペットの入力変数として使用されるタイプ。ロボットでこのスニペットを使用するには、これらの各タイプに対応するタイプ・ファイルが存在している必要があります。
返されるタイプ	スニペットによって返される値のタイプ。ロボットでこのスニペットを使用するには、これらの各タイプに対応するタイプ・ファイルが存在している必要があります。
格納されるタイプ	スニペットによってデータベースに格納される値のタイプ。ロボットでこのスニペットを使用するには、これらの各タイプに対応するタイプ・ファイルが存在している必要があります。
使用されるスニペット	このスニペットによって使用されるスニペットの名前。スニペットがスニペットAを使用し、スニペットAがスニペットBを使用する場合は、スニペットAのみがここにリストされます。
サイズ	スニペットのサイズ(バイト数)。
削除	リポジトリからスニペットを削除する場合に、このボタンをクリックします。ファイル・システムにそのスニペットのコピーが存在しない場合、そのスニペットは完全に失われます。
作成者(デフォルトでは非表示)	このスニペットを最初にアップロードしたユーザーのユーザー名。この機能を使用できるのは、LDAPを使用してスタンドアロンのKapow Katalystを実行している場合のみです。
変更者(デフォルトでは非表示)	このスニペットを最後に変更したユーザーのユーザー名。この機能を使用できるのは、LDAPを使用してスタンドアロンのKapow Katalystを実行している場合のみです。

列	説明
最終変更	このスニペットの最後の変更に対するタイムスタンプ。
ダウンロード	リポジトリからスニペットのコピーを取得してファイル・システムに保存する場合に、このボタンをクリックします。

左上隅にあるスニペットの追加ボタンをクリックすると、新しいスニペットをアップロードするためのダイアログが開きます。スニペットは、Design Studioによって暗黙的にアップロードされる場合があることに注意してください。これが実行されるのは、Design Studioを使用してこのスニペットを使用するロボットをアップロードした場合です。Design Studioではロボットとスニペット間の依存関係が認識されるため、ロボットとともに必要なスニペットが常にアップロードされます。

「削除」列をクリックすると、削除を確認するプロンプトが表示されます。そのスニペットが他のスニペットまたはロボットによって使用されている場合、確認メッセージには使用件数が含まれます。ロボット(またはロボットが使用しているスニペット)が使用しているスニペットを削除すると、そのロボットは実行できなくなります。「ロボット」タブでは、必要なスニペットが欠落しているロボットは、その名前が赤のフォントで表示されてリストされます。

リソース

このタブには、Management Consoleにアップロードされているリソースが表示されます。これらのリソースは、バイナリ属性の入力変数が設定されたスケジュール済ロボットに対する入力として使用できます。(ロボットへの変数などの追加方法およびロード方法については、「ロボットでローカル・ファイルを使用する方法」を参照。)

リソースごとに次の情報が表示されます。

表11.8 リソースの情報

列	説明
名前	リソースの名前。
サイズ	リソースのサイズ(バイト数)。
プロジェクト名	リソースが属するプロジェクトの名前(「すべて」プロジェクトが表示されている場合に便利)
削除	リポジトリからリソースを削除する場合に、このボタンをクリックします。ファイル・システムにそのリソースのコピーが存在しない場合、そのリソースは完全に失われます。
ダウンロード	リポジトリからリソースのコピーをダウンロードしてファイル・システムに保存する場合に、このボタンをクリックします。
作成者(デフォルトでは非表示)	このリソースを最初にアップロードしたユーザーのユーザー名。この列に関連情報が含まれるのは、個々のユーザー・ログインが設定されるスタンドアロン・バージョンのKapow Katalystを実行している場合のみです。
変更者(デフォルトでは非表示)	このリソースを最後に変更したユーザーのユーザー名。この列に関連情報が含まれるのは、個々のユーザー・ログインが設定されるスタンドアロン・バージョンのKapow Katalystを実行している場合のみです。

リソースをアップロードするには、タブの左上隅にあるリソースの追加ボタンをクリックして、スケジュール済ロボットに対する入力を構成するダイアログでアップロードするか、Design Studioから直接アップロードします。

OAuth

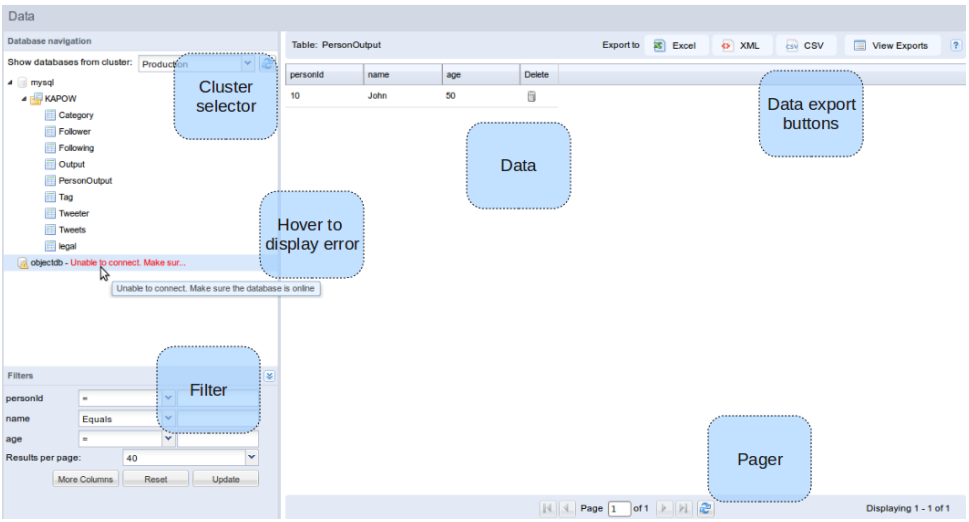
このタブには、Management Consoleを使用して認証されたOAuthアプリケーションおよびユーザーが含まれています。ユーザーの資格証明をスケジュールのロボットへの入力として使用できるため、ロ

ボットは、認証されたユーザーのかわりにユーザー名およびパスワードにアクセスせずにAPIにアクセスできます。

OAuthによって保護されたAPIにアクセスするロボットの作成方法および管理方法の詳細は、このドキュメントのOAuthの項を参照してください。

データ

データ・ビューでは、ロボットによって抽出されたデータを表示およびエクスポートできます。



データ・ビュー

データ・ビューの左上隅では、クラスタ・セレクタおよびデータベース・ナビゲーション・ツリーを表示できます。各クラスタのデータベースのデータを表示できます。特定のクラスタからデータを表示するには、クラスタを単に選択することで、データベース・ナビゲーション・ツリーには、そのクラスタの設定に定義されているデータベースが表示されます。クラスタ・セレクタの横には、クラスタのデータベースを変更した場合に使用するリフレッシュ・ボタンがあり、このボタンを使用すると、データベース・ナビゲーション・ツリーが再移入されて新しい情報が表示されます。

データベース名をクリックすると、ツリーが開き、そのデータベースの様々なスキーマが表示されます。スキーマをクリックすると、そのスキーマのKapow表が表示されます。表をクリックすると、表の内容が右側のデータ・グリッドにロードされます。データがロードされると、データベース・ナビゲーション・ツリーの下にいくつかのフィルタが表示されます。このフィルタは、ロギング・タブのフィルタとまったく同じように機能します。バイナリおよび長いテキスト属性はフィルタできません。

「削除」列をクリックすると、表の1つまたは複数の行を削除できるウィンドウが開きます。行をダブルクリックすると、その行の内容を含むウィンドウが表示されるため、データをコピーできます。

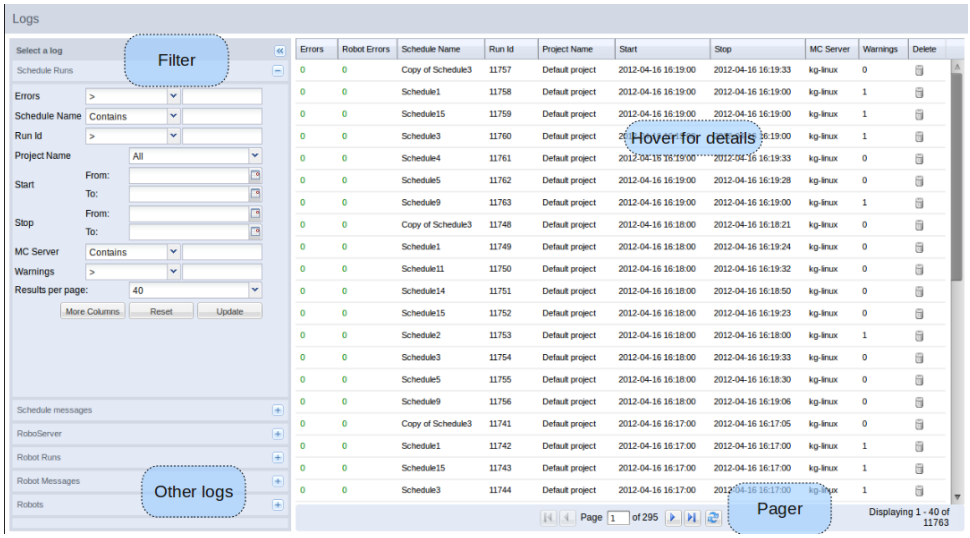
前述のデータ・グリッドには、4つのボタンが含まれるエクスポート・バーが表示されます。最初の3つのボタンでは、表のデータをExcel、XMLまたはCSV形式にエクスポートできます。4番目のボタンでは、以前のエクスポートを表示して再度ダウンロードできます。エクスポートは次回Management Consoleを再起動したときに自動的に削除され、また、エクスポート数が100を超えたときには最も古いエクスポートが削除されます。CSVまたはXMLにエクスポートできる行数に制限はありませんが、Excelファイルは、メモリーの不足を防ぐために10000レコードに制限されています。

ナビゲーション・ツリーの各データベースで使用できるスキーマ/カタログのリストは、(クラスタ設定のデータベース構成で)使用されるユーザーの資格証明の権限によって制御されます。

ログ

このタブを使用すると、Management ConsoleおよびRoboServerのデータベース・ロギングで作成されたログを表示できます。スケジュールの実行およびスケジュールのメッセージは、スケジュールに関する情報をレポートするManagement Consoleのログです。残りのログは、RoboServerのス

テータス、ロボットおよびロボットの実行に関する情報を含むRoboServerのログです。Management Consoleのログは Management Consoleのデータベースに書き込まれるため、常に使用可能です。ただし、RoboServerのログの場合は、ロギング・データベースがManagement Consoleの「設定」(「設定」->「ログ・データベース」を参照)で設定されて、ロギングのクラスタ設定を使用してRoboServerで有効である必要があります。



ログ・ビュー

6つのログのビューのいずれかを選択するには、表示するログの横にある  アイコンをクリックします。

6つのログのそれぞれのページ・レイアウトは同じです。  をクリックすると、左側にはいくつかのフィルタが表示され、右側のグリッドにはデータがロードされます。フィルタの下には選択ボックスと3つのボタンがあります。

データ・グリッドへの列の追加
列の追加ボタンは、右側のグリッドで列を追加および削除するために使用します。「リセット」は、入力されたフィルタの構成を消去します。「更新」は、フィルタの構成に基づいたデータを含むグリッドをロードします。選択ボックスでは、(次の更新に対する)各ページの結果数を制御します。選択した各ページの結果数よりも結果が多い場合は、データ・グリッドの下にあるコントロールを使用して次のページにナビゲートできます。任意のフィルタ・テキスト・フィールドで戻るボタンをクリックして、更新をトリガーすることもできます。

ログ・メッセージを削除しない場合は、ログが特定のサイズを超えて増加しないように、自動クリーン・アップ・システムが準備されています。RoboServerのログの場合は、RoboServerの自動クリーン・アップ・システムによって実行され、最も古いメッセージが最初に削除されます。「設定」->RoboServerログ・データベースでは、クリーン・アップのトリガー前に必要なメッセージ数を構成できます。スケジュールのログ・クリーン・アップしきい値は、「設定」->スケジュール・ログで定義します。

スケジュールの実行

スケジュールの開始時間や終了時間など、実行する各スケジュールの実行情報が表示されます。

コンテキスト・メニューを使用すると、個々のスケジュール・メッセージまたはスケジュールの実行の一部として実行されたロボットにナビゲートできます。

「削除」列内をクリックすると、実行およびメッセージを削除できるウィンドウが開きます。削除時には、メッセージを常に削除する必要がありますが、実行情報は必要に応じて保持できます。この実行およびメッセージを削除することも、

現在のフィルタに一致するすべての実行またはメッセージを削除することもできます。多数の実行またはメッセージの削除には時間がかかる場合があります。

スケジュールのメッセージ　特定のスケジュール実行の個々のメッセージ・エントリが表示されます。これによって、スケジュールの実行が失敗した理由を正確に確認できます。

注意: 特定のスケジュール実行のロボット・エラーは、コンテキスト・メニューからロボット・ビュー・エラーの表示を選択することで検出できます。

1つ以上のレコードを削除するには、「削除」列をクリックします。これによってウィンドウが開き、このメッセージのみ、現在のフィルタに一致するすべてのメッセージ、またはすべてのRoboServerログ・メッセージの削除を選択できます(フィルタに一致するメッセージを削除するオプションは、結果の一致が500を超える場合は無効になります。これは、パフォーマンスのためです)。

RoboServer　RoboServerまたはManagement Consoleの一般的なメッセージが表示されます。ダブルクリックするか、コンテキスト・メニューを使用してメッセージを別のウィンドウに開くと、エラー・メッセージのコピーが容易になります。

1つ以上のレコードを削除するには、「削除」列をクリックします。これによってウィンドウが開き、このメッセージのみ、現在のフィルタに一致するすべてのメッセージ、またはすべてのRoboServerログ・メッセージの削除を選択できます(フィルタに一致するメッセージを削除するオプションは、結果の一致が500を超える場合は無効になります。これは、パフォーマンスのためです)。

ロボット　これは、今まで実行されたすべてのロボットに関する単純なサマリー・ビューです(データが使用可能である場合)。行をダブルクリックするか、コンテキスト・メニューを使用すると、特定のロボットのすべての実行にナビゲートできます。

ロボットの実行　各ロボットの実行に関する情報が表示されます。このログには、デフォルトでは表示されない追加のフィールドがいくつかありますが、フィルタの下にある列の追加をクリックすると追加できます。行をダブルクリックすると、この実行時にログに記録されたすべてのメッセージが表示されますが、コンテキスト・メニューでも表示できます。コンテキスト・メニューでは、同じロボットに対するすべての実行を表示することも、この実行の実行時にこのロボットに提供された入力を表示することもできます。

「削除」列をクリックすると、実行およびメッセージを削除できるウィンドウが開きます。削除時には、メッセージを常に削除する必要がありますが、実行情報は必要に応じて保持できます。この実行およびメッセージを削除することも、現在のフィルタに一致するすべての実行またはメッセージを削除することもできます。多数の実行またはメッセージの削除には時間がかかる場合があります。

ロボットのメッセージ　ロボットの実行に属する個々のエラー・メッセージが表示されます。行をダブルクリックすると、エラー・メッセージを簡単にコピーできるウィンドウが表示されます。コンテキスト・メニューを使用すると、このメッセージが属する実行にナビゲートできます。

ロボット・エラーの場合は、データ・グリッドの位置コード列にリンクが含まれます。このリンクをクリックすると、小さい.robotDebugファイルがダウンロードされます。Design Studioを使用してファイルを開くと、ロボットがロードされてこの実行の入力が設定され、ロボットによってエラーの位置にナビゲートされるため、エラーをすばやくデバッグできます。

管理

このタブには、Management Consoleの管理に使用する複数のサブタブが含まれています。

タスク・ビュー　Management Consoleで実行中のロボットおよびその他タスクが表示されます。

クラスタ　RoboServerおよびクラスタを追加または削除したり、クラスタ設定を構成します。

プロジェクト　プロジェクトを作成および削除します。

ユーザー　Management Consoleのユーザーに関する情報を表示します。

設定　様々なManagement Consoleの設定を構成します。

バックアップ　バックアップを作成および復元したり、プロジェクトをインポート/エクスポートします。

ライセンス　ライセンス情報を入力したり、現在のライセンスおよび使用中のDesign Studioのシートを表示します。

タスク・ビュー

タスク・ビューには、Management Consoleのスケジューラによって開始されたスケジュール済ロボットおよび前後処理タスクが表示されます。その目的は、すべてのサーバーおよびロボットにわたる現在のアクティビティの概要を提供することであるため、以前のロボット実行の結果を把握する必要がある場合は、ログを検査する必要があります。

更新の遅延の原因となるため、このタブには、短時間実行のロボットは表示されません(あるいはほとんど表示されません)。

現在実行中のすべてのロボットに関する次の情報が表示されます。

表11.9 実行の情報

列	説明
名前	実行中のタスクの名前。
スケジュール	タスクを実行するスケジュールの名前。
プロジェクト名	このタスクが属するプロジェクトの名前。
ステータス	タスクの状態は次のいずれかになります。 キュー待機中: タスクはキューに入って実行を待機しています。タスクは次の理由でキューで待機している可能性があります。 サーバーなし: 現在クラスタにサーバーがないか、またはすべてのサーバーがオフラインです。サーバーが追加されるか、オフライン・サーバーがオンラインになると、タスクは実行を開始します。 容量なし: すべてのサーバーが最大の容量で実行中です。他のタスクが終了して容量が使用可能になるか、または追加サーバーがクラスタに追加されると、タスクは実行を開始します。 他のタスクの待機中: タスクは他のタスクの終了を待機している可能性があります。後処理はすべてのロボットが終了するまで実行されず、ロボットは前処理が終了するまで実行されない場合があります。また、スケジュールのロボットが順次実行されるように構成されている場合、ロボットは、前のロボットが完了するまでキュー待機中の状態になります。 実行中: このタスクは現在実行中です。

列	説明
最終ステータス変更	最終ステータス変更の時間。
停止	実行が終了していない場合でもタスクを停止する場合に、このボタンをクリックします。

クラスタ

このセクションでは、Management Consoleで認識されているクラスタおよびRoboServerを管理できます。リストのすべてのサーバーは、ダッシュボードのポートレットを使用して監視できます。デフォルトでは、このリストには、1つのRoboServer(つまり、Management Consoleの機能も実行するサーバー)を含む1つのクラスタが表示されます。複数のRoboServerおよびクラスタを使用する大規模設定では、Management Consoleをスタンドアロンの Webコンテナ(ライセンスで許可されている場合)にデプロイするか、またはロボットの実行に使用しないRoboServerにデプロイすることをお薦めします。

サーバーごとに次の情報が表示されます。

表11.10 サーバーの情報

列	説明	
クラスタ/サーバー	クラスタの場合	クラスタの名前。クラスタでSSLが使用されている場合は、接尾辞に- SSLが付きます。クラスタに未適用の設定がある場合は、ここでもそれが示され、名前が青で表示されます。クラスタに無効な設定がある場合は、名前が赤で表示されます。
	RoboServerの場合	RoboServerの名前およびポート。RoboServerの構成が不適切な場合は、名前が赤で表示されます。赤い名前の上にマウスのポインタを重ねると、ツールチップによってエラーが通知されます。
バージョン	実行中のRoboServer上でのソフトウェアのバージョンが表示されます。	
KCU	このクラスタに割り当てられたKCU数。クラスタ内のKCUは、クラスタ内のオンラインRoboServer間で均等に配分されます。クラスタのKCUを調整するには、KCUの割当てボタンをクリックします。	
実行中のロボット	このRoboServerで現在実行中のロボット数が表示されます。	
キュー待機中のロボット	このRoboServerでのキュー待機中のロボット数。	
最大ロボット	このRoboServerでの同時ロボットの最大数。クラスタ設定で構成できます。	
最大キュー	このRoboServerでキュー待機可能なロボットの最大数。クラスタ設定で設定できます。	
ライセンス・タイプ	RoboServerのライセンス・タイプで、本番または非本番のいずれかです。	
クラスタ・モード/サーバー・ステータス	クラスタの場合は、クラスタ・モードが表示されます。RoboServerの場合は、サーバーがオンラインかオフラインかが表示されます。	
削除	Management Consoleからクラスタ/サーバーを削除する場合に、このボタンをクリックします。これによって、ダッシュボードのポートレットから関連する情報が削除されます。	

クラスタの作成

クラスタの作成時には、クラスタの名前およびクラスタ・タイプを指定します。非本番クラスタを作成する場合は、非本番ライセンスからKCUを割り当て、同様に本番クラスタを作成する場合は、本番ライセンスからKCUを割り当てることができます。SSLオプションを選択する場合は、クラスタ内のすべてのRoboServerでSSL RQLサービスを使用する必要があります。

クラスタを作成した後は、そのクラスタにRoboServerを追加できます。

コンテキスト・メニュー

グリッドにはコンテキスト・メニューが表示され、このメニューには、ボタン・メニューからは使用できない使用頻度の少ない次の機能が含まれています。

クラスタ設定　クラスタ設定ダイアログが表示されます。

RoboServerの停止　選択したRoboServerを停止/再起動できるダイアログが表示されます。

スレッドのダンプ　フル・スレッド・ダンプを実行するために、選択したRoboServerにリクエストを送信すると、スレッド・ダンプが別のウィンドウに開きます。Management Consoleのスレッド・ダンプを取得することも可能です(詳細は、「Hazelcast構成」を参照)。

負荷分散およびフェイルオーバー

クラスタでロボットの実行が必要な場合は、使用可能なスロットが最も多いRoboServerが検索されます。使用可能なスロットは、そのRoboServerですでに実行中のロボット数および同時実行可能なロボット数(クラスタ設定の最大同時ロボット)に基づいて計算されます。

クラスタ内のあるRoboServerがオフラインの場合、KCUは残りのRoboServer間で自動的に均等に配分されます。

クラスタ・モード

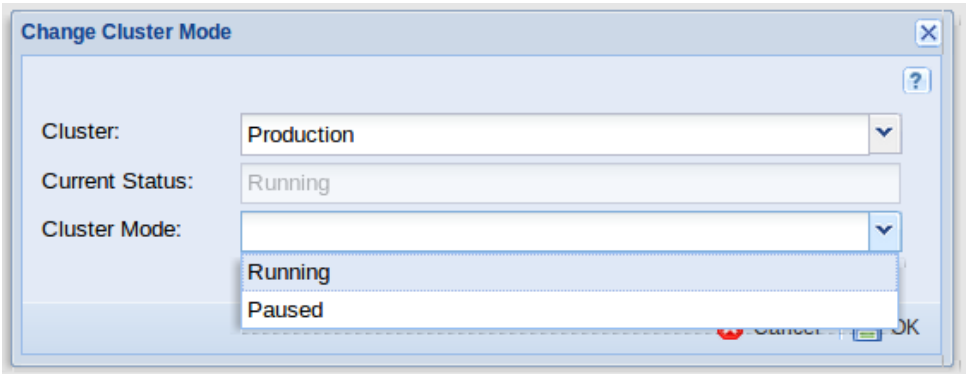
クラスタには、標準、メンテナンスまたはメンテナンス保留中の3つの異なるモードがあります。メンテナンス・モードは、クラスタを何も実行されていない状態のモードに設定する場合に使用します。このモードのポイントは、特定スケジュールのすべてのロボットが確実に同じ設定を使用し、その設定がスケジュール実行の途中で変更されないように、クラスタ設定を更新できることです。次の表では、個々のモードについて説明します。

表11.11 クラスタ・モード

クラスタ・モード	説明
標準	標準クラスタ・モードでは、クラスタは通常の動作で、スケジュールおよび個々のロボットは期待どおりに実行されます。スケジュール実行の途中で設定が変更されないように、このモードでは、クラスタ設定をRoboServerに適用しません。
メンテナンス	メンテナンス・モードでは、クラスタでのロボットの実行は許可されません(APIから開始しないかぎり(次を参照))。クラスタ設定をRoboServerに適用する場合、クラスタはメンテナンス・モードである必要があります。
メンテナンス保留中	クラスタをメンテナンス・モードに設定するには、3つの方法があります。これらによって、現在実行中またはキュー待機中のロボットおよびスケジュールの処理方法が指定されます。クラスタのメンテナンス・モードへの設定時にこの3つの方法のいずれかを選択すると、クラスタは、そのクラスタで何も実行されなくなるまで(実行可能なAPIロボットを除く(次を参照))、メンテナンス保留中モードになります。次に、クラスタをメンテナンス・モードに設定する3つの方法について説明します。 すべてのロボットおよびスケジュールをすぐに停止　これによって、現在実行中のすべてのロボットの停止が試行されます。キュー待機中のロボットはデキューされて実行されません。これは、クラスタをメンテナ

クラスタ・モード	説明	
		ンス・モードに設定する最も速い方法です。
	現在実行中のタスクの終了後に停止	これによって、クラスタがメンテナンス・モードになる前に開始されていて、現在実行中のすべてのロボットが確実に終了します。一部がキュー待機中で一部が実行中の複数のロボットを実行しているスケジュールでは、現在実行中のロボットのみが実行されて、キュー待機中のロボットはデキューされます。
	現在実行中のスケジュールの終了後に停止	これによって、実行中およびキュー待機中のすべてのロボットが確実に終了します。つまり、クラスタがメンテナンス・モードに設定される前に、開始済のスケジュールが終了します。

クラスタ・モードを変更するには、クラスタ・リストの上部にあるクラスタ・モードの変更ボタンをクリックします。これによって、次のダイアログが開きます。

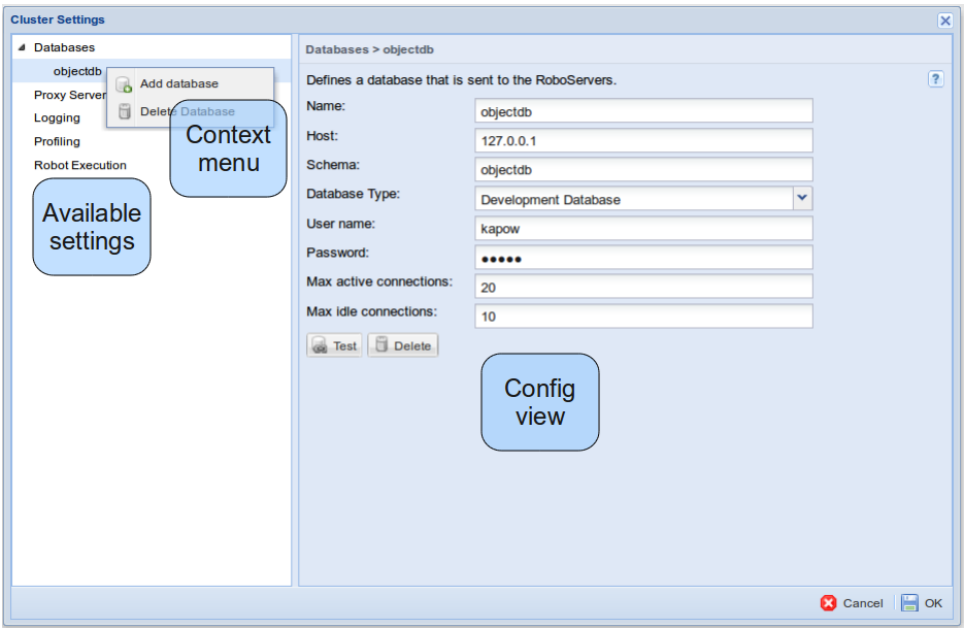


クラスタ・モードの変更
このダイアログでは、モードを変更する必要があるクラスタを選択し、メンテナンス・モードへの遷移に前述の方法のいずれかを選択する必要があります。ヒント: クラスタ・モードは、リスト内のクラスタを右クリックして、クラスタ・モードの変更で適切なサブメニュー項目を選択して変更することもできます。

名前が示すように、クラスタ・モードはクラスタ・レベルでのみ関係します。そのため、これは、タスクをクラスタで実行できる場合にManagement Consoleで制御する方法です。そうでない場合は、個々のRoboServerを制御します。これは、APIから開始されたロボットは、メンテナンス・モードになる場合も停止が試行されないことを意味します。したがって、RoboServerの設定はAPIロボットの実行中に更新できます。ただし、実行中のロボットの設定は更新されないことが保証されます。たとえば、1つ以上のAPIロボットの実行中にデータベースを更新する場合、これらのロボットでは開始時に構成されていたデータベースが使用されます。次回の実行時には、新しい設定が使用されます。

クラスタ設定

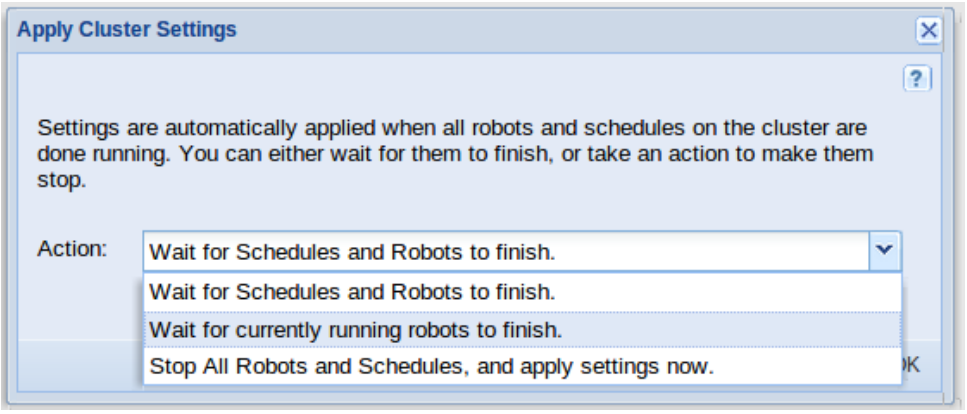
クラスタ設定は、クラスタ内の各RoboServerに送信される設定です。たとえば、クラスタ設定を使用して、RoboServerで実行中のロボットが使用できるデータベースを構成できます。クラスタ設定ダイアログを開くには、クラスタを右クリックしてクラスタ設定...メニュー項目を選択するか、「設定」列のボタンをクリックします。



クラスタ設定

クラスタ設定を変更すると、変更の設定が太字で示されて、「OK」ボタンをクリックして確定する前に変更内容の概要が示されます。

設定をRoboServerに送信するには、最初にその設定を適用する必要があります。設定の適用を実行するには、クラスタをメンテナンス・モードに設定します(詳細は、「クラスタ・モード」の項を参照)。「OK」ボタンをクリックしてクラスタ設定ダイアログを閉じ、複数の設定変更を確定したときに、クラスタがすでにメンテナンス・モードの場合は、その設定が即時に適用されます。クラスタがメンテナンス・モードでない場合は、次のダイアログが表示されます。



クラスタ設定の適用
設定の適用ダイアログで、設定を適用するために、クラスタのメンテナンス・モードへの遷移方法を選択します。設定の適用後、クラスタは標準モードに戻ります。

次の各項では、クラスタ設定の各部分の詳細を説明します。

データベース

クラスタに定義されているデータベースは、クラスタのRoboServerで実行中のロボットが使用できるデータベースです。これは、ロボットがクラスタ・データベースにアクセスすることを意味し、ロボットは、クラスタ設定で指定されたデータベースの名前を使用する必要があります。データベースでは、データベース・タイプが使用されます。データベース・タイプは、Management Consoleの「設定」で定義します。

クラスタをDesign Studioに共有データベースを提供するように選択している場合、クラスタに定義されているデータベースは、Design Studioにも送信されます。

9.2.0より前のバージョンからアップグレードしている場合は、データベースを、以前に定義したレガシーのdb.propertiesファイルからインポートできます。これを実行するには、ツリー内の「データベース」ノードを選択し、ファイルからのデータベースのインポート・ボタンをクリックします。これによって、ファイルの内容を貼付けできるウィンドウが開きます。

データベースのプロパティは、ここで説明する内容と同じです。

新しいデータベースを追加するには、データベース・カテゴリを選択して、データベースの追加ボタンをクリックするか、データベース・カテゴリを右クリックして追加ボタンをクリックします。レガシーのプロパティ・ファイルからデータベースをインポートできます。これを実行するには、データベース・カテゴリを選択し、ファイルからのデータベースのインポート・ボタンをクリックします。これによって、インポートするファイルの内容を貼付けできるウィンドウが開きます。

プロキシ・サーバー

このセクションでは、RoboServerで使用するプロキシ・サーバーのリストを構成できます。定義済のプロキシ・サーバーが1つのみの場合は、そのプロキシ・サーバーが使用されます。プロキシ・サーバーのリストを定義した場合は、最初の接続でランダムなプロキシ・サーバーが選択されます。後続の接続では、循環的なラウンド・ロビン順序でプロキシ・サーバーが使用されます。

プロキシ・サーバーのプロパティは、ここで説明する内容と同じです。

新しいプロキシ・サーバーを追加するには、プロキシ・サーバー・カテゴリを選択して、プロキシ・サーバーの追加ボタンをクリックするか、プロキシ・サーバー・カテゴリを右クリックして追加ボタンをクリックします。レガシーのプロパティ・ファイルからプロキシ・サーバーをインポートできます。これを実行するには、プロキシ・サーバー・カテゴリを選択し、ファイルからのプロキシ・サーバーのインポート・ボタンをクリックします。これによって、インポートするファイルの内容を貼付けできるウィンドウが開きます。このファイルは、この項の下部で説明している形式である必要があります。

ロギング

このセクションでは、クラスタのRoboServerに対してロギングを有効または無効にしたり、ログの定義方法を設定します。可能なロギングには3つのタイプがあり、これらについて次に説明します。

データベース・ロギング　データベース・ロギングが有効な場合、RoboServerでは、Design Studioの設定で定義されるRoboServerログ・データベースにログを記録します。ロボット入力をデータベースに記録が選択されている場合は、ロボットに提供される入力がデータベースに記録されます。注意: データベース・ロギングを機能させるには、RoboServerログ・データベースを構成して有効にする必要があります。データベースが構成されていない場合は、クラスタ設定が無効になります(この問題を修正できるように通知が送信されます)。

Log4J　ログの記録にlog4jを使用すると、通常の**log4j.properties**ファイルを使用してログの場所を制御できます。**log4j.properties**ファイルは、アプリケーション・データ・フォルダの**Configuration**フォルダにあります。デフォルトの**log4j.properties**ファイルでは、すべてのロボット実行情報、ロボット・メッセージおよびサーバー・メッセージを1つのファイルに記録します。このログは、アプリケーション・データ・フォルダの**Logs**フォルダに配置されます。より高度な設定には、Windowsのイベント・ログの格納、ファイルのローテーション、およびsyslogがあります。詳細は、log4jのマニュアルを参照してください。

このログを有効にすると、RoboServerでは、3つの異なるロガーを使用してログが記録されます。

表11.12 Log4Jロガー

ロガー	説明
log4j.logger.kapow.server.messages	このロガーは、特定のロボットの実行に関連付けられていないサーバー・ロギング用に使用されます。

ロガー	説明
log4j.logger.kapow.robotrun.log	このロガーはロボットの開始および停止に関する情報用です。これには、ロボットの実行時間が含まれます。
log4j.logger.kapow.robotmessage.log	Log Messageによって記録されるメッセージ(たとえば、各ステップでのエラー処理の一環として)。これには、プロファイリングも含まれます(プロファイリングをログに出力するように設定されている場合)。

電子メール・ロギング　これによって、ロボットがエラー・メッセージを記録した場合、またはサーバーにエラー・メッセージがある場合に電子メールが送信されます。

表11.13 電子メール構成

プロパティ	説明
送信先アドレス	電子メールの受信者。複数のアドレスは「,」で区切って追加できます。
送信元アドレス	電子メールで使用される送信元アドレス。

プロファイリング

ロボットのプロファイリングを有効にすると、個々のステップおよびロボット全体の実行時間を表示できます。これは、実行の遅いロボットのパフォーマンスを向上させる必要がある場合に便利です。

プロファイリングは、次のプロパティを使用して構成します。

表11.14 プロファイリングのプロパティ

プロパティ	説明
出力タイプ	「サマリー」を選択すると、実行の要約文のみがロボットの実行ごとにプロファイリング・ログに書き込まれます。詳細を選択すると、ステップの実行時間が「しきい値」プロパティで定義されたしきい値を超えた場合に、詳細な文がロボットで実行されたステップごとにプロファイリング・ログに書き込まれます。
出力ターゲット	これによって、プロファイリング情報の送信先が制御されます。可能な送信先は、コンソール、ファイルまたはログです。
しきい値	このしきい値では、ステップのプロファイリング情報を含めるための、そのステップの最短実行時間(ミリ秒)が定義されます。
待機メッセージにページURLを記録	選択すると、ページ・ロードの待機時間の前にページのURLが出力されます。

ロボットの実行

このセクションでは、RoboServerのロボット実行のプロパティを定義できます。(個々の)RoboServerで同時に実行可能なロボット数およびキュー待機可能なロボット数を指定できます。数字の微調整に関するヘルプについては、この項を参照してください。

表11.15 ロボット実行のプロパティ

プロパティ	説明
最大同時ロボット	各RoboServerで同時実行を許可するロボット数。
最大キュー待機ロボット	各RoboServerでキューに置くことを許可するロボット数。

プロジェクト

「管理」タブのこのセクションでは、Management Consoleで使用可能なプロジェクトを構成できます。

プロジェクトは、ロボット、タイプ、スニペット、リソースおよびスケジュールを区分する方法です。ロボットがアクセスできるのは、属しているプロジェクトに含まれるタイプ、スニペットおよびリソースのみです。また、ロボットやタイプなどの名前はプロジェクト内でのみ区別する必要があります。

プロジェクトを削除すると、そのプロジェクトに関連付けられているすべてのロボット、タイプ、スニペット、リソースおよびスケジュールも削除されることに注意してください。

デフォルトでは、Management Consoleには単一のプロジェクトが含まれています。

Management Consoleをスタンドアロンの Webコンテナにデプロイする場合は、グループ・メンバーシップ(LDAPグループなど)に基づいてユーザーのプロジェクト権限を構成することも可能です(詳細は、「プロジェクトの権限」を参照)。

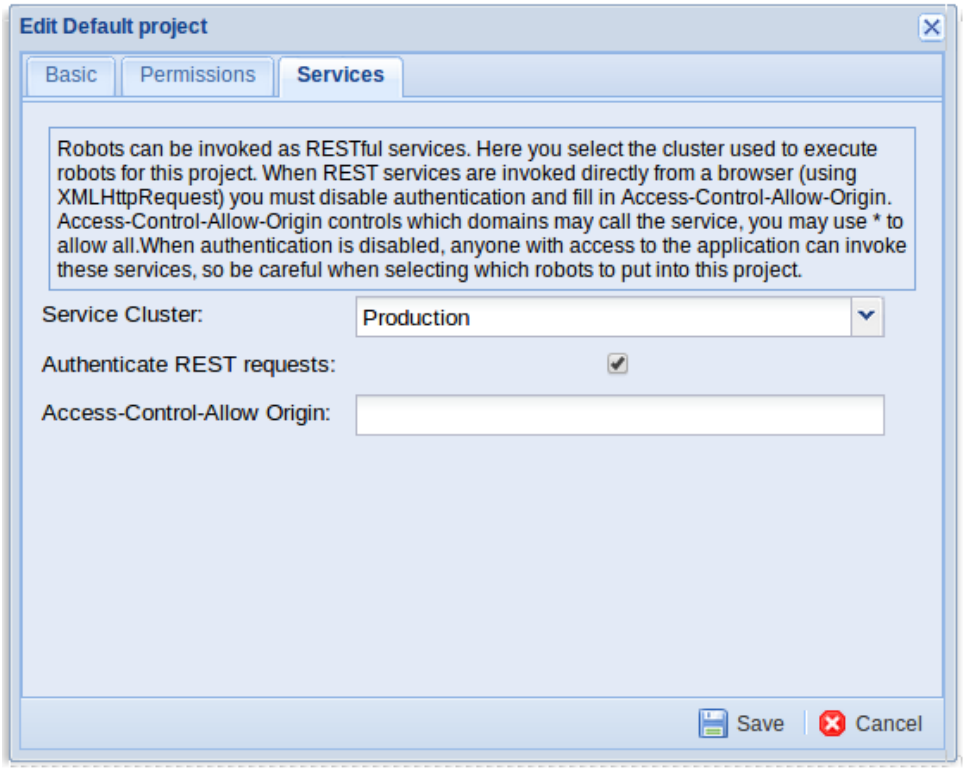
プロジェクト名は、ログ表の外部キーとして使用されます。これは、ログ・ファイルの表示権限があるユーザーを判断するために必要です。これは、プロジェクトの名前を変更する場合は、そのプロジェクトに属する既存のすべてのロボットの実行およびロボットのメッセージを更新して、新しい名前を反映する必要があることを意味します。そうしないと、ログはプロジェクトでフィルタされると表示されなくなります。プロジェクトの名前の変更時にManagement Consoleがロギング・データベースに接続している場合は、Management Consoleによって、ログ・データベースにあるロボットの実行/メッセージのエントリの名前が自動的に変更されます。ただし、接続していない場合、またはこの更新時に接続が失われている場合、管理者は、次のSQLを手動で実行してログ表を更新する必要があります。

```
UPDATE ROBOT_RUN SET projectName = '<newName>'
where projectName = '<oldName>'; UPDATE ROBOT_MESSAGE SET
projectName = '<newName>' where projectName = '<oldName>';
```

ここで、**<oldName>**は古いプロジェクト名で、**<newName>**は新しいプロジェクト名です。

Restサービス

ロボットは、Restサービスとして公開できます。「プロジェクトの編集」ダイアログを開くと、「サービス」タブでこれらのサービスのランタイムを構成できます。



プロジェクト - 「サービス」タブ

デフォルトでは、Restサービスは基本認証によって保護されています。ブラウザから(XMLHttpRequestを使用して) Restサービスを直接起動する場合は、認証を無効にする必要があり、そうしないと、JavaScriptソース・ファイル内のログインの資格証明を公開することになります。Java、Ruby、C#などのプログラミング言語からRestサービスを呼び出す場合は、認証によってサービスを保護することをお薦めします(資格証明をセキュアな方法で格納できると仮定)。

ブラウザからRestサービスを呼び出す場合は、そのサービスが起動元のWebページと同じWebサーバーに配置されていないかぎり、特定の制限があります。別のドメインからRestサービスを呼び出す(Cross-Origin Resource Sharing (CORS)と呼ばれます)場合は、クライアントが別のドメインのリソースを処理できるように、特定のヘッダーを含める必要があります。Access-Control-Allow Originはそのようなヘッダーの1つです。ドメインをまたぐ方法でRestサービスを呼び出す場合は、リクエストを生成したページがロードされたドメインを指定する必要があります。http://example.comのページに、http://kapowsoftware.comに配置されているサービスに対するリクエストを生成するJavaScriptが含まれたページがある場合は、http://kapowsoftware.comからのサービス・レスポンスには、ヘッダーのAccess-Control-Allow-Origin: http://example.comを含める必要があり、含めないと、ブラウザの組込みセキュリティ・メカニズムによってCross-Originレスポンスの処理が妨げられます。前述のスクリーン・ショットに示したように、ワイルドカードとして*を使用でき、これは任意のドメインがRestサービスをドメインをまたぐ方法で起動できることを意味します。

ユーザー

「ユーザー」タブでは、ログインしたユーザーに関するデータを表示できます。また、表内のそのユーザーの行にある「削除」ボタンを押すことによって、ユーザーが格納した構成を削除することもできます。

設定

このタブでは、Management Consoleを構成できます。変更内容を有効にするには、「保存」ボタンをクリックする必要があります。Management Consoleへの接続方法に影響を与える追加のオプション(ポート番号やセキュリティ設定など)は、「埋込みManagement Consoleの構成」の説明に従ってSettingsアプリケーションで構成します。

スケジュール・ログのクリーン・アップ

ここでは、スケジュール・ログのクリーン・アップに対するしきい値を構成します。

Management Consoleでは、スケジュールの実行数およびメッセージ数を10分ごとに参照します。ここで定義したしきい値を超えるレコードがある場合はクリーン・アップが発生し、レコード数がリストされたしきい値を下回るまでレコードが削除されます。

多数のスケジュールがある場合、またはスケジュールの実行が頻繁な場合は、しきい値を上げる必要があり、しきい値を上げないと、「ログ」タブで使用可能な情報が十分でない場合があります。

表11.16 RoboServerログ・データベースのクリーン・アップ

プロパティ	説明
最大スケジュール実行数	クリーン・アップのトリガー前にデータベースに記録されるスケジュールの最大実行数。
最大スケジュール・メッセージ数	クリーン・アップのトリガー前にデータベースに記録される各スケジュール・メッセージの最大数。

RoboServer認証

このセクションでは、構成済のクラスタに属するRoboServerでロボットを実行する際に、Management Consoleが使用する資格証明(ユーザー名およびパスワード)を構成します。Management Consoleでは、すべてのRoboServerに対して同じ資格証明のセットを使用します。これらの資格証明は、構成済のすべてのRoboServerについて、「必須認証」の説明に従って設定した構成と一致する必要があります。

RoboServerログ・データベース

これはロギング・データベースで、データベース・ロギングがクラスタ設定で有効化されているクラスタに属するすべてのRoboServerが使用します。このデータベースは、クラスタ・データベースと同じ方法で構成します。

RoboServerログ・データベースのクリーン・アップしきい値は、スケジュール・ログと同じ方法で構成できます。

表11.17 RoboServerログ・データベースのクリーン・アップ

プロパティ	説明
ロボット・メッセージ件数のしきい値	データベースのロボット・メッセージ数がこのしきい値を超えると、ログではそのデータベースが自動的にクリーン・アップされます。クリーン・アップでは、削除の実行対象として最も古いロボットの実行およびメッセージが削除されます。ログ・データベースによるパフォーマンス問題を認識している場合は、このしきい値を下げることができます。履歴メッセージの格納数を多くする場合は、このしきい値を上げることができます。
ロボット実行件数のしきい値	データベースのロボット実行件数がこのしきい値を超えると、ログではそのデータベースが自動的にクリーン・アップされます。クリーン・アップでは、削除の実行対象として最も古いロボットの実行およびメッセージが削除されます。ログ・データベースによるパフォーマンス問題を認識している場合は、このしきい値を下げることができます。履歴メッセージの格納数を多くする場合は、このしきい値を上げることができます。
サーバー・メッセージ件数のしきい値	データベースのサーバー・メッセージ数がこのしきい値を超えると、ログではそのデータベースが自動的にクリーン・アップされます。クリーン・アップによって、最も古いサーバー・メッセー

プロパティ	説明
	ジが削除されます。ログ・データベースによるパフォーマンス問題を認識している場合は、このしきい値を下げることができます。履歴メッセージの格納数を多くする場合は、このしきい値を上げることができます。

データベースをロギングに使用するには、新しいデータベース(スキーマ)を作成するか、または既存のデータベースが使用可能であることを単純に確認して、データベース・サーバーを準備する必要があります。データベースに対する表の作成、表の削除、索引の作成、索引の削除、選択、挿入、更新および削除を実行できる権限を備えたユーザー名とパスワードを取得する必要があります。

Management ConsoleおよびRoboServerの両方で、起動時にログ表が自動的に作成されます(表が存在しない場合)。ただし、ログ表は、次のスクリプトを使用して作成することもできます。

表11.18 ログ表に対するSQLスクリプト(右クリックして「名前を付けて保存」を選択)

データベース	表の作成	表の削除
IBM DB2	create	drop
Derby	create	drop
MySQL	create	drop
Oracle	create	drop
Microsoft SQL Server	create	drop
Sybase	create	drop

バージョン7.2からアップグレードしていて、すでに作成済のロギング表を使用する場合は、新しい列が追加されているため、**ROBOT_RUN**表を変更する必要があります。**VARCHAR(255)**タイプの**PROJECTNAME**という列を**ROBOT_RUN**表(Oracleでは**NVARCHAR2**、SybaseおよびMicrosoft SQL Serverでは**NVARCHAR**)に追加する必要があります。

バージョン8.1以前からアップグレードしていて、すでに作成済のロギング表を使用する場合は、新しい列が追加されているため、ROBOT_MESSAGE表を変更する必要があります。**VARCHAR(255)**タイプの**PROJECTNAME**という列を**ROBOT_MESSAGE**表(Oracleでは**NVARCHAR2**、SybaseおよびMicrosoft SQL Serverでは**NVARCHAR**)に追加する必要があります。

SMTPサーバー

このセクションでは、Kappletユーザーへの通知の送信(Kappletの起動時に通知が選択されている場合)、および電子メール・ロギングがクラスタ設定で有効になっている場合に電子メール・ログ・メッセージの送信に使用される電子メール・サーバーを構成できます。注意: Kappletの電子メール通知が機能するには、Kapplet結果で送信元アドレスも指定する必要があります。

SMTPサーバーは、次のプロパティを使用して構成します。

表11.19 SMTPサーバー

プロパティ	説明
ホスト	SMTPサーバーのホスト名。
ポート	SMTPサーバーのポート。
ユーザー	SMTPサーバーに認証が必要な場合は、ここにユーザー名を入力します。
パスワード	SMTPサーバーに認証が必要な場合は、ここにパスワードを入力します。

プロパティ	説明
暗号化	· なし: 資格証明および電子メールはクリア・テキストで送信されます。 · TLS: TLS暗号化が使用されます。これは、SMTPサーバーに信頼できる証明書がある場合のみ機能します(サーバーに自己署名の証明書がある場合は、keytoolユーティリティを使用してその証明書をエクスポートし、JVMの信頼ストアにインポートする必要があります)。STARTTLS SMTP拡張機能を使用します。 · SMTPS: SMTP over SSL。電子メールの送信に資格証明を介することで通信のセキュアなチャネルが確立されます。これは、SMTPサーバーによってまれにサポートされますが、サーバーで自己署名の証明書を使用する場合も機能します。

ベースURL

これは、アプリケーション・ベースURLです。これは、Kappletsでの結果のリンクの生成時に使用され、電子メールには、Management Consoleサーバーに存在する結果へのリンクを含めることができます。通常、ベースURLは自動的に構成され、変更する必要はありません。

プロキシ・サーバー

このセクションでは、Management Consoleが外部サーバーへの接続に使用するプロキシ・サーバーを指定できます(たとえば、外部APIにアクセスするためにOAuthセキュリティ・トークンを生成するときです)。

プロキシ・サーバーは、次のプロパティを使用して構成します。

表11.20 プロキシ・サーバー

プロパティ	説明
プロキシ・サーバーの使用	プロキシ・サーバーを使用する場合はtrue、直接接続を使用する場合はfalse。
ホスト	プロキシ・サーバーのホスト名。
ポート	プロキシ・サーバーのポート。
ユーザー	プロキシ・サーバーに認証が必要な場合は、ここにユーザー名を入力します。
パスワード	プロキシ・サーバーに認証が必要な場合は、ここにパスワードを入力します。

共有データベース

共有データベースは、Management Consoleから(ライセンスの取得によって)アクティブ化されているすべてのDesign Studioクライアントに送信されるデータベースです。ユーザー・インタフェースを簡素化するために、これらのデータベースがクラスタで構成されています。特定のクラスタで使用可能なデータベースをDesign Studioクライアントでも使用できるようにするには、ここでクラスタを単に選択します。Design Studioで使用可能な特別なデータベースのセット(Design Studioへの本番データベースの送信は不要など)が必要な場合は、共有データベースが定義済の特別なDesign Studioのクラスタを作成できます。共有データベースのデフォルトのクラスタは、デフォルトで自動的に作成される本番クラスタです。

Kapplet結果

このセクションでは、Kapplet結果の設定を構成します。

表11.21 Kapplet結果

プロパティ	説明
指定期間後に結果を削除(時間単位)	Kappletの結果が自動的にクリーン・アップ(削除)される前にストレージに存在する時間数。
送信元アドレス(結果の電子メール送信時)	結果の電子メールの送信時に使用される送信元アドレス。結果の電子メールには、適切に構成されたSMTPサーバーも必要です。

データベース・タイプ

データベース・タイプでは、MySQLなど、特定のタイプのデータベースを定義します。データベース・タイプは、次のプロパティで構成されています。

表11.22 データベース・タイプのプロパティ

プロパティ	説明
名前	データベース・タイプを識別する名前。
JDBCドライバ	JDBCドライバ・クラスのドライバ(このドライバは、データベース・ドライバの下にアップロードする必要があります)。
接続URLテンプレート	特定タイプのデータベースの接続URLを定義するテンプレート文字列。テンプレート文字列で使用可能な変数は次のとおりです。 ${**ServerName**} データベース・サーバーのサーバー名(ホスト)を定義します。 ${Schema} データベースのスキーマ(または、データベース・ベンダーに応じてデータベース/カタログ)を定義します。 **jdbc:mysql://${ServerName}/${Schema}**が接続文字列テンプレートの例です。この文字列では、デフォルトのポート(ポートの指定なし)および${**ServerName**}で指定されたサーバーで稼働し、${Schema}で指定されたスキーマを使用するMySQLデータベースを接続文字列で定義しています。変数は、特定のタイプのデータベースが作成されたときに提供される値です(「クラスタ・データベース」の項を参照)。
検証の問合せ	データベースの検証の問合せは、特定のデータベースに対する接続を検証するときに使用する問合せです。そのような問合せは、タイプの異なるデータベース間で大きく異なります。

データベース・タイプは、Design Studioクライアントにも送信されます。

新しいデータベース・タイプを追加するには、データベース・タイプ・カテゴリを選択して、データベース・タイプの追加ボタンをクリックするか、データベース・タイプ・カテゴリを右クリックして追加ボタンをクリックします。注意: 新しいデータベース・タイプの追加は、拡張機能であり、場合によっては実験的機能とみなす必要があり、デフォルトのデータベース・タイプはサポートされますが、新しいタイプのデータベース全体が製品全体で期待どおりに機能することは保証されていません。ただし、たとえば非標準ポートで稼働するMySQLサーバーを定義するタイプを追加するために、データベース・タイプを変更することは通常の使用事例です。

データベース・ドライバ

「設定」のデータベース・ドライバ・セクションには、アップロードされたデータベースJDBCドライバが含まれます。これらのドライバは様々なデータベース・タイプで必要です。データベース・タイプと同様に、アップロードされたデータベース・ドライバもDesign Studioクライアントに送信されます。Management ConsoleをMySQLデータベースなどで稼働する場合は、TomcatにMySQLドライバを提供する必要があります。これは、MySQLデータベースが機能するのは、Management Consoleで(「ログ」タブで、または接続のテスト時に)使用される場合であることを意味します。ただし、MySQL

データベースがすべてのRoboServerでも機能するには、MySQLドライバをRoboServer（およびDesign Studio)に送信できるように、ここでアップロードする必要があります。

ヒント: 特定のデータベース・タイプでは、特定のサイズを超える大規模なファイルを格納できるように、パラメータまたは設定の微調整が必要な場合があります。たとえば、MySQLデータベースの多数のインストールで、**max_allowed_packet**変数の値が1 MBに設定されている(つまり、1 MBを超える大規模なデータベース・ドライバを格納できない)場合は、この変数の値を上げる必要がある場合があります。ドライバのアップロード時に問題が発生した場合は、データベースに関するドキュメントを参照してください。問題の識別に役立つエラー・メッセージを受信し、Management Consoleログには詳細が含まれています。

セキュリティに関する注意: localhostからManagement Consoleにアクセスする場合、JDBCドライバをアップロードできるのは、デフォルトでは管理者ユーザーのみです。Management Consoleを埋込みモードで実行している場合は、RoboServer Settingsアプリケーションでこの動作を変更できます(詳細は、ここを参照)。Management ConsoleをTomcatで実行している場合は、かわりにこの項を参照してください。

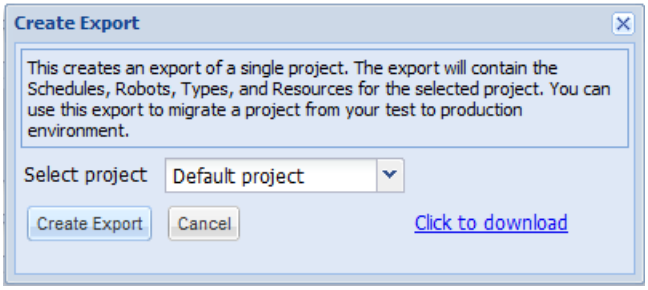
バックアップ

このタブでは、Management Consoleの構成のすべてまたは一部を1つのファイルにコピー(バックアップまたはエクスポート)したり、この方法で作成したファイルから情報を読み取る(復元またはインポート)ことができます。

バックアップの作成/復元およびプロジェクトのエクスポート/インポート間の相違を理解することが重要です。バックアップには、リポジトリ内のすべてのスケジュールやプロジェクトを含む、Management Consoleの構成のすべてが含まれ、その復元は全体でのみ可能です。これは、データの喪失後にシステムを復元するために使用したり、Kapow Katalystの後続バージョンへのアップグレード時に使用できます。後者の場合は、Management Consoleの古いインスタンスのバックアップを作成して、新しいインスタンスに復元します(リリース8.1より前のManagement Consoleのインスタンスからの復元の詳細は、次を参照してください。また、リリース8.1よりも前では、「バックアップの作成」は「エクスポート」という名前で、「エクスポート」という用語は現在「プロジェクトのエクスポート」に対してのみ使用されることに注意してください。)

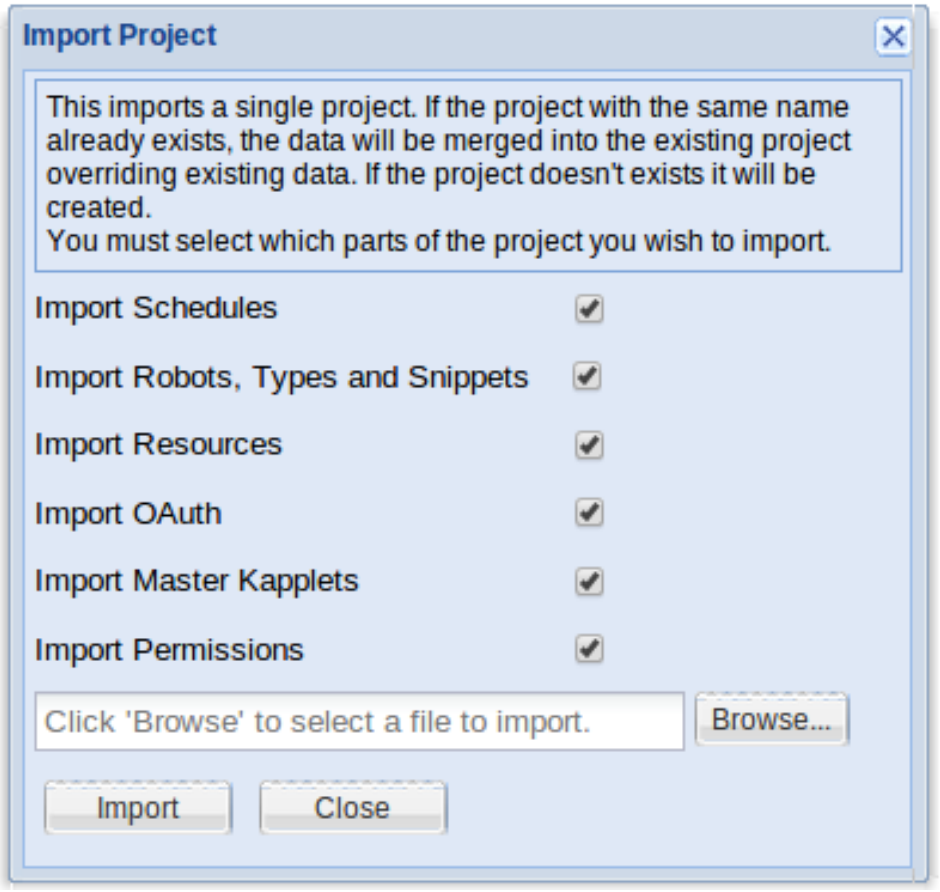
「プロジェクトのエクスポート」では、単一プロジェクトに関する情報が含まれた1つのファイルが作成されます。これは、プロジェクト内のスケジュール、ロボット、タイプおよびリソースで構成されます。そのようなファイルは、Kapow Katalystシステム間でスケジュール、ロボットなどをコピーするために使用できます(テスト環境から本番環境へ移動する場合など)。ターゲット・システムにある既存のプロジェクトへのインポートが可能で、この場合、ファイルのアイテムはプロジェクトにマージされ、名前が一致する場合は既存のアイテムが上書きされます。また、一部の種類のアイテムのみを含める選択的インポートの実行も可能です。

バックアップ・ファイルの作成またはプロジェクトのエクスポート・ファイルの作成は、次に示すように複数ステップのプロセスです。バックアップの作成ボタンまたはプロジェクトのエクスポート・ボタンをクリックすると、ダイアログが開きます。プロジェクトをエクスポートする場合は、このダイアログで目的のプロジェクトを選択する必要があります。次に、目的のバックアップ/エクスポート・ファイルを作成するために、「作成...」ボタンをクリックします。これには時間がかかる場合があります。このファイルの準備が整うと、ダウンロード・リンクが表示されます。このリンクがアクティブなのは、ダイアログが開いている間のみで、ダイアログを閉じた後はリンクをコピーして使用できないことに注意してください。次に、プロジェクトのエクスポート用のアクティブ・リンクが表示された(つまり、エクスポート・ファイルの作成後の)ダイアログを示します。



プロジェクトのエクスポートのダイアログ

バックアップからの復元とプロジェクトのインポートは似ており、最初にバックアップの復元ボタンまたはプロジェクトのインポート・ボタンをクリックします。表示されたダイアログで、復元/インポートするバックアップ/プロジェクトを検索します。プロジェクトをインポートするときは、(次の例に示すように)復元するアイテムの種類も選択します。最後に、「復元」ボタンまたは「インポート」ボタンをクリックして、実際に復元またはインポートを実行します。



プロジェクトのインポート・ダイアログ

リリース8.1よりも前のManagement Consoleからのエクスポートの復元に関する注意: 前述のとおり、Management Consoleの以前のリリースで作成されたバックアップからの復元が可能です。3つの注意事項を忘れないでください。最初に、リリース8.1で用語が変更されました。現在バックアップと呼ばれている内容に対して、以前はエクスポートという用語を使用していました。つまり、リリース8.1より前のManagement Consoleの内容を移行する場合は、そのエクスポート機能を使用してエクスポート・ファイルを作成する必要があります。2番目に、Design Studioの「ツール」メニューにある8.1より前のManagement Consoleのエクスポートのアップグレード機能を使用して、このファイルを新しいバックアップ・ファイル形式に変換する必要があります。3番目に、8.2ではプロジェクトベースの権限が新たに導入されたため、以前のバージョンの権限は自動的にアップグレードできません。古いバックアップの復元後は、プロジェクト権限を手動で割り当てる必要があります。

ライセンス

このタブでは、現在のライセンスに関する情報が表示され、新しいライセンスを入力できます。Management Consoleには、本番キーと非本番キーの2つのライセンス・キー用の場所があります。本番環境が開発/QA環境と完全に分離している場合は、2つのManagement Consoleをインストールし、一方には本番ライセンス・キーを含め、他方には非本番ライセンス・キーを含める必要があります。混合環境では、単一のManagement Consoleに両方のキーを含める必要があります。

このタブには、現在使用中のDesign Studioのシート(このManagement Consoleからライセンスを取得し、現在アクティブなDesign Studioクライアント)に関する情報も表示されます。

JMX

Management Consoleでは、JMXプロトコルを介して少量の情報を提供します。JMXを介して公開される情報は実験的で、パブリックAPIとはみなされません。

Management Consoleでは、次の4つのMBeanが公開されています。

表11.23 トップ・レベルのMBean

名前	説明
DistributionController	スケジュールの実行に関する情報およびクラスタ構成。
FileBackedDataExporter	データ・ビューから作成されたエクスポートに関する情報。
RobotLibraryGenerator	Management Console内のロボット・ライブラリの内部キャッシュを制御できます。
TaskQueuePerformanceTracker	2つのクラスタ化されたManagement Consoleインスタンス間で分散されたタスク・キューをプロファイルできます。

MBeanには、JConsoleとJava VisualVM (およびMBeanプラグイン)を介してアクセスできます。両方とも、JDK 1.6以上またはその他のJMXクライアントに含まれています。

OAuth

OAuthは、オープン・スタンダードの認証であり、TwitterやFacebookなどの一般的な多数のAPIで使用されます。これは、アプリケーション(およびKapow Katalystの場合はロボット)が、ユーザーのログイン資格証明に直接アクセスせずに、ユーザーのかわりにデータにアクセスしたり、アクションを実行する手段を提供します。たとえば、ロボットは、TwitterのAPIを使用してユーザー(例: @KapowSoftware)の最新の言及を抽出できますが、これにはそのユーザーによって提供されたアクセス・トークンを使用し、@KapowSoftwareのTwitterパスワードへの明示的なアクセスは不要です。

Management Consoleは、アクセス・トークンを生成してキー・ストアに安全に格納するために使用します。アクセス・トークンは、スケジュールのロボットに入力として渡すことができます。通常どおり、実際のAPI呼出しを実行して返されたデータを処理するロボットは、Design Studioで作成します。次の例では、両方の手順を示します。

サポートされるサービス・プロバイダ

OAuth用語では、サービス・プロバイダは、アクセスするWebサービスのプロバイダです。Kapow Katalystでは、次のサービス・プロバイダをサポートしています。

- Dropbox
- Facebook
- Google
- LinkedIn
- Salesforce
- Twitter

・ Yelp

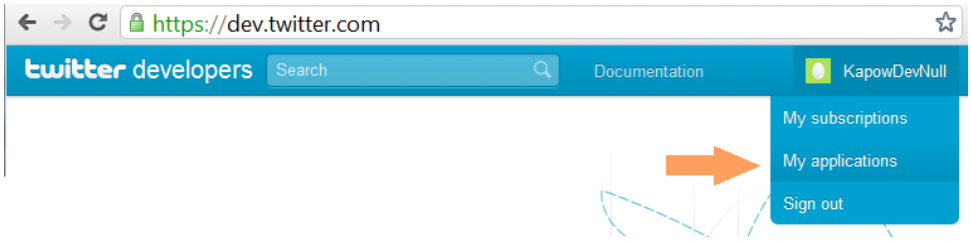
各サービス・プロバイダのAPIに関するドキュメントは、それぞれのWebサイトを参照してください。

アプリケーションの追加

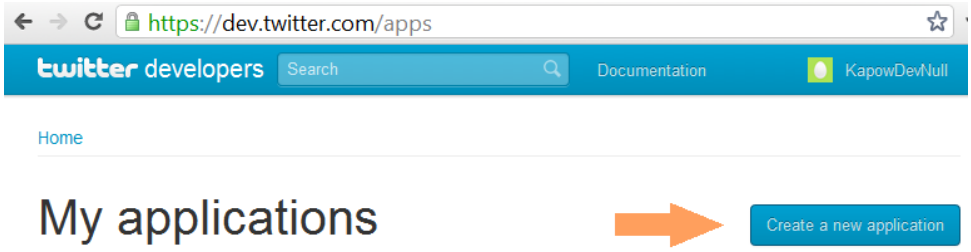
サービス・プロバイダのAPIにアクセスするには、そのサービス・プロバイダでアプリケーションを作成する必要があります。アプリケーションの作成によって、コンシューマ・キー(APIキーまたはアプリケーション・キーとも呼ばれます)およびコンシューマ・シークレット(APIシークレットまたはアプリケーション・シークレットとも呼ばれます)が提供されます。

アプリケーションを作成するには通常、サービス・プロバイダの開発者コミュニティにログインして、「新規アプリケーションの作成」などを選択し、必要な情報を入力します。これをTwitterで実行する方法を説明します。

最初に、http://dev.twitter.comにログイン(必要に応じて新しいアカウントを作成)し、右上隅にあるユーザー名をクリックして「My Applications」を選択します。

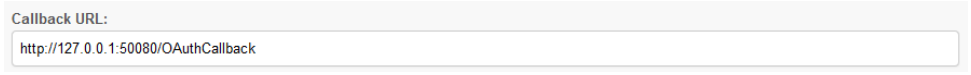


次に、「Create a new Application」ボタンをクリックします。



アプリケーションの名前や説明などの必要な情報を入力し、Twitterの利用規約を読んでから受諾します。

フォーム内のフィールドの1つに「Callback URL」があります。これは、作成するアプリケーションがユーザーのかわりにユーザーのTwitterアカウントと対話することをユーザーが承諾した後、ユーザーのブラウザがTwitterによってリダイレクトされる先のURLです。このフィールドには、Management Consoleがデプロイされているフォルダの下にある**OAuthCallback**へのパスを設定する必要があります。たとえば、埋込みManagement Consoleで実行する場合は、**http://localhost:50080/**で実行します。この場合は、コールバックURLを**http://localhost:50080/OAuthCallback**に指定しますが、サービス・プロバイダによっては、コールバックURLに**localhost**を含めることを許可していない場合があることに注意してください。Twitterはそのようなプロバイダの1つであるため、かわりに**http://127.0.0.1:50080/OAuthCallback**を使用します。

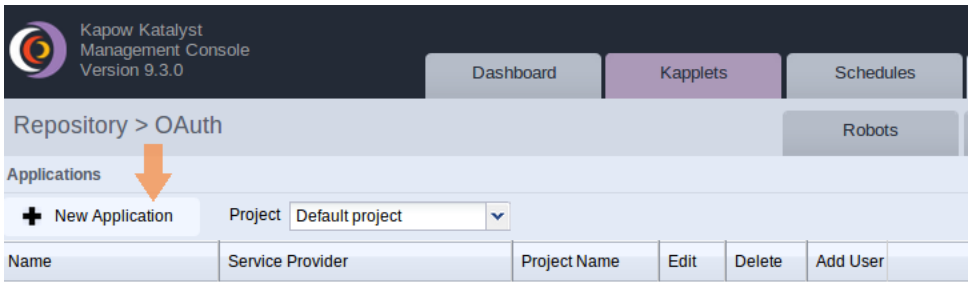


あるいは(一部のサービス・プロバイダではこれが必須)、Management Consoleを実行しているマシンのホスト名または非ループバックIPアドレスを指定する必要があります。このページは認証しているユーザーのブラウザによってロードされるため、これは、パブリック・ホスト名またはIPアドレスである必要はありません。

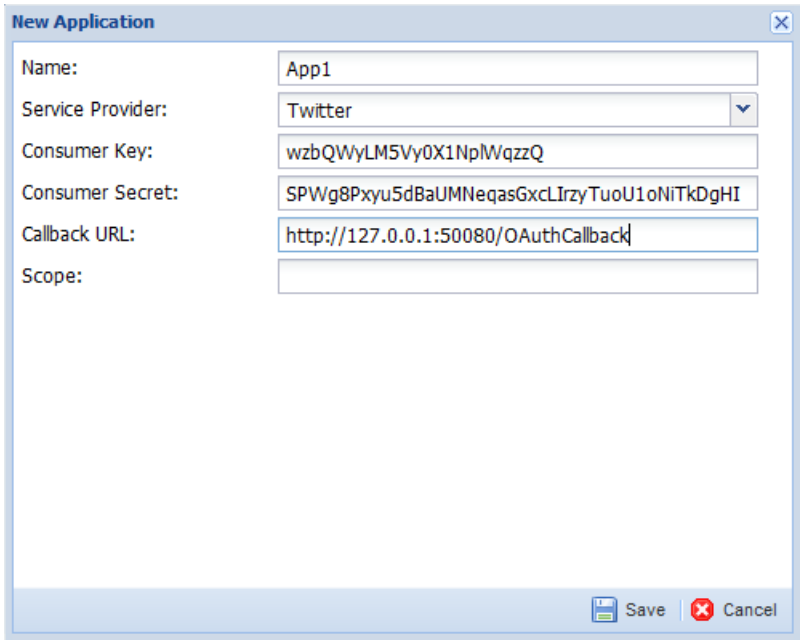
アプリケーションの作成後は、そのアプリケーションのサマリーが表示されます。これらの値の一部はManagement Consoleにコピーする必要があるため、ブラウザで開いたManagement Consoleに進みま

す。コールバックURLとして入力したのと同じIPアドレスまたはホスト名を使用する必要があります(したがって、この例ではブラウザを**http://127.0.0.1:50080/**に指定します)。

ここで、「リポジトリ」タブのサブタブであるOAuthタブにナビゲートして、「新規アプリケーション」ボタンをクリックします。



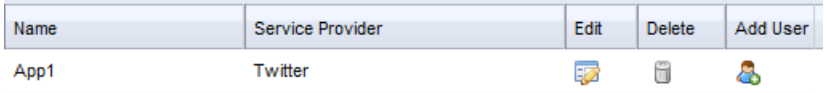
アプリケーションの名前(サービス・プロバイダでのアプリケーションの作成時に使用したのと同じ名前にする必要はありません)を選択して、サービス・プロバイダ(この場合はTwitter)を選択します。



コンシューマ・キーおよびコンシューマ・シークレットは、サービス・プロバイダで表示されたアプリケーションのサマリー・ページからコピーする必要があります。

以前入力したのと同じコールバックURLを入力して、「保存」をクリックします。サービス・プロバイダによっては、さらにスコープ(ユーザーがアプリケーションにアクセス権限を付与するAPIの部分)の指定が必要な場合があります。たとえば、アプリケーションによるGoogleへのアクセス時に、Google Analytics Data APIへのアクセスを許可する場合は、スコープに**https://www.google.com/analytics/feeds/**を指定する必要があります。Twitterでは、スコープ・フィールドを使用しないため、この例ではこのフィールドは空白のままにします。

これで、Management ConsoleでのOAuthアプリケーションの設定が完了しました。



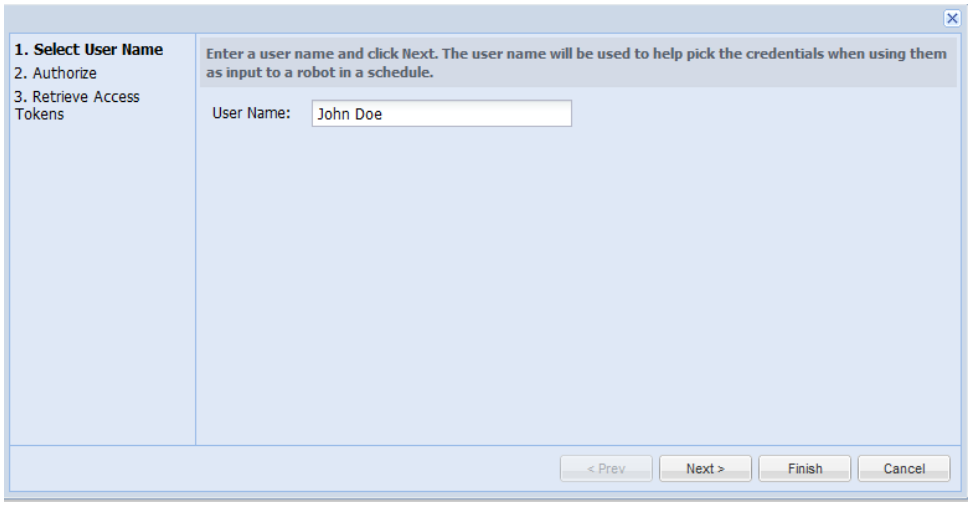
アプリケーションを後で編集する場合は、コンシューマ・シークレットがセキュリティ上の理由から「(暗号化)」と表示されることに注意してください。コンシューマ・シークレットを変更するには、入力フィールドのこの値を新しいコンシューマ・シークレットで置換し、それ以外の場合は、アプリケーションの編集時にこのフィールドはそのままにします。

次は、アプリケーションにユーザーを追加します。

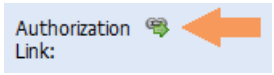
ユーザーの追加

前のステップで作成したアプリケーションの横にある「ユーザーの追加」列のアイコンをクリックします。

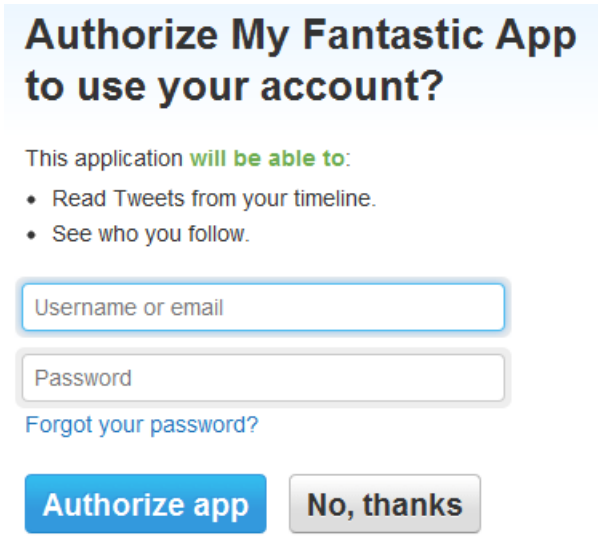
これによって、ウィザードが開始されます。最初のステップとして、ユーザーの名前を選択します。これをサービス・プロバイダで使用するユーザー名にマップする必要はなく、Management Console内でのみ使用されます。



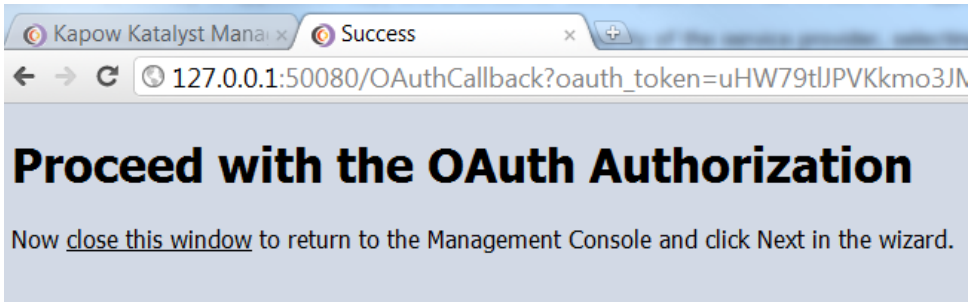
次の画面には、クリックする必要がある認証リンクがあります。



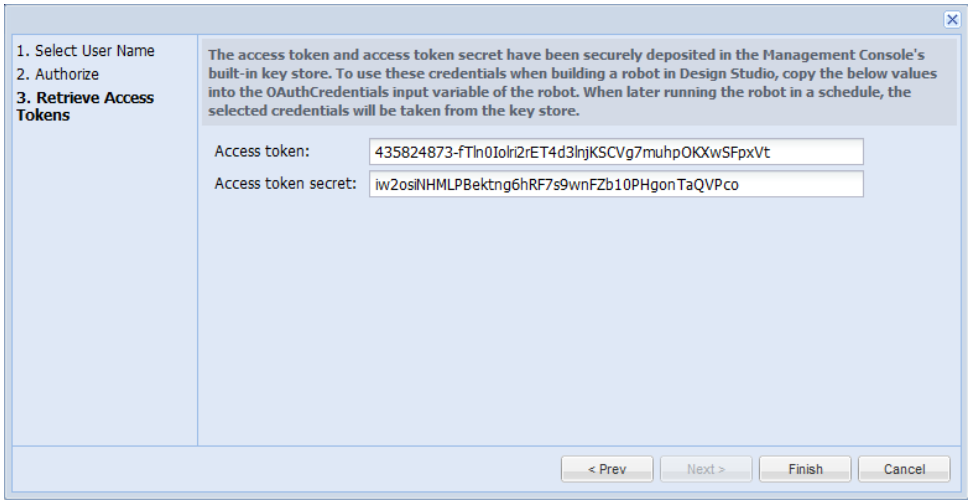
このリンクをクリックすると、サービス・プロバイダのWebサイトが表示されます。Twitterでは、次のように表示されます。



ユーザー名およびパスワードを入力して、「Authorize app」をクリックします。サービス・プロバイダによってコールバックURLに転送されて、認証が成功した場合は、次のページが表示されます。



ブラウザのタブを閉じて、Management Consoleに戻ります。ウィザードの「次」をクリックすると、ユーザーのかわりにサービス・プロバイダにアクセスするために使用できるアクセス・トークンが表示されます。これらは、Management Consoleのキー・ストアに安全に格納され、スケジュールに対する入力として使用できます。ただし、後のステップで作成するロボットに対するテスト入力として、サンプルのアクセス・トークンが必要になるため、これらの値をメモ帳などのテキスト・エディタにコピーします。「終了」のクリック後は、セキュリティ上の理由から、これらはキー・ストアから非暗号化形式で取得できなくなります。



Twitterでは、アクセス・トークンおよびアクセス・トークン・シークレットの両方を取得します。OAuth 2.0を使用するサービス・プロバイダでは、アクセス・トークンのみを返すため、アクセス・トークン・シークレットは使用されません。サービス・プロバイダによっては、さらにリフレッシュ・トークンを返す場合があります。これは、サービス・プロバイダによって返されるアクセス・トークンの有効期間が短い場合に使用されます。ロボットは、このリフレッシュ・トークンを使用して新しいアクセス・トークンを取得でき、ユーザーがManagement Consoleを介して再度認証する必要はありません。サービス・プロバイダのAPIに対するロボットを作成するには、このウィザードの最後のステップに表示されるすべてのトークンをコピーする必要があります。

「終了」をクリックすると、OAuthタブの「ユーザー」セクションにユーザーが表示されます。

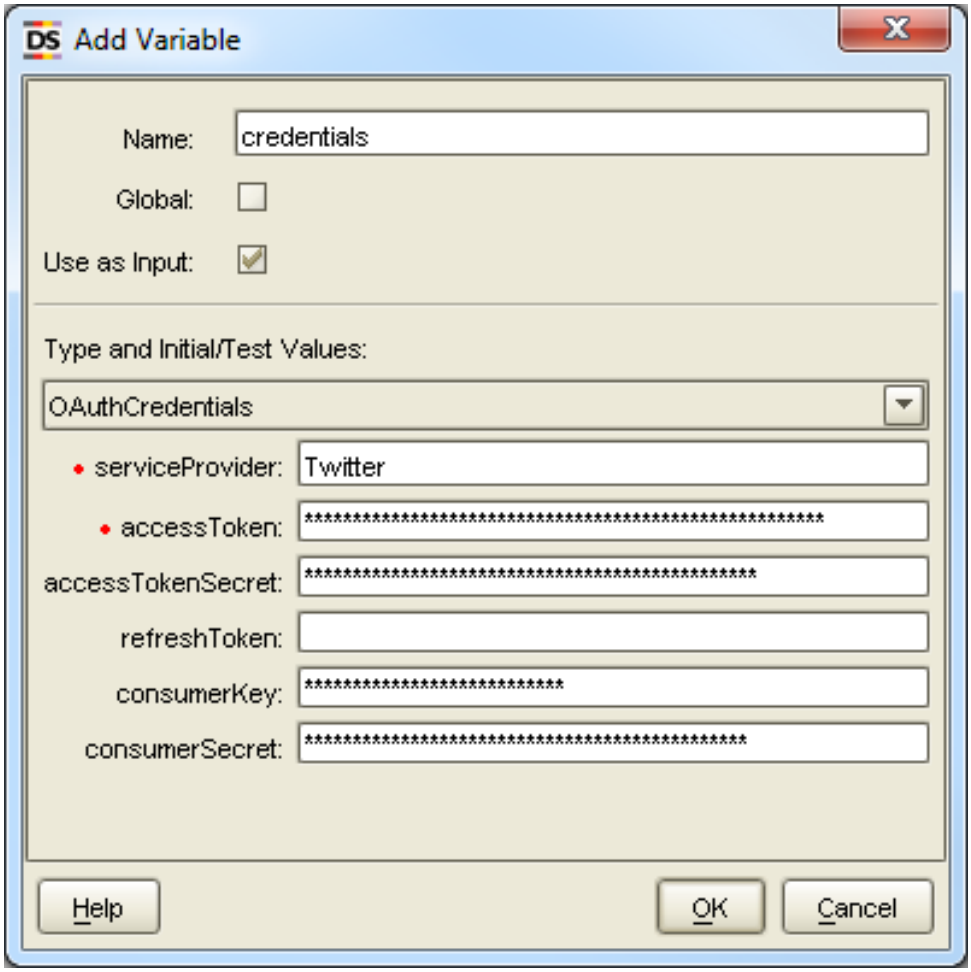
このユーザーを後で編集する場合、アクセス・トークン、アクセス・トークン・シークレットおよびリフレッシュ・トークンは、セキュリティ上の理由から「(暗号化)」と表示されることに注意してください。これらを変更するには、入力フィールドのこの値を変更後の値で置換し、それ以外の場合は、ユーザーの編集時にこのフィールドはそのままにします。

次は、OAuthを使用するAPIにアクセスするロボットの記述方法を示します。

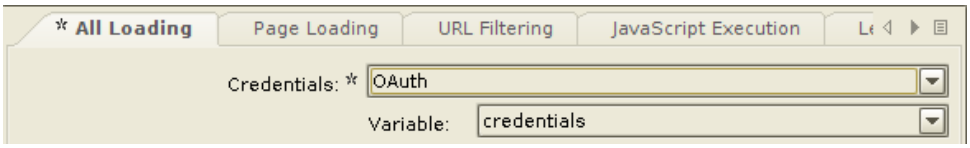
ロボットの記述

次に、認証メカニズムとしてOAuthを使用するREST APIにアクセスするロボットの記述方法を示します。この例では、TwitterのREST APIを使用して、認証しているユーザーおよびそのユーザーをフォローしているユーザー別の最新ステータスを取得します。

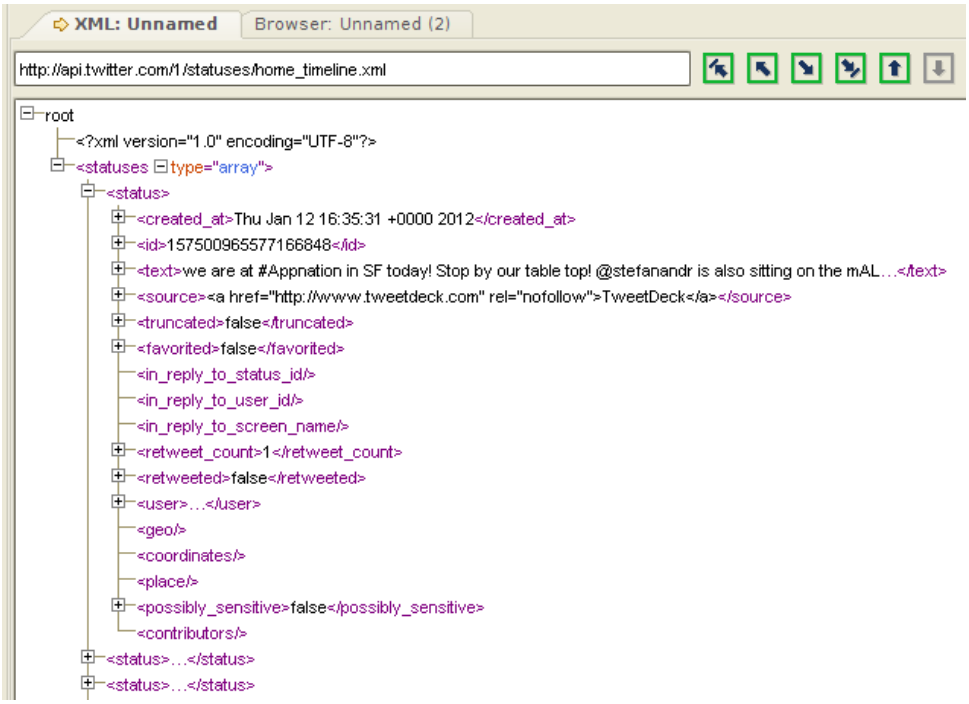
Design Studioを起動し、新しいロボットを作成します。認証前はREST APIにアクセスできないため、ウィザードにはURLを入力しないでください。**OAuthCredentials**タイプの新しい入力変数を追加します。サービス・プロバイダとして「Twitter」と入力し、Management Consoleウィザードのユーザー認証プロセスの完了時に取得したアクセス・トークンおよびアクセス・トークン・シークレットを入力し、Twitterアプリケーションのコンシューマ・キーおよびコンシューマ・シークレットを入力します。



変数の追加後は、  ボタンをクリックしてロボット構成を開きます。「基本」タブの「構成...」ボタンをクリックします。すべてロード・タブには、「資格証明」オプションがあります。これを標準のユーザー名/パスワード認証からOAuthに切り替えて、直前に追加した入力変数を選択します。



両方のダイアログで、「OK」を押します。これでロボットはOAuthを使用するように構成されたため、Design Studioでの実行時に指定の資格証明が使用されます。TwitterのAPIへのアクセスを開始できます。たとえば、認証しているユーザーおよびそのユーザーをフォローしているユーザー別の最新ステータスの更新を表示するには、**http://api.twitter.com/1/statuses/home_timeline.xml**のURLにアクセスできます。Design Studioのアドレス・バーにこのURLを入力して、[Enter]を押します。

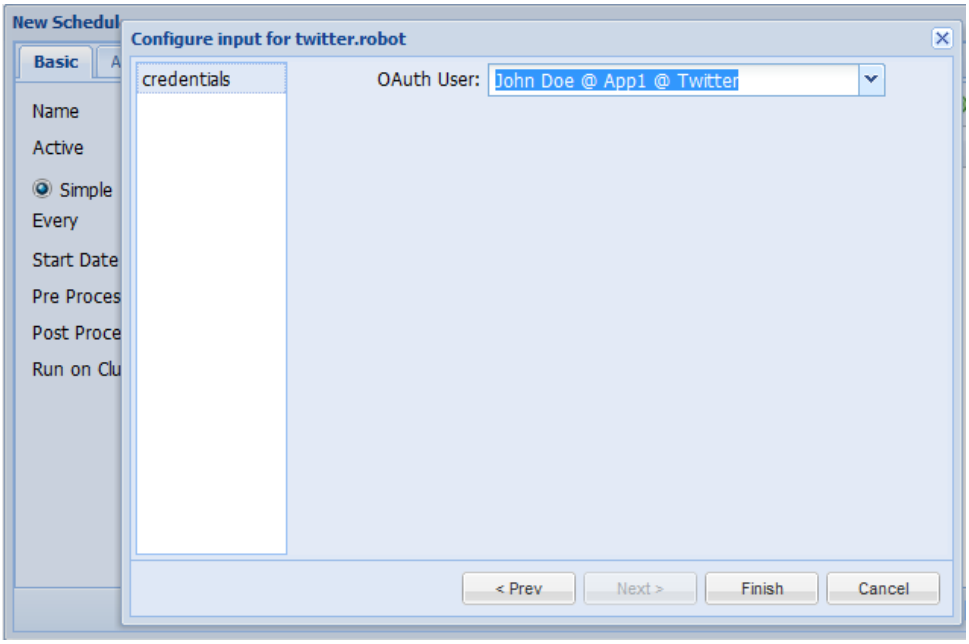


前述のように、ユーザーのタイムラインの最新ステータスが含まれたXMLが返されて表示されます。

次は、ロボットへの入力としてManagement Consoleに格納されている資格証明の使用方法を説明します。

資格証明を使用するロボットのスケジュール

最初に、前のステップで作成したロボットを保存して、OAuthのTwitterアプリケーションに対するユーザーを作成したManagement Consoleインスタンスにアップロードします。ブラウザでManagement Consoleを開いて、新しいスケジュールを作成します。スケジュールにロボットを追加して、ステップ4: 入力の構成に到達するまで、ロボットの追加ウィザードの「次」をクリックします。



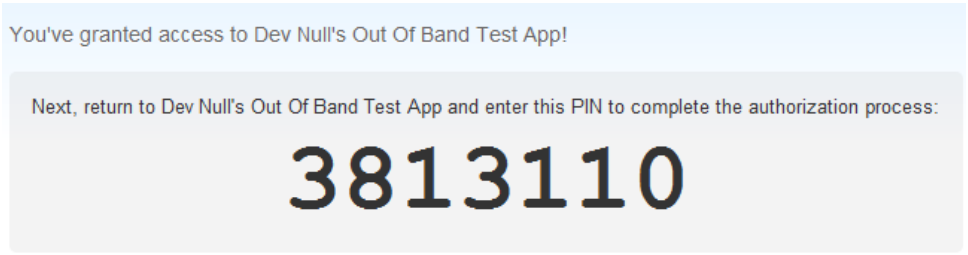
このスケジュールでのロボットの実行時に使用する資格証明のOAuthユーザーを選択します。これによって、RoboServerでのロボットの実行時に、サービス・プロバイダ、アクセス・トークン、コンシューマ・キーおよびコンシューマ・シークレットの各フィールドが自動移入されます。

「終了」をクリックしてスケジュールを保存すると、実行の準備が整います。

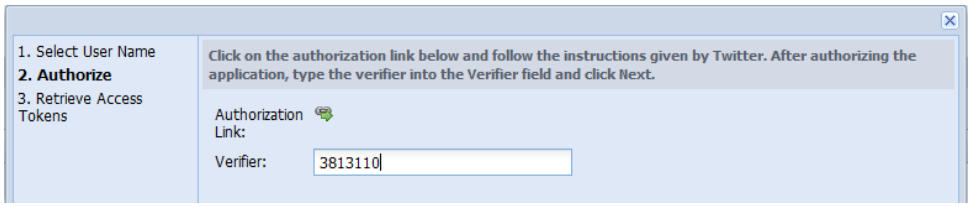
アウト・オブ・バンド・アプリケーション

一部のサービス・プロバイダでは、コールバックを使用せずにOAuthアプリケーションを認証するメカニズムも提供されています。これは「アウト・オブ・バンド」と呼ばれます。Management Consoleでもアウト・オブ・バンド認証がサポートされています。サービス・プロバイダで作成されたアプリケーションをアウト・オブ・バンド・アプリケーションとして登録する必要があります。Twitterでこれを実行するには、単に、コールバックURLフィールドを空白のままにします。その他のサービス・プロバイダでは、特別な値の「oob」を使用してアウト・オブ・バンド・アプリケーションを示します。

Management Consoleでのアプリケーションの登録時には、ここのコールバックURLフィールドも空白のままにします。アプリケーションへのユーザーの追加時には、認証リンクをクリックしてもManagement Consoleに戻るようにリダイレクトされません。かわりに、サービス・プロバイダによってベリファイア(つまりPIN)が表示されます。



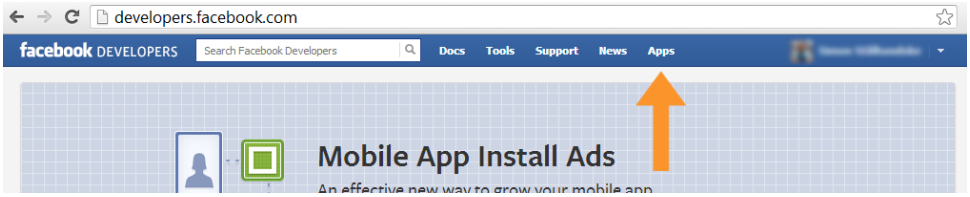
ウィザードの「次」を押す前に、このPINをベリファイア・フィールドにコピーする必要があります。



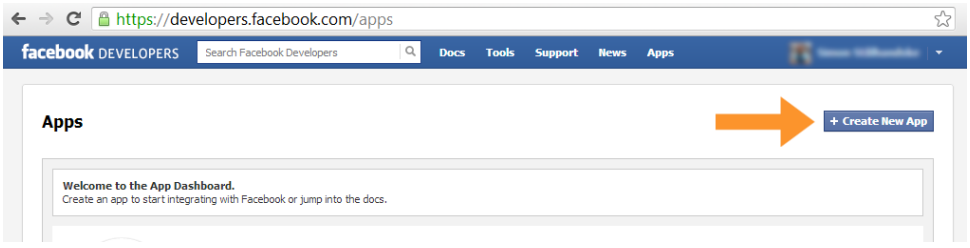
Facebook

これは、Graph APIを介してFacebookにアクセスするロボットの作成に関する簡単なガイドです。Graph APIは、Facebookのアカウントに対する読取りおよび書込みに使用するAPIです。詳細は、**https://developers.facebook.com/docs/reference/api/**を参照してください。

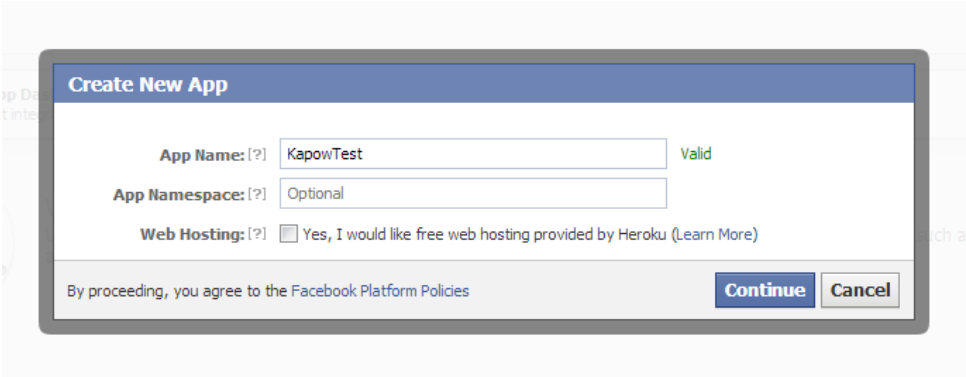
最初に、Facebookのアカウントを使用してhttp://developers.facebook.comにログインするか、必要に応じて新しいアカウントを作成します。ページの上部のバーにある「Apps」をクリックします。



次に、「Create New App」をクリックします。

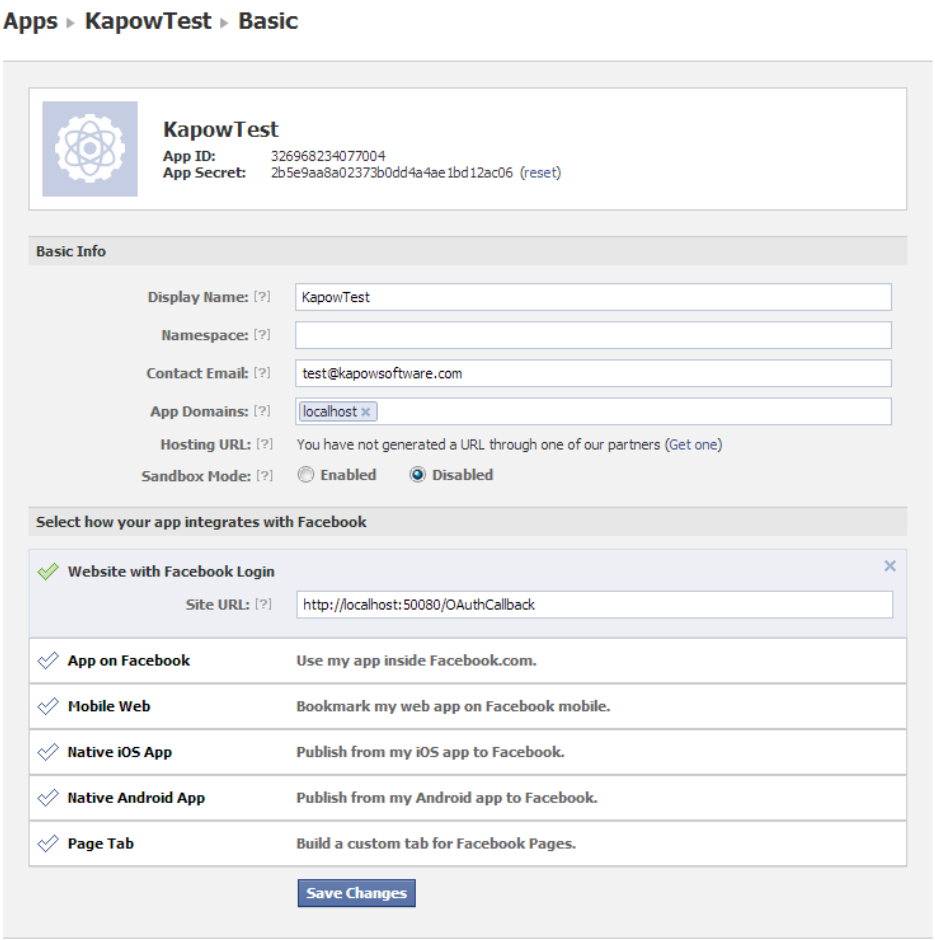


新しいアプリケーションの名前を入力して「Continue」をクリックします。



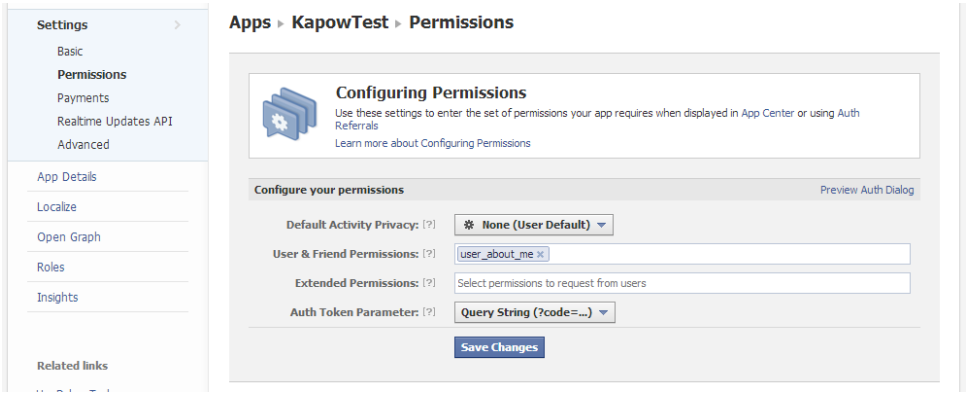
新しいアプリケーションの構成画面が表示されます。フィールドの1つの「App Domains」というフィールドは、アプリケーションで使用されるドメインを示すために使用します。これは、Management Consoleをホストしているドメインに設定する必要があります。埋込みManagement Consoleで実行する場合、このフィールドには「localhost」と記述します。ページの下部では、アプリケーションのFacebookとの統合方法を選択できます。「Website with Facebook Login」を選択して、「Site URL」にOAuthコールバックURLを記述します。コールバックURLは、Management Consoleがデプロイされているフォルダの下にある**OAuthCallback**へのパスです。たとえば、埋込みManagement Consoleで実行する場合は、**http://localhost:50080/**で実行します。この場合は、コールバックURLを**http://localhost:50080/OAuthCallback**に指定します。

このページでは、後でManagement Consoleでコンシューマ・キーおよびコンシューマ・シークレットとして使用するApp IDおよびAppシークレットをメモしておくことも重要です。

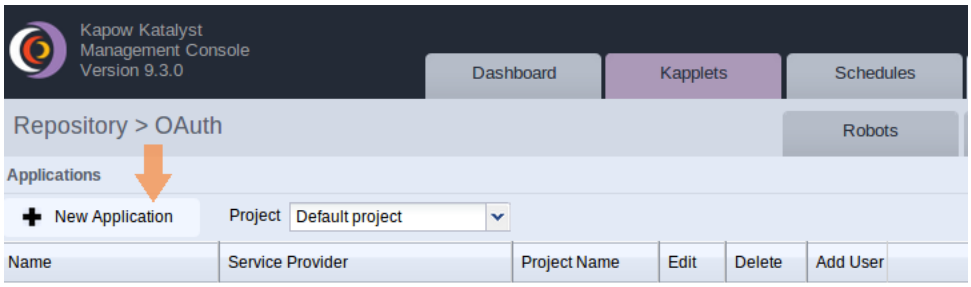


アプリケーションの基本構成の保存後は、ページの左側にあるメニューで「Permissions」をクリックして「Permission Configuration」に移動します。ここで、認証時にアプリケーションに必要な権限を構成できます。基本的なユーザー情報の場合は、「User & Friend Permission」フィールド

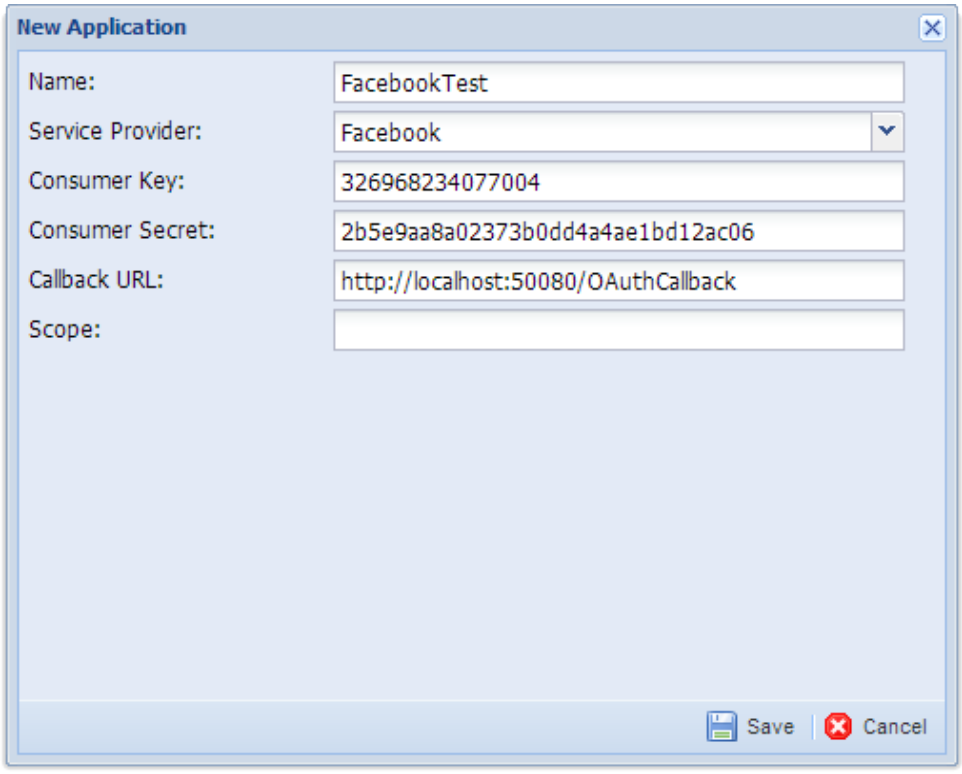
に「user_about_me」と入力します。指定可能な権限の詳細は、https://developers.facebook.com/docs/reference/login/を参照してください。



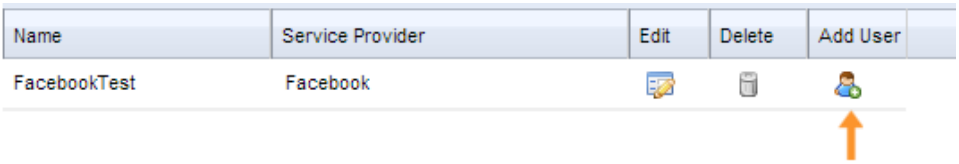
アプリケーションを設定して権限を保存した後は、Management Consoleを開きます。アプリケーションに対して指定したURLでManagement Consoleが開くことを確認してください。前述で使用した情報では、これは**http://localhost:50080**です。次に、OAuthのリポジトリにナビゲートして「新規アプリケーション」をクリックします。



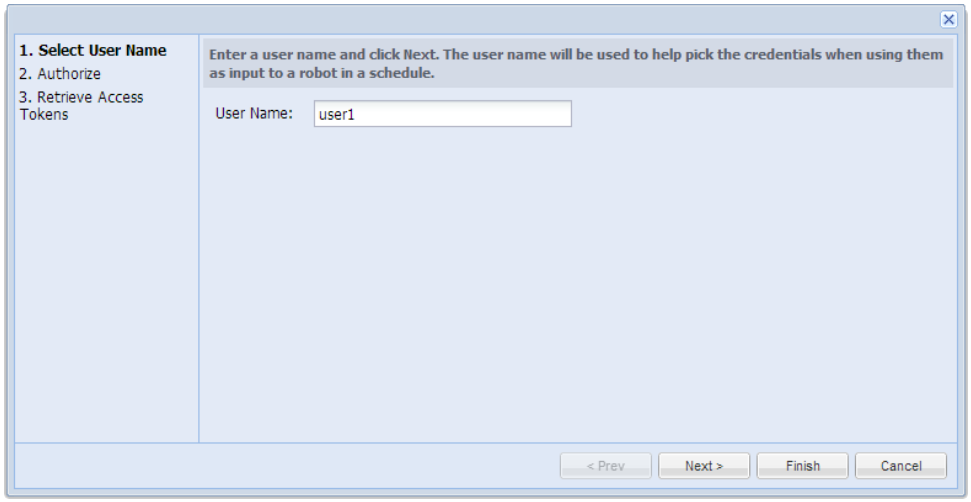
新しいアプリケーションの名前を選択し、サービス・プロバイダとしてFacebookを選択します。コンシューマ・キーおよびコンシューマ・シークレットには、以前にメモしたApp IDおよびAppシークレットをそれぞれ設定します。コールバックURLは、すでに正しく記述されています。



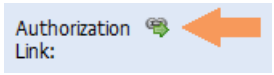
保存すると、アプリケーションがリポジトリに表示されます。「ユーザーの追加」の下にあるアイコンをクリックして、アプリケーションにユーザーを追加します。



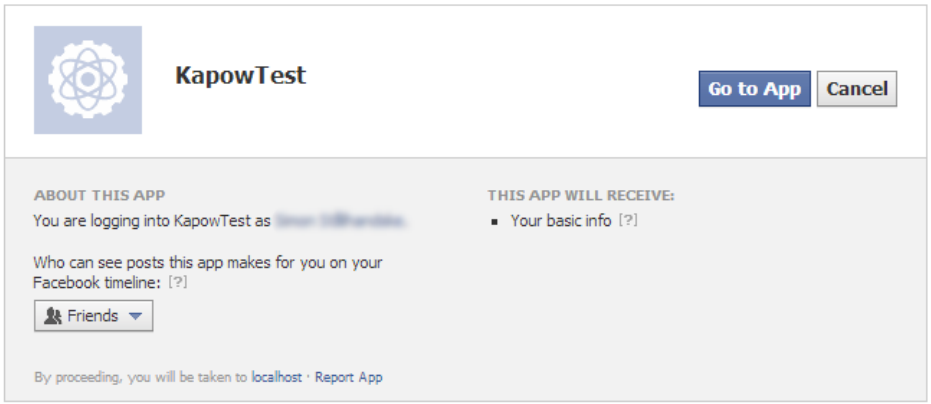
ユーザーの名前を選択して「次」をクリックします。



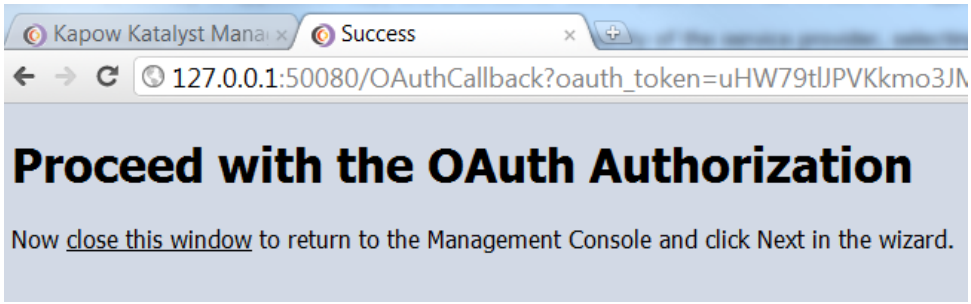
ここで、認証リンクをクリックします。



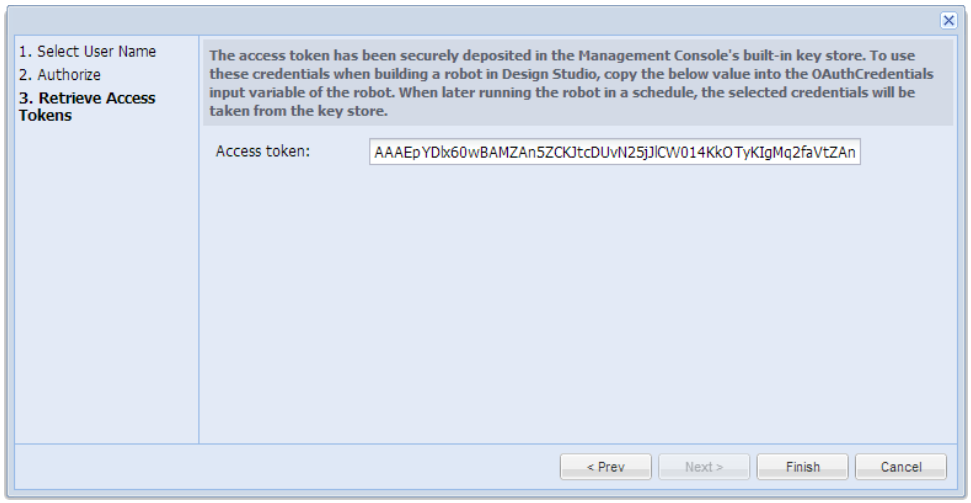
これによって、アプリケーションがユーザーのFacebookアカウントに対するアクセス権限をリクエストするページが開きます。



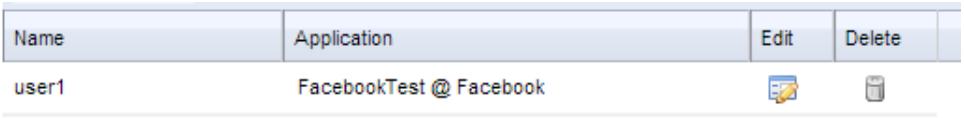
アプリケーションが認証されると、Management ConsoleのOAuthコールバックのページに転送されます。



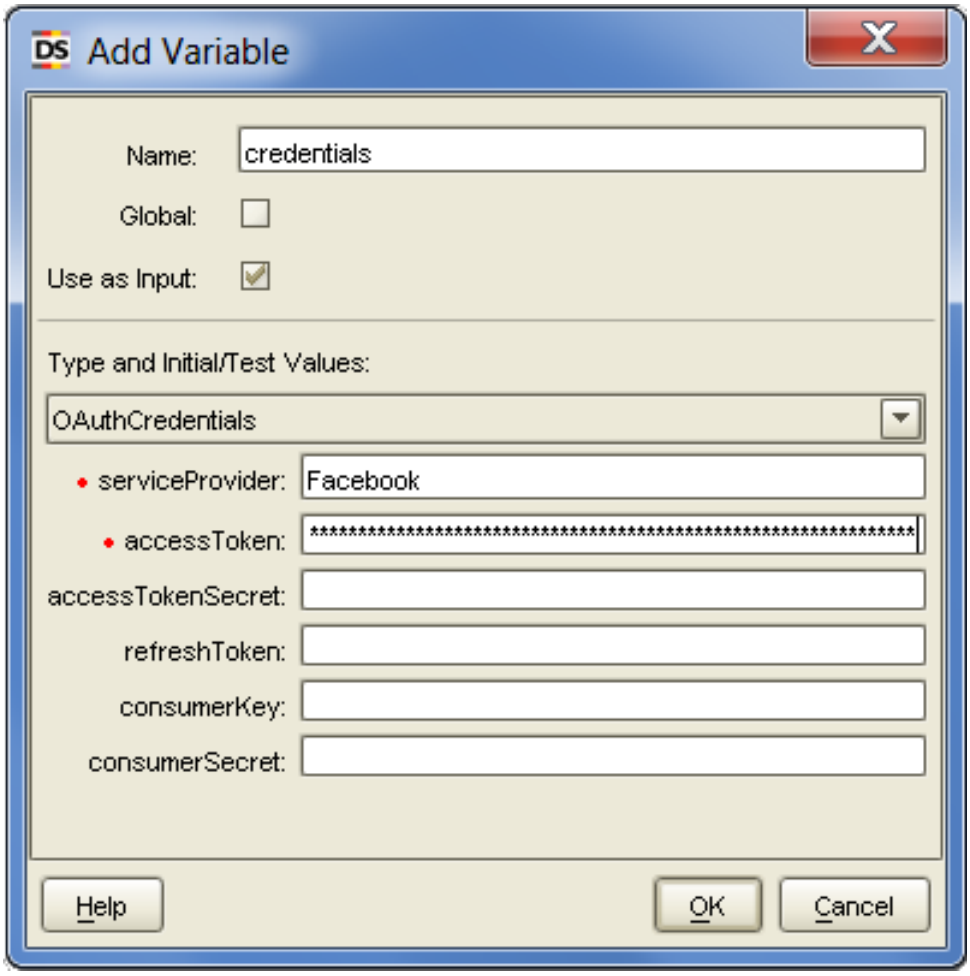
OAuthコールバックのページを閉じて、Management Consoleに戻ります。ウィザードの「次」をクリックすると、ユーザーのかわりにFacebookにアクセスするために使用できるアクセス・トークンが表示されます。このトークンは、セキュリティ上の理由から後でアクセスできないため、メモ帳にコピーしておきます。



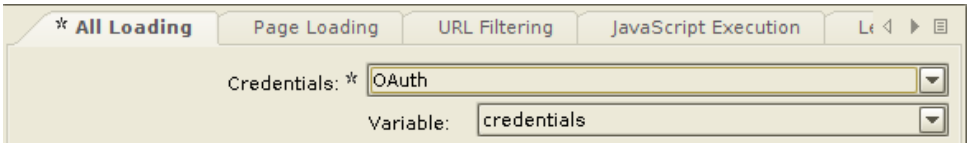
「終了」をクリックすると、OAuthタブの「ユーザー」セクションにユーザーが表示されます。



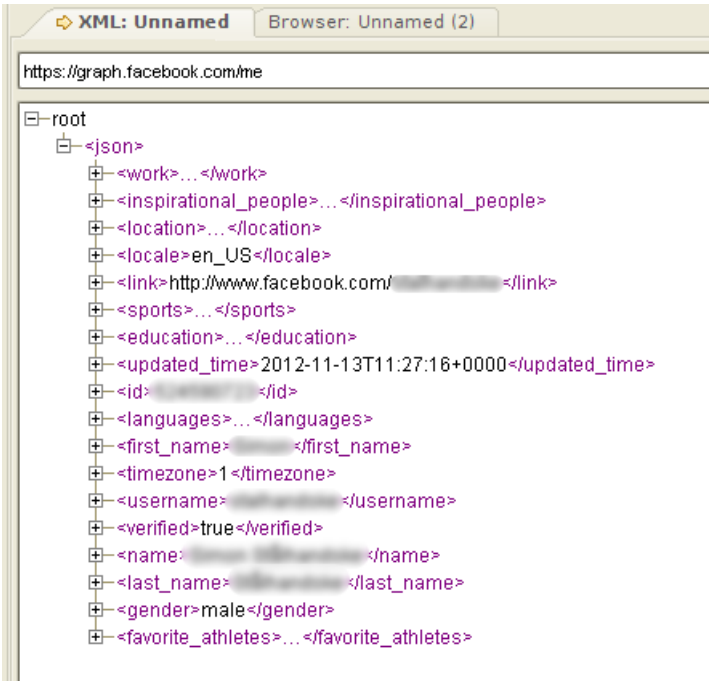
ここで、Design Studioを開いて新しいロボットを作成するか、FacebookのAPIにアクセスするロボットを開きます。ロボットがFacebookのGraph APIを使用できるようにするには、サービス・プロバイダ名として「Facebook」を、および以前にManagement Consoleからコピーしたアクセス・トークンが含まれた、複雑なタイプの「OAuthCredentials」の入力変数を確実に設定します。



次に、OAuthの資格証明を使用するロボットを確実に構成します。



これでロボットは、FacebookのGraph APIにアクセスできます。たとえば、ブラウザ・ビューのアドレス・バーに「**https://graph.facebook.com/me**」と入力して、[Enter]を押します。

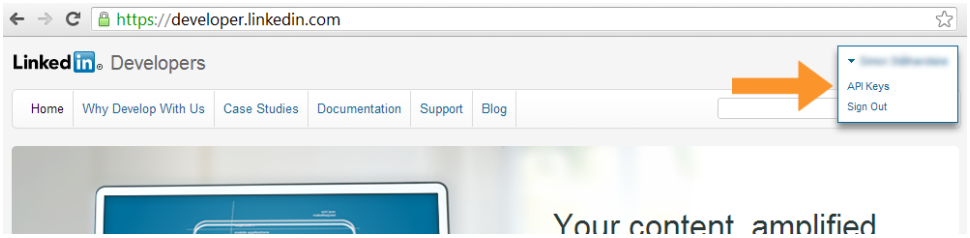


FacebookのGraph APIの詳細を学習するには、https://developers.facebook.com/docs/getting-started/graphapi/を参照してください。ロボットを作成した後は、Management Consoleでロボットの実行をスケジュールできます。

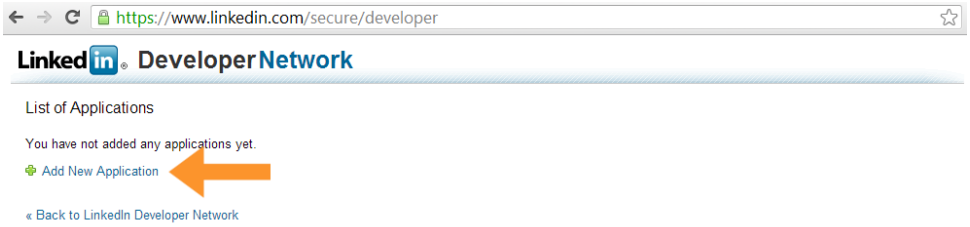
LinkedIn

これは、LinkedInアカウントに対する読取りおよび書込みに使用できるREST APIを介して、LinkedInにアクセスするロボットの作成方法に関する簡単なガイドです。

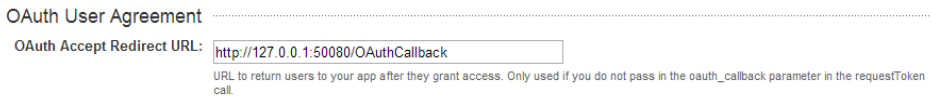
最初に、LinkedInのアカウントを使用してhttp://developer.linkedin.comにログインするか、必要に応じて新しいアカウントを作成します。右上隅にあるユーザーの名前の上にマウス・ポインタを重ねて、表示されるメニューで「API Keys」をクリックします。



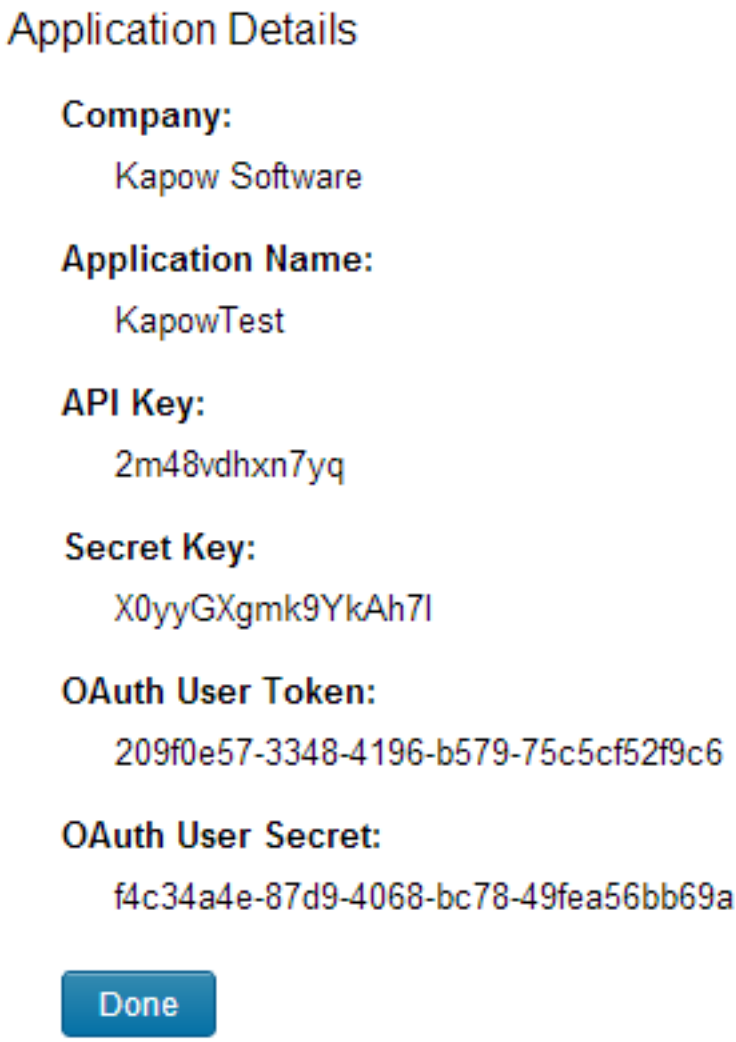
次に、「Add New Application」をクリックします。



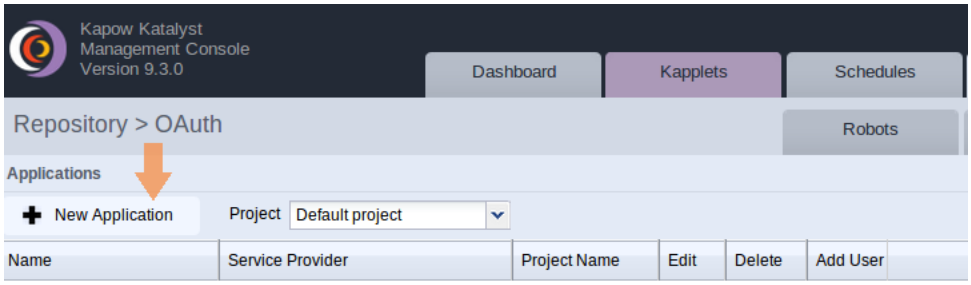
新しいアプリケーションの構成画面が表示されます。関連する情報をフォームに入力します。フィールドの1つに「OAuth Accept Redirect URL」と呼ばれるフィールドがあります。このフィールドには、OAuthコールバックURLを記述します。コールバックURLは、Management Consoleがデプロイされているフォルダの下にある**OAuthCallback**へのパスです。たとえば、埋込みManagement Consoleで実行する場合は、**http://localhost:50080/**（または、同様に**http://127.0.0.1:50080/**）で実行します。後者の場合は、コールバックURLを**http://127.0.0.1:50080/OAuthCallback**に指定します。



LinkedInのAPIの利用規約を読んで受諾し、アプリケーションを追加すると、アプリケーションの詳細を含むページが開きます。このページでは、後でManagement Consoleでコンシューマ・キーおよびコンシューマ・シークレットとして使用するAPIキーおよびシークレット・キーをメモしておくことが重要です。ページには、ユーザー・トークンおよびユーザー・シークレットも表示されますが、これらは独自に作成するため無視できます。

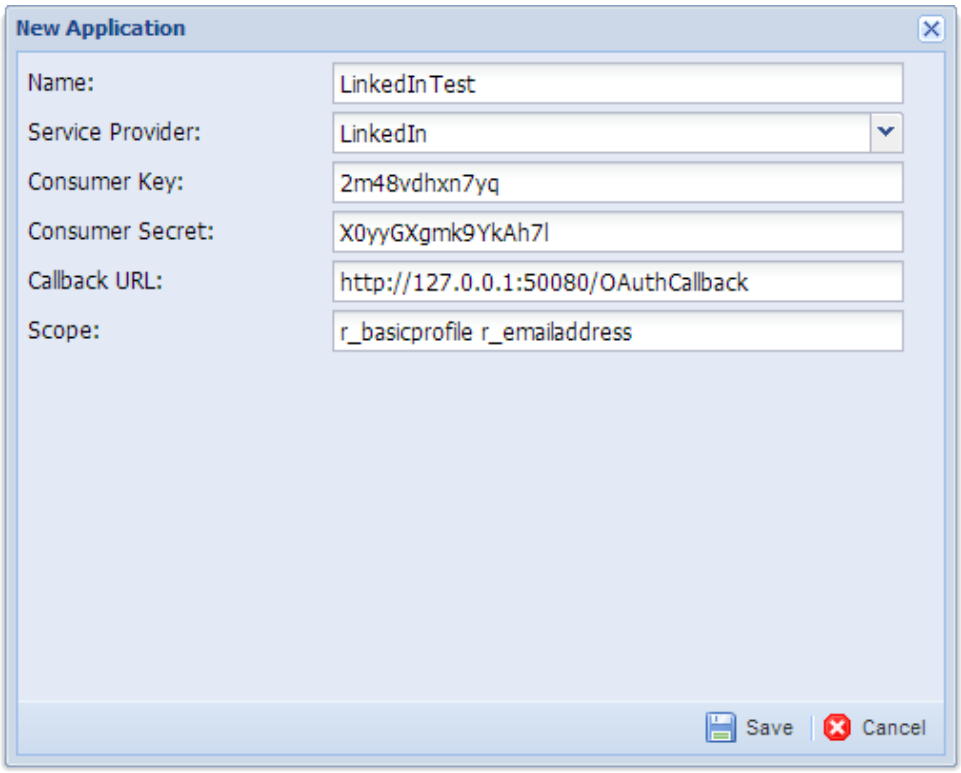


ここで、Management Consoleを開きます。アプリケーションに対して指定したURLでManagement Consoleが開くことを確認してください。前述で使用した情報では、これは**http://127.0.0.1:50080**です。次に、OAuthのリポジトリにナビゲートして「新規アプリケーション」をクリックします。

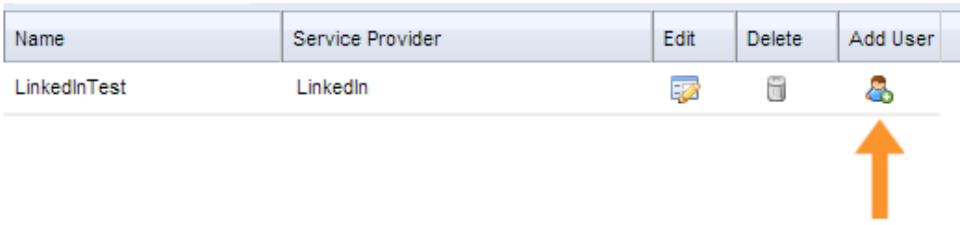


新しいアプリケーションの名前を選択し、サービス・プロバイダとしてLinkedInを選択します。コンシューマ・キーおよびコンシューマ・シークレットには、以前にメモしたAPIキーおよびシークレット・キーをそれぞれ設定します。コールバックURLは、すでに正しく記述されています。

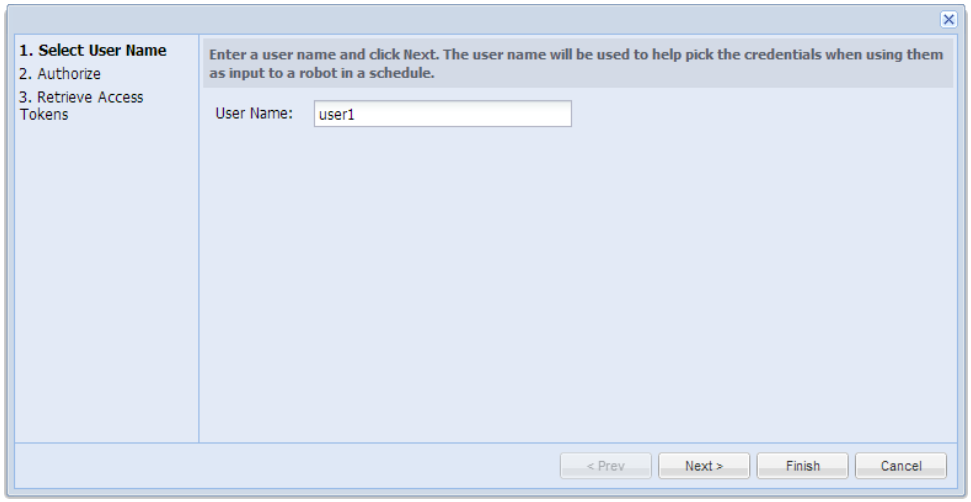
スコープ・フィールドには、アプリケーション・ユーザーから要求された権限の種類を指定します。次の図では、基本的なプロファイル情報の読取り権限を付与する「r_basicprofile」、およびユーザーの電子メール・アドレスへのアクセス権限を付与する「r_emailadress」の2つの異なる権限がリクエストされています。スコープ・オプションの詳細は、https://developer.linkedin.com/documents/authentication#grantingを参照してください。



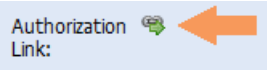
保存すると、アプリケーションがリポジトリに表示されます。「ユーザーの追加」の下にあるアイコンをクリックして、アプリケーションにユーザーを追加します。



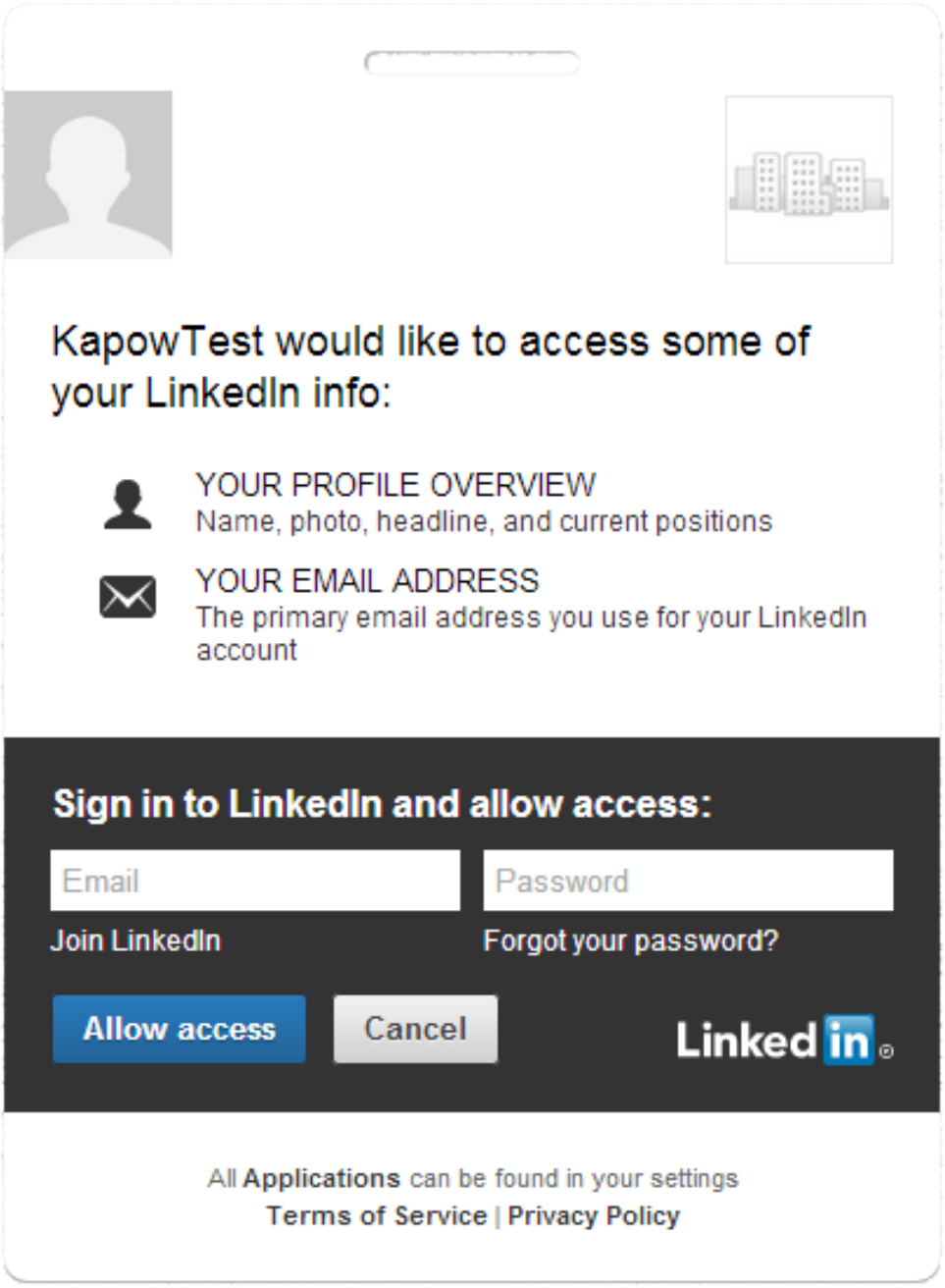
ユーザーの名前を選択して「次」をクリックします。



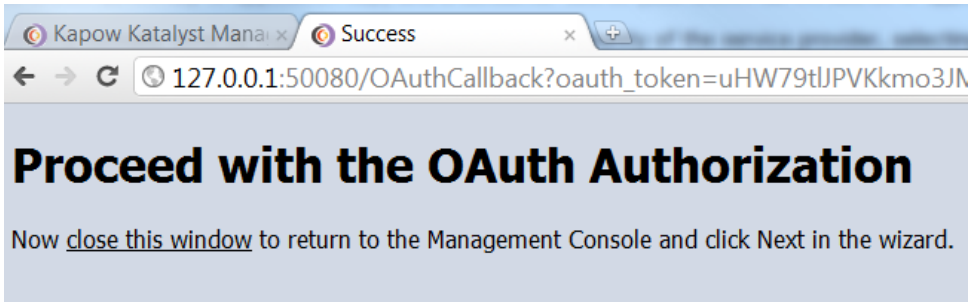
ここで、認証リンクをクリックします。



これによって、アプリケーションがユーザーのLinkedInアカウントに対するアクセス権限をリクエストするページが開くため、リクエストされている2つの権限を指定します。

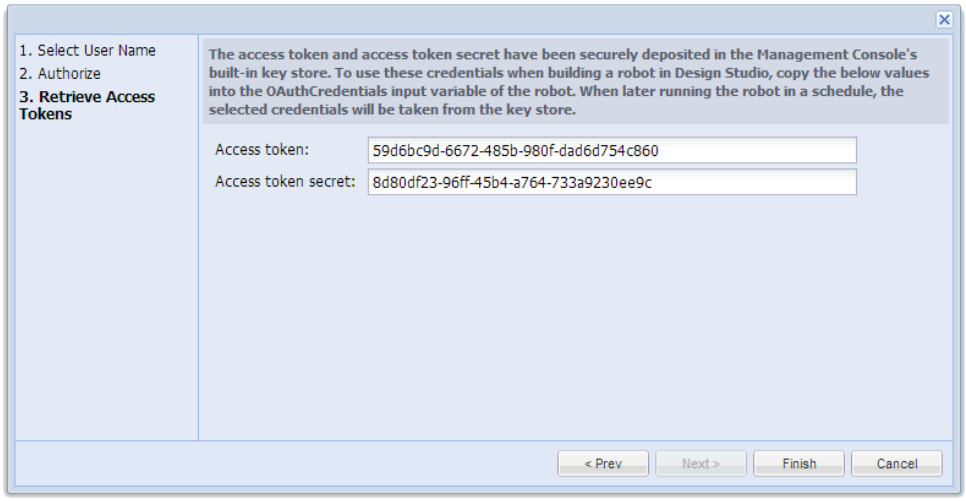


アプリケーションが認証されると、Management ConsoleのOAuthコールバックのページに転送されます。

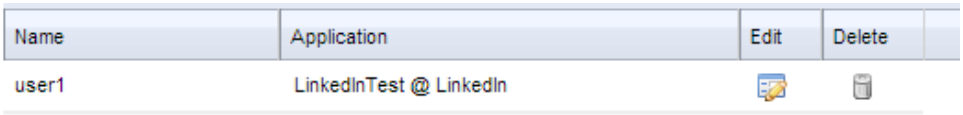


OAuthコールバックのページを閉じて、Management Consoleに戻ります。ウィザードの「次」をクリックすると、ユーザーのかわりにLinkedInにアクセスする権限が付与された、アクセス・トークン

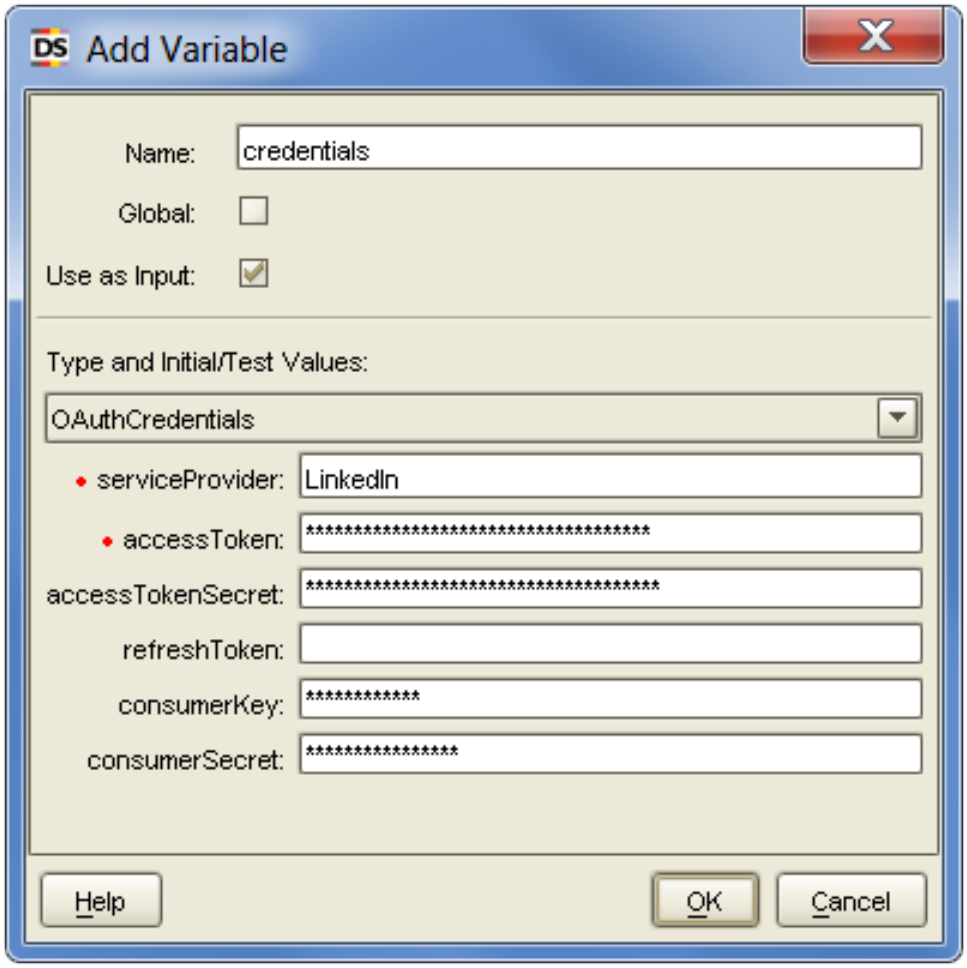
およびアクセス・トークン・シークレットが表示されます。これらのトークンは、セキュリティ上の理由から後でアクセスできないため、メモ帳にコピーしておきます。



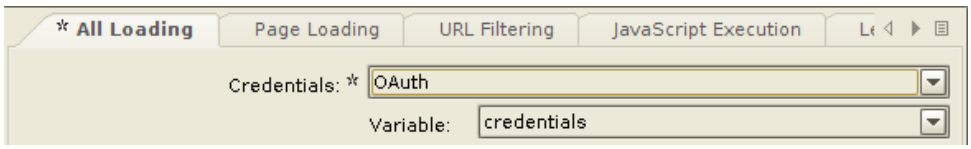
「終了」をクリックすると、OAuthタブの「ユーザー」セクションにユーザーが表示されます。



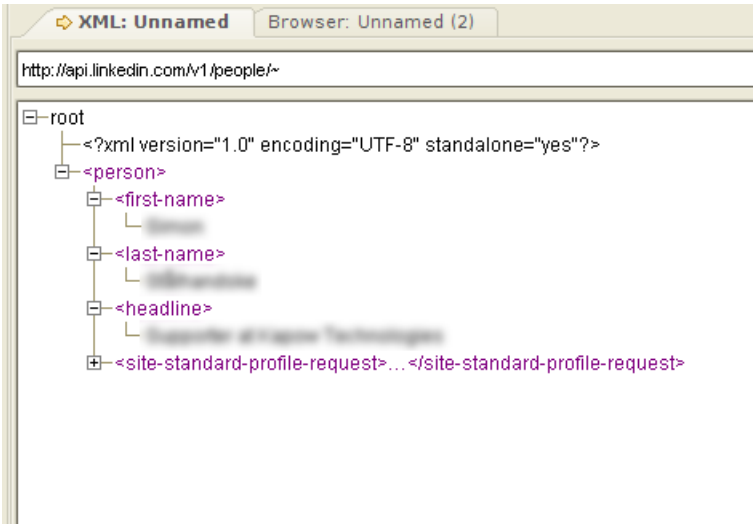
ここで、Design Studioを開いて新しいロボットを作成するか、LinkedInのAPIにアクセスするロボットを開きます。ロボットがLinkedInのAPIを使用できるようにするには、複雑なタイプの「OAuthCredentials」の入力変数を確実に設定します。変数には、サービス・プロバイダ名として「LinkedIn」を指定する必要があります。また、アクセス・トークンおよびアクセス・トークン・シークレットとともに、以前にメモしたAPIキーおよびシークレット・キーを入力します。後者の2つは、コンシューマ・キーおよびコンシューマ・シークレットとしてそれぞれ使用します。



次に、OAuthの資格証明を使用するロボットを確実に構成します。



これでロボットは、LinkedInのAPIにアクセスできます。たとえば、ブラウザ・ビューのアドレス・バーに「**http://api.linkedin.com/v1/people/~**」と入力して、[Enter]を押します。これによって、基本的なユーザー情報が含まれたxmlページが返されます。また、「**http://api.linkedin.com/v1/people/~/email-address**」を試行すると、ユーザーの電子メール・アドレスが返されます。

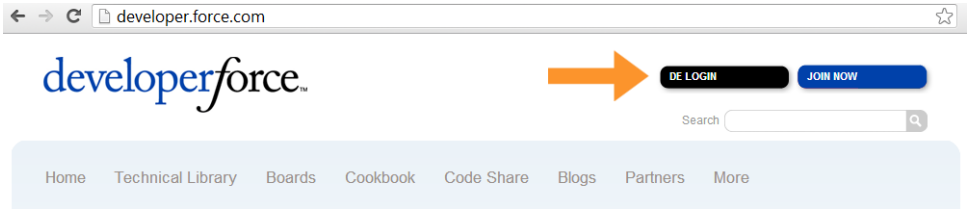


LinkedInのREST APIの詳細を学習するには、https://developer.linkedin.com/restを参照してください。ロボットを作成した後は、Management Consoleでロボットの実行をスケジュールできます。

Salesforce

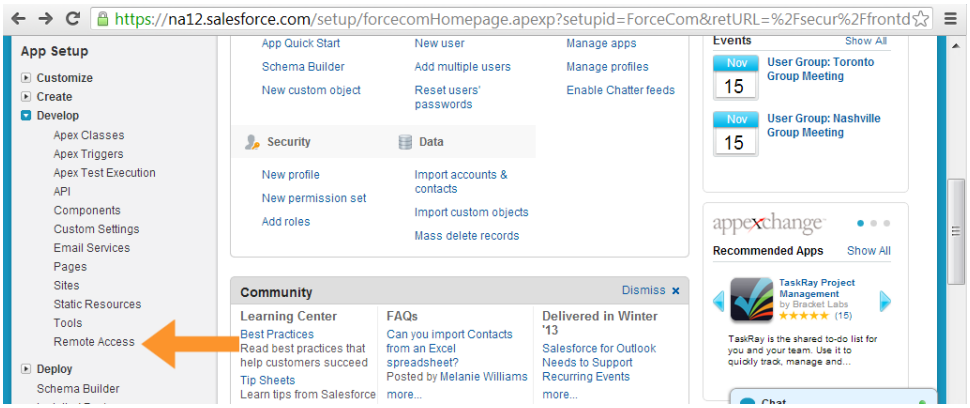
これは、REST APIおよびOAuthを使用してSalesforceにアクセスするロボットの作成方法に関する簡単なガイドです。

最初に、http://developer.force.comでページの上部にある「DE LOGIN」をクリックして、Salesforceアカウントを使用してログインします。必要な場合は、新しいアカウントを作成します。

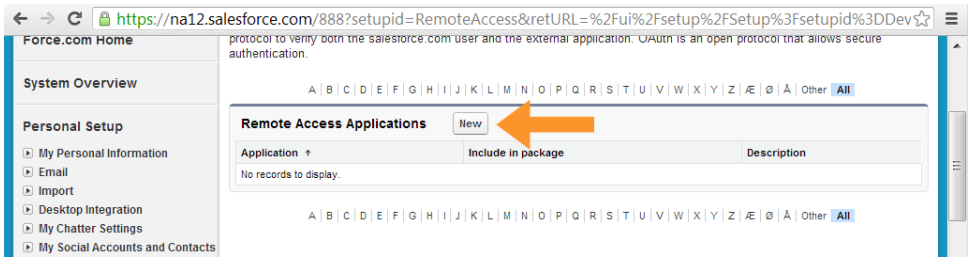


ユーザーのアカウントのホーム・ページで、アカウントを実行するインスタンスをメモしてください。インスタンスは、URLのサーバー名の部分に表示されます。次に示す例では、インスタンスは「na12」です。

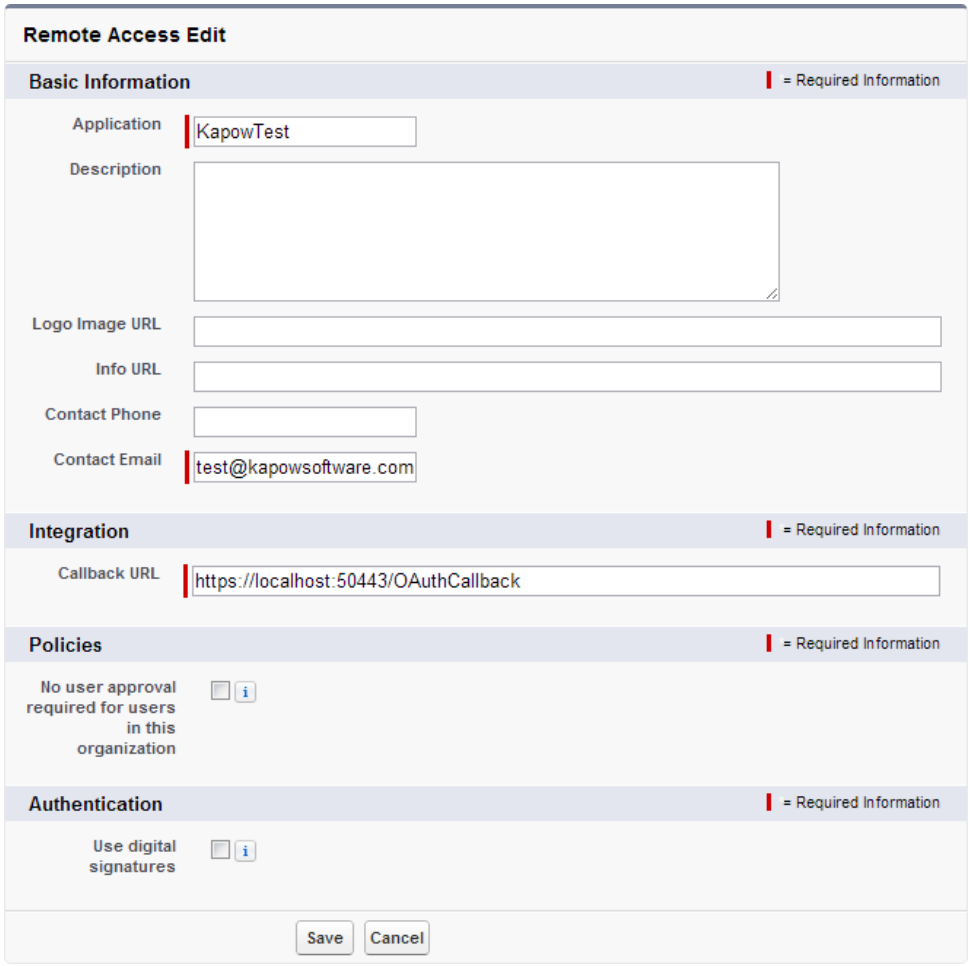
左側のメニューから「Remote Access」選択します。これは、「App Setup」>>「Develop」の下にあります。



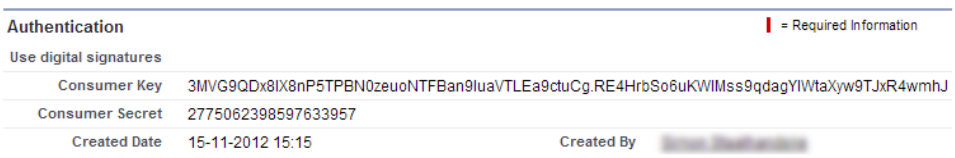
ここで、新しいリモート・アクセス・アプリケーションを作成します。このアプリケーションをOAuthを介してSalesforceのAPIにアクセスできるようにします。



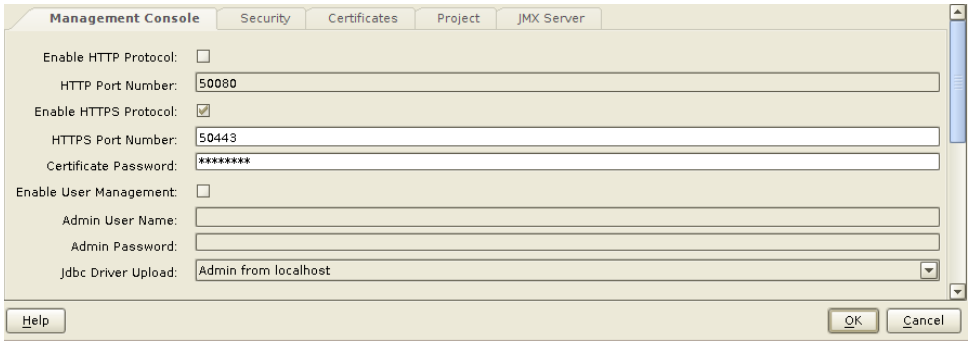
新しいアプリケーションの構成画面が表示されます。関連する情報をフォームに入力します。フィールドの1つに「Callback URL」と呼ばれるフィールドがあります。このフィールドには、OAuthコールバックURLを記述します。Salesforceでは、コールバックURLを**https://localhost:50443/OAuthCallback**に転送するには、HTTPSを使用する必要があります。すぐに気付くように、**50443**は、HTTPSを使用する埋込みManagement Consoleの設定時のデフォルトのポートです。



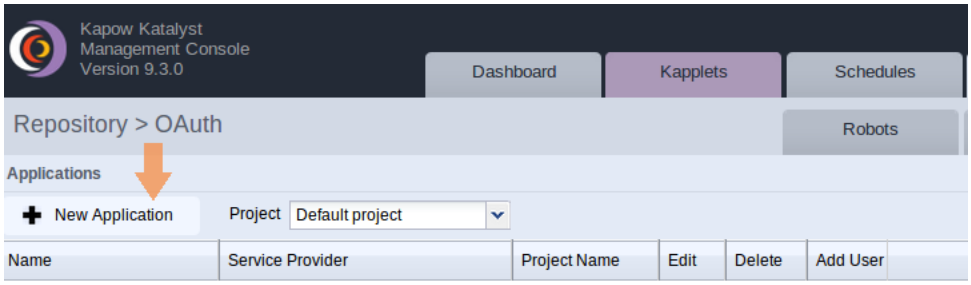
「save」をクリックすると、アプリケーションの詳細が含まれたページが開きます。このページでは、後でManagement Consoleで使用するコンシューマ・キーおよびコンシューマ・シークレットをメモしておくことが重要です。



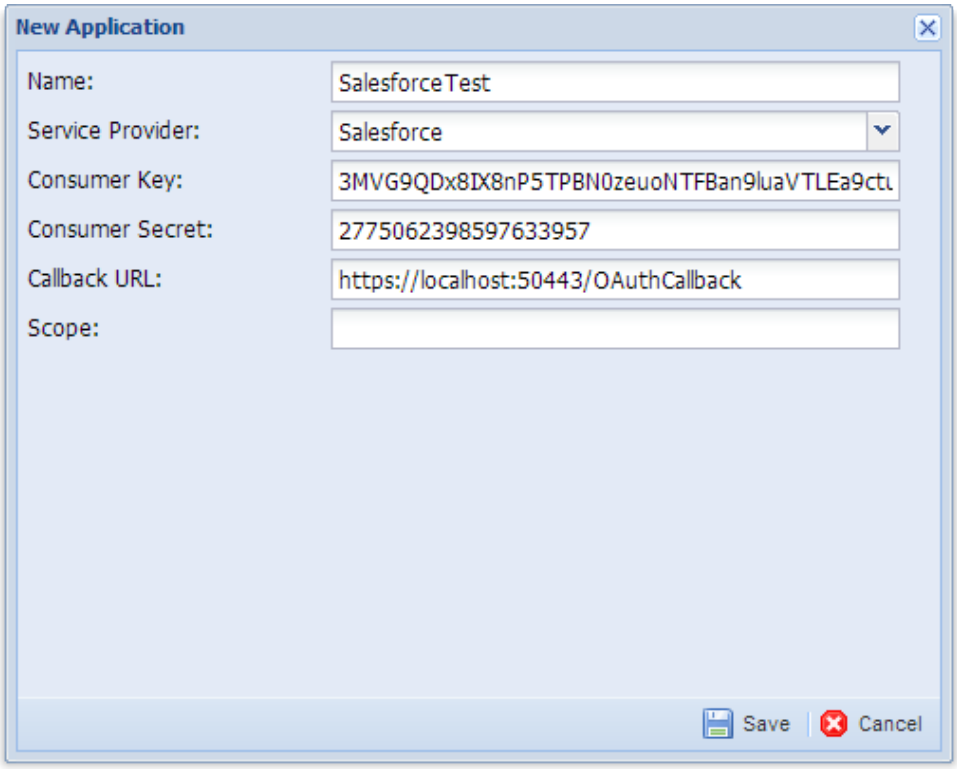
HTTPSプロトコルを使用するローカルのManagement Consoleを設定するために、Settingsを開きます。Windowsの場合、Settingsは、「スタート」メニューのDesign Studioと同じフォルダにあります。Settingsで「Management Console」タブに移動し、HTTPプロトコルを無効にして、かわりにHTTPSを有効にします。



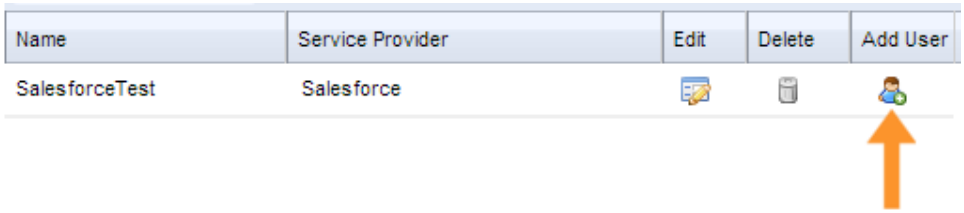
「OK」をクリックして、Management Consoleを起動または再起動します。起動後に、Management Consoleを開きます。アプリケーションに対して指定したURLでManagement Consoleが開くことを確認してください。前述で使用した情報では、これは**https://localhost:50443**です。次に、OAuthのリポジトリにナビゲートして「新規アプリケーション」をクリックします。



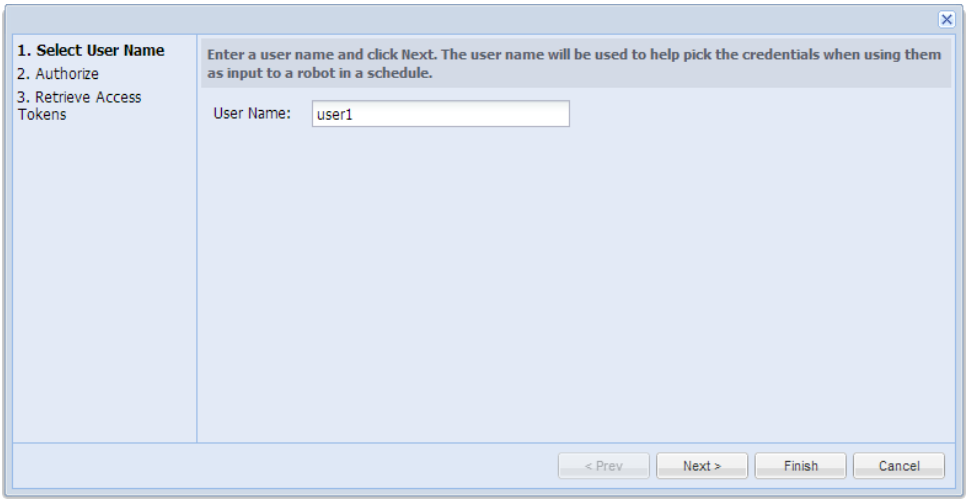
新しいアプリケーションの名前を選択し、サービス・プロバイダとしてSalesforceを選択します。以前にメモしたコンシューマ・キーおよびコンシューマ・シークレットを入力します。コールバックURLは、すでに正しく記述されています。アプリケーションがリクエストするユーザー権限を指定する場合は、https://help.salesforce.com/help/doc/en/remoteaccess_oauth_scopes.htmに示されているパラメータを使用して、スコープ・フィールドに入力できます。複数のスコープ・パラメータは空白で区切ります。



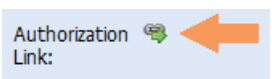
保存すると、アプリケーションがリポジトリに表示されます。「ユーザーの追加」の下にあるアイコンをクリックして、アプリケーションにユーザーを追加します。



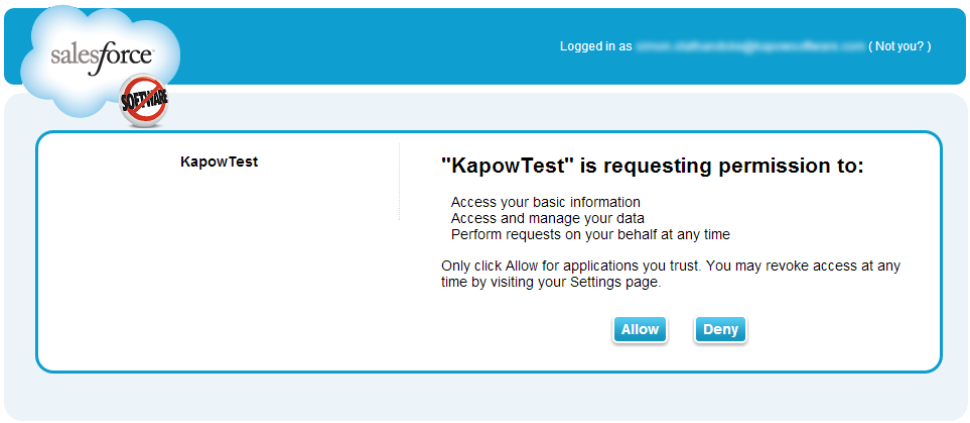
ユーザーの名前を選択して「次」をクリックします。



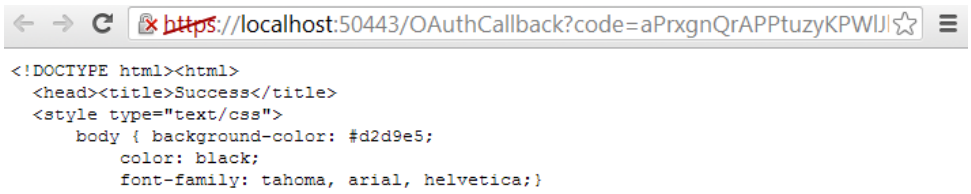
ここで、認証リンクをクリックします。



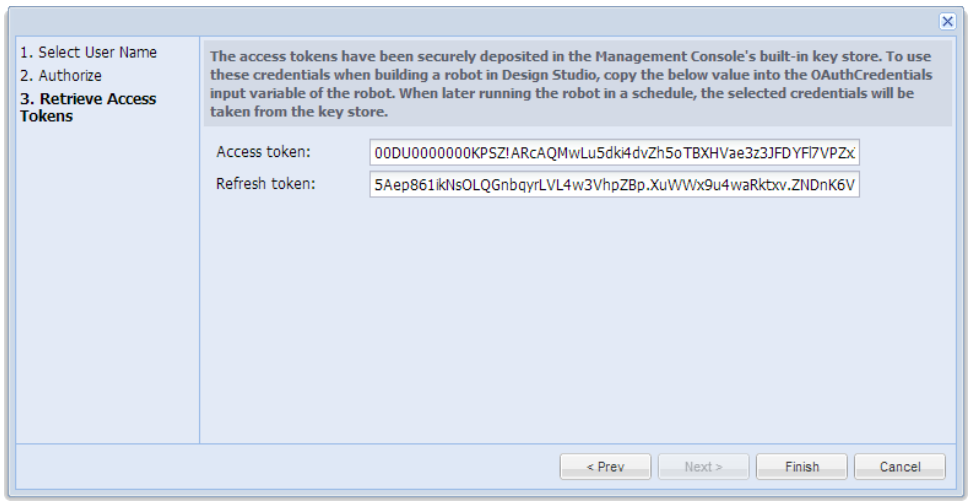
これによって、アプリケーションがユーザーのSalesforceアカウントに対するアクセス権限をリクエストするページが開くため、リクエストされている特別な権限を指定します。デフォルトでリクエストされる権限を次に示します。



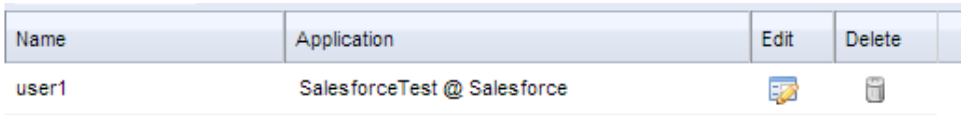
アプリケーションが認証されると、Management ConsoleのOAuthコールバックのページに転送されます。現時点では、HTTPSバージョンのManagement Consoleには、コールバックのページがスクランブルする不具合があります。ただし、これは認証プロセスには影響を与えません。



OAuthコールバックのページを閉じて、Management Consoleに戻ります。ウィザードの「次」をクリックすると、ユーザーのかわりにSalesforceにアクセスする権限が付与された、アクセス・トークンおよびリフレッシュ・トークンが表示されます。これらのトークンは、セキュリティ上の理由から後でアクセスできないため、メモ帳にコピーしておきます。

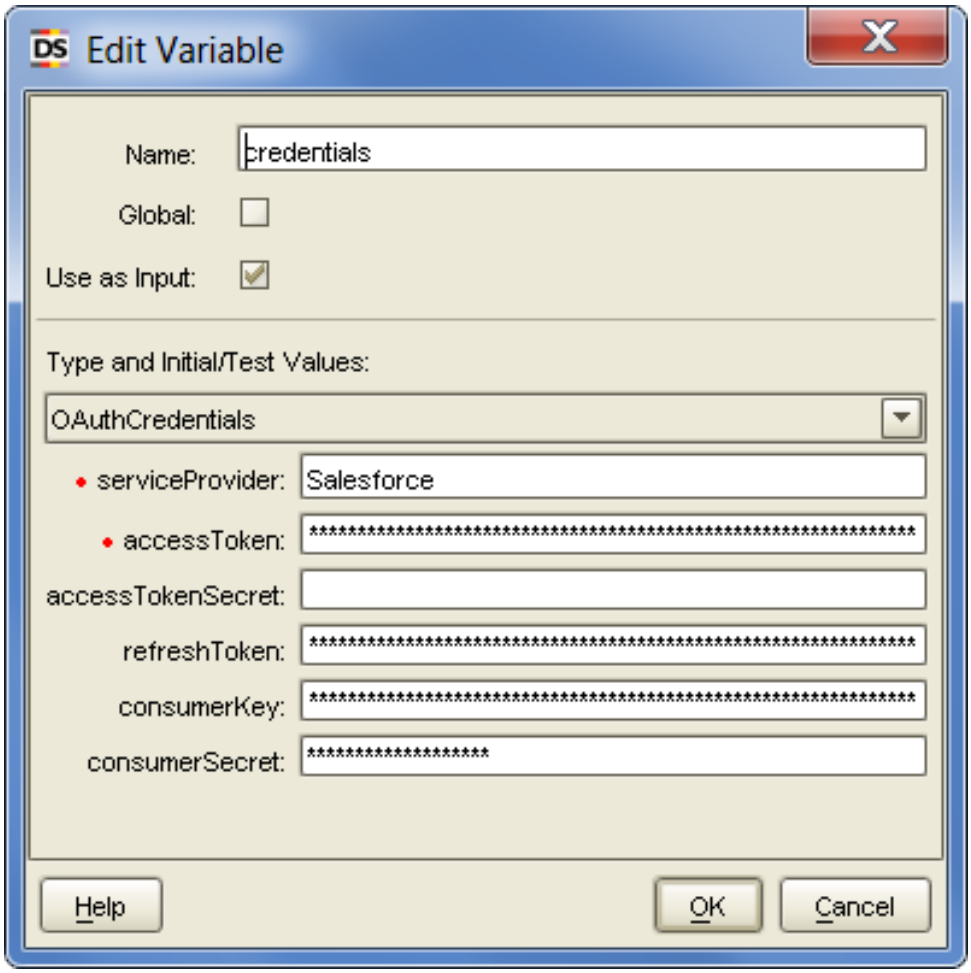


「終了」をクリックすると、OAuthタブの「ユーザー」セクションにユーザーが表示されます。

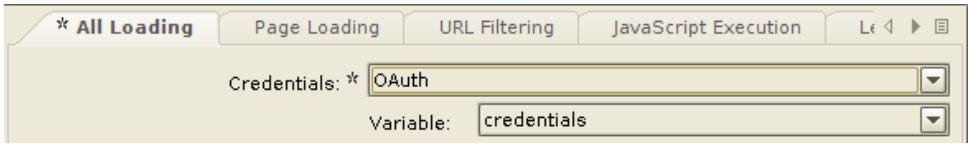


ここで、Design Studioを開いて新しいロボットを作成するか、SalesforceのAPIにアクセスするロボットを開きます。ロボットがSalesforceのAPIを使用できるようにするには、複雑なタイプの「OAuthCredentials」の入力変数を確実に設定します。変数には、サービス・プロバイダ名として

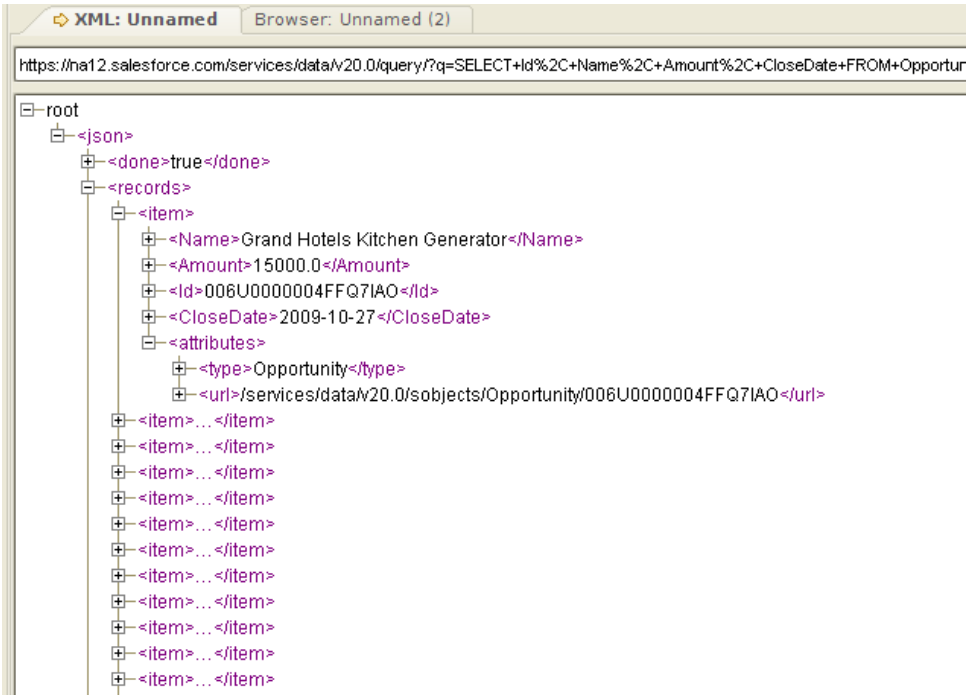
「Salesforce」を指定する必要があります。また、アクセス・トークンおよびリフレッシュ・トークンとともに、以前にメモしたコンシューマ・キーおよびコンシューマ・シークレットを入力します。



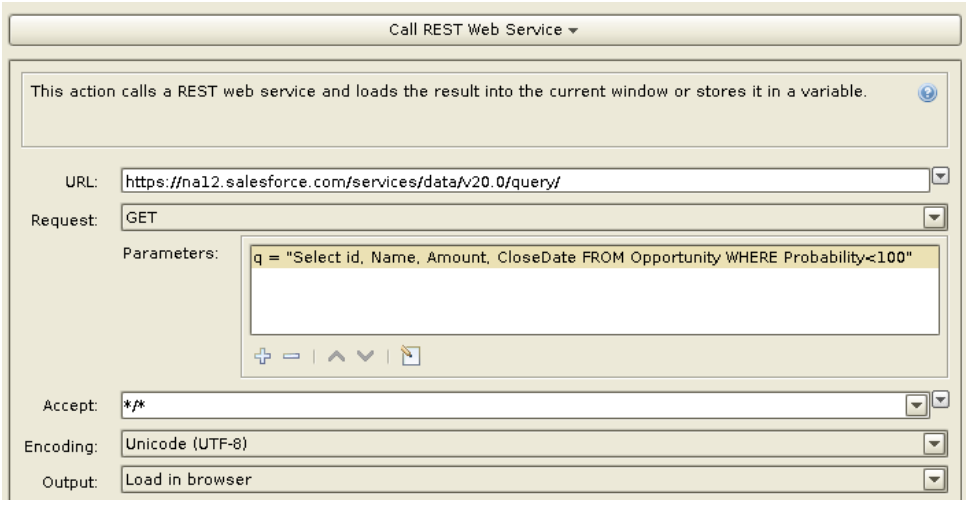
次に、OAuthの資格証明を使用するロボットを確実に構成します。



これでロボットは、SalesforceのAPIにアクセスできます。たとえば、ブラウザ・ビューのアドレス・バーに**https://instance.salesforce.com/services/data/v20.0/query/?q=SELECT+Id%2C+Name%2C+Amount%2C+CloseDate+FROM+Opportunity+WHERE+Probability%3C100**をコピーし、"instance"を、事前に確定したユーザーのインスタンスに置換し、[Enter]を押します。これによって、商談(opportunities)が含まれたJSONページが返されます。



SalesforceのREST APIの詳細を学習するには、http://www.salesforce.com/us/developer/docs/api_rest/を参照してください。SalesforceではREST APIを使用するため、「REST Webサービスの呼出し」のステップを使用してこのAPIにアクセスできます。これは、前述の例で使用されている問合せなどのパラメータの指定が容易になるため、特に便利です。

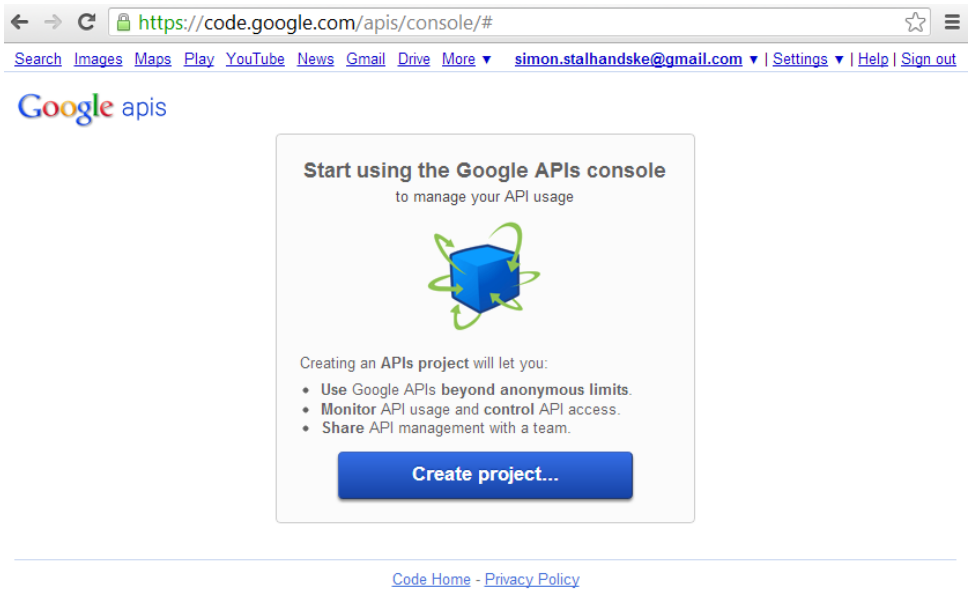


前述の指定した問合せはSalesforce Object Query Language (SOQL)で、SalesforceのAPIへのアクセス時に非常に便利です。SOQLに関するドキュメントは、http://www.salesforce.com/us/developer/docs/soql_sosl/index_Left.htmを参照してください。ロボットを作成した後は、Management Consoleでロボットの実行をスケジュールできます。

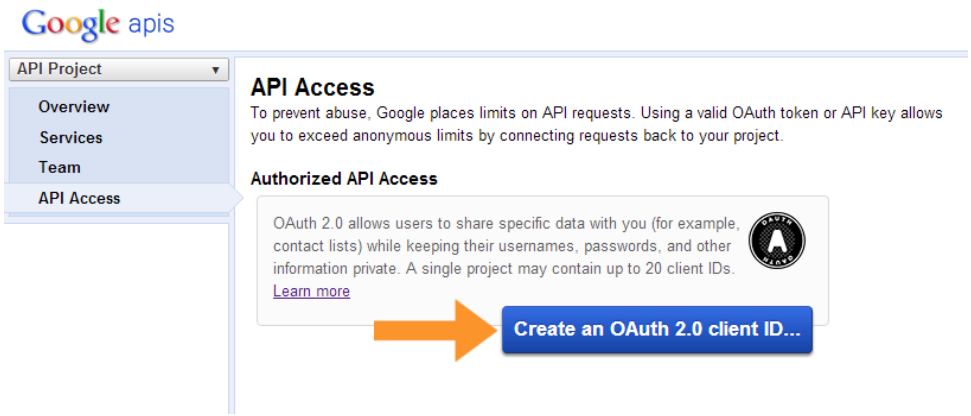
Google

これは、Googleによるサービスに対する読取りおよび書込みに使用できる様々なGoogleのAPIにアクセスする、ロボットの作成方法に関する簡単なガイドです。これには、YouTube、Google Drive、Bloggerなどの多数のサービスが含まれます。

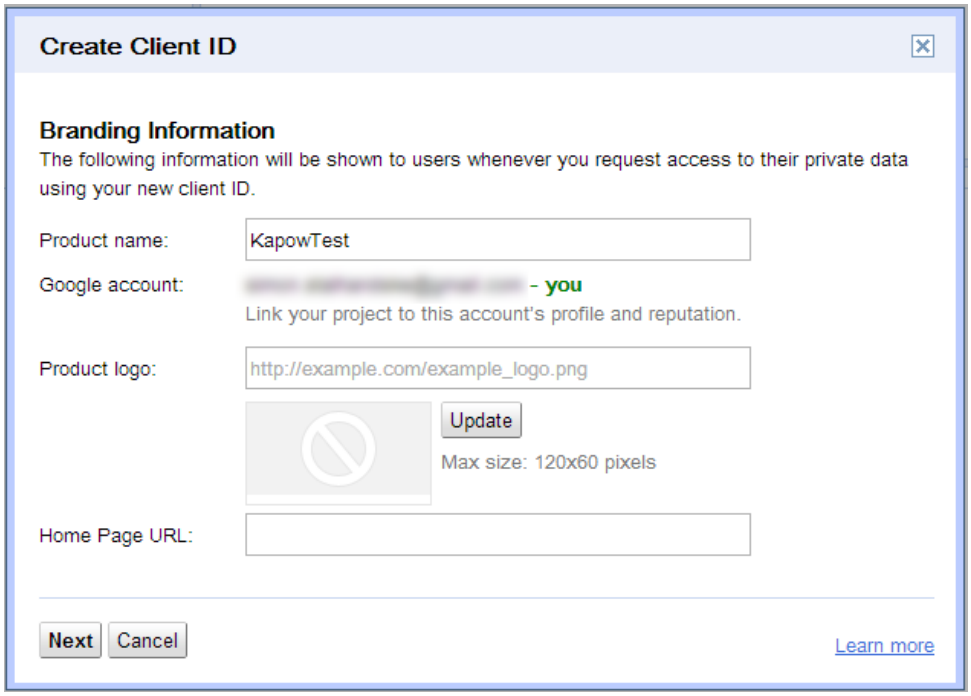
最初に、Googleのアカウントを使用してhttp://code.google.com/apis/consoleにログインするか、必要に応じて新しいアカウントを作成します。「Create Project...」をクリックして、新しいプロジェクトを作成します。



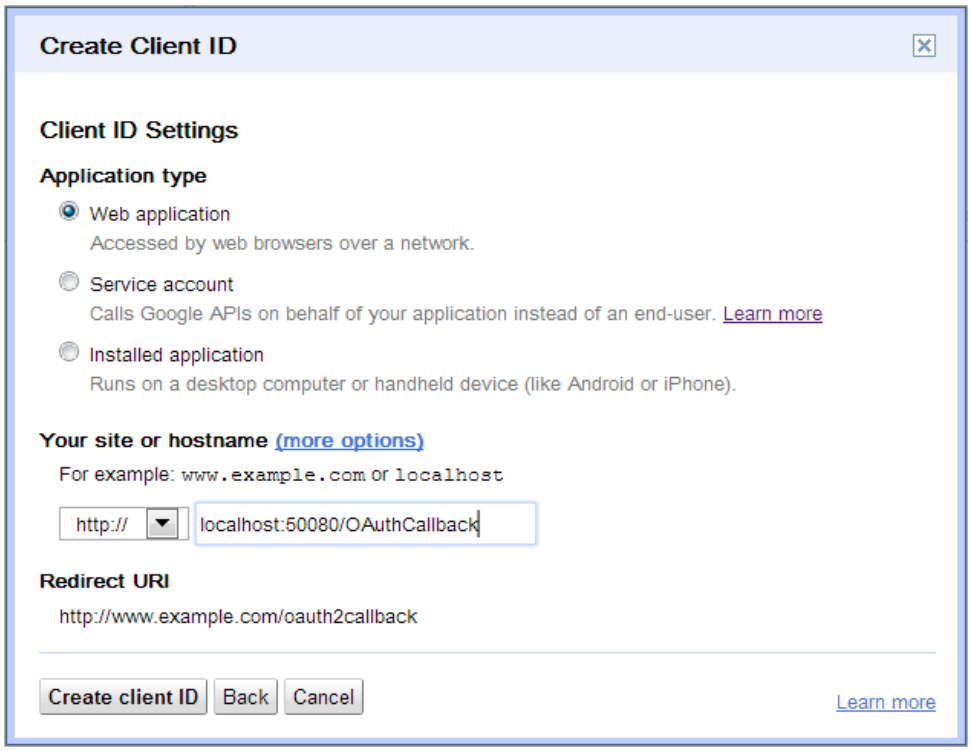
左側のメニューで「API Access」を選択し、次に「Create an OAuth 2.0 client ID...」クリックします。



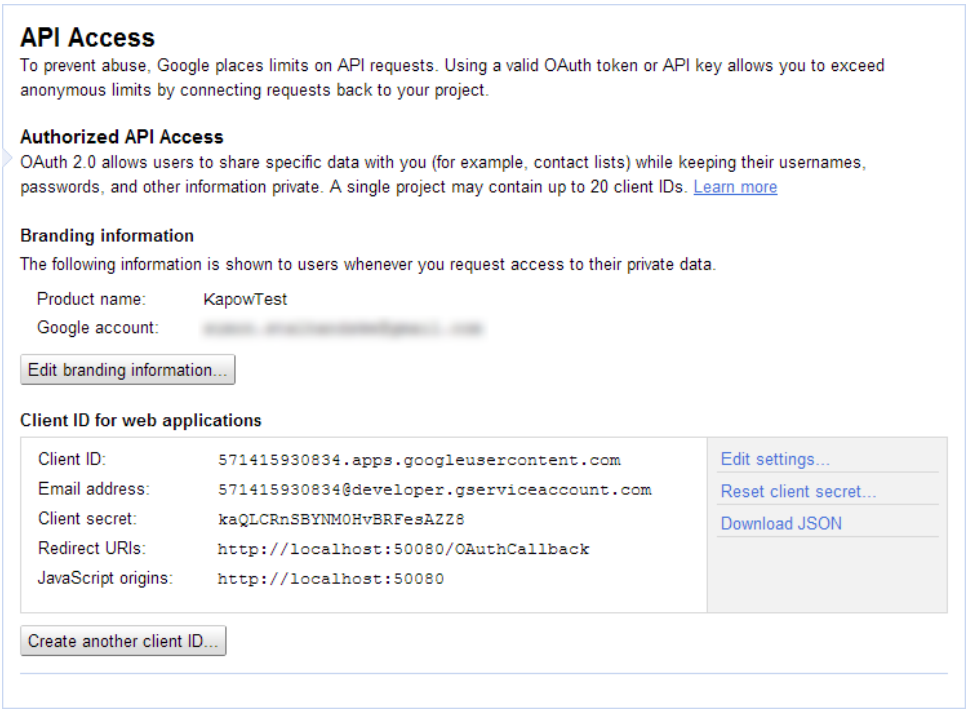
これで、構成画面が表示されます。関連情報を入力します(「product name」のみ必須です)。



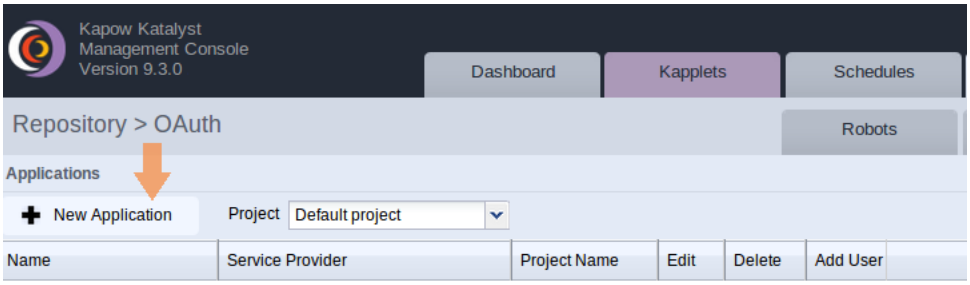
「next」をクリックすると、新しいフォームが表示されます。「Application type」に「Web Application」を選択し、「Your site or hostname」というラベルのフィールドにOAuthコールバックURLを記述します。コールバックURLは、Management Consoleがデプロイされているフォルダの下にある**OAuthCallback**へのパスです。たとえば、埋込みManagement Consoleで実行する場合は、**http://localhost:50080/**で実行するため、コールバックURLは**http://localhost:50080/OAuthCallback**です。コールバックURLの入力後、[Enter]を押します。フォームの下部にある「Redirect URL」が入力したURLに変更されます。Management Consoleで使用するプロトコルに応じて、ドロップダウン・メニューから「**http://**」または「**https://**」を選択することを忘れないでください。



「Create client ID」をクリックすると、アプリケーションの詳細が含まれたページが開きます。このページでは、後でManagement Consoleでコンシューマ・キーおよびコンシューマ・シークレットとして使用するクライアントIDおよびクライアント・シークレットをメモしておくことが重要です。

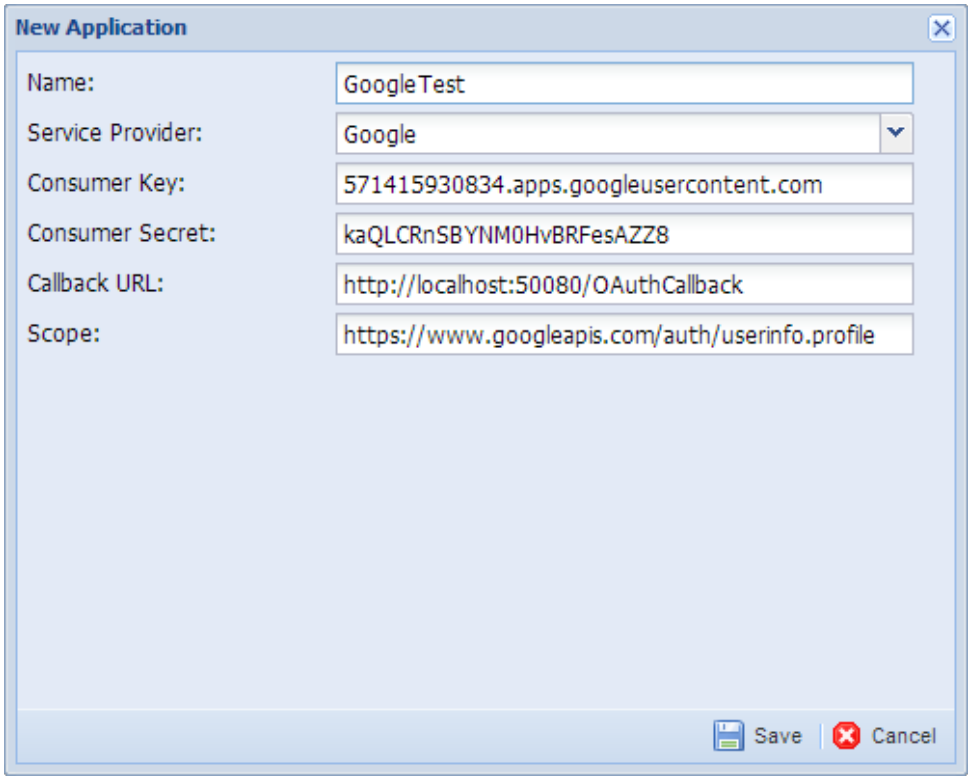


ここで、Management Consoleを開きます。アプリケーションに対して指定したURLでManagement Consoleが開くことを確認してください。前述で使用した情報では、これは**http://localhost:50080**です。次に、OAuthのリポジトリにナビゲートして「新規アプリケーション」をクリックします。

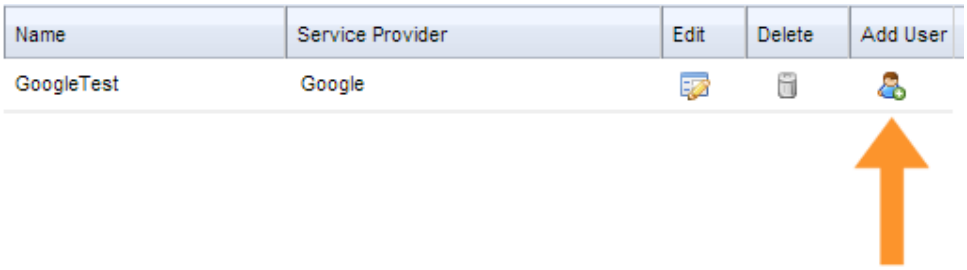


新しいアプリケーションの名前を選択し、サービス・プロバイダとしてGoogleを選択します。コンシューマ・キーおよびコンシューマ・シークレットには、以前にメモしたクライアントIDおよびクライアント・シークレットをそれぞれ設定します。コールバックURLは、すでに正しく記述されています。

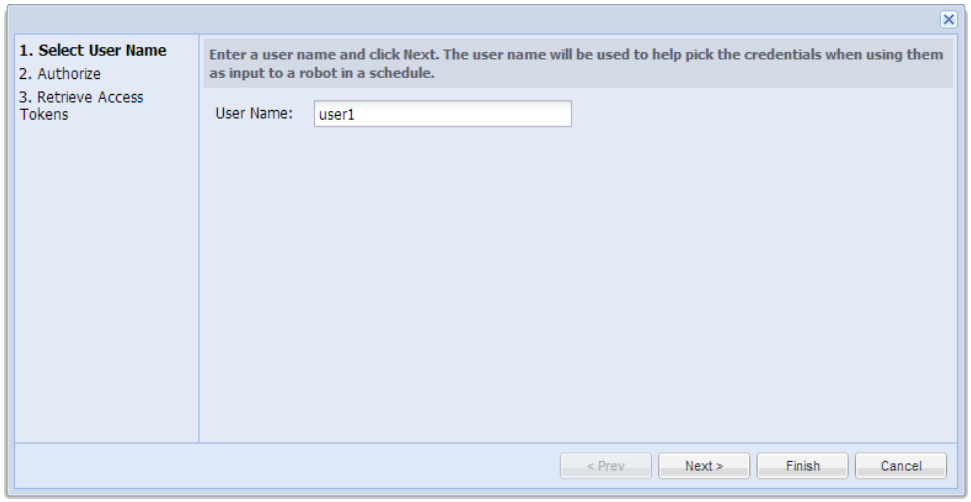
スコープ・フィールドには、アプリケーション・ユーザーから要求されたGoogleのAPI権限を指定します。次の図では、基本的なユーザー情報がリクエストされています。APIとそれぞれに関連付けられているスコープの総括的なリストは、「Google APIs Discovery Service」を参照してください。このページの下部に、2012年12月の時点で使用可能なAPIのリストの表を示します。この表では、各APIの説明、ドキュメントへのリンクおよびスコープURLを提供しています。



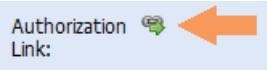
保存すると、アプリケーションがリポジトリに表示されます。「ユーザーの追加」の下にあるアイコンをクリックして、アプリケーションにユーザーを追加します。



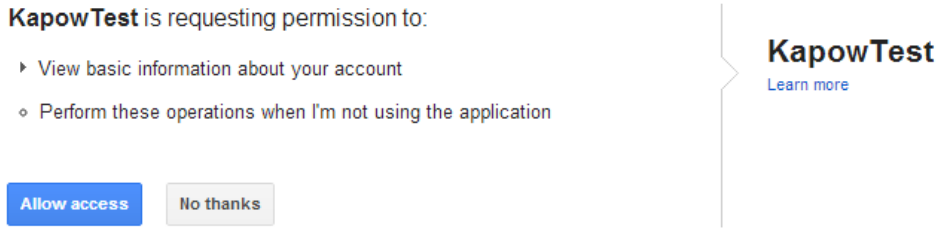
ユーザーの名前を選択して「次」をクリックします。



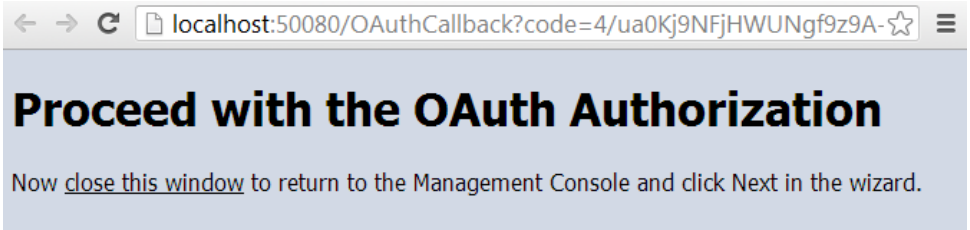
ここで、認証リンクをクリックします。



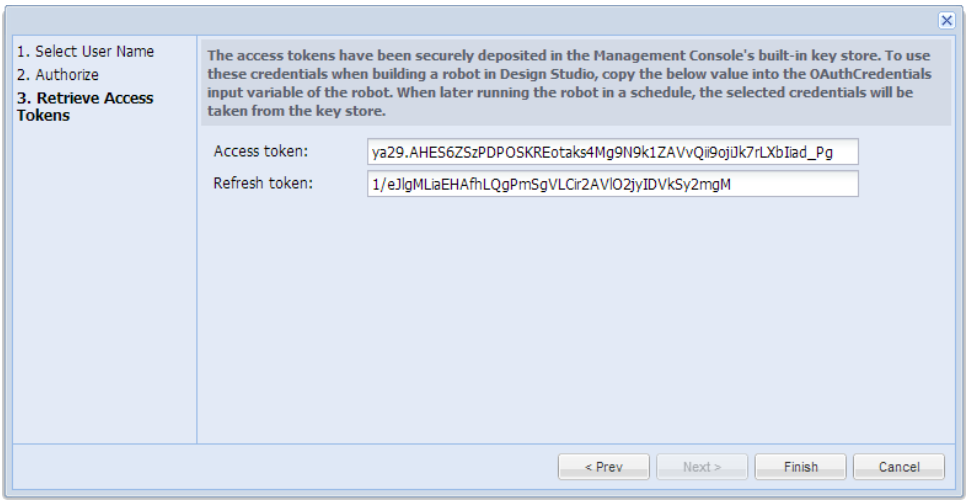
これによって、アプリケーションのスコープを介して指定した、ユーザーのGoogleアカウントの特定の側面に対するアクセス権限を、アプリケーションがリクエストするページが開きます。



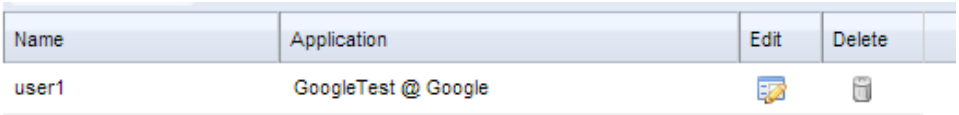
アプリケーションが認証されると、Management ConsoleのOAuthコールバックのページに転送されます。



OAuthコールバックのページを閉じて、Management Consoleに戻ります。ウィザードの「次」をクリックすると、ユーザーのかわりにGoogleにアクセスする権限が付与された、アクセス・トークンおよびリフレッシュ・トークンが表示されます。これらのトークンは、セキュリティ上の理由から後でアクセスできないため、メモ帳にコピーしておきます。

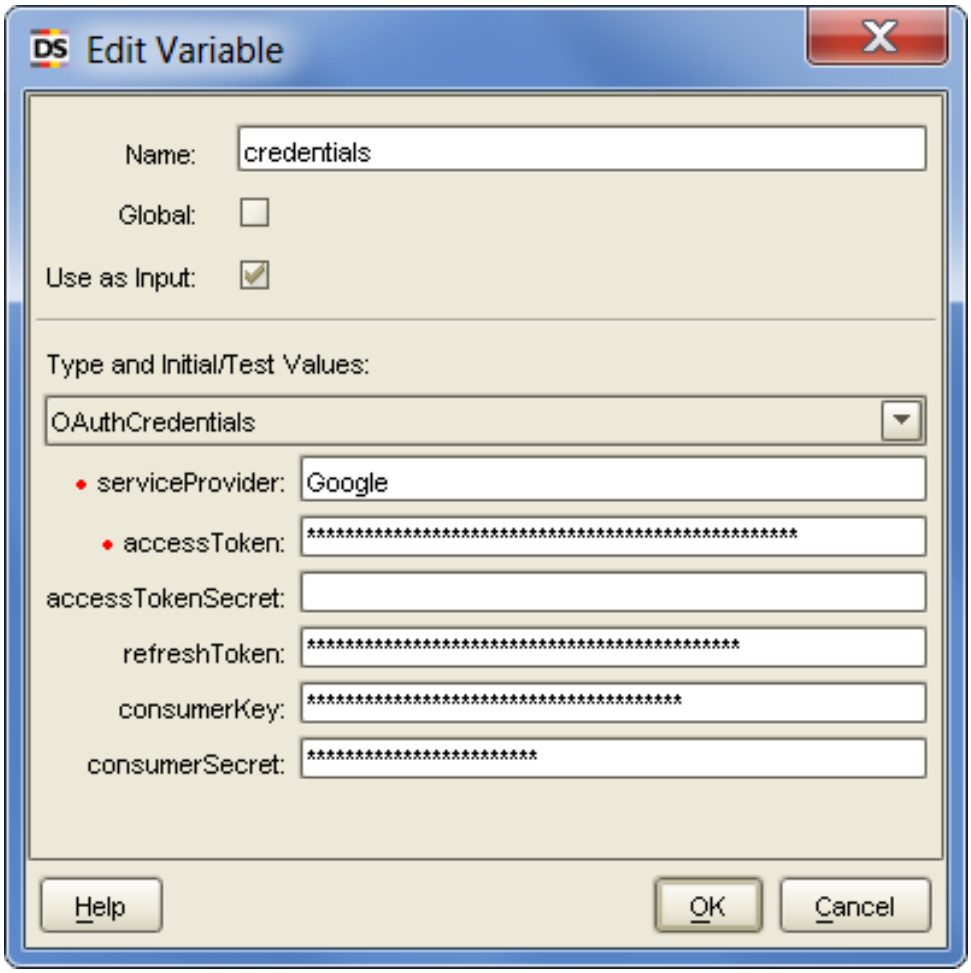


「終了」をクリックすると、OAuthタブの「ユーザー」セクションにユーザーが表示されます。

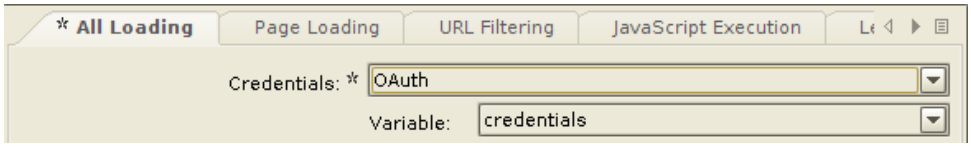


ここで、Design Studioを開いて新しいロボットを作成するか、選択したGoogleのAPIにアクセスするロボットを開きます。ロボットがGoogleのAPIを使用できるようにするには、複雑なタイプの「OAuthCredentials」の入力変数を確実に設定します。変数には、サービス・プロバイダ名として「Google」を指定する必要があります。また、アクセス・トークンおよびリフレッシュ・トークンと

ともに、以前にメモしたクライアント・キーおよびクライアント・シークレットを入力します。後者の2つは、コンシューマ・キーおよびコンシューマ・シークレットとしてそれぞれ使用します。



次に、OAuthの資格証明を使用するロボットを確実に構成します。



これでロボットは、選択したGoogleのAPIにアクセスできます。たとえば、ブラウザ・ビューのアドレス・バーに「**https://www.googleapis.com/oauth2/v1/userinfo**」と入力して、[Enter]を押します。これによって、基本的なユーザー情報が含まれたJSONページが返されます。

Googleの APIの詳細を学習するには、「Google's OAuth 2.0 Playground」の体験を試みてください。また、次の表にあるドキュメントへのリンクは、各APIの詳細を学習する際に役立つリソースです。ロボットを作成した後は、Management Consoleでロボットの実行をスケジュールできます。

表11.24 2012年12月の時点での様々なGoogle APIの概要。更新リストを取得するには、「Google APIs Discovery Service」を使用してください。

API	説明	ドキュメント	スコープ
Search API For Shopping	アド・エクスチェンジ購入者アカウントの構成の管理	https://developers.google.com/ad-exchange/buyer-rest	https://www.googleapis.com/auth/adexchange.buyer
AdSense Management API	AdSenseデータの表示および管理	https://developers.google.com/adsense/management/	https://www.googleapis.com/auth/adsense
AdSense Management API	AdSenseデータの表示	https://developers.google.com/adsense/management/	https://www.googleapis.com/auth/adsense.readonly
AdSense Host API	AdSenseホスト・データおよび関連するアカウントの表示および管理	https://developers.google.com/adsense/host/	https://www.googleapis.com/auth/adsensehost
Google Analytics API	Google Analyticsデータの表示および管理	https://developers.google.com/analytics/	https://www.googleapis.com/auth/analytics
Google Analytics API	Google Analyticsデータの表示	https://developers.google.com/analytics/	https://www.googleapis.com/auth/analytics.readonly
BigQuery API	Google BigQueryでのデータの表示および管理	https://developers.google.com/bigquery/docs/overview	https://www.googleapis.com/auth/bigquery
BigQuery API	Google Cloud Storageでのデータの管理	https://developers.google.com/bigquery/docs/overview	https://www.googleapis.com/auth/devstorage.read_write
BigQuery API	Google Cloud Storageでのデータおよび権限の管理	https://developers.google.com/bigquery/docs/overview	https://www.googleapis.com/auth/devstorage.full_control
BigQuery API	Google Cloud Storageでのデータの表示	https://developers.google.com/bigquery/docs/overview	https://www.googleapis.com/auth/devstorage.read_only
Blogger API	Bloggerアカウントの管理	https://developers.google.com/blogger/docs/3.0/getting_started	https://www.googleapis.com/auth/blogger
Blogger API	Bloggerアカウントの表示	https://developers.google.com/blogger/docs/3.0/getting_started	https://www.googleapis.com/auth/blogger.readonly
Books API	本の管理	https://developers.google.com/books/docs/v1/getting_started	https://www.googleapis.com/auth/books

API	説明	ドキュメント	スコープ
Calendar API	カレンダの管理	https://developers.google.com/google-apps/calendar/firstapp	https://www.googleapis.com/auth/calendar
Calendar API	カレンダの表示	https://developers.google.com/google-apps/calendar/firstapp	https://www.googleapis.com/auth/calendar.readonly
Compute Engine API	Google Compute Engineリソースの表示および管理	https://developers.google.com/compute/docs/reference/v1beta13	https://www.googleapis.com/auth/compute
Compute Engine API	Google Compute Engineリソースの表示	https://developers.google.com/compute/docs/reference/v1beta13	https://www.googleapis.com/auth/compute.readonly
Compute Engine API	Google Cloud Storageでのデータの表示	https://developers.google.com/compute/docs/reference/v1beta13	https://www.googleapis.com/auth/devstorage.read_only
Google Maps Coordinate API	Google Coordinateジョブの表示	https://developers.google.com/coordinate/	https://www.googleapis.com/auth/coordinate.readonly
Google Maps Coordinate API	Google Maps Coordinateジョブの表示および管理	https://developers.google.com/coordinate/	https://www.googleapis.com/auth/coordinate
DFA Reporting API	DoubleClick for Advertisersレポートの表示および管理	https://developers.google.com/doubleclick-advertisers/reporting/	https://www.googleapis.com/auth/dfareporting
Drive API	Google Driveでのファイルおよびドキュメントの表示	https://developers.google.com/drive/	https://www.googleapis.com/auth/drive.readonly
Drive API	Google Driveアプリケーションの表示	https://developers.google.com/drive/	https://www.googleapis.com/auth/drive.apps.readonly
Drive API	Google Driveでのファイルおよびドキュメントの表示および管理	https://developers.google.com/drive/	https://www.googleapis.com/auth/drive
Drive API	Google Driveでのファイルおよびドキュメントのメタ・データの表示	https://developers.google.com/drive/	https://www.googleapis.com/auth/drive.metadata.readonly
Drive API	このアプリケーションで開かれた、または作成されたGoogle Driveファイルの表示および管理	https://developers.google.com/drive/	https://www.googleapis.com/auth/drive.file

API	説明	ドキュメント	スコープ
Freebase API	アカウントを使用したFreebaseへのサインイン	http://wiki.freebase.com/wiki/API	https://www.googleapis.com/auth/freebase
Fusion Tables API	Fusion Tablesの表示	https://developers.google.com/fusiontables	https://www.googleapis.com/auth/fusiontables.readonly
Fusion Tables API	Fusion Tablesの管理	https://developers.google.com/fusiontables	https://www.googleapis.com/auth/fusiontables
Google Affiliate Network API	GANデータの表示	https://developers.google.com/affiliate-network/	https://www.googleapis.com/auth/gan.readonly
Google Affiliate Network API	GANデータの管理	https://developers.google.com/affiliate-network/	https://www.googleapis.com/auth/gan
Groups Settings API	Google Apps Groupの設定の表示および管理	https://developers.google.com/google-apps/groups-settings/get_started	https://www.googleapis.com/auth/apps.groups.settings
Google Latitude API	市区町村レベルの場所の管理	https://developers.google.com/latitude/v1/using	https://www.googleapis.com/auth/latitude.current.city
Google Latitude API	最適で利用可能な場所および場所の履歴の管理	https://developers.google.com/latitude/v1/using	https://www.googleapis.com/auth/latitude.all.best
Google Latitude API	最適で利用可能な場所の管理	https://developers.google.com/latitude/v1/using	https://www.googleapis.com/auth/latitude.current.best
Google Latitude API	市区町村レベルの場所および場所の履歴の管理	https://developers.google.com/latitude/v1/using	https://www.googleapis.com/auth/latitude.all.city
Moderator API	Google Moderatorのアクティビティの管理	http://code.google.com/apis/moderator/v1/using_rest.html	https://www.googleapis.com/auth/moderator
Google OAuth2 API	ユーザーのアカウントに関する基本情報の表示	https://developers.google.com/accounts/docs/OAuth2	https://www.googleapis.com/auth/userinfo.profile
Google OAuth2 API	電子メール・アドレスの表示	https://developers.google.com/accounts/docs/OAuth2	https://www.googleapis.com/auth/userinfo.email
Google OAuth2 API	Googleでのユーザーに関する認識	https://developers.google.com/accounts/docs/OAuth2	https://www.googleapis.com/auth/plus.me

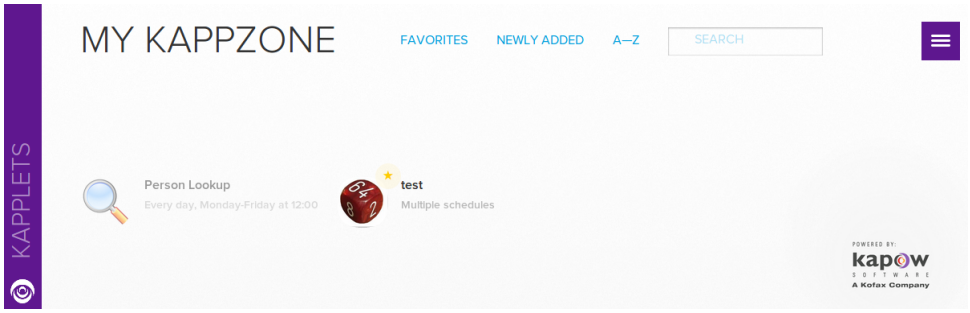
API	説明	ドキュメント	スコープ
Orkut API	Orkutアクティビティの管理	http://code.google.com/apis/orkut/v2/reference.html	https://www.googleapis.com/auth/orkut
Orkut API	Orkutデータの表示	http://code.google.com/apis/orkut/v2/reference.html	https://www.googleapis.com/auth/orkut.readonly
Google+ API	Googleでのユーザーに関する認識	https://developers.google.com/+/api/	https://www.googleapis.com/auth/plus.me
Prediction API	Google Prediction APIでのデータの管理	https://developers.google.com/prediction/docs/developer-guide	https://www.googleapis.com/auth/prediction
Prediction API	Google Cloud Storageでのデータの表示	https://developers.google.com/prediction/docs/developer-guide	https://www.googleapis.com/auth/devstorage.read_only
Search API For Shopping	製品データの表示	https://developers.google.com/shopping-search/v1/getting_started	https://www.googleapis.com/auth/shoppingapi
Google Site Verification API	Googleによる新しいサイトの検証の管理	http://code.google.com/apis/siteverification/	https://www.googleapis.com/auth/siteverification.verify_only
Google Site Verification API	管理するサイトおよびドメインのリストの管理	http://code.google.com/apis/siteverification/	https://www.googleapis.com/auth/siteverification
Cloud Storage API	Google Cloud Storageでのデータの管理	https://developers.google.com/storage/docs/json_api/	https://www.googleapis.com/auth/devstorage.read_write
Cloud Storage API	Google Cloud Storageでのデータおよび権限の管理	https://developers.google.com/storage/docs/json_api/	https://www.googleapis.com/auth/devstorage.full_control
Cloud Storage API	Google Cloud Storageでのデータの表示	https://developers.google.com/storage/docs/json_api/	https://www.googleapis.com/auth/devstorage.read_only
TaskQueue API	タスクおよびタスク・キューの管理	http://code.google.com/appengine/docs/python/taskqueue/rest.html	https://www.googleapis.com/auth/taskqueue
TaskQueue API	タスク・キューのタスクの消費	http://code.google.com/appengine/docs/	https://www.googleapis.com/auth/taskqueue.consumer

API	説明	ドキュメント	スコープ
		python/taskqueue/rest.html	
Tasks API	タスクの管理	http://code.google.com/apis/tasks/v1/using.html	https://www.googleapis.com/auth/tasks
Tasks API	タスクの表示	http://code.google.com/apis/tasks/v1/using.html	https://www.googleapis.com/auth/tasks.readonly
URL Shortener API	goo.gl短縮URLの管理	http://code.google.com/apis/urlshortener/v1/getting_started.html	https://www.googleapis.com/auth/urlshortener
YouTube API	YouTubeのユーザーの資産および関連するコンテンツの表示および管理	https://developers.google.com/youtube	https://www.googleapis.com/auth/youtubepartner
YouTube API	YouTubeビデオの管理	https://developers.google.com/youtube	https://www.googleapis.com/auth/youtube.upload
YouTube API	YouTubeアカウントの管理	https://developers.google.com/youtube	https://www.googleapis.com/auth/youtube
YouTube API	YouTubeアカウントの表示	https://developers.google.com/youtube	https://www.googleapis.com/auth/youtube.readonly
YouTube Analytics API	YouTubeコンテンツに関するYouTubeアナリティクス・レポートの表示	http://developers.google.com/youtube/analytics/	https://www.googleapis.com/auth/yt-analytics.readonly

第12章 Kapow Kappletsユーザーズ・ガイド

Kapow Kappletsには、ロボットに対するフレンドリなユーザー・インタフェースが用意されています。Kapplet Administratorは、1つ以上のロボットをユーザーが実行できるようにし、そのユーザーは、提供されたKappletを介してロボットと対話するため、ロボットに関する知識は何も必要ありません。Kappletは、エンド・ユーザーの用語にあわせてカスタマイズでき、アイコンおよび説明も添付できます。

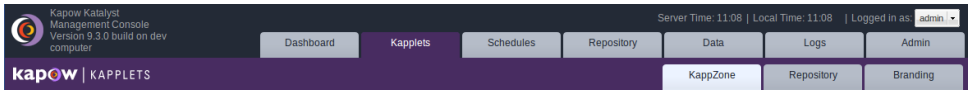
Kappletは、Kapplet AdministratorがKapplet Studioで構築および保守します。Kappletを使用する準備が整うと、すべてのKapplet UsersがKappZoneから使用できるようになり、ここで、ユーザーは個人のMy KappZoneにKappletsをインストールできます。Kapplet Usersは、My KappZoneにInstalled Kappletsの独自のリストを表示するようにしておくか、またはブラウザにKappletsをブックマークできます。



Kappletsは、Management Consoleのユーザー/ロール管理に基づいて構築されます。Management Consoleがスタンドアロン・アプリケーション・サーバーにデプロイされていると、Kapplet Administratorsは、LDAPを使用して、どのユーザーがどのプロジェクト内のKapplet公開済ロボットにアクセスできるかを管理できます。

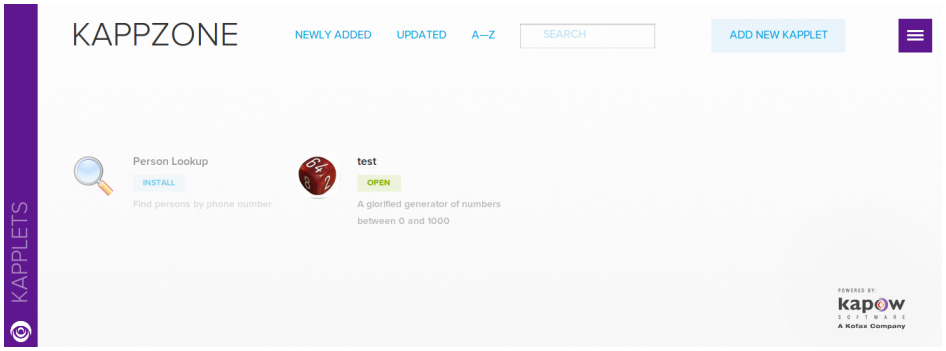
Kappletsの構築と保守

Management Consoleでは、Kappletsは、「Kapplets」というタブから管理できる別々のエンティティとして表されます。

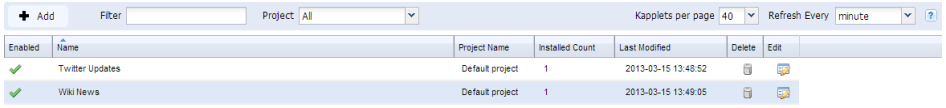


この部分のManagement Consoleには、3つのサブセクションがあります。

1. 「KappZone」には、現在使用できるすべてのKappletsが表示されます。ここから、My KappZoneで使用できるようにするKappletsをインストールできます。このサブセクションは、「Kappletsの起動」で詳しく説明しており、ここではこれ以上説明しません。



2. 「Kapplet Repository」には、現在定義されているKappletsのリストが表示されます。ここで、Kapplet定義を表示、追加、削除および編集できます。このサブセクションは、後述の「Kappletsの作成、削除および編集」の項で詳しく説明します。



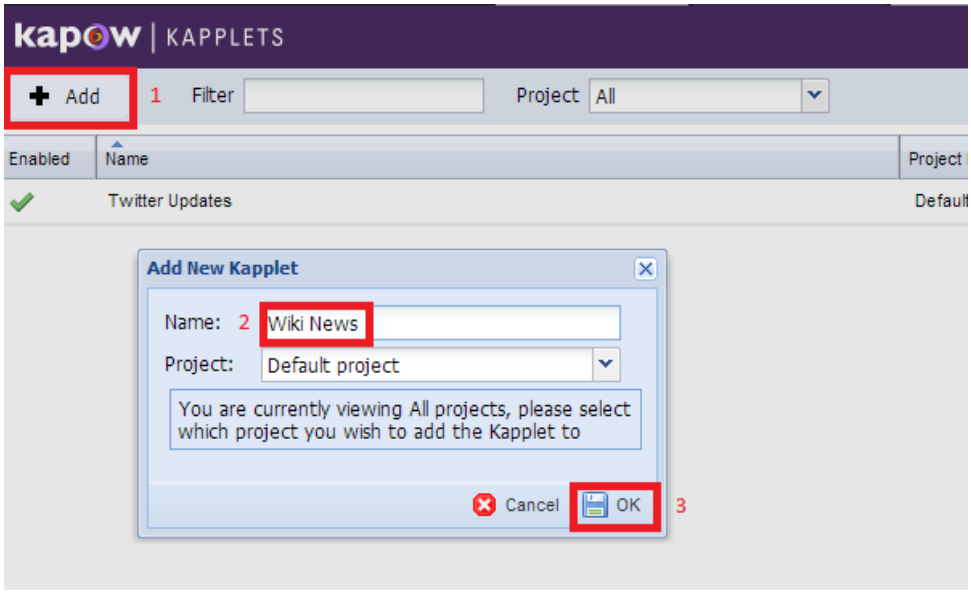
3. 「Branding」タブを使用すると、Kapplet Administratorは、Kappletsの基本的な外観のカスタマイズに使用できるインタフェースにアクセスできます。このインタフェースは、「ブランドのカスタマイズ」で詳しく説明しており、ここではこれ以上説明しません。

すべてのManagement Consoleユーザーが、Kappletsの管理はもちろん、使用もできるわけではないことを理解することが重要です。このような権限を特定のユーザーに割り当てるには、プロジェクトの「権限」タブに移動する必要があります。ここで、この特定のプロジェクトに対して、Kapplet Administratorsとするユーザー・グループおよびkapplet Usersとするユーザー・グループを構成できます。標準エディションのセキュリティ・オプションはさらに緩やかで、すべてのユーザーにKappletsを表示および編集する権限があります。

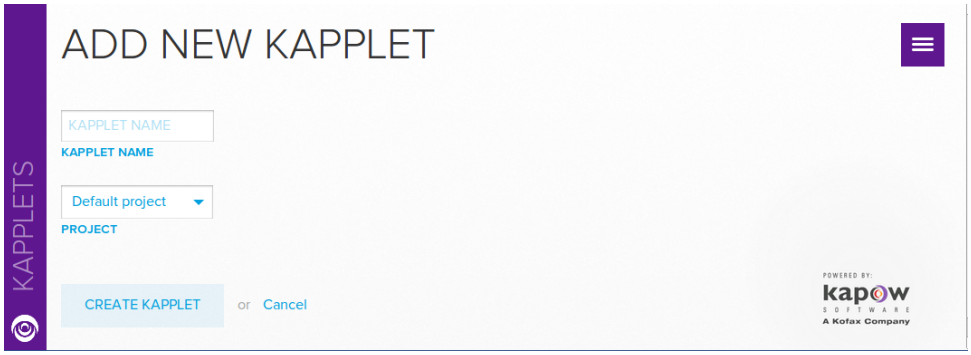
Kappletsの作成

Kapplet Administratorが新しいKappletを作成する方法はいくつかあります。以前のバージョンと同様に、Kapplet Administratorは「Kapplet Repository」サブセクションに移動でき、左上隅にある

「追加」をクリックするとダイアログが開き、このダイアログで、Kapplet Administratorは新しいKappletをホストするプロジェクトを選択して、そのKappletに名前を設定する必要があります。

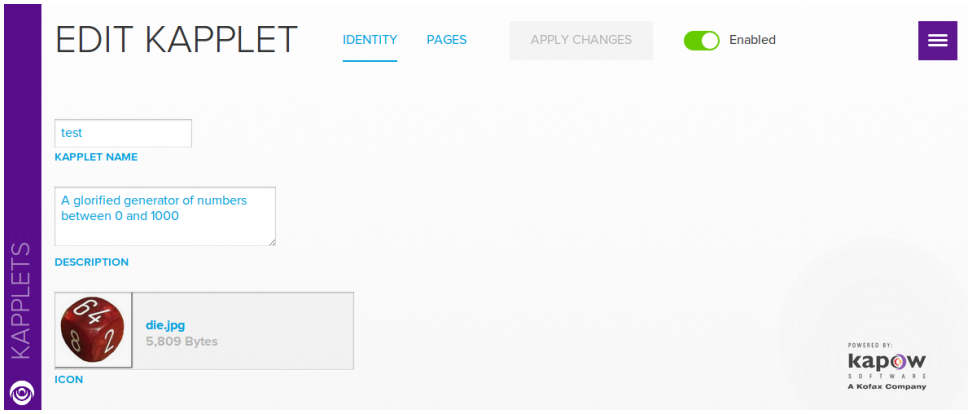


Kapplet Administratorは、新しいKappletをKapplet User領域、つまりKappZoneから直接作成することもできます。この場合、新規Kappletの追加ボタンをクリックするか、またはMain Navigation Menuの新規Kapplet項目をクリックすると、プロセスを開始できます。ページが開き、次に示すように、このページで新しいKappletに名前を設定して、既存プロジェクトに関連付けることができます。

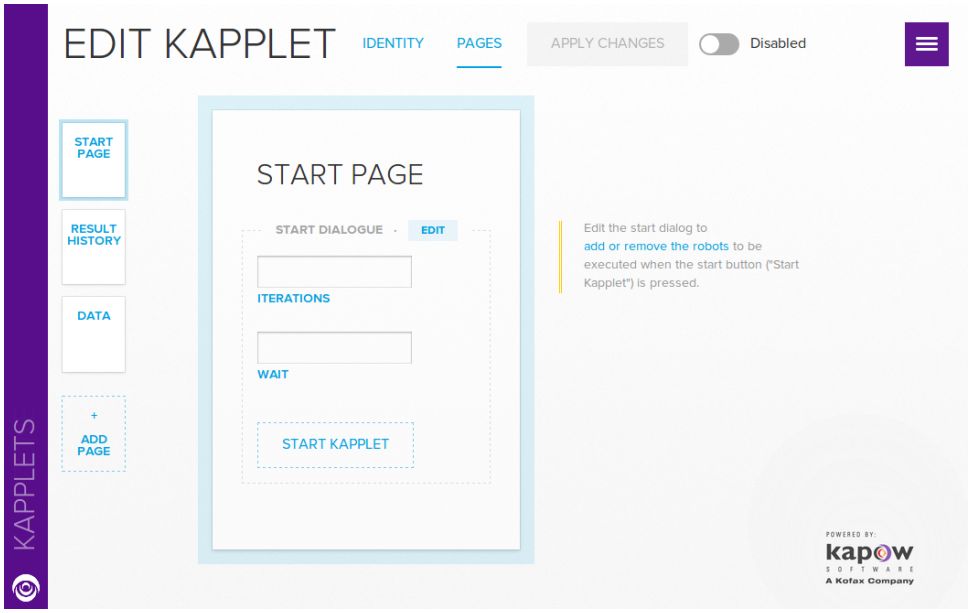


適切に名前を設定して関連付けると、新しいKappletは、Kappletの作成ボタンを押すのみで作成され、その後Kapplet Studioが開き、Kapplet Administratorは新しいKappletをさらに構成できます。

Kapplet Studioの使用

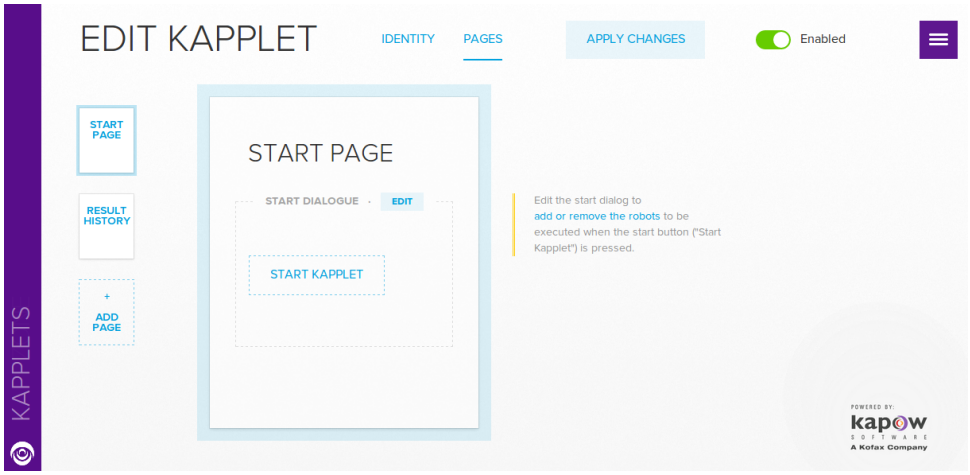


新しく作成したKappletsの場合、Kapplet Studioが開いたときにIdentity Sectionが表示されます。このページでは、前述のように、Kappletの名前、コメント、アイコンなどの一般情報を入力または変更できます。既存のKappletを編集用に開く場合、Identity Sectionが表示されるのは、ロボットがKappletに追加されていない場合のみであることに注意してください。それ以外の場合、Kapplet Studioは、Kappletの機能を構成するPages Sectionで開きます。Pages Sectionは、Kapplet Studioの上部にある「ページ」ヘッダーをクリックして開くこともできます。セクションを変更する前に、Kapplet Studioの上部にある変更の適用ボタンをクリックして変更を保存する必要があり、保存しないと変更が失われることに注意してください。



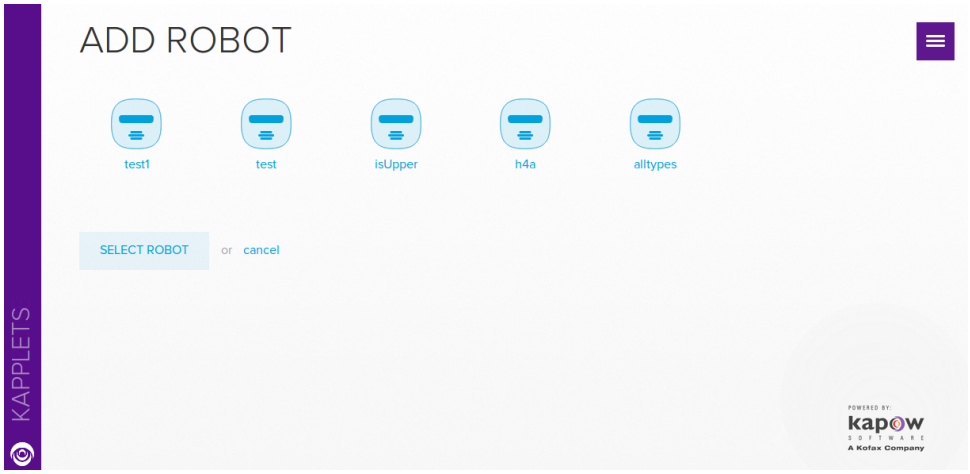
Pages Sectionがアクティブ化すると、Kapplet Administratorに、前述の例と同様のページが表示されます。左側に、Kapplet Page Definitionsの現在のコレクションが、クリック可能なタイトル付きのページ・アイコンの形式で列挙されます。これらの各アイコンは、構築中のKappletの機能ページ/パートを示します。一部のページはすべての機能Kappletsに対して必須ですが、その他は実際の機能に依存します。2つの必須ページはStart PageとResult History Pageであり、それぞれ、Start Actionに対する入力の収集方法と結果履歴の表示方法を管理します。残りのページは、Start Actionによって生成される結果の表示方法および対話方法を管理します。

Start Pageの構成

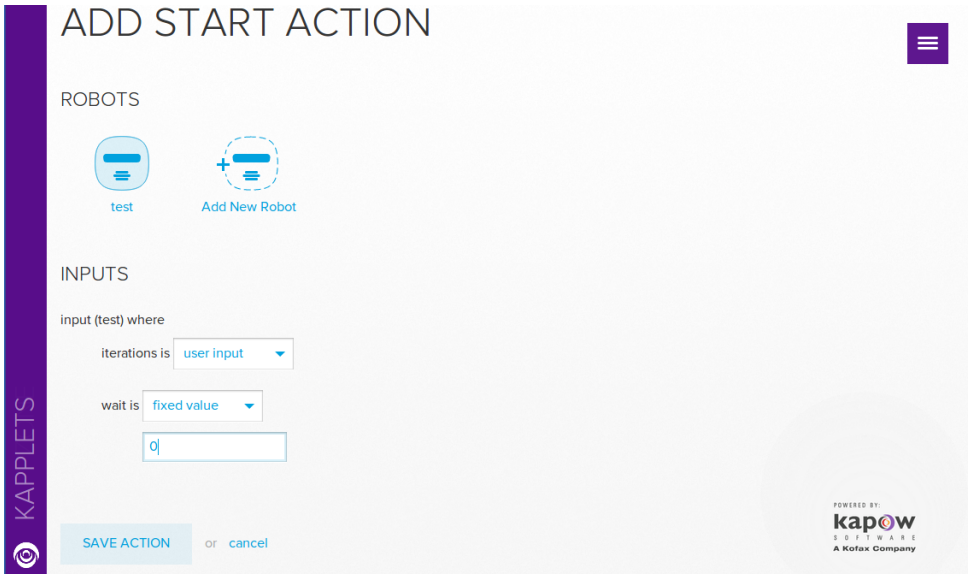


Pages Sectionが開くと、Start Pageが選択された状態になります。このページの実際のコンテンツは、Start Actionに関連付けられているロボットに依存します。ロボットが関連付けられていない場合、前述のようにページは空です。その後、アクションの追加領域をクリックするのみでロボット・ベースのStart Actionを関連付けることができます。

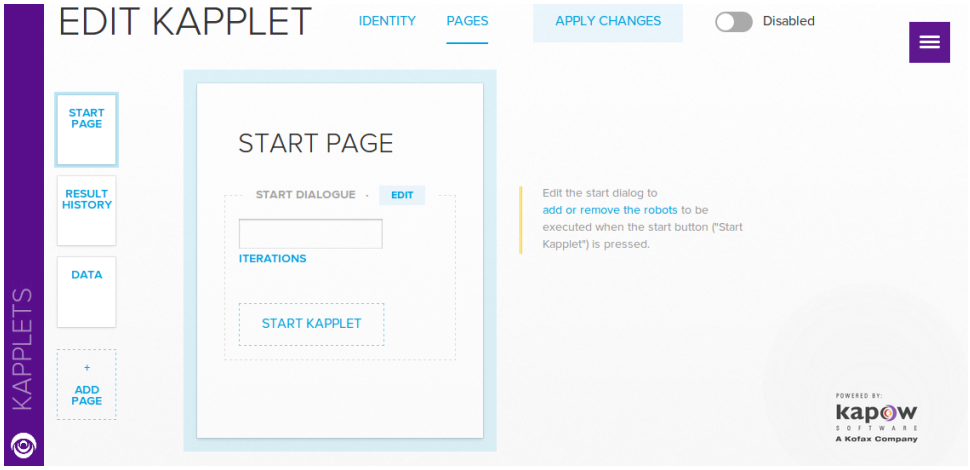
前述の開始アクションの追加ページで、アクションに対するロボットを選択できます。これを実行するには、最初に、新規ロボットの追加アイコンをクリックして選択ページを開きます。この結果、次に示すように使用可能なロボットがリストされ(ここでは単一のロボットのみ表示されます)、次に、ロボットのアイコンを最初にクリックした後でロボットの選択ボタンをクリックすることによって選択を実行します。



ロボットが追加されると、開始アクションの追加ページには、追加したロボットで定義されている入力変数の属性がリストされます。各属性に対して、Kapplet Administratorは、固定値を関連付けるかどうか(およびフィールドをユーザーに表示するかどうか)、または入力フィールドを表示するかどうかを決定できます。次に示す例では、追加したtestロボットに、2つの属性(1つはユーザー入力、1つは固定)がある単一の変数inputが必要です。



アクションの保存ボタンをクリックすると、Kapplet AdministratorはPages Sectionに戻り、Start Pageに今度は、次に示すように、Kappletの実行時にKapplet Usersに表示するページのアウトラインが表示されます。必要な場合、Kapplet Administratorはアウトラインと対話して、ユーザー操作性を変更できます。ページ・タイトル、フィールドの順序とラベル、および開始ボタンのラベルのすべてを変更できます。

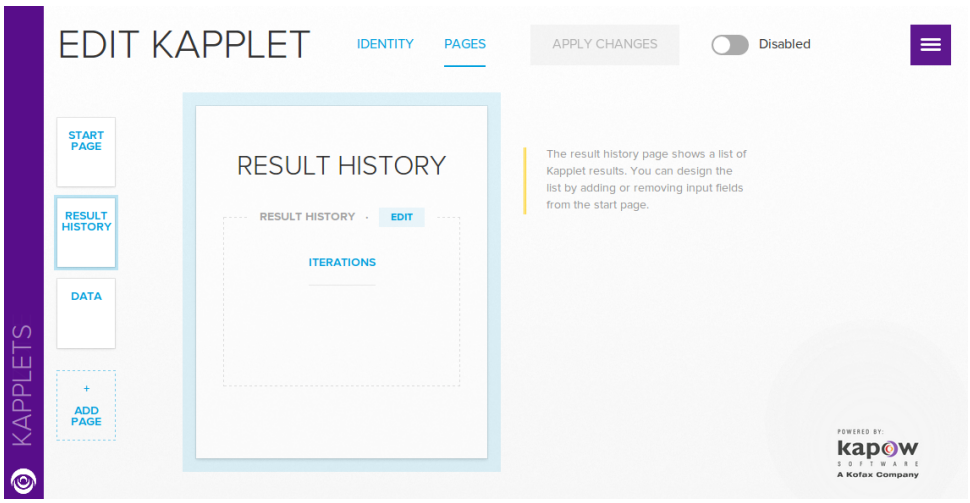


Pages Sectionの他の変更点は、Start Actionの出力タイプ(ある場合)に対応するページが、左側のページ一覧に追加されることです。前述の例では、以前に追加したtestロボットによって発行された出力タイプ「データ」対応する、「データ」という単一のページが該当します。

新しく追加した出力ページについてKapplet結果ページで説明する前に、特殊な結果履歴ページについてKapplet結果履歴ページで説明します。

Result History Pageの構成

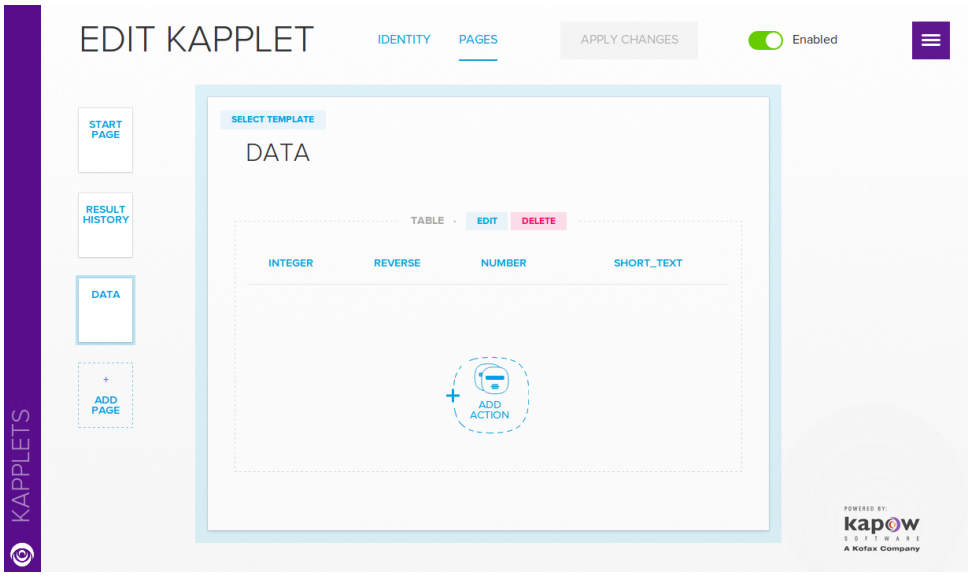
すべてのKappletにResult History Pageがあり、Kapplet Usersは、Kappletが実行した過去および現在の結果にアクセスできます。このページには、結果が項目のリストとして表示され、それぞれが特定の過去の実行に対応しており、タイムスタンプと、対象となる実行の開始アクションに指定した入力パラメータの一覧で構成されています。Kapplet Studioで、Result History Pageのアウトラインをプレビューでき、一部はカスタマイズ可能(ただし、現在はページ・タイトルのみ)です。次に示すように、testの例のロボットでは、タイムスタンプ(表示されていません)と、繰返しパラメータに対応する1つの入力値で構成される要素のリストが表示されます。



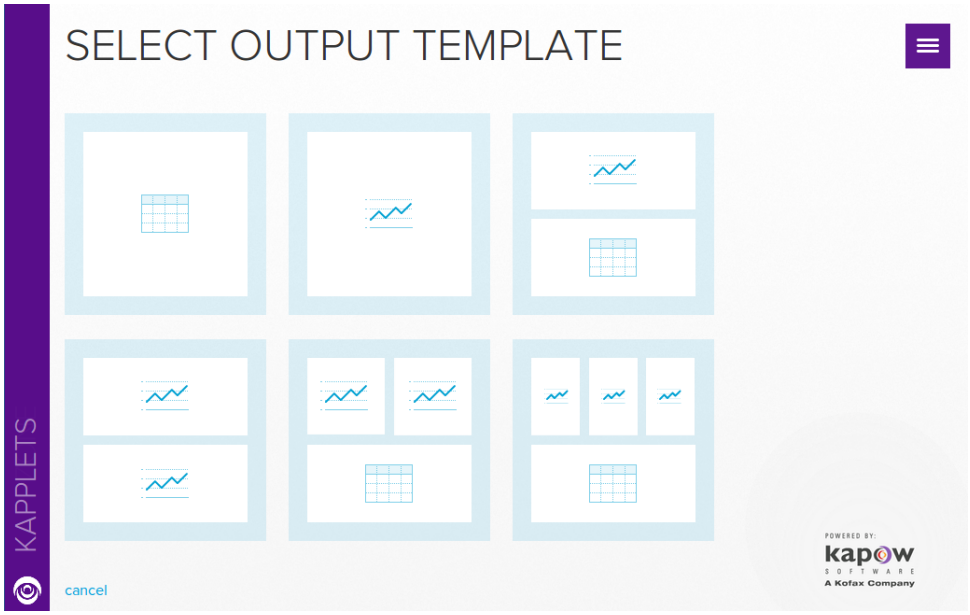
Result Pagesの構成

前述のように、Start Actionを追加すると、Start Action(実際に基礎となるロボット)によって返された出力タイプに対応する新しいページが多数表示されます。これらの各ページは、Kapplet Studio

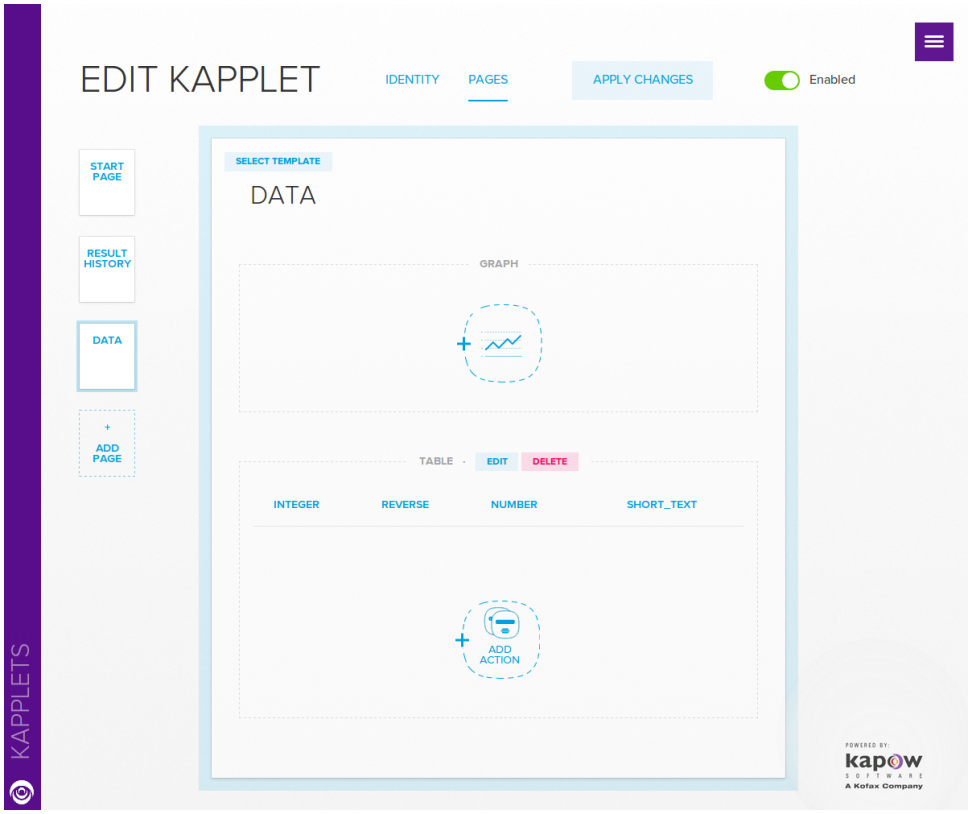
で次の例に示すように開くことができ、この例では、testロボットから発生した「データ」ページが開かれています。



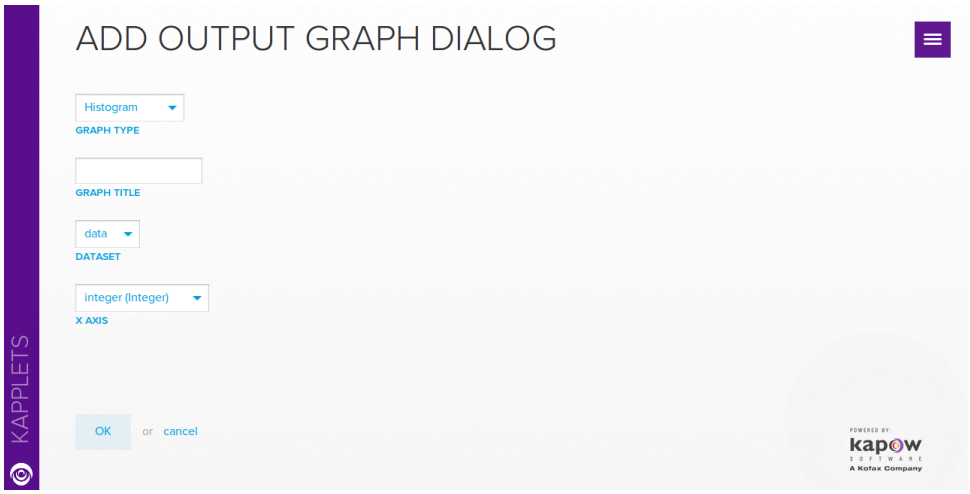
最初に、このページは、各属性に対する列がある表に返された結果を表示するように構成されます。各列ヘッダーおよびページ・タイトルは、適切な名前になるように編集できます。ただし、さらに大きい変更も多数実行できます。たとえば、「削除」ボタンをクリックすると表を削除でき、「編集」ボタンをクリックすると、ソースとして使用するデータセットを変更できます。ビューのタイプも「テンプレートの選択」ボタンをクリックして変更でき、このボタンをクリックすると、次に示すように選択ページが表示されます。



テンプレート・オプションを使用すると、1ページに表を0または1つ、およびグラフを0から3つまで表示できるため、多少高度な検査と選択オプションがサポートされます。次の例では、1つのグラフと1つの表が表示されるテンプレートを選択し、「データ」Result Pageのアウトラインがそれに従って変更されています。

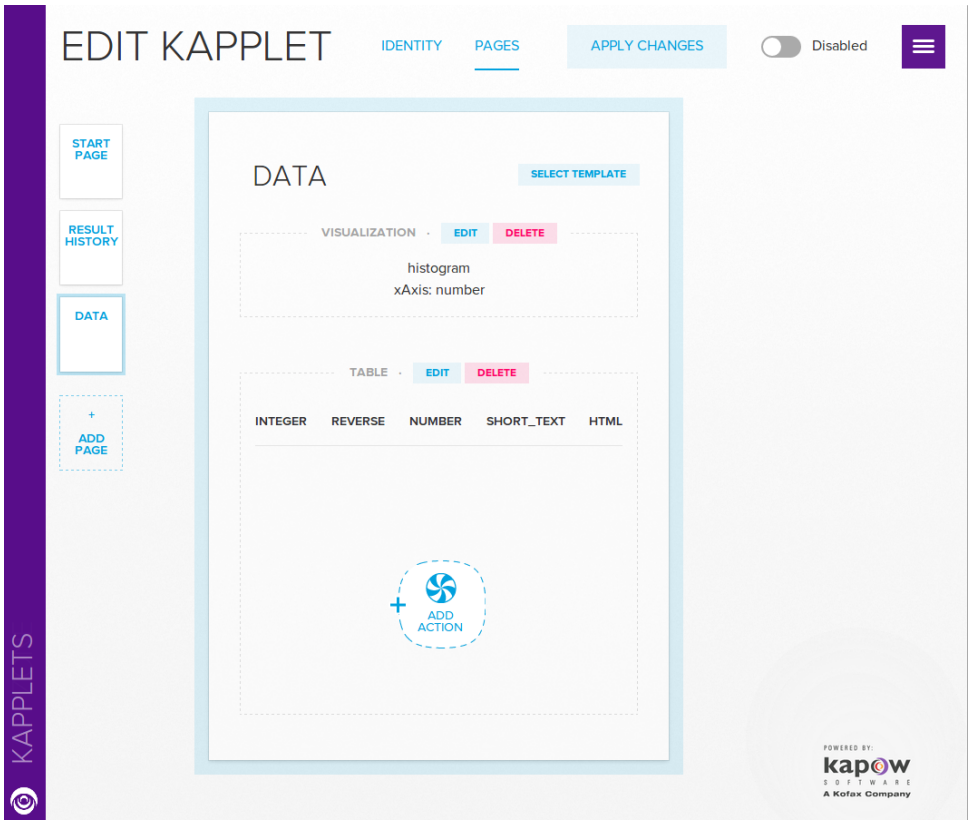


表示領域の中央にある+グラフ・アイコンをクリックすると、Kapplet Administratorは、表示するグラフの実際のタイプを選択できます。多数のグラフ・タイプの中から1つを選択し、構成する必要があります。テキスト・フィールドは通常、Kapplet Administratorが必要とする値に設定できますが、ドロップダウンは常に、特定のデータセットのコンテキスト内で適切な選択ができるように制限されます。これは、次の例でも明らかであり、データセットボックスではtestロボットから設定された「データ」のみ、x軸ボックスでは「データ」タイプの数値属性に対応する値のみが許可されます。



グラフを構成して保存すると、Result Pageが再度表示され、+グラフ・アイコンのかわりに、グラフ構成に関する情報の主要な部分がいくつか表示されます。次に示す例では、ヒストグラムの追加後に、「データ」のResult Pageが表示されます。

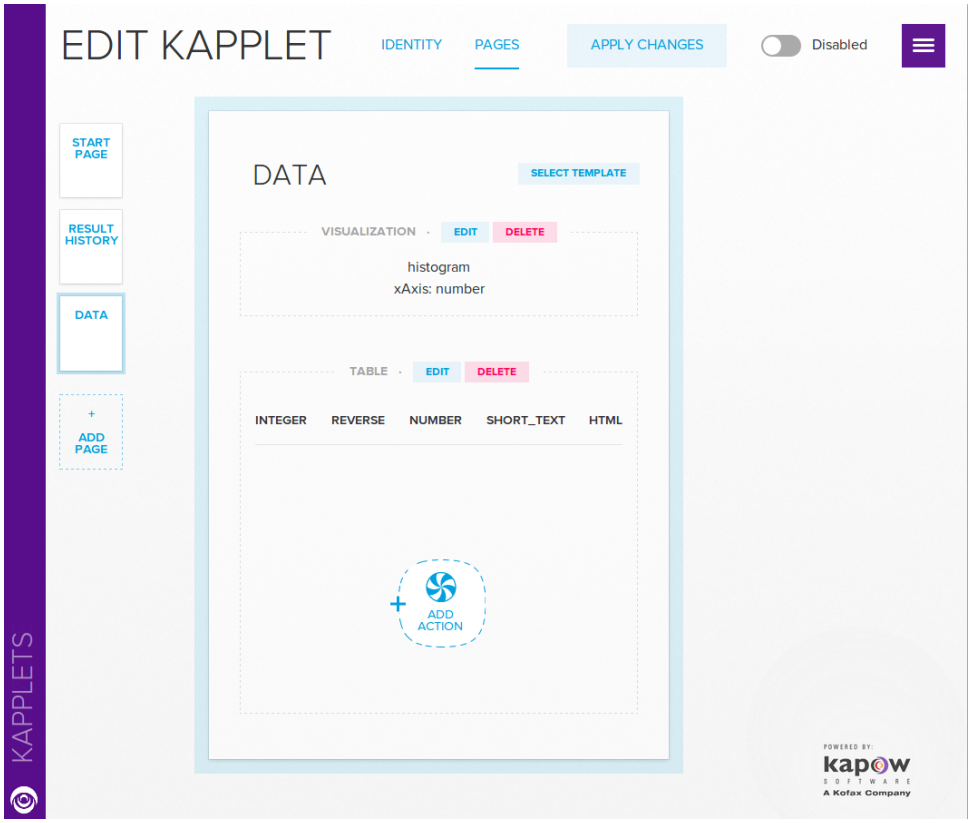
単一のテンプレートに由来するすべての表およびグラフの選択/フィルタリング機能は、非常に高度なデータ調査を容易にするために相互に作用することに注意してください。同じデータセットに関する複数のビューを左側のページ一覧の「ページの追加」機能を使用して作成できますが、これらのビューは独立していて、このように相互に作用しません。



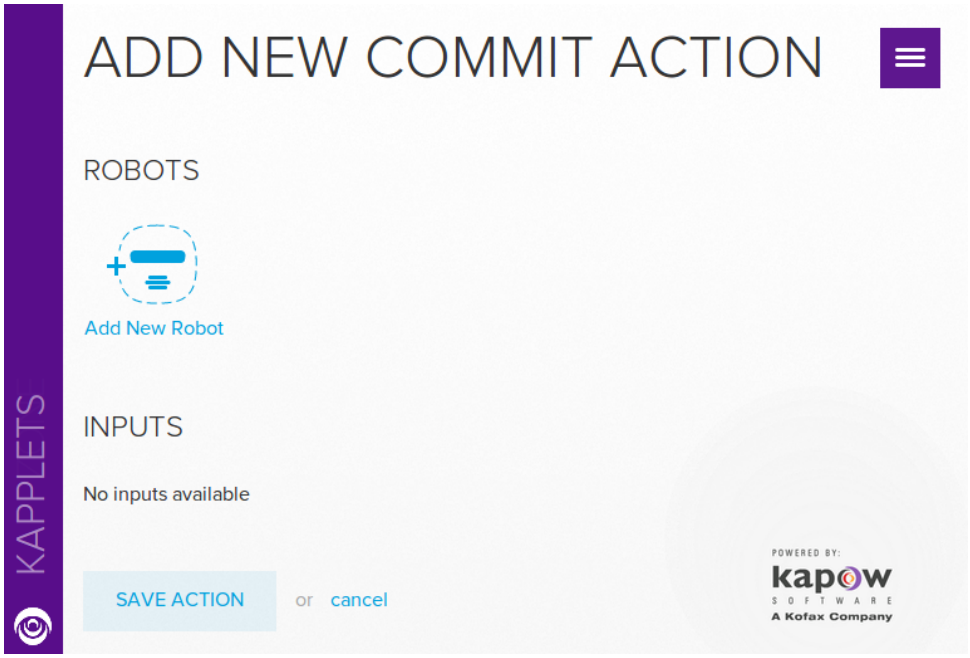
グラフ・タイプを選択すると、Kappletが正しく構成され、データの収集および調査に使用できるようになります。「Commit Actionの追加」で説明するように、Kapplet Administratorは、収集したデータの処理をユーザーに許可することもできます。

Commit Actionの追加

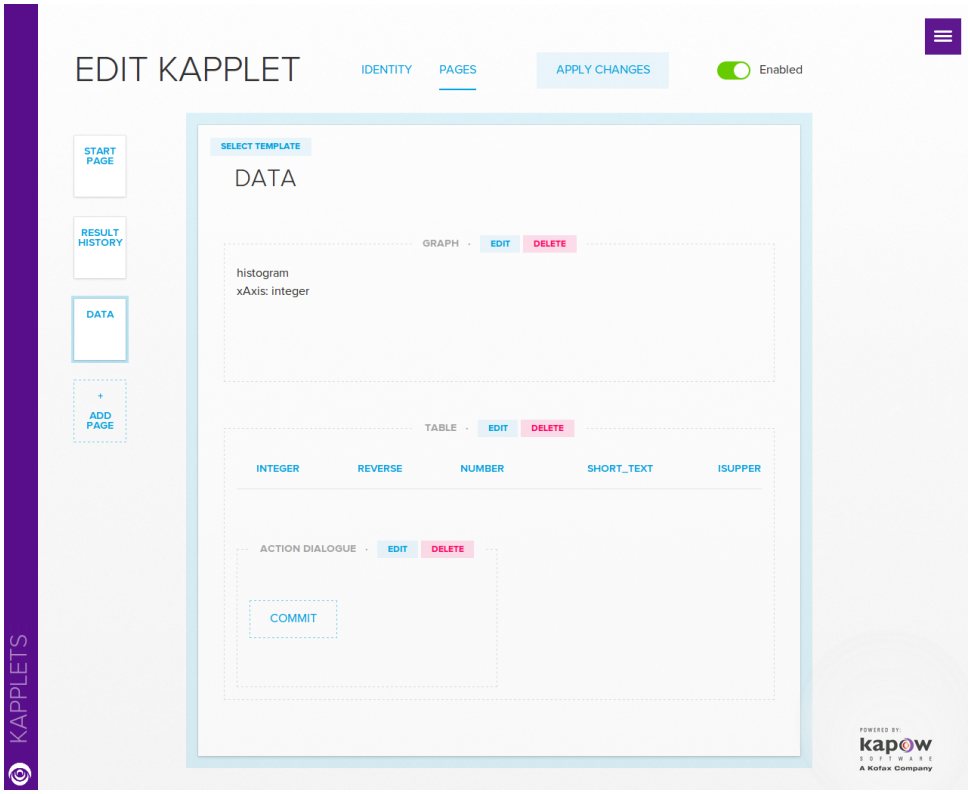
Start Actionで生成されるデータセットの表形式ビューを表示するようにResult Pageを設定すると、Result Pageアウトラインの表アウトラインにあるアクションの追加ボタンをクリックすることによって、いわゆるCommit ActionをKappletに追加できます。「Result Pagesの構成」にも示されている次の例では、下部の表アウトラインにアイコンが表示されています。



アイコンをクリックすると、次の例のようなダイアログがKapplet Administratorに表示されます。ダイアログでKapplet Administratorは、選択した表の行をデータベース、Webサービス、実際のWebサイトなどに簡単にコミットするためのロボットを選択できます。ロボットは、選択表に表示されている入力データと同じタイプの入力データを取得して、コミットの結果を入力タイプと異なるタイプの単一オブジェクトの形式で返す必要があります。



返されるタイプによって、実際の選択表に変更が発生するため、Commit Actionが完了すると、返された情報を元の情報と一緒に表示できます。この変更は次に示されており、この場合は単純なロボットisUpperが、testの例のCommit Actionをサポートするために追加されています。元の表に対応するタイプ「データ」の入力を与えると、isUpperによって、指定した「データ」値の属性「数値」が500以上かどうかを識別する、単一のブール属性isUpperのみを含むメタ・タイプのインスタンスが返されます。

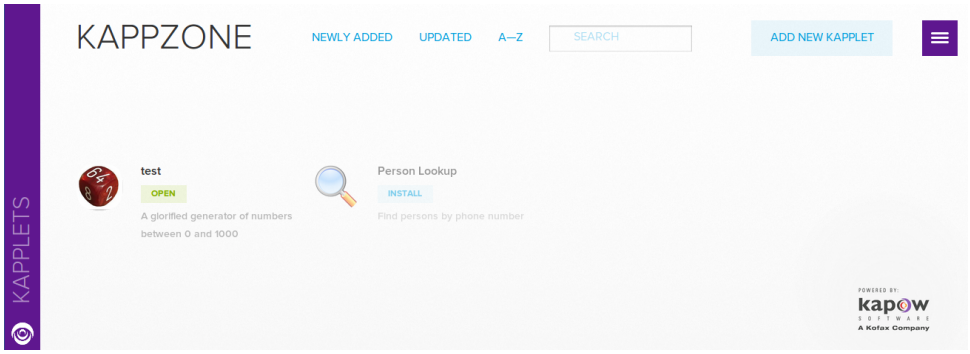


1つ以上の行のコミット時に、Commit Actionによってその内容に従って表が更新され、ユーザーは必要に応じてさらに行を選択してコミットできます。

Commit Actionを追加すると、Kappletが(再度)正しく構成され、データの収集、選択およびコミットに使用できるようになります。Kapplet Administratorは、Kapplet UsersがKappletを通常使用できるように、画面上部にある適切なトグルを使用してKappletを有効化する必要があることに注意してください。

Kappletsのインストールと使用

現在はKapplet AdministratorsであるKapplet Usersの場合、Kapow Kappletsには通常、**kappzone**をManagement Console URLに追加することによって、Webブラウザから直接アクセスします。たとえば、Management Consoleが**http://localhost:50080/**にデプロイされている場合は、**http://localhost:50080/kappzone**でKappletsに直接アクセスできます。

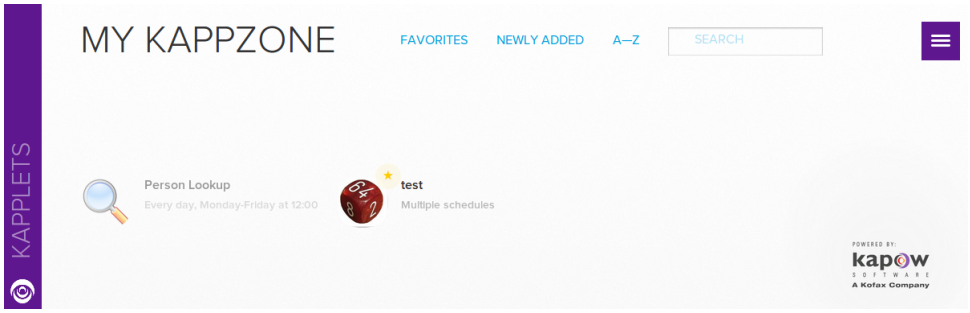


使用可能なKapplets。非Installed Kappletsはユーザーがインストール可能で、Installed Kappletsはユーザーが開くこと(実行)が可能であることに注意してください。

ユーザーは、「KappZone」タブでKappletsにアクセスできます。ユーザーがKappletをインストールすると、Kappletのインスタンスがユーザーの My KappZoneで使用可能になります。指定されたKappletを特定のKapplet Userが1回のみインストールできます。Installed Kappletは、My KappZone(それ自体をクリック)とKappZone(「開く」をクリック)の両方から実行できます。

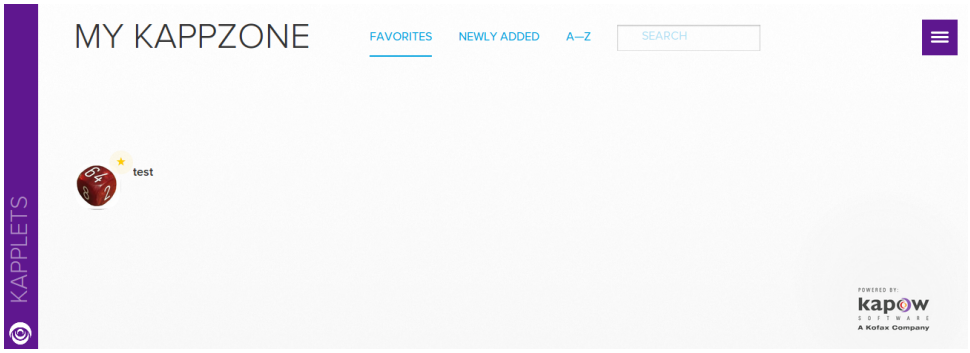
ユーザーは、自分のブラウザのMy KappZoneからInstalled Kappletsをブックマークでき、Installed Kappletsに星を付けると、お気に入りの表示に追加できます。このように、ユーザーは最もよく使用するKappletsにスムースにナビゲートできます。

Kappletをブックマークするために、Kapplet UserはKappletのURLを識別する必要があります。これはブラウザに表示されており、現在のページを単純にブックマークできます。



Installed Kappletsでは、test Kappletにユーザーがお気に入りの星を付けていることがわかります。

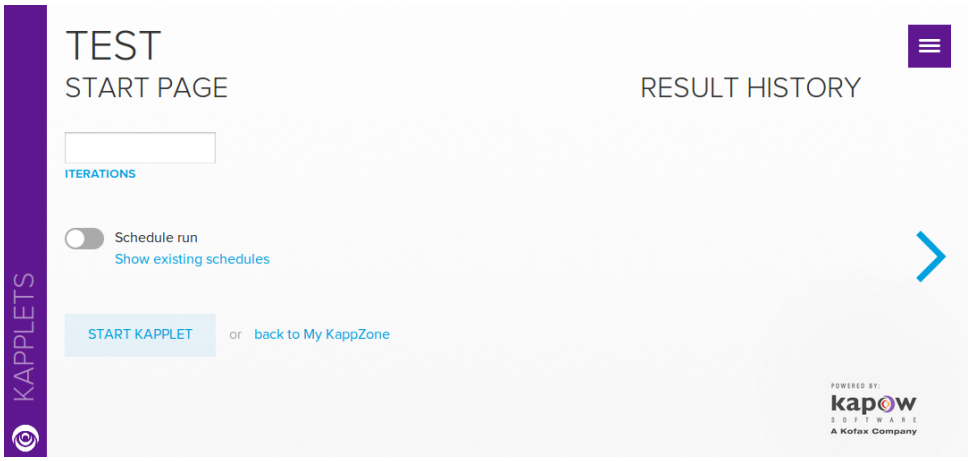
KappZoneを開いた後、Kapplet Userは、お気に入りをクリックすると、Kapplet Userのお気に入りの Installed Kappletsを表示できます。



お気に入りKappletsでは、星付きのKappletsのみが表示されることがわかります。

Kappletsの起動

Installed Kappletを実行するには、My KappZoneまたはお気に入りのいずれかでそれを単にクリックし、必須フィールドをすべて入力して、「開始」ボタンをクリックします。次の例のtest Kappletでは、繰返しフィールドのみ入力する必要があります。



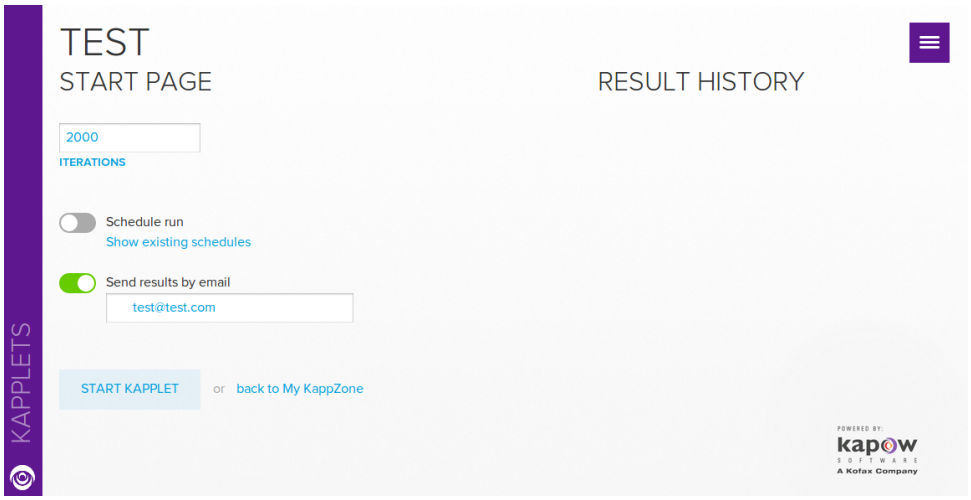
Kappletを使用する際、Kapplet Userには通常、前述のようなビューが表示され、ここには、アクティブなKapplet Pagesがすべて、左から右の順序で、Start Pageから始まり、Result History Page、次にResult Pages(関連している場合)の順に同時に表示されます。Kappletが開始するまで、またはアクティブなKapplet RunがResult History Pageから選択されるまでは、Start PageとResult History Pageのみが表示されることに注意してください。

Kappletが開始すると、対応する新しいKapplet RunがResult History Pageにリストされ、対応するResult PagesがResult History Pageの右側に表示されます。アクティブな(まだ削除されていない)Kapplet Runsは常に保存され、対応するInstalled KappletのResult History Pageで再検索できます。対応するResult Pagesは、関連するResult History Pageのエントリをクリックすると開くことができます。前述の例では、test Kappletの結果が前半部分で構成されていることが示されています。表内の選択項目をコミットでき、出力は次のようになります。

ユーザーが、サード・パーティのプログラムで、表形式のKappletの結果をさらに処理する場合は、Kapplet結果をExcel形式でダウンロードできます。

Kappletsからの電子メール通知

長時間実行のKappletsの場合、Kapplet Userは、結果ページが移入されるのを何もせずに待機することを希望しない場合があります。この場合、Kapplet Userは、Kapplet Runの開始時に電子メール・アドレスを入力し、結果が準備できたときに電子メールを受信します。電子メール・アドレスを入力するには、結果を電子メールで送信オプションをオンに切り替えて通知を有効化する必要があります。このトグルが表示されるのは、(SMTPがManagement Consoleで構成されている)場合で、Kappletsに対して送信元アドレスが構成されている場合のみです。次に、SMTPサーバーが構成されていて、Kapplet Userの電子メール・アドレスが入力された状態のInstalled Kappletを示します。

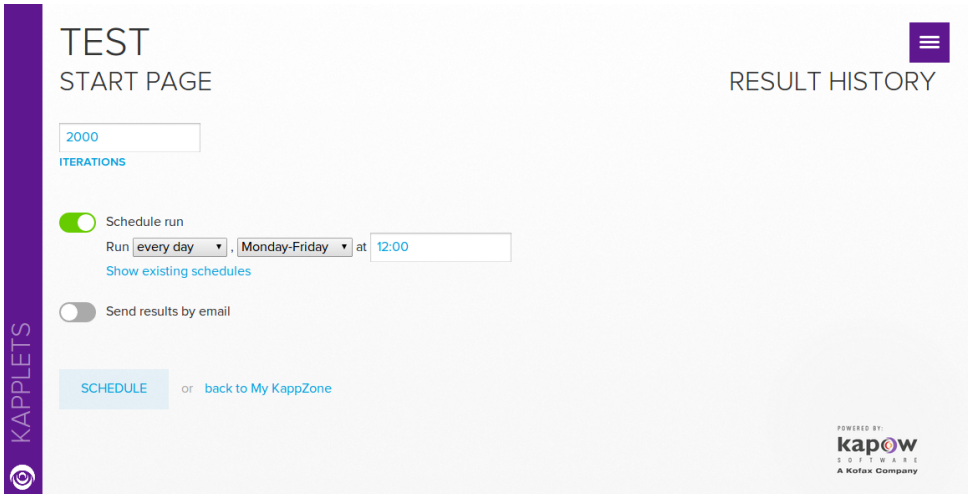


Kapplet Userが「開始」ボタンをクリックすると、Kappletが通常どおり実行され、Start Actionが終了したときに電子メールがユーザーに送信されます。

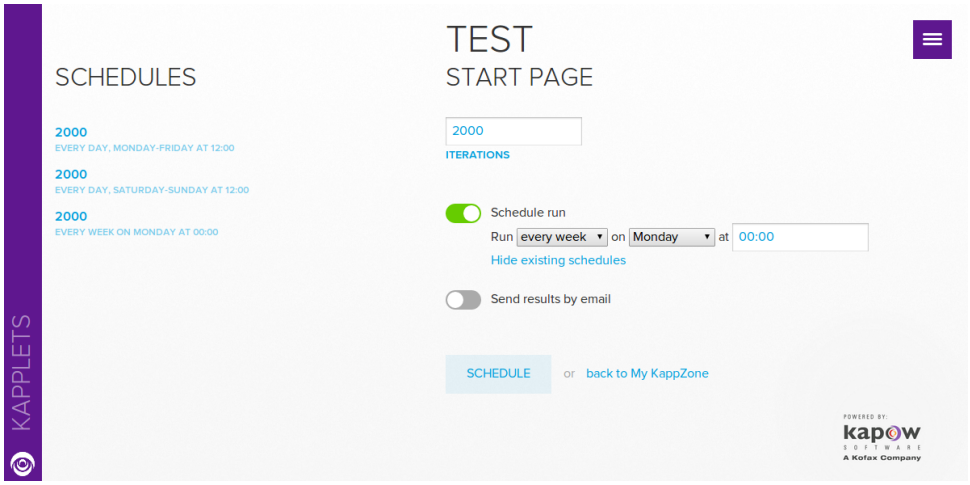
Kappletsのスケジュール

一部のKappletsは、毎日正午または毎週金曜日午後4時50分など、事前定義の間隔で実行する必要があります。Kappletをスケジュールするには、Kapplet Userは、次の例に示すように、実行のスケ

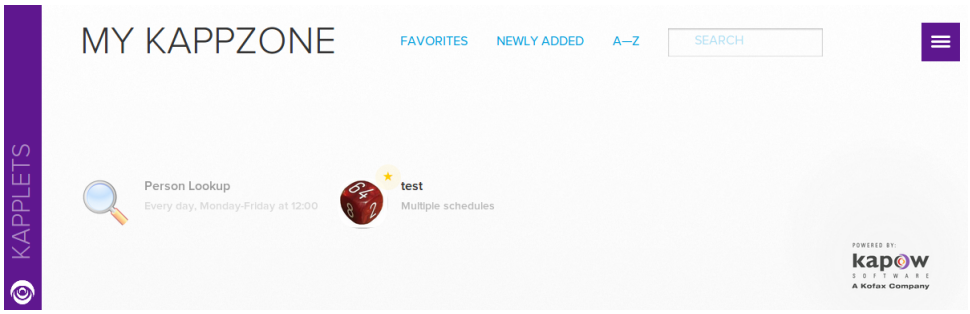
ジュール・オプションをオンに切り替えて、適切なスケジュール情報を入力する必要があります。時間指定は、Management Consoleが実行されているサーバーのタイムゾーンであると解釈されることに注意してください。また、実行のスケジュール・オプションをオンに切り替えると、「開始」ボタンが「スケジュール」ボタンに変わることにも注意してください。



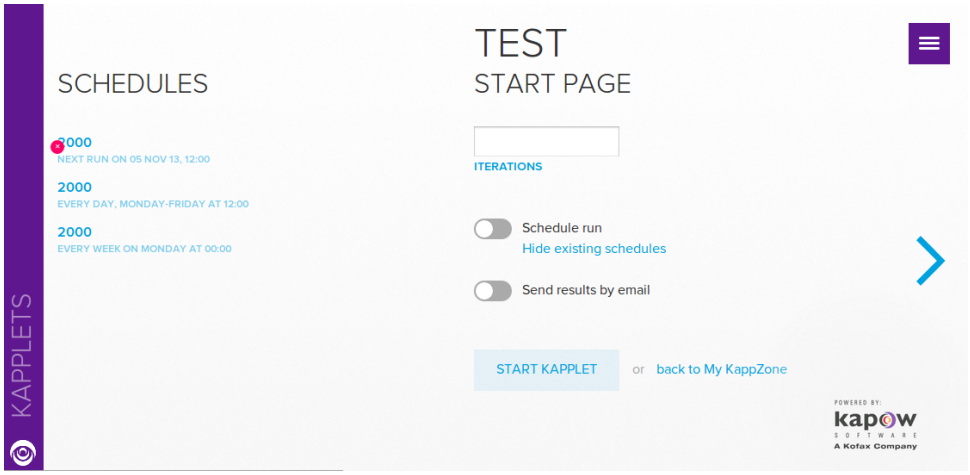
Kappletがスケジュールされると、Kapplet Userが指定したとおりに定期的に実行されます。Installed Kappletに対して構成されたスケジュール(ある場合)は、既存のスケジュールの表示リンクをクリックすると、画面の左側にリストできます。このリストの例を次に示します。



Kapplet UserがInstalled Kappletsをリストしている場合は、スケジュールがKappletに表示されます。



スケジュールを削除するには、左側のスケジュール・リストのスケジュールに対する削除アイコンをクリックします。



スケジュールおよび電子メール通知は、Kappletsに対して結合できることに注意してください。

ブランドのカスタマイズ

KappZoneおよびMy KappZoneのブランドは、自社のカラー・スキームに準拠するように、さらにKapowのロゴを自社のロゴに置き換えるようにカスタマイズできます。

KappZoneおよびMy KappZoneのブランドを変更するには、Management ConsoleのKappletsセクションのブランド・タブに移動します。最初は、デフォルト(Kapow)ブランドが使用されます。選択したロゴおよび色のプレビューが、右側にどのように表示されるか注意してください。

ブランドを変更するには、最初にカスタム・ブランドを選択します。次に、「My KappZone」セクションの上部バーの背景として使用する色コードを指定します。デフォルトのブランドで使用されている紫などの濃色を選択した場合は、対照色として淡色を選択する必要があり、これは、選択した背景色の上に表示される「My KappZone」のテキストおよびいくつかのアイコンに使用されます。淡い背景色を選択した場合は、濃い対照色をお薦めします。

My KappZoneおよびKappZoneの上部、および一部のサブページの縮小版にあるKapowロゴを置換するロゴをアップロードするには、グレーのドロップ・ゾーンにドラッグするか、またはドラッグ・アン

ド・ドロップがサポートされていないブラウザを使用している場合は、ドロップ・ゾーンをクリックします。

第13章 ロボットを使用したプログラミング

ロボットは、API (Javaまたは.Net)を介してRoboServerで実行されます。独自のアプリケーションでAPIを直接使用するか、またはManagement Consoleを使用してロボットを実行する場合は間接的にAPIを使用できます。

この部分のドキュメントは、Javaおよび.Net APIを使用したロボットの直接実行について説明します。

- 14章「Javaプログラマーズ・ガイド」では、Javaプログラムで使用できるAPIについて説明します。
- 15章「.NETプログラマーズ・ガイド」では、C#プログラムを含め、.NETアプリケーションで使用するAPIについて説明します。

Java、.NETに関するAPI参照ドキュメントを使用できます(ガイドには適切なリンクが含まれています)。参照ドキュメントは、インストールの一部としてインストールもされ、開発環境から簡単に使用できます。

第14章 Javaプログラマーズ・ガイド

このガイドでは、Kapow Katalyst 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。ガイドでは、Design Studioチュートリアルを完了していて、単純なロボットの作成方法を理解していること、およびJavaプログラミング言語の知識が十分あることを前提としています。

プログラマーズ・ガイドは、APIの大部分が非推奨になり、新しい実行APIが作成されたため、バージョン8.3用に完全に書き直されました。APIはまだ下位互換性がありますが、非推奨のクラスは将来のリリースで削除されるため、新しいAPIを理解し、新しいAPIを使用するように既存のアプリケーションの書換えを検討することをお薦めします。

古いRobotExecutorは、次の理由から非推奨になっています。

- APIによってRQLExceptionがスローされた後も、ロボットがRoboServer上で実行し続けていました。
- RoboServerがクライアントとの接続を失った場合でも、ロボットが実行し続け、その結果、実行されたすべての戻り値に対するログ・エラーが発生していました。
- リクエストの配布時に、配布ポリシーでサーバー容量を調べていませんでした。
- カスタムRQLHandlerの実装時に表面上はわからない危険性が多数あったため、オブジェクト・ストリーミングの実装が面倒でした。

古いJavaプログラマーズ・ガイドは、http://help.kapowtech.com/8.2/topic/doc/java/Top.htmlでまだ参照できます。

特定のクラスに関する詳細は、JavaDocを参照してください。

基本

Management Consoleで実行されるロボットは、Java APIを使用して実行されます。Java APIを使用すると、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RoboServerに送信できます。これは、Management Consoleがクライアントとして機能し、RoboServerがサーバーとして機能する標準的なクライアント-サーバー設定です。

APIの使用によって、JavaベースのアプリケーションはRoboServerに対するクライアントになることができます。データベースにデータを格納するロボットの実行に加えて、ロボットがデータをクライアント・アプリケーションに直接戻すようにすることもできます。いくつか例を示します。

- 複数のロボットを使用してフェデレーテッド検索を実行し、複数のソースからの結果をリアルタイムで集計します。
- アプリケーション・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ないWebベース・システムにアカウントを作成します。

このガイドの基本の項では、中心的なクラス、およびロボットを実行するためにそれらを使用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、およびRoboServer上でそれらの実行を制御する方法についても説明します。

Java APIはjarファイルであり、Kapow Katalystインストール・フォルダ内の**/API/robosuite-java-api/lib/robosuite-api.jar**に格納されています。詳細は、「重要なフォルダ」を参照してください。このガイドのすべての例も**/API/robosuite-java-api/examples**にあります。Java APIと並んで配置されているのは、APIの外部依存を構成する5つの追加jarファイルです。ロボットの実行などの最も基本的なAPIタスクは、これらのいずれのサード・パーティ・ライブラリも使用せずに行うことができますが、一部の拡張機能には、これらのサード・パーティ・ライブラリを1つ以上使用する必要があります。このガイドの例では、このようなライブラリが必要な場合は指定します。

最初の例

NewsMagazine.robotというロボットの実行に必要なコードの説明から開始します(このロボットはデフォルト・プロジェクトのTutorialsフォルダにあります)。ロボットは、その結果を値を返すステップ・アクションを使用して出力し、これによって、APIを使用したプログラムによる出力の処理を容易にします。その他のロボット(通常はManagement Consoleによってスケジュールで実行されるロボット)は、データベースに格納ステップ・アクションを使用して、それらのデータをデータベースに直接格納し、この場合、ロボットによって収集されたデータは、APIクライアントには戻されません。

次に、NewsMagazineロボットを実行し、その出力をプログラムによって処理する方法について説明します。

表14.1 入力なしのロボットの実行

```
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);
            }
       }
       catch (RQLException e) {
             e.printStackTrace();
       }
     }
   }
```

次に、関係するクラスとその役割について説明します。

表14.2

RoboServer	これは、ロボットを実行できるRoboServerを識別する単純な値オブジェクトです。各RoboServerは、Management

	Consoleによってアクティブ化され、使用前にKCUが割り当てられている必要があります。
Cluster	クラスタは、1つの論理ユニットとして機能するRoboServerのグループです。
Request	このクラスは、ロボット・リクエストの作成に使用されます。リクエストを実行する前に、クラスタをリクエスト・クラスに登録する必要があります。
DefaultRobotLibrary	ロボット・ライブラリは、リクエストで指定されたロボットを検索する場所をRoboServerに指示します。後の例で、様々なロボット・ライブラリのタイプおよびそれらを使用する場合/方法について検討します。
RQLResult	これには、ロボット実行の結果が含まれます。結果には、値レスポンス、ログおよびサーバー・メッセージが含まれます。
RQLObject	値を返すアクションを使用してロボットから返される各値には、RQLObjectとしてアクセスできます。

ここからは、例の各行を見直して、具体的に説明していきます。

これは、RoboServerがlocalhostのポート50000で実行されていることをAPIに通知します。

表14.3

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

これは、1つのRoboServerを含むクラスタを定義しています。クラスタは**Request**クラスに登録され、このクラスタでリクエストを実行できるようになります。各クラスタを1回のみ登録できます。

表14.4 クラスタの登録

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

これは、Library:/TutorialsにあるNewsMagazine.robotというロボットを実行するリクエストを作成します。Library:/は、リクエスト用に構成されたロボット・ライブラリを表します。ここでは、**DefaultRobotLibrary**が使用されており、これは、RoboServerに対してサーバーのローカル・ファイル・システムでロボットを検索するように指示します。ロボット・ライブラリの使用方法の詳細は、「???を参照してください。

表14.5

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

これは、**MyCluster**というクラスタ(以前に登録したクラスタ)でロボットを実行し、ロボットが終了すると結果を返します。デフォルトでは、ロボットでAPI例外が生成されると、実行で例外がスローされます。

表14.6

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

ここでは、抽出した値を処理します。最初に、Postというタイプの抽出されたすべての値を取得し、それらを順に処理します。各RQLObjectに対して、Postタイプの属性にアクセスして、結果を印刷します。属性とマッピングについては、後述の項で説明します。

表14.7

```
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);
}
```

ロボット入力

APIを介して実行されるほとんどのロボットが、検索キーワードまたはログイン資格証明などの入力によってパラメータ化されます。ロボットへの入力はRoboServerに対するリクエストの一部であり、リクエストで**createInputVariable**メソッドを使用して提供されます。短いコード・フラグメントについて説明します。

表14.8 暗黙的な**RQLObjectBuilder**を使用した入力

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

ここでは、**Request**を作成し、**createInputVariable**を使用してuserLoginという入力変数を作成します。次に、**setAttribute**を使用して、入力変数のユーザー名およびパスワードの属性を構成します。

前述の例は、一般的な簡易表記ですが、**RQLObjectBuilder**を使用して処理を冗長に表すこともできます。

表14.9 明示的な**RQLObjectBuilder**を使用した入力

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

2つの例は同じです。最初は、匿名**RQLObjectBuilder**でカスケード・メソッド呼出しを利用しているため、短くなっています。

RoboServerがこのリクエストを受信すると、次のことが発生します。

- RoboServerによって、(リクエスト用の**RobotLibrary**が構成されているすべての場所から)Input.robotがロードされます。
- RoboServerによって、ロボットにuserLoginという変数があり、この変数に入力のマークが付けられていることが検証されます。

- RoboServerによって今度は、**setAttribute**を使用して構成した属性が、変数userLoginのタイプと互換性があることが検証されます。これは、タイプにusernameおよびpasswordという属性があり、これらが両方ともテキストベースの属性であることを意味します(次の項でAPIとDesign Studio属性の間のマッピングについて説明します)。

- すべての入力変数に互換性がある場合は、RoboServerによってロボットの実行が開始されます。

ロボットに複数の入力変数が必要な場合は、ロボットを実行するためにそれらをすべて作成する必要があります。構成する必要があるのは必須の属性のみであり、APIを使用して構成しない非必須の属性はnull値になります。FacebookとTwitterの両方にログインする必要があるロボットがあるとすると、入力を次のように定義できます。

表14.10

```
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");
```

属性タイプ

Design Studioで新しいタイプを定義する場合は、各属性に対する属性タイプを選択します。これらの属性の一部には、短いテキスト、長いテキスト、パスワード、HTML、XMLなどのテキストを含めることができ、ロボット内部で使用する場合は、これらの属性に格納するテキストに対する要件がある場合があります。XML属性にテキストを格納する場合、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、タイプがロボット内で使用されるときに行われますが、APIではタイプに関して何もわからないため、同じ方法での属性値の検証は行われません。このため、APIの属性タイプが8のみであるのに対して、Design Studioでは19使用可能です。次の表に、APIとDesign Studio属性タイプの間のマッピングを示します。

表14.11 APIとDesign Studioのマッピング

API属性タイプ	Design Studio属性タイプ
テキスト	**Short Text, Long Text, Password, HTML, XML, Properties, Language, Country, Currency, Refind Key**
整数	**Integer**
ブール	**Boolean**
数値	**Number**
文字	**Character**
日付	**Date**
セッション	**Session**
バイナリ	**Binary, Image, PDF**

その後、API属性タイプは次のようにJavaにマップされます。

表14.12 属性に対するJavaタイプ

属性タイプ	Javaクラス
テキスト	**java.lang.String**
整数	**java.lang.Long**
ブール	**java.lang.Boolean**

属性タイプ	Javaクラス
数値	**java.lang.Double**
文字	**java.lang.Character**
日付	**java.util.Date**
セッション	**com.kapowtech.robosuite.api.construct.Session**
バイナリ	**com.kapowtech.robosuite.api.construct.Binary**

RQlObjectBuilderの**setAttribute**メソッドはオーバーロードされているため、引数として正しいJavaクラスを使用しているかぎり、APIを使用した属性の構成時に属性タイプを明示的に指定する必要はありません。次の例は、考えられるすべての(Design Studio)属性タイプを使用してオブジェクトに対する属性を設定する方法を示しています。

表14.13 推奨のsetAttribute使用方法

```
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>bla</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");
```

前述の例では、new Long(42L)およびnew Double(12.34)を明示的に使用していますが、Autoboxingのため42Lおよび12.34で十分です。また、null値をキャストする必要があることに注意が必要であり、これは、そうしないとJavaコンパイラでは、オーバーロードされている**setAttribute**メソッドのいずれを呼び出す必要があるかを判断できないためです。ただし、未構成の属性は自動的にnullになるため、nullを明示的に設定する必要はありません。

APIを使用した入力の作成時に、AttributeおよびAttributeTypeを明示的に指定することは可能です。この方法は、お薦めしませんが、まれに必要になることがあり、次のようになります。

表14.14 非推奨のsetAttribute使用方法

```
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));
```

このように、すべての属性値を文字列の形式で提供する必要があります。文字列値はその後、提供した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。これが役立つのは、Kapow Katalyst Java APIに加えて他の汎用APIを構築する場合のみです。

実行パラメータ

createInputVariableメソッド以外に、**Request**には、RoboServerでのロボットの実行方法を制御するメソッドが多数あります。

表14.15 Requestの実行制御メソッド

setMaxExecutionTime(int seconds)	ロボットの実行時間を制御します。この時間を経過すると、ロボットはRoboServerによって停止されます。タイマーは、ロボットが実行し始めるまで開始しないため、ロボットがRoboServerのキューに入れられた場合、これは考慮されません。
setStopOnConnectionLost(boolean)	true（デフォルト）の場合、RoboServerでクライアント・アプリケーションとの接続が失われたことが検出された場合はロボットが停止します。この値をfalseに設定するには、十分な理由が必要であり、これを処理するようにコードが作成され

	ていない場合、アプリケーションは期待どおりに実行されません。
setStopRobotOnApiException(boolean)	true（デフォルト）の場合は、最初のAPI例外の発生後に、RoboServerによってロボットが停止されます。デフォルトでは、ステップで実行に失敗すると、ロボットのほとんどのステップでAPI例外が発生します。これはステップ・エラー処理タブで構成します。 falseに設定すると、ロボットはAPI例外に関係なく実行し続けますが、アプリケーションで**RequestExecutor**ストリーミング実行モードを使用していないかぎり、例外はやはりexecute()によってスローされるため、これをfalseに設定する場合は十分注意してください。
setUsername(String), setPassword(String)	RoboServer資格証明を設定します。RoboServerは、認証を必要とするように構成できます。このオプションを有効化すると、クライアントは資格証明を提供する必要があり、提供しないとRoboServerによってリクエストを拒否されます。
setRobotLibrary(RobotLibrary)	ロボット・ライブラリは、リクエストで指定されたロボットを検索する場所をRoboServerに指示します。後の例で、様々なロボット・ライブラリのタイプおよびそれらを使用する場合/方法について検討します。
setExecutionId(String)	このリクエストの実行IDを設定できます。提供しない場合は、RoboServerによって自動的に生成されます。実行IDは、ロギングのために使用され、クライアントがロボットをプログラムによって停止できる必要がある場合にも必要です。IDは、(経時的に)グローバルに一意である必要があります。2つのロボットが同じ実行IDを使用した場合は、ログに一貫性がなくなります。
setProject(String)	これは、ロギングの目的にのみ使用します。Management Consoleは、このフィールドを使用してログ・メッセージをプロジェクトにリンクするため、ログ・ビューをプロジェクトでフィルタできます。 アプリケーションで**RepositoryRobotLibrary**を使用していない場合はおそらく、この値を設定して、このロボットが属しているプロジェクト(ある場合)をRoboServerロギング・システムに通知する必要があります。

ロボット・ライブラリ

Design Studioで、ロボットはプロジェクトにグループ化されます。ファイル・システムを調べると、これらのプロジェクトは**Library**というフォルダで識別されることがわかります。詳細は、「ライブラリおよびプロジェクト」を参照してください。

RoboServerに対する実行リクエストを作成するとき、次のようにロボットのURLでロボットを識別します。

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

ここで、**Library:/**は、ロボット・ライブラリのシンボリック参照であり、この中でRoboServerはロボットを検索する必要があります。**RobotLibrary**は、ビルダーで次のように指定されます。

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

3つの異なるロボット・ライブラリ実装があり、いずれを選択するかはデプロイメント環境に依存します。

表14.16 ロボット・ライブラリ

ライブラリ・タイプ	説明
DefaultRobotLibrary	これは、現在のプロジェクト・フォルダでロボットを検索するようにRoboServerを構成します。このフォルダはSettingsアプリケーションで定義します。 複数のRoboServerがある場合は、すべてのRoboServerにロボットをデプロイする必要があります。 このロボット・ライブラリがキャッシュされていない場合は、ロボットはすべての実行でディスクからリロードされます。これは、ロボットが頻繁に変更される開発環境でライブラリを使用できるようにしますが、本番環境には適していません。

EmbeddedFileBasedRobotLibrary	このライブラリは、RoboServerに送信される実行リクエストに埋め込まれます。このライブラリを作成するには、ロボットとそのすべての依存項目(タイプ、スニペットおよびリソース)を含むzipファイルを作成する必要があります。これを実行するには、Design Studioの「ツール」->ロボット・ライブラリ・ファイルの作成メニューを使用します。
	
	ライブラリはすべてのリクエストとともに送信され、大規模ライブラリの場合はオーバーヘッドがある程度大きくなりますが、ライブラリはRoboServerにキャッシュされ、最大限可能なパフォーマンスが提供されます。
	
	1つの利点は、ロボットとコードを1つのユニットとしてデプロイできるため、QA環境から本番環境への完全な移行が提供されることです。ただし、ロボットが頻繁に変更される場合は、頻繁に再デプロイする必要があります。
	
	次のコードを使用すると、リクエストに対して埋込みのロボット・ライブラリを構成できます。
	
	`Request request =`
	`  new Request`
	`  ("Library:/Tutorials/NewsMagazine.`
	`  robot");`
	`RobotLibrary library =`
	`new EmbeddedFileBasedRobotLibrary`
	`  (new FileInputStream`

ライブラリ・タイプ	説明
	("c:\\embeddedLibrary.robotlib")); **request.setRobotLibrary(library);**
RepositoryRobotLibrary	これは最も柔軟な**RobotLibrary**です。 このライブラリは、Management Consoleの組込みリポジトリをロボット・ライブラリとして使用します。このライブラリを使用する場合、RoboServerがManagement Consoleと通信すると、Management Consoleによってロボットとその依存項目を含むロボット・ライブラリが送信されます。 キャッシュは、Management Console内とRoboServer内の両方で、ロボットごとに発生します。Management Console内では、生成されたライブラリは、ロボットとその依存項目に基づいてキャッシュされます。RoboServerでは、キャッシュはタイムアウトに基づいているため、リクエストごとにManagement Consoleに問い合せる必要はありません。さらに、RoboServerとManagement Consoleの間のライブラリ・ロードでは、バンド幅をさらに削減するために、HTTP public/privateキャッシュが使用されます。 NewsMagazine.robotがManagement Consoleにアップロードされると、ロボットの実行時に、次のようにリポジトリ・ロボット・ライブラリを使用できます。 **Request request = new Request** **("Library:/Tutorials/NewsMagazine.** **robot");** **RobotLibrary library =** **new RepositoryRobotLibrary** **("http://localhost:50080",** **"Default Project", 60000);** **request.setRobotLibrary(library);** これは、ローカルのManagement Consoleからロボットをロードし、それを1分間キャッシュした後、新しいバージョンのロボット(そのタイプとスニペット)が変更されたかどうかManagement Consoleを確認するようにRoboServerに指示します。 さらに、**Library:/**プロトコルを介してロードされるリソースでは、RoboServerはManagement Consoleから直接リソースをリクエストします。

拡張

この項では、APIの拡張機能の一部についてもう少し詳しく説明します。これらには、出力ストリーミング、ロギングおよびSSL構成、さらに並列実行があります。

負荷分散およびフェイルオーバー

表14.17 ロード・バランス・エグゼキュータ

```
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);
```

RequestExecutor内部で発生している内容についてもう少し詳しく説明します。エグゼキュータには、RoboServerの配列が与えられます。エグゼキュータが構成されると、各RoboServerに接続しようとします。接続すると、各RoboServerにサーバーの構成方法を検出するためにpingリクエストを送信します。

負荷は、RoboServer上の未使用実行スロットの数に基づいて、クラスタ内の各オンラインRoboServerに分散されます。次のリクエストは常に、最も使用可能なスロットがあるRoboServerに配布されます。使用可能な実行スロットの数は、初期pingレスポンスから取得し、エグゼキュータは、それが開始した各ロボットおよびその完了時を追跡します。RoboServer上の実行スロットの数は、「ーバー」タブの最大同時ロボット数によって決まります。

RoboServerがオフラインになると、pingリクエストに正常に応答するまでは、ロボットの実行リクエストは受け入れられません。

2つのクライアント・ルール

RoboServerの特定のクラスタを使用するAPIクライアントは1つのみにする必要があります。同じRoboServerに対してロボットを実行しているJVMが複数ある場合は、結果的にパフォーマンスが低下します。

エグゼキュータ・ロガー

リクエストを実行したとき、ロボットでエラーが生成された場合は、実行メソッドで例外がスローされます。他のタイプのエラーおよび警告は、**ExecutorLogger**インタフェースを介してレポートされます。前述の例では、ロボットの実行時に**ExecutionLogger**は提供しておらず、これは、システム出力に書き込むデフォルト実装を使用することを意味します。次に、RoboServerの1つがオフラインになった場合の**ExecutorLogger**のレポート方法について説明します。

例では、オンラインではないRoboServerを含むクラスタを構成します。

表14.18 エグゼキュータ・ロガー、オフライン・サーバーの例

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

この例を実行すると、コンソールに次が出力されます。

表14.19 エグゼキュータ・ロガー、オフラインRoboServerのコンソール出力

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

アプリケーションで**System.out**に直接書き込まないようにすることが多く、その場合は異なる**ExecutorLogger**実装を提供でき、これは、次のようにクラスタの登録時に実行できます。

表14.20 **DebugExecutorLogger**の使用

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

この例では、**System.out**にも出力する**DebugExecutorLogger**()を使用しますが、出力されるのはAPIデバッグが有効化されている場合のみです。または、エラー・メッセージの処理方法を制御するために、**ExecutorLogger**の独自の実装を提供できます。その他の詳細は、**ExecutorLogger** JavaDocを参照してください。

データ・ストリーミング

ロボット実行の結果をリアルタイムで提示する必要がある場合があります。このような場合は、ロボットがその実行を終了し、RQLResultにアクセスするのを待機するかわりに、抽出した値をAPIで即時に返す必要があります。

APIでは、ロボットが返した値をAPIが受信するたびに、コールバックを受信する機能が提供されています。これは、**RobotResponseHandler**インタフェースを介して実行します。

表14.21 **AbstractFailFastRobotResponseHandler**を使用したレスポンス・ストリーミング

```
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
   RQLException {
   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);
   }
   catch (RQLException e) {
   e.printStackTrace();
   }
 }
}
```

前述の例では、**Request**の2番目の実行メソッドが使用されており、これは、ロボットを実行するクラスタ名の他に**RobotResponseHandler**を必要とします。この例では、デフォルトのエラー処理を提供する**AbstractFailFastRobotResponseHandler**を拡張して**RobotResponseHandler**を作成しているため、処理する必要があるのはロボットが返す値のみです。

handleReturnedValueメソッドは、APIがRoboServerから戻り値を受信するたびに呼び出されます。この例で使用した**AbstractFailFastRobotResponseHandler**では、非ストリーミングの実行メソッドと同じ方法で例外がスローされます。これは、ロボットで生成された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にはいくつかのメソッドがあり、これは3つのカテゴリにグループ化できます。

ロボット・ライフサイクル・イベント	ロボットの起動時やその実行の終了時など、RoboServer上のロボットの実行状態が変わったときに呼び出されるメソッド。
ロボット・データ・イベント	ロボットがデータまたはエラーをAPIに返したときに呼び出されるメソッド。
追加のエラー処理	RoboServer内部またはAPIのエラーのいずれかによって呼び出されるメソッド。

表14.22 RobotResponseHandler - ロボット・ライフサイクル・イベント

メソッド名	説明
void equestSent(RoboServer roboServer, ExecuteRequest request)	リクエストを実行するサーバーを**RequestExecutor**が検出したときに呼び出されます。
void requestAccepted(String executionId)	検出したRoboServerがリクエストを受け入れ、それをキューに入れたときに呼び出されます。
void robotStarted(Stoppable stoppable)	RoboServerがロボットの実行を開始したときに呼び出されます。これは通常、RoboServerに過度の負荷がかかっている場合または複数のAPIクライアントによって使用されている場合を除いて、ロボットがキューに入れられた直後に発生します。
void robotDone(RobotDoneEvent reason)	ロボットがRoboServer上での実行を終了したときに呼び出されます。**RobotDoneEvent**は、実行が正常に終了したか、エラーで終了したか、または停止されたかを指定するために使用されます。

表14.23 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)	ロボットが値を返すアクションを実行し、値がソケット経由でAPIに返されたときに呼び出されます。
void handleRobotError (RobotErrorResponse response, Stoppable stoppable)	ロボットでAPI例外が発生し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例外後に実行を停止します。この動作は、**Request.setStopRobotOnApiException(false)**を使用して上書きでき、この場合、このメソッドは複数回呼び出されます。これは、生成されたエラーに関係なく、データ・ストリーミング・ロボットで実行を継続する場合に有効です。
void handleWriteLog (RobotMessageResponse response, Stoppable stoppable)	ロボットがログの作成アクションを実行した場合に呼び出されます。これは、ロボット内から追加のロギング情報を提供する場合に有効です。

表14.24 RobotResponseHandler - 追加のエラー処理

メソッド名	説明
void handleServerError (ServerErrorResponse response, Stoppable stoppable)	RoboServerがエラーを生成した場合、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RoboServer内部でエラーが発生してロボットを起動できない場合に呼び出されます。
handleError(RQLException e, Stoppable stoppable)	API内部でエラーが発生した場合に呼び出されます。クライアントがRoboServerとの接続を失った場合が最も一般的です。

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

これらのメソッドの一部ではRQLExceptionをスローでき、これを行う場合は、結果に注意する必要があります。ハンドラを呼び出すスレッドは、**Request.execute()**を呼び出すスレッドであり、これは、スローされたいずれかの例外によって呼出しスタックが処理されず、実行メソッドが範囲外となることを意味します。**handleReturnedValue**、**handleRobotError**または**handleWriteLog**に応じて例外をスローした場合、**Stoppable.stop()**の起動はユーザーが行う必要があり、これを起動しないと、**Request.execute()**の呼出しが完了した場合でもロボットが実行を継続する場合があります。

データ・ストリーミングは、次のユースケースの1つで最もよく使用されます。

- 結果がユーザーにリアルタイムで提示される、Ajaxベースのアプリケーション。データがストリーミングされなかった場合、結果はロボットの実行が終了するまで表示できません。
- ロボットの実行全体でクライアントがメモリーにデータを格納できないほどの大量のデータを返すロボット。
- 抽出した値がロボット実行で同時に処理されるように最適化する必要があるプロセス。
- データベースにデータをカスタム・フォーマットで格納するプロセス。
- API例外のカスタム処理を無視するまたは必要とするロボット(次を参照)。

表14.25 **AbstractFailFastRobotResponseHandler**を使用したレスポンスおよびエラー収集

```
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());
 }
 }
 catch (RQLException e) {
 e.printStackTrace();
 }
 }
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 RQLException {
_output.add(response);
 }
@Override
public void handleRobotError
(RobotErrorResponse response, Stoppable stoppable)
throws RQLException {
// 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;
    }
  }
}
```

前述の例は、返された値およびエラーを収集する**RobotResponseHandler**の使用方法を示しています。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必要がある場合に有効であり、これは、Webサイトが不安定で、タイムアウトになることがある場合に役立ちます。ハンドラによって収集されるのはロボット・エラー(API例外)のみであることに注意が必要であり、RoboServerに対する接続が失われた場合は、やはり**Request.execute()**によって**RQLException**がスローされます(また、ロボットがRoboServerによって停止されます)。

詳細は、**RobotResponseHandler** JavaDocを参照してください。

SSL

APIは、RQLServiceを介してRoboServerと通信します。RQLServiceはRoboServerのコンポーネントの1つであり、特定のネットワーク・ポートでAPIリクエストをリスニングします。RoboServerを起動する際に、RoboServerで暗号化SSLサービスを使用するか、プレーン・ソケット・サービスを使用するか、あるいは両方(2つの異なるポートを使用)を使用するかを指定します。クラスタ内のすべてのRoboServerで同じRQLService (ただしポートは異なる場合があります)を実行している必要があります。

RoboServerを、次のようにポート50043でSSL RQLServiceを使用して起動したとします。

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

次のコードを使用できます。

表14.26 SSL構成

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

実行する必要があるのは、SSLクラスタとしてクラスタを作成し、各RoboServerで使用されるSSLポートを指定することのみです。これで、RoboServerとAPI間のすべての通信が暗号化されます。

この例が動作するには、アプリケーション・クラスパスに**commons-ssl-0.3.8.jar**が必要であり、これはKapowインストール内のAPI jarファイルの隣にあります。

データ暗号化以外に、SSLには、リモート・パーティのアイデンティティを検証する機能があります。不正なWebサイトはそれ自身ではない何かになりすますことがあるため、このタイプの検証はインターネットに関して非常に重要です。多くの場合、APIクライアントとRoboServerは同じローカル・ネットワーク上に存在するため、他のパーティのアイデンティティの検証が必要なることはあまりありませんが、必要になった場合に備えてAPIではこの機能がサポートされています。

アイデンティティ検証はほとんど使用されないため、このガイドでは説明しません。関心がある場合は、Java APIに付属のSSLの例を参照してください。

並列実行

Requestの両方の実行メソッドがブロック状態であるということは、各ロボット実行にスレッドが必要であることを意味します。これまで使用した例はすべて、メイン・スレッドでロボットを直接実行しており、これは、順次方式で一度に1つのロボットしか実行できないため、通常はお薦めしません。

2つのチュートリアル・ロボットを並列で実行する例について説明します。この例では、マルチスレッド化のために**java.util.concurrent**ライブラリが使用されます。

表14.27 マルチスレッド化の例

```
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);
   }
   catch (RQLException e) {
   e.printStackTrace();
   }
  }
 }
}
```

前述の例では、2つのスレッドがある**ThreadPoolExecutor**が作成され、次に4つの**RobotRunnable**を作成して、スレッド・プールでそれらを実行します。スレッド・プールには2つのスレッドがあるため、2つのロボットが即時に実行を開始し、残りの2つは**LinkedBlockingQueue**に入れられ、最初の2つのロボットの実行が終了してスレッド・プールのスレッドが使用可能になると、順に実行されます。

Requestは、実行メソッド内で複製される状況が発生しないように、可変であることに注意してください。**Request**は可変であるため、同じ**Request**を別々のスレッドで変更しないでください。

リポジトリ統合

Management ConsoleでRoboServerのクラスタも指定し、これらは、スケジュール済ロボット、およびRESTサービスとして実行されるロボットを実行するために使用します。APIでは、RepositoryClientを使用してManagement Consoleからクラスタ情報を取得できました。詳細はRepositoryClientのドキュメントを参照してください。

表14.28 リポジトリ統合

```
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);
  }
}
```

前述の例は、localhostにデプロイされたManagement Consoleに接続する**RepositoryClient**の作成方法を示しています。この例が動作するには、**commons-logging-1.1.1.jar,commons-codec-1.4.jar,commons-httpclient-4.1.jar**がクラスパスに含まれている必要があります。

認証は有効化されていないため、ユーザー名とパスワードの両方に**null**が渡されます。**RepositoryClient**を登録するとき、Management Consoleに存在するクラスタの名前を指定すると、これはManagement Consoleに問い合せてこのクラスタ用に構成されているRoboServerのリストを取得し、Management Console上でクラスタ構成が更新されていないかどうかを2分間隔で確認します。

この統合によって、Management Consoleでクラスタを作成でき、これはManagement Consoleユーザー・インタフェースを使用して動的に変更できます。Management Consoleを使用する場合は、API使用を伴うクラスタは、2つのクライアント・ルールに違反しているため除外して、ロボットのスケジュールには使用しないでください。

内部処理

この項では、クラスタを登録し、**Request**を実行したときに、内部で行われている処理について説明します。

Clusterを**Request**に登録する際に、背後で**RequestExecutor**が作成されます。この**RequestExecutor**は、クラスタ名をキーとして使用して、**Map**に格納されます。リクエストを実行すると、提供したクラスタ名を使用して関連する**RequestExecutor**が検索され、リクエストが実行されます。

短い例について説明します。

表14.29 通常の実行

```
public static void main(String[] args) throws
InterruptedException, RQLException {
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");
RQLResult result = request.execute("MyCluster");
System.out.println(result);
}
```

今度は、同じ例を非表示の**RequestExecutor**を使用して直接作成します。

表14.30 内部処理の実行

```
public static void main(String[] args) throws InterruptedException,
RQLException {
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); }
```

RequestExecutorがデフォルトで非表示の理由は、追跡する必要がないためです。クラスタごとに**RequestExecutor**を1つのみ作成できるため、それを直接使用する場合は、アプリケーション全体でそれに対する参照を保管する必要があります。 **Request.registerCluster(cluster)**の使用は、**RequestExecutor**およびライフサイクルのルールを最良の方法で無視できることを意味します。

RequestExecutorには、必要な状態、およびロード・バランシングとフェイルオーバー機能を提供するロジックが含まれています。**RequestExecutor**の直接使用では、いくつかの特別な機能も提供されており、これについてこの後説明します。

RequestExecutorの機能

RequestExecutorがリポジトリに接続されていない場合は、**addRoboServer(..)**および**removeRoboServer(..)**を呼び出して、RoboServerを動的に追加および削除できます。これらのメソッドでは、**RequestExecutor**内で使用される配布リストが変更されます。

RequestExecutor.getTotalAvailableSlots()では、内部配布リストのすべてのRoboServerの未使用実行スロット数が返されます。

これらのメソッドを使用すると、使用可能な実行スロットの数が少なくなったときにすぐに、RoboServerを**RequestExecutor**に動的に追加できます。

RequestExecutorの作成時に、オプションで**RQLEngineFactory**を提供できます。**RQLEngineFactory**を使用すると、RoboServerの接続時にいずれの**RQLProtocol**を使用するかをカスタマイズできます。これが必要になる状況はほとんどなく、たとえば、セキュリティの向上のためにクライアント証明書を使用する場合などです。詳細は、「APIクライアント証明書」を参照してください。

Webアプリケーション

RequestExecutorには、RoboServerとの間のリクエストの送受信、および既知の各RoboServerの定期的なpingに使用する内部スレッドが多数含まれています。これらのスレッドにはすべてデーモンのマークが付けられ、これは、メイン・スレッドが存在する場合にJVMの停止を妨げないことを意味しています。デーモン・スレッドの詳細は、Thread JavaDocを参照してください。

Webアプリケーション内で**RequestExecutor**を使用する場合、JVMの存続期間はWebアプリケーションより長くなり、Webコンテナの実行中は、Webアプリケーションをデプロイおよびアンデプロイできます。これは、Webアプリケーションは、それ自体で作成したスレッドを停止する必要があり、停止しない場合は、Webアプリケーションのアンデプロイ時にメモリー・リークが発生します。メモリー・リークが発生する理由は、実行中のスレッドが参照しているオブジェクトはそのスレッドが停止するまでガベージ・コレクションされず、その結果、これらのスレッドがアプリケーションのアンデプロイ時に停止されない場合は、オブジェクトがガベージ・コレクションされることがないためです。

Webアプリケーション内で**RequestExecutor**を使用する場合は、コードでこれらの内部スレッドをシャットダウンする必要があり、これは、コードで**RequestExecutor**を明示的に作成した場合は、**Request.shutdown()**または**RequestExecutor.shutdown()**を呼び出すことによって行います。

この例は、Webアプリケーションのアンデプロイ時に、**ServletContextListener**を使用してAPIを正しくシャットダウンする方法を示しています。アプリケーションのweb.xmlにコンテキスト・リスナーを定義する必要があります。

表14.31 Webアプリケーションでの正しいシャットダウン

```
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は、Webコンテナによってアプリケーションがアンデプロイされたときに呼び出されます。ここでは、**Request.shutdown()**を呼び出して、非表示の**RequestExecutor**内のすべての内部スレッドが確実に正しく停止されるようにします。

contextInitializedは、未確認の例外をスローできないため、**ClusterAlreadyDefinedException**を**RunTimeException**にラップする必要があります。開発者は、この場所の**ClusterAlreadyDefinedException**を無視しがちですが、これは、アプリケーションで他のクラスタを定義していないためスローできないという主張によるものです。ただし、Java Webコンテナのクラス・ローダー階層によって、実際には、アプリケーションが2回デプロイされた場合はこの例外を取得することが可能です。ただし、これは、APIのjarファイルが共通のクラス・ローダーによってロードされた場合のみ発生し、個々のアプリケーションのクラス・ローダーによってロードされた場合は発生しません。

APIデバッグ

これが必要になることはほとんどありませんが、APIでは、デバッグ目的の追加情報を提供できます。APIデバッグを有効にするには、システム・プロパティ**DEBUG_ON**を構成する必要があります。このプロパティの値は、APIのパッケージ/クラス名にする必要があります。

たとえば、APIとRoboServerの間のデータ伝送に関心がある場合は、パッケージ**com.kapowtech.robosuite.api.java.rql.io**のデバッグ情報を問い合せることができます。開発中は、次のように、システム・プロパティをコードに直接設定して実行できます。

表14.32 デバッグの有効化

```
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);
```

本番でアプリケーションをデバッグしている場合は、次のように、コマンドラインを介してシステム・プロパティを定義します。

表14.33 デバッグの有効化

```
java -DDEBUG_ON=com.kapowtech.robosuite.api.java.rql.io Tutorial1
```

複数パッケージからのデバッグに関心がある場合は、パッケージ名を,(カンマ)で区切ります。パッケージ名のかわりに、すべてのパッケージからのデバッグが出力されるように引数**ALL**を指定できます。

リポジトリAPI

リポジトリAPIを使用すると、Management Consoleのリポジトリを問い合せて、プロジェクト、ロボットおよびロボットの呼出しに必要な入力のリストを取得できます。また、ロボット、タイプおよびリソース・ファイルをプログラムによってデプロイすることもできます。

依存性

リポジトリAPIを使用するには、次のライブラリが必要であり、すべてのライブラリがKapow Katalystインストール・フォルダ内の**API/robosuite-java-api/lib**フォルダにあります。

- **commons-logging-1.1.1.jar**以上
- **commons-codec-1.4.jar**以上
- **commons-httpclient-4.1.jar**以上
- **commons-ssl-0.3.8.jar**以上 - Management ConsoleがHTTPを介してアクセスする必要がある場合

また、Java 1.5以上を使用する必要があります。

リポジトリ・クライアント

リポジトリとの通信は、**com.kapowtech.robosuite.api.java.repository.engine**にある**RepositoryClient**を介して確立されます。

表14.34 RepositoryClientの作成

```
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();
}
}
```

ここでは、**http://localhost:50080/** でManagement Consoleのリポジトリにユーザー名とパスワードを使用して接続するように構成された**RepositoryClient**について説明します。

RepositoryClientが作成されたら、**getProjects()**メソッドを使用して、プロジェクトのリストをリポジトリに問い合せます。**RepositoryClient**メソッドのいずれかを呼び出す際に、エラーが発生した場合は**RepositoryClientException**がスローされることに注意してください。

RepositoryClientには、次の11のメソッドがあります。

表14.35 RepositoryClientのメソッド

メソッド・シグネチャ	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い場合にエラーは生成されません。
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。
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。
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。
void deployLibrary(String projectName, EmbeddedFileBasedRobotLibrary library, boolean failIfExists)	ライブラリをサーバーにデプロイします。ロボット、タイプおよびリソースは、**failIfExists**がtrueでないかぎり、オーバーライドされます。
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。指定した名前のリソースがすでに存在する場合は、**failIfExists**をfalseに設定するとオーバーライドできます。
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。指定した名前のロボットがすでに存在する場合は、**failIfExists**をfalseに設定するとオーバーライドできます。
void deploySnippet(String projectName, String snippetName, byte[] snippetBytes, boolean failIfExists)	ロボットをプロジェクトにデプロイします。指定した名前のスニペットがすでに存在する場合は、**failIfExists**をfalseに設定するとオーバーライドできます。
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。指定した名前のタイプがすでに存在する場合は、**failIfExists**をfalseに設定するとオーバーライドできます。
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。
Cluster[] getRoboServerClusters()	Management Consoleのクラスタ構成を返します。
Robot[] getRobotsInProject(String projectName)	指定した名前のプロジェクトで使用可能なロボットを返します。
RobotSignature getRobotSignature(String projectName, String robotName)	ロボット・シグネチャ、つまり、この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または保存するタイプのリストを返します。

RepositoryClientの作成時に、プロキシ・サーバーを明示的に指定する必要があります。標準の認証なしのhttpプロキシ・サーバーがサポートされています。認証付きのNTLMプロキシ・サーバーもサポートされています。

その他の詳細は、**RepositoryClient** JavaDocを参照してください。

リポジトリ・クライアント経由のデプロイメント

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およびタイプをデプロイする方法を示しています。

表14.36 RepositoryClientを使用したデプロイメント

```
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 (RepositoryClientException e) {
// an error connecting to the repository e.printStackTrace();
}
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());
}
```

リポジトリREST API

リポジトリAPIは実際には、RESTfulサービス(およびデータが送信される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・メソッドによって、リポジトリにXMLが送信され、リポジトリからはXMLで応答されます。すべてのデプロイ・メソッドによってリポジトリにバイト(URLでエンコードされた情報)が送信され、リポジトリからはXMLが確認に返されます。送受信されるXMLのフォーマットは、ここから入手できるDTDで管理されます。

ここで、すべてのXMLベースのリクエストの例を示しますが、すべてのメッセージが次の宣言で始まる必要があります。

表14.37

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE repository-request PUBLIC
"-//Kapow Technologies//DTD Repository 1.3//EN"
"http://www.kapowtech.com/robosuite/repository_1_3.dtd">
```

Management Consoleが**http://localhost:8080/ManagementConsole**でデプロイされる場合、リクエストは**http://localhost:8080/ManagementConsole/secure/RepositoryAPI?format=xml**に送信される必要があります。

表14.38 REST操作

メソッド	リクエストの例	レスポンスの例
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>**
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>7.2</version> <last-modified>2011-10-11 18:24:12.648</last-modified>**

メソッド	リクエストの例	レスポンスの例
		</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>7.2</version> <last-modified>2011-10-11 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-

メソッド	リクエストの例	レスポンスの例
		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-list> </returned-type></returned-type-list> <stored-type-list/> </robot-signature></repository-response>

メソッド	リクエストの例	レスポンスの例
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>**

デプロイメントは、rawバイト(**octet-stream**をPOSTボディとして送信)を次のURLに送信することによって行います。次に、リポジトリが**http://localhost:8080/ManagementConsole**でデプロイされる例を示します。

表14.39 デプロイ操作のメソッド

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**
deploy library	**http://localhost:8080/ManagementConsole/secure/RepositoryAPI?format=bytes&operation=deployLibrary&projectName=Default project&fileName=NA&failIfExists=true**

認証がManagement Consoleで有効化されている場合、URL **http://localhost:8080/ManagementConsole/secure/RepositoryAPI**は基本認証によって保護されます。この結果、**http://username:password@localhost:8080/ManagementConsole/secure/RepositoryAPI**の方法でURLに資格証明を組み込むことができます。

第15章 .NETプログラマーズ・ガイド

このガイドでは、Kapow Katalyst .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。ガイドでは、Design Studioチュートリアルを完了していて、単純なロボットの作成方法を理解していること、およびC#プログラミング言語の知識が十分あることを前提としています。

プログラマーズ・ガイドは、APIの大部分が非推奨になり、新しい実行APIが作成されたため、バージョン9.1用に完全に書き直されました。APIはまだ下位互換性がありますが、非推奨のクラスは将来のリリースで削除されるため、新しいAPIを理解し、新しいAPIを使用するように既存のアプリケーションの書換えを検討することをお薦めします。

古いRobotExecuterは、次の理由から非推奨になっています。

- APIによってRQLExceptionがスローされた後も、ロボットがRoboServer上で実行し続けていました。
- RoboServerがクライアントとの接続を失った場合でも、ロボットが実行し続け、その結果、実行されたすべての戻り値に対するログ・エラーが発生していました。
- リクエストの配布時に、配布ポリシーでサーバー容量を調べていませんでした。
- カスタムRQLHandlerの実装時に表面上はわからない危険性が多数あったため、オブジェクト・ストリーミングの実装が面倒でした。

古い.NETプログラマーズ・ガイドは、http://help.kapowtech.com/8.2/topic/doc/dotnet/Top.htmlでまだ参照できます。

特定のクラスに関する詳細は、Kapow Katalystインストール・フォルダ内の\API\robosuite-dotnet-api\docsにあるコンパイル済ヘルプrobosuite-dotnet-api.chmを参照してください。

基本

.NET APIの使用によって、すべての.NETベースのアプリケーション(.NET 4.0が必要)がRoboServerに対するクライアントになることができます。データベースにデータを格納するロボットの実行に加えて、ロボットがデータをクライアント・アプリケーションに直接戻すようにすることもできます。いくつか例を示します。

- 複数のロボットを使用してフェデレーテッド検索を実行し、複数のソースからの結果をリアルタイムで集計します。
- アプリケーション・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ないWebベース・システムにアカウントを作成します。

このガイドの基本の項では、中心的なクラス、およびロボットを実行するためにそれらを使用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、およびRoboServer上でそれらの実行を制御する方法についても説明します。

.NET APIは.dllファイルであり、Kapow Katalystインストール・フォルダ内の**/API/robosuite-dotnet-api/lib/robosuite-dotnet-api.dll**に格納されています。詳細は、「重要なフォルダ」を参照してください。このガイドのすべての例も**/API/robosuite-dotnet-api/examples**にあります。.NET APIと並んで配置されているのは、必須のサード・パーティ・ライブラリであるlog4net.dllです。

最初の例

NewsMagazine.robotというロボットの実行に必要なコードの説明から開始します(このロボットはデフォルト・プロジェクトのTutorialsフォルダにあります)。ロボットは、その結果を値を返すステップ・アクションを使用して出力し、これによって、APIを使用したプログラムによる出力の処理を容易にします。その他のロボット(通常はManagement Consoleによってスケジュールで実行される

ロボット)は、データベースに格納ステップ・アクションを使用して、それらのデータをデータベースに直接格納し、この場合、ロボットによって収集されたデータは、APIクライアントには戻されません。

次に、NewsMagazineロボットを実行し、その出力をプログラムによって処理する方法について説明します。

表15.1 入力なしのロボットの実行

```
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();
   }
 }
}
```

次に、関係するクラスとその役割について説明します。

表15.2

RoboServer	これは、ロボットを実行できるRoboServerを識別する単純な値オブジェクトです。各RoboServerは、Management Consoleによってアクティブ化され、使用前にKCUが割り当てられている必要があります。
Cluster	クラスタは、1つの論理ユニットとして機能するRoboServerのグループです。
Request	このクラスは、ロボット・リクエストの作成に使用されます。リクエストを実行する前に、クラスタをリクエスト・クラスに登録する必要があります。
DefaultRobotLibrary	ロボット・ライブラリは、リクエストで指定されたロボットを検索する場所をRoboServerに指示します。後の例で、様々なロボッ

	ト・ライブラリのタイプおよびそれらを使用する場合/方法について検討します。
RQLResult	これには、ロボット実行の結果が含まれます。結果には、値レスポンス、ログおよびサーバー・メッセージが含まれます。
RQLObject	値を返すアクションを使用してロボットから返される各値には、RQLObjectとしてアクセスできます。

ここからは、例の各行を見直して、具体的に説明していきます。

最初の行は、RoboServerがlocalhostのポート50000で実行されていることをAPIに通知します。

表15.3

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

次の3行は、1つのRoboServerを含むクラスタを定義しています。クラスタは**Request**クラスに登録され、このクラスタでリクエストを実行できるようになります。各クラスタはアプリケーションごとに1回のみ登録でき、これは通常、アプリケーションの初期化時に行われます。

表15.4 クラスタの登録

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

この後に、Library:/TutorialsにあるNewsMagazine.robotというロボットを実行するリクエストを作成するコードが続きます。Library:/は、リクエスト用に構成されたロボット・ライブラリを表します。ここでは、**DefaultRobotLibrary**が使用されており、これは、RoboServerに対してサーバーのローカル・ファイル・システムでロボットを検索するように指示します。ロボット・ライブラリの使用方法の詳細は、ロボット・ライブラリを参照してください。

表15.5

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

次の行は、**MyCluster**というクラスタ(以前に登録したクラスタ)でロボットを実行し、ロボットが終了すると結果を返します。ロボットの実行中にエラーが発生すると、ここで例外がスローされます。

表15.6

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

最後に、抽出した値を処理します。最初に、Postというタイプの抽出されたすべての値を取得し、それらを順に処理します。各RQLObjectに対して、Postタイプの属性にアクセスして、結果を印刷します。属性とマッピングについては、後述の項で説明します。

表15.7

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

```
Console.WriteLine(title + ", " + preview);
}
```

ロボット入力

APIを介して実行されるほとんどのロボットが、検索キーワードまたはログイン資格証明などの入力によってパラメータ化されます。ロボットへの入力はRoboServerに対するリクエストの一部であり、リクエストで**createInputVariable**メソッドを使用して提供されます。短いコード・フラグメントについて説明します。

表15.8 暗黙的な**RQLObjectBuilder**を使用した入力

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

ここでは、**Request**を作成し、**CreateInputVariable**を使用してuserLoginという入力変数を作成します。次に、**setAttribute**を使用して、入力変数のユーザー名およびパスワードの属性を構成します。

前述の例は、一般的な簡易表記ですが、**RqlObjectBuilder**を使用して処理を冗長に表すこともできます。

表15.9 明示的な**RQLObjectBuilder**を使用した入力

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

2つの例は同じです。最初は、匿名**RqlObjectBuilder**でカスケード・メソッド呼出しを利用しているため、短くなっています。

RoboServerがこのリクエストを受信すると、次のことが発生します。

- RoboServerによって、(リクエスト用の**RobotLibrary**が構成されているすべての場所から)Input.robotがロードされます。
- RoboServerによって、ロボットにuserLoginという変数があり、この変数に入力のマークが付けられていることが検証されます。
- RoboServerによって今度は、**setAttribute**を使用して構成した属性が、変数userLoginのタイプと互換性があることが検証されます。これは、タイプにusernameおよびpasswordという属性があり、これらが両方ともテキストベースの属性であることを意味します(次の項でAPIとDesign Studio属性の間のマッピングについて説明します)。
- すべての入力変数に互換性がある場合は、RoboServerによってロボットの実行が開始されます。

ロボットに複数の入力変数が必要な場合は、ロボットを実行するためにそれらをすべて作成する必要があります。構成する必要があるのは必須の属性のみであり、APIを使用して構成しない非必須の属性はnull値になります。FacebookとTwitterの両方にログインする必要があるロボットがあるとすると、入力を次のように定義できます。

表15.10

```
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");
```

属性タイプ

Design Studioで新しいタイプを定義する場合は、各属性に対する属性タイプを選択します。これらの属性の一部には、短いテキスト、長いテキスト、パスワード、HTML、XMLなどのテキストを含めることができ、ロボット内部で使用する場合は、これらの属性に格納するテキストに対する要件がある場合があります。XML属性にテキストを格納する場合、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、タイプがロボット内で使用されるときに行われますが、APIではタイプに関して何もわからないため、同じ方法での属性値の検証は行われません。このため、APIの属性タイプが8のみであるのに対して、Design Studioでは19使用可能です。次の表に、APIとDesign Studio属性タイプの間のマッピングを示します。

表15.11 APIとDesign Studioのマッピング

API属性タイプ	Design Studio属性タイプ
テキスト	**Short Text, Long Text, Password, HTML, XML, Properties, Language, Country, Currency, Refind Key**
整数	**Integer**
ブール	**Boolean**
数値	**Number**
文字	**Character**
日付	**Date**
セッション	**Session**
バイナリ	**Binary, Image, PDF**

その後、API属性タイプは次のように.NETにマップされます。

表15.12 属性に対する.NETタイプ

属性タイプ	.NETクラス
テキスト	**System.String (string)**
整数	**System.Int64**
ブール	**System.Boolean (bool)**
数値	**System.Double (double)**
文字	**System.Char (char)**
日付	**System.DateTime**
セッション	**Com.Kapowtech.Robosuite.Api.Construct.Session**
バイナリ	**Com.Kapowtech.Robosuite.Api.Construct.Binary**

RqlObjectBuilderの**setAttribute**メソッドはオーバーロードされているため、引数として正しい.NETクラスを使用しているかぎり、APIを使用した属性の構成時に属性タイプを明示的に指定する必要はありません。次の例は、考えられるすべての(Design Studio)属性タイプを使用してオブジェクトに対する属性を設定する方法を示しています。

表15.13 推奨のsetAttribute使用方法

```
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>bla</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 this as input");
```

前述の例で、null値をキャストする必要があることに注意が必要であり、これは、そうしないとC#コンパイラでは、**SetAttributeEntry**メソッドのオーバーロードされているバージョンのいずれを呼び出す必要があるかを判断できないためです。ただし、未構成の属性は自動的にnullになるため、nullを明示的に設定する必要はありません。

APIを使用した入力の作成時に、AttributeおよびAttributeTypeを明示的に指定することは可能です。この方法は、お薦めしませんが、まれに必要になることがあり、次のようになります。

表15.14 非推奨のsetAttribute使用方法

```
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>bla</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));
```

このように、すべての属性値を文字列の形式で提供する必要があります。文字列値はその後、提供したAttributeEntryTypeに基づいて適切な.NETオブジェクトに変換されます。これが役立つのは、Kapow Katalyst .NET APIに加えて他の汎用APIを構築する場合のみです。

実行パラメータ

CreateInputVariableメソッド以外に、**Request**には、RoboServerでのロボットの実行方法を制御するプロパティが多数あります。

表15.15 Requestの実行制御プロパティ

MaxExecutionTime	このプロパティでは、ロボットが実行可能な最大秒数を制御します。この時間を経過すると、ロボットはRoboServerによって停止されます。タイマーは、ロボットが実行し始めるまで開始しないため、ロボットがRoboServerのキューに入れられた場合、これは考慮されません。
StopOnConnectionLost	このプロパティ(デフォルトはtrue)では、RoboServerでクライアント・アプリケーションとの接続が失われたことが検出された場合に、ロボットを停止するかどうかを制御します。この値をfalseに設定するには、十分な理由が必要であり、これを処理するようにコードが作成されていない場合、アプリケーションは期待どおりに実行されません。
StopRobotOnApiException	このプロパティ(デフォルトはtrue)では、最初のAPI例外の発生時に、RoboServerにロボットを停止するよう指示します。デフォルトでは、ステップで実行に失敗すると、ロボットのほとんどのステップでAPI例外が発生します。これはステップ・エラー処理タブで構成します。 falseに設定すると、ロボットはAPI例外に関係なく実行し続けますが、アプリケーションで結果のストリーミングに**IRobotResponseHandler**を使用していないかぎり、例外はやはりExecute()によってスローされるため、これをfalseに設定する場合は十分注意してください。
Username, Password	これらのプロパティは、資格証明の設定に使用します。これは、RoboServerが認証を必要とするように構成される場合に使用します。このオプションを有効化すると、クライアントは資格証明を提供する必要があり、提供しないとRoboServerによってリクエストを拒否されます。
RobotLibrary	このプロパティは、RobotLibraryをリクエストに割り当てるために使用します。ロボット・ライブラリは、リクエストで指定されたロボットを検索する場所をRoboServerに指示します。後の例で、様々なロボット・ライブラリのタイプおよびそれらを使用する場合/方法について検討します。

ExecutionId	このリクエストの実行IDを設定できます。提供しない場合は、RoboServerによって自動的に生成されます。実行IDは、ロギングのために使用され、クライアントがロボットをプログラムによって停止できる必要がある場合にも必要です。IDは、(経時的に)グローバルに一意である必要があります。2つのロボットが同じ実行IDを使用した場合は、ログに一貫性がなくなります。 ロボットがより規模の大きいワークフローの一部であり、クライアント・アプリケーションにすでに一意の識別子がある場合は、これを設定すると、ロボットのログをシステムの残りの部分に容易に結合できるため便利です。
setProject(String)	これは、ロギングの目的にのみ使用します。Management Consoleは、このフィールドを使用してログ・メッセージをプロジェクトにリンクするため、ログ・ビューをプロジェクトでフィルタできます。 アプリケーションで**RepositoryRobotLibrary**を使用していない場合はおそらく、この値を設定して、このロボットが属しているプロジェクト(ある場合)をRoboServerロギング・システムに通知する必要があります。

ロボット・ライブラリ

Design Studioで、ロボットはプロジェクトにグループ化されます。ファイル・システムを調べると、これらのプロジェクトは、**Library**というフォルダを含むことが唯一の制約であるフォルダで表されることがわかります。詳細は、「ライブラリおよびプロジェクト」を参照してください。

RoboServerに対する実行リクエストを作成するとき、次のようにロボットのURLでロボットを識別します。

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

ここで、**Library:/**は、ロボット・ライブラリのシンボリック参照であり、この中でRoboServerはロボットを検索する必要があります。**RobotLibrary**は、ビルダーで次のように指定されます。

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

3つの異なるロボット・ライブラリ実装があり、いずれを選択するかはデプロイメント環境に依存します。

表15.16 ロボット・ライブラリ

ライブラリ・タイプ	説明
DefaultRobotLibrary	これは、現在のプロジェクト・フォルダでロボットを検索するようにRoboServerを構成します。このフォルダはSettingsアプリケーションで定義します。 複数のRoboServerがある場合は、すべてのRoboServerにロボットをデプロイする必要があります。 このロボット・ライブラリがキャッシュされていない場合は、ロボットはすべての実行でディスクからリロードされます。これは、ロボットが頻繁に変更される開発環境でライブラリを使用できるようにしますが、本番環境には適していません。
EmbeddedFileBasedRobotLibrary	このライブラリは、RoboServerに送信される実行リクエストに埋め込まれます。このライブラリを作成するには、ロボットとそのすべての依存項目(タイプ、スニペットおよびリソース)を

ライブラリ・タイプ	説明
	含むzipファイルを作成する必要があります。これを実行するには、Design Studioの「ツール」->ロボット・ライブラリ・ファイルの作成メニューを使用します。 ライブラリはすべてのリクエストとともに送信され、大規模ライブラリの場合はオーバーヘッドがある程度大きくなりますが、ライブラリはRoboServerにキャッシュされ、最大限可能なパフォーマンスが提供されます。 1つの利点は、ロボットとコードを1つのユニットとしてデプロイできるため、QA環境から本番環境への完全な移行が提供されることです。ただし、ロボットが頻繁に変更される場合は、頻繁に再デプロイする必要があります。 次のコードを使用すると、リクエストに対して埋込みのロボット・ライブラリを構成できます。 **var request =** **new Request** **("Library:/Tutorials/NewsMagazine.robot");** **var stream =** **new FileStream("c:\\embeddedLibrary.robotlib",** **FileMode.Open);** **request.RobotLibrary =** **new EmbeddedFileBasedRobotLibrary(stream);**
RepositoryRobotLibrary	これは最も柔軟な**RobotLibrary**です。 このライブラリは、Management Consoleの組込みリポジトリをロボット・ライブラリとして使用します。このライブラリを使用する場合、RoboServerがManagement Consoleと通信すると、Management Consoleによってロボットとその依存項目を含むロボット・ライブラリが送信されます。 キャッシュは、Management Console内とRoboServer内の両方で、ロボットごとに発生します。Management Console内では、生成されたライブラリは、ロボットとその依存項目に基づいてキャッシュされます。RoboServerでは、キャッシュはタイムアウトに基づいているため、リクエストごとにManagement Consoleに問い合せる必要はありません。さらに、RoboServerとManagement Consoleの間のライブラリ・ロードでは、バンド幅をさらに削減するために、HTTP public/privateキャッシュが使用されます。 NewsMagazine.robotがManagement Consoleにアップロードされると、ロボットの実行時に、次のようにリポジトリ・ロボット・ライブラリを使用できます。 **var request =** **new Request** **("Library:/Tutorials/NewsMagazine.robot");** **request.RobotLibrary =** **new RepositoryRobotLibrary** **("http://localhost:50080",** **"Default Project", 60000);**

ライブラリ・タイプ	説明
	これは、ローカルのManagement Consoleからロボットをロードし、それを1分間キャッシュした後、新しいバージョンのロボット(そのタイプとスニペット)が変更されたかどうかManagement Consoleを確認するようにRoboServerに指示します。 さらに、**Library:/**プロトコルを介してロードされるリソースでは、RoboServerはManagement Consoleから直接リソースをリクエストします。

拡張

この項では、APIの拡張機能の一部についてもう少し詳しく説明します。これらには、出力ストリーミング、ロギングおよびSSL構成、さらに並列実行があります。

負荷分散

表15.17 ロード・バランス・エグゼキュータ

```
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);
```

RequestExecutor内部で発生している内容についてもう少し詳しく説明します。エグゼキュータには、RoboServerの配列が与えられます。エグゼキュータが構成されると、各RoboServerに接続しようとします。接続すると、各RoboServerにサーバーの構成方法を検出するためにpingリクエストを送信します。

負荷は、RoboServer上の未使用実行スロットの数に基づいて、クラスタ内の各オンラインRoboServerに分散されます。次のリクエストは常に、最も使用可能なスロットがあるRoboServerに配布されます。使用可能な実行スロットの数は、初期pingレスポンスから取得し、エグゼキュータは、それが開始した各ロボットおよびその完了時を追跡します。RoboServer上の実行スロットの数は、「ーバー」タブの最大同時ロボット数によって決まります。

RoboServerがオフラインになると、pingリクエストに正常に応答するまでは、ロボットの実行リクエストは受け入れられません。

2つのクライアント・ルール

RoboServerの特定のクラスタを使用するAPIクライアントは1つのみにする必要があります。同じRoboServerに対してロボットを実行している.Netアプリケーションが複数ある場合は、結果的にパフォーマンスが低下します。

エグゼキュータ・ロガー

リクエストを実行したとき、ロボットでエラーが生成された場合は、実行メソッドで例外がスローされます。他のタイプのエラーおよび警告は、**IExecutorLogger**インタフェースを実装するクラスであるエグゼキュータ・ロガー介してレポートされます。前述の例では、ロボットの実行時にエグゼキュータ・ロガーは提供しておらず、これは、システム出力に書き込むデフォルト実装**ExecutorLogger**を使用することを意味します。次に、RoboServerの1つがオフラインになった場合の**ExecutorLogger**のレポート方法について説明します。

例では、オンラインではないRoboServerを含むクラスタを構成します。

表15.18 エグゼキュータ・ロガー、オフライン・サーバーの例

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

この例を実行すると、コンソールに次が出力されます。

表15.19 エグゼキュータ・ロガー、オフラインRoboServerのコンソール出力

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

アプリケーションで**System.out**に直接書き込まないようにすることが多く、その場合は異なる**IExecutorLogger**実装を提供でき、これは、次のようにクラスタの登録時に実行できます。

表15.20 **DebugExecutorLogger**の使用

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

この例では、**System.out**にも出力する**DebugExecutorLogger**()を使用しますが、出力されるのはAPIデバッグが有効化されている場合のみです。または、エラー・メッセージの処理方法を制御するために、**ExecutorLogger**の独自の実装を提供できます。

データ・ストリーミング

ロボット実行の結果を、ロボットの実行中にリアルタイムで提示する必要がある場合があります。このような場合、ロボットがその実行を終了し、RqlResultにアクセスするのを待機するかわりに、抽出した値をAPIで即時に返す必要があります。

APIでは、ロボットが返した値をAPIが受信するたびに、コールバックを受信する機能が提供されています。これは、**IRobotResponseHandler**インタフェースを介して実行します。

表15.21 **AbstractFailFastRobotResponseHandler**を使用したレスポンス・ストリーミング

```
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);
  }
 }
}
```

前述の例では、**Request**の2番目の実行メソッドが使用されており、これは、ロボットを実行するクラスタ名の他に**RobotResponseHandler**を必要とします。この例では、デフォルトのエラー処理を提供する**AbstractFailFastRobotResponseHandler**を拡張して**IRobotResponseHandler**を作成しているため、処理する必要があるのはロボットが返す値のみです。

handleReturnedValueメソッドは、APIがRoboServerから戻り値を受信するたびに呼び出されます。この例で使用した**AbstractFailFastRobotResponseHandler**では、非ストリーミングの実行メソッドと同じ方法で例外がスローされます。これは、ロボットで生成されたAPI例外に応じて、例外がスローされることを意味します。

IRobotResponseHandlerにはいくつかのメソッドがあり、これは3つのカテゴリにグループ化できます。

ロボット・ライフサイクル・イベント	ロボットの起動時やその実行の終了時など、RoboServer上のロボットの実行状態が変わったときに呼び出されるメソッド。
ロボット・データ・イベント	ロボットがデータまたはエラーをAPIに返したときに呼び出されるメソッド。
追加のエラー処理	RoboServer内部またはAPIのエラーのいずれかによって呼び出されるメソッド。

表15.22 IRobotResponseHandler - ロボット・ライフサイクル・イベント

メソッド名	説明
void RequestSent(RoboServer roboServer, ExecuteRequest request)	リクエストを実行するサーバーを**RequestExecutor**が検出したときに呼び出されます。
void RequestAccepted(String executionId)	検出したRoboServerがリクエストを受け入れ、それをキューに入れたときに呼び出されます。
void RobotStarted(IStoppable stoppable)	RoboServerがロボットの実行を開始したときに呼び出されます。これは通常、RoboServerに過度の負荷がかかっている場合または複数のAPIクライアントによって使用されている場合を除いて、ロボットがキューに入れられた直後に発生します。
void RobotDone(RobotDoneEvent reason)	ロボットがRoboServer上での実行を終了したときに呼び出されます。**RobotDoneEvent**は、実行が正常に終了したか、エラーで終了したか、または停止されたかを指定するために使用されます。

表15.23 IRobotResponseHandler - ロボット・データ・イベント

メソッド名	説明
void HandleReturnedValue (RobotOutputObjectResponse	ロボットが値を返すアクションを実行し、値がソケット経由でAPIに返されたときに呼び出されます。

メソッド名	説明
response, IStoppable stoppable)	
void HandleRobotError (RobotErrorResponse response, IStoppable stoppable)	ロボットでAPI例外が発生し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例外後に実行を停止します。この動作は、**Request.StopRobotOnApiException = false**を使用して上書きでき、この場合、このメソッドは複数回呼び出されます。これは、生成されたエラーに関係なく、データ・ストリーミング・ロボットで実行を継続する場合に有効です。
void HandleWriteLog (RobotMessageResponse response, IStoppable stoppable)	ロボットがログの作成アクションを実行した場合に呼び出されます。これは、ロボット内から追加のロギング情報を提供する場合に有効です。

表15.24 IRobotResponseHandler - 追加のエラー処理

メソッド名	説明
void HandleServerError (ServerErrorResponse response, IStoppable stoppable)	RoboServerがエラーを生成した場合、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RoboServer内部でエラーが発生してロボットを起動できない場合に呼び出されます。
void handleError (RQLException e, IStoppable stoppable)	API内部でエラーが発生した場合に呼び出されます。クライアントがRoboServerとの接続を失った場合が最も一般的です。

メソッドの多くには、**IStoppable**オブジェクトが含まれており、このオブジェクトは、たとえば、特定のエラーまたは返される値に応じて停止するために使用できます。

これらのメソッドの一部ではRQLExceptionをスローでき、これを行う場合は、結果に注意する必要があります。ハンドラを呼び出すスレッドは、**Request.Execute()**を呼び出すスレッドであり、これは、スローされたいずれかの例外によって呼出しスタックが処理されず、実行メソッドが範囲外となることを意味します。**handleReturnedValue**、**handleRobotError**または**handleWriteLog**に応じて例外をスローした場合、**Stoppable.stop()**の起動はユーザーが行う必要があり、これを起動しないと、**Request.Execute()**の呼出しが完了した場合でもロボットが実行を継続する場合があります。

データ・ストリーミングは、次のユースケースの1つで最もよく使用されます。

- 結果がユーザーにリアルタイムで提示される、Ajaxベースのアプリケーション。データがストリーミングされなかった場合、結果はロボットの実行が終了するまで表示できません。
- ロボットの実行全体でクライアントがメモリーにデータを格納できないほどの大量のデータを返すロボット。
- 抽出した値がロボット実行で同時に処理されるように最適化する必要があるプロセス。
- データベースにデータをカスタム・フォーマットで格納するプロセス。
- API例外のカスタム処理を無視するまたは必要とするロボット(次を参照)。

表15.25 **AbstractFailFastRobotResponseHandler**を使用したレスポンスおよびエラー収集

```
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;
  }
 }
}
```

前述の例は、返された値およびエラーを収集する**IRobotResponseHandler**の使用方法を示し
ています。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必要

がある場合に有効であり、これは、Webサイトが不安定で、タイムアウトになることがある場合に役立ちます。ハンドラによって収集されるのはロボット・エラー(API例外)のみであることに注意が必要であり、RoboServerに対する接続が失われた場合は、やはり**Request.Execute()**によって**RQLException**がスローされます(また、ロボットがRoboServerによって停止されます)。

詳細は、**/docs**フォルダ内の**IRobotResponseHandler** chmドキュメントを参照してください。

SSL

APIは、RQLServiceを介してRoboServerと通信します。RQLServiceはRoboServerのコンポーネントの1つであり、特定のネットワーク・ポートでAPIリクエストをリスニングします。RoboServerを起動する際に、RoboServerで暗号化SSLサービスを使用するか、プレーン・ソケット・サービスを使用するか、あるいは両方(2つの異なるポートを使用)を使用するかを指定します。クラスタ内のすべてのRoboServerで同じRQLService (ただしポートは異なる場合があります)を実行している必要があります。

RoboServerを、次のようにポート50043でSSL RQLServiceを使用して起動したとします。

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

次のコードを使用できます。

表15.26 SSL構成

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

実行する必要があるのは、SSLクラスタとしてクラスタを作成し、各RoboServerで使用されるSSLポートを指定することのみです。これで、RoboServerとAPI間のすべての通信が暗号化されます。

データ暗号化以外に、SSLには、リモート・パーティのアイデンティティを検証する機能があります。不正なWebサイトはそれ自身ではない何かになりすますことがあるため、このタイプの検証はインターネットに関して非常に重要です。多くの場合、APIクライアントとRoboServerは同じローカル・ネットワーク上に存在するため、他のパーティのアイデンティティの検証が必要なることはあまりありませんが、必要になった場合に備えてAPIではこの機能がサポートされています。

組み込まれているSSLの例をコンパイルおよび実行する方法は、ここを参照してください。

リポジトリ統合

Management ConsoleでRoboServerのクラスタも指定し、これらは、スケジュール済ロボット、およびRESTサービスとして実行されるロボットを実行するために使用します。APIでは、RepositoryClientを使用してManagement Consoleからクラスタ情報を取得できました。詳細はRepositoryClientのドキュメントを参照してください。

表15.27 リポジトリ統合

```
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());
         }     } }
```

前述の例は、localhostのポート50080にデプロイされたManagement Consoleに接続する**RepositoryClient**の作成方法を示しています。

Management Consoleに認証が必要な場合はユーザー名とパスワードを渡す必要があり、それ以外の場合は、両方にnullを渡すことができます。**RepositoryClient**を登録するとき、Management Consoleに存在するクラスタの名前を指定すると、これはManagement Consoleに問い合せてこのクラスタ用に構成されているRoboServerのリストを取得し、Management Console上でクラスタ構成が更新されていないかどうかを2分間隔で確認します。

この統合によって、Management Consoleでクラスタを作成でき、これはManagement Consoleユーザー・インタフェースを使用して動的に変更できます。Management Consoleを使用する場合は、API使用を伴うクラスタは、2つのクライアント・ルールに違反しているため除外して、ロボットのスケジュールには使用しないでください。

内部処理

この項では、クラスタを登録し、**Request**を実行したときに、内部で行われている処理について説明します。

Clusterを**Request**に登録する際に、背後で**RequestExecutor**が作成されます。この**RequestExecutor**は、クラスタ名をキーとして使用して、**Map**に格納されます。リクエストを実行すると、提供したクラスタ名を使用して関連する**RequestExecutor**が検索され、リクエストが実行されます。

短い例について説明します。

表15.28 通常の実行

```
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);
}
```

今度は、同じ例を非表示の**RequestExecutor**を使用して直接作成します。

表15.29 内部処理の実行

```
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);
}
```

RequestExecutorがデフォルトで非表示の理由は、追跡する必要がないためです。クラスタごとに**RequestExecutor**を1つのみ作成できるため、それを直接使用する場合は、アプリケーション全体でそれに対する参照を保管する必要があります。 **Request.RegisterCluster(cluster)**の使用は、**RequestExecutor**およびライフサイクルのルールを最良の方法で無視できることを意味します。

RequestExecutorには、必要な状態、およびロード・バランシングとフェイルオーバー機能を提供するロジックが含まれています。**RequestExecutor**の直接使用では、いくつかの特別な機能も提供されており、これについてこの後説明します。

RequestExecutorの機能

RequestExecutorがリポジトリに接続されていない場合は、**AddRoboServer(..)**および**RemoveRoboServer(..)**を呼び出して、RoboServerを動的に追加および削除できます。これらのメソッドでは、**RequestExecutor**内で使用される配布リストが変更されます。

RequestExecutor.TotalAvailableSlotsプロパティには、内部配布リストのすべてのRoboServerの未使用実行スロット数が含まれます。

これらのメソッドを使用すると、使用可能な実行スロットの数が少なくなったときにすぐに、RoboServerを**RequestExecutor**に動的に追加できます。

RequestExecutorの作成時に、オプションで**IRqlEngineFactory**を提供できます。**IRqlEngineFactory**を使用すると、RoboServerの接続時にいずれの**RQLProtocol**を使用するかをカスタマイズできます。これが必要になる状況はほとんどなく、たとえば、セキュリティの向上のためにクライアント証明書を使用する場合などです。詳細は、「APIクライアント証明書」を参照してください。

リポジトリAPI

リポジトリAPIを使用すると、Management Consoleのリポジトリを問い合せて、プロジェクト、ロボットおよびロボットの呼出しに必要な入力のリストを取得できます。また、ロボット、タイプおよびリソース・ファイルをプログラムによってデプロイすることもできます。

リポジトリ・クライアント

リポジトリとの通信は、ネームスペース**Com.KapowTech.RoboSuite.Api.Repository.Engine**にある**RepositoryClient**を介して確立されます。

例について説明します。

表15.30 リポジトリからのプロジェクトの取得

```

```

```
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);
}
```

ここでは、**http://localhost:50080/**でManagement Consoleのリポジトリにユーザー名とパスワードを使用して接続するように構成された**RepositoryClient**について説明します。Management Consoleがパスワードで保護されていない場合は、ユーザー名とパスワードに**null**を指定する必要があります。

RepositoryClientが作成されたら、**GetProjects()**メソッドを使用して、プロジェクトのリストをリポジトリに問い合せます。**RepositoryClient**メソッドのいずれかを呼び出す際に、エラーが発生した場合は**RepositoryClientException**がスローされることに注意してください。

RepositoryClientには、次の11のメソッドがあります。

表15.31 **RepositoryClient**のメソッド

メソッド・シグネチャ	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。
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。
void DeleteType(string projectName, string typeName, boolean silent)	プロジェクトからタイプを削除します。
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。
void DeployLibrary(string projectName, EmbeddedFileBasedRobotLibrary library, boolean failIfExists)	ライブラリをサーバーにデプロイします。ロボット、タイプおよびリソースは、**failIfExists**がtrueでないかぎり、オーバーライドされます。
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。指定した名前のリソースがすでに存在する場合は、**failIfExists**をfalseに設定するとオーバーライドできます。
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。指定した名前のロボットがすでに存在する場合は、**failIfExists**をfalseに設定するとオーバーライドできます。
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。指定した名前のタイプがすでに存在する場合は、**failIfExists**をfalseに設定するとオーバーライドできます。
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。指定した名前のスニペットがすでに存在する場合は、**failIfExists**をfalseに設定するとオーバーライドできます。
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・シグネチャ	説明
Cluster[] GetRoboServerClusters()	Management Consoleのクラスタ構成を返します。
Robot[] GetRobotsInProject(string projectName)	指定した名前のプロジェクトで使用可能なロボットを返します。
RobotSignature GetRobotSignature(string projectName, string robotName)	ロボット・シグネチャ、つまり、この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タイプのリストを返します。

詳細は、.Netドキュメントを参照してください。.Netドキュメントは、Kapow Katalystインストール内の**/API/robosuite-dotnet-api/docs/RoboSuite .NET API.chm**にあります。

認証がリポジトリで有効化されている場合、指定した資格証明に十分なアクセス権がない場合、リクエストは拒否される場合があります。

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を使用している場合は、Internet Explorer用に構成されたプロキシ・サーバーがリポジトリAPIによって使用されます。

リポジトリ・クライアント経由のデプロイメント

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およびタイプをデプロイする方法を示しています。

表15.32 リポジトリへのデプロイ

```
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);
```

RESTとしてのリポジトリAPI

リポジトリには、RESTfulサービス経由でアクセスすることもできます。

例

Kapow Katalystインストールには追加のAPIコード例が6つ含まれており、これらの例は**API \robosuite-dotnet-api\example**にあります。

例のコンパイルおよび実行

例をコンパイルするには、コマンド・プロンプトからbuild.batを実行します。この結果、直接実行できる6つの.exeファイルが生成されます。

.exeファイルは**robosuite-dotnet-api.dll**と**log4net.dll**に依存し、その両方がexamplesディレクトリにあります。ファイルは両方とも、binフォルダにあるファイルの同一コピーであり、例を簡単に実行できるようにここにコピーされます。

各例のプログラムでは、引数なしで実行すると、小規模な使用方法テキストが出力されます。

C#コンパイラの問題

build.batファイルは、パスでC#コンパイラが使用可能であることを前提としています。

.NET Framework 4.0

APIと付属のlog4netは、.Net Framework 4.0 Client Profileをターゲットとして作成されています。.Net Framework 4.0 Client Profileの詳細は、http://msdn.microsoft.com/en-us/library/cc656912.aspxを参照してください。

SSLの例

SSLの例RunSslRobot.exeを実行するには、RoboServerがSSLを使用するように構成されていて、証明書がクライアント・マシンにインポートされている必要があります。このガイドでは、ローカルのRoboServerを実行しているWindows PCで、自己署名証明書を使用してSSLを構成する方法を示します。

自己署名証明書の作成

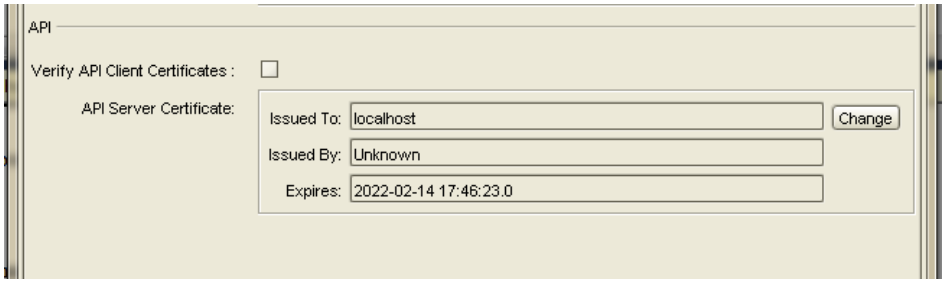
examplesディレクトリに含まれているbatファイル**MakeCertificates.bat**によって、.net APIクライアント用の自己署名証明書が作成されます。**MakeCertificates.bat**を次のように使用できます。

1. Visual Studioコマンド・プロンプトを起動し、examplesディレクトリに移動します。
2. **MakeCertificates.bat**と入力します。
3. ここで、証明書のパスワードの入力を求められます(この場合は単に「123」と入力)。
4. 今度は、証明書が存在するホスト名の入力を求められます。ローカルのRoboServerの場合は、「localhost」と入力します。これは、プロトコルの作成時に、クライアント・コードで使用した同じホスト名であることが重要です。
5. パスワードに関して再度3回プロンプトが表示されます。各ダイアログに「123」と入力します。

証明書は、「APIクライアント証明書」の説明に従ってKeytoolを使用して作成することもできます。

RoboServerの構成

1. RoboServerで、「開始」->すべてのプログラム->「Kapow Katalyst」にある「Settings」アプリケーションを起動します。
2. 「Settings」アプリケーションで、証明書タブに移動します。
3. 「変更」をクリックし、ファイル**API\robosuite-dotnet-api\example\server.pfx**を選択します。
4. パスワードの入力を求められた場合は、「123」と入力します。



5. これで、RoboServerは完全に構成され、ポート50443でSSL構成を使用するように、スイッチ**-service ssl:50443**を使用して起動できます。

APIクライアントの構成

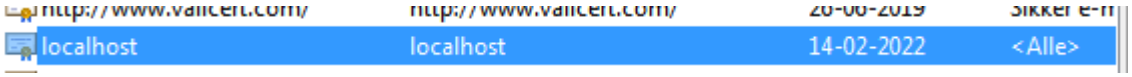
1. コマンド**mmc.exe**を実行します。

2. コンソールメニューで、スナップインの追加/削除をクリックします。

3. スナップインで、証明書をダブルクリックし、ローカル・コンピュータに対して証明書を管理することを選択し、「終了」をクリックします。

4. 証明書スナップインがロードされたら、ノードの証明書->信頼できるルート認証局を展開し、証明書ノードを右クリックして、メニュー項目のすべてのタスク->「インポート」をクリックします。

5. この結果、証明書インポート・ウィザードが起動します。証明書ファイルの選択を求められた場合は、**API\robosuite-dotnet-api\example\server.pub.cer**を参照して、インポートを完了します。



これらのステップの完了後は、サーバーとクライアントの両方がSSLを使用するように構成され、例のRunSslRobot.exeの実行を使用して構成を検証できます。

第16章 ランタイム

Kapow Katalystには、開発したロボットを実行するためのツールが多数用意されています。以降の各項でこれらのツールについて説明します。

- RoboServerは、リモート・クライアントでロボットを実行できるようにするサーバー・アプリケーションです。これは、Management ConsoleおよびRoboServer Settingsアプリケーション(セキュリティや認証などの拡張構成の場合)の両方を使用して構成します。
- Management Consoleを使用すると、ロボットの実行のスケジュール、ログおよび抽出データの表示ができます。また、システムの状態を監視するためのダッシュボード、およびRoboServerのクラスタに関する設定を構成できる集中管理の場所も用意されています。
- Control Centerを使用すると、RoboServerとそのロボットをリモートで監視できます。

第17章 RoboServerユーザーズ・ガイド

RoboServerによって、Design Studioで作成したロボットが実行されます。ロボットは様々な方法で起動でき、Management Consoleによって特定の時間に実行するようにスケジュールする、REST Webサービスを介して呼び出す、Javaまたは.NET APIを使用する、Kappletから起動する方法があります。

RoboServerでロボットを実行できるようにするには、Management Consoleでアクティブ化する必要があります。RoboServerは、有効なライセンスがあるManagement Consoleのクラスタに属していて、そのクラスタに十分なKCUが割り当てられている場合にアクティブです。RoboServerでは、それらがクラスタ上に構成されているManagement Consoleから設定も受信します。RoboServerとこれらのクラスタの管理に関する詳細は、11章「Management Consoleユーザーズ・ガイド」を参照してください。

RoboServerの起動

RoboServerはいくつかの異なる方法で起動できます。

- RoboServerプログラム・アイコン(またはManagement ConsoleとRoboServerの両方を起動する、Management Consoleの起動プログラム・アイコン)をクリックする方法。
- コマンドラインから起動する方法(詳細は後述)。
- サービスとして実行する方法。RoboServerをサービスとして実行する方法の詳細は、「サーバーの自動起動」を参照してください。

コマンドラインからRoboServerを起動するには、コマンド・ウィンドウを開いて次を入力します。

```
RoboServer
```

次に[Enter]を押します。RoboServerを「インストール・ガイド」の説明に従って正しくインストールした場合は、コマンド・ウィンドウに次が出力され、その後でRoboServerが終了します。

表17.1 RoboServerヘルプ・テキスト

```
Detected the following plugins: ...
Usage:
RoboServer [-maxClippingSessions <num>] [-verbose] [-version] [-MC]
[-port] [-sslPort] [-help] -service
<service:params>
Available services: ...
```

ドットは、インストールに依存する情報を示します。前述と異なる内容がコマンド・ウィンドウに出力された場合は、「インストール・ガイド」の説明に従ってRoboServerをインストールしたことを確認してください。

例

次のコマンドでは、RoboServerを起動し、ポート番号50000でリスニングするソケットベースのRQLサービスに対するソケット接続を受け入れます。

```
RoboServer -p 50000
```

RoboServerパラメータ

RoboServerの起動方法に関係なく、次のパラメータが受け入れられます。

表17.2 RoboServerパラメータ

パラメータ	説明
-c <num> | -maxClippingSessions <num>	このパラメータは、このRoboServerに存在できるクリップ・セッションの最大数を指定します。このパラメータはオプションです。デフォルト値は50です。最大値は0です。 例: **-maxClippingSessions 20**
-v | -verbose	このオプションのパラメータを指定すると、RoboServerはステータスおよびランタイム・イベントを出力します。
-V | -version	このオプションのパラメータを指定すると、RoboServerはバージョン番号を出力した後、終了します。
-MC	このオプションのパラメータを指定すると、Management ConsoleがRoboServerの一部として起動します。Management Consoleは、Settingsアプリケーションを使用して構成された埋込みWebサーバー上で実行されます。
-s <service-name: service-parameter> | -service <service-name: service-parameter>	このパラメータでは、RoboServerで起動するRQLまたはJMXサービスを指定します。このパラメータは少なくとも1回指定する必要があり、同じRoboServerで複数のサービスを起動するために複数回指定できます。使用可能なサービスは、インストールに依存します。 例: **-service socket:50000** 例: **-service jmx:50100**
-p <port-number> | -port <port-number>	これは、**-s socket:<port-number>**を呼び出すための短縮形です。 例: **-port 50000**
-P <port-number> -sslPort <port-number>	これは、**-s ssl:<port-number>**を呼び出すための短縮形です。 例: **-sslPort 50001**

パラメータの必須の順序はありません。JMXサービスでは、指定したポートのRoboServerの管理情報が提供されます。

RoboServerのシャットダウン

RoboServerは、コマンドライン・ツールのShutDownRoboServerを使用してシャットダウンできます。ShutDownRoboServerを引数を指定せずに実行すると、サーバーのシャットダウン方法に関する様々なオプション、特にサーバーで現在実行中のロボットの処理方法に関するオプションが表示されます。

本番構成

安定して稼働する本番環境にするには、デフォルトのRoboServerパラメータの一部を微調整する必要がある場合があります。次の構成オプションについて説明します。

- RoboServerインスタンスの数
- メモリー割当
- 同時ロボット数
- 自動メモリー・オーバーロード検出

RoboServerはOracleのJava仮想マシン(JVM)で実行され、Java仮想マシンはオペレーティング・システム(OS)上で実行され、オペレーティング・システムはハードウェア上で実行されます。JVMおよびOSにはパッチが適用され、ハードウェア・アーキテクチャが変更され、それぞれの新しいイテレーションはパフォーマンスの向上を目的としているため、パフォーマンスに関する一般的なガイドラインをいくつか示すことはできますが、確実に最適な構成にする唯一の方法は、構成をテストすることです。

一般的に、RoboServerの2つのインスタンスを起動すると、パフォーマンスが多少向上します。JVMでは、ガベージ・コレクション(GC)と呼ばれるメモリー管理が使用されます。ほとんどのハードウェアで、GC中は1つのCPUコアがアクティブであり、この場合、クアッドコアCPU上ではCPUの75%がアイドル状態です。RoboServerの2つのインスタンスを起動すると、1つのインスタンスがGCを実行している間に、もう1つのインスタンスは今までどおり全CPUを使用できます。

RoboServerで実行できる同時ロボットの数は、使用可能なCPUの量、およびRoboServerで処理する必要があるデータの取得可能速度に依存します。同時ロボット数は、Management Consoleのクラスタ設定で構成します。低速のWebサイトに対して実行するロボットは、応答時間が速いWebサイトに対して実行するロボットよりCPU使用率が大幅に低く、ここに理由があります。プログラムで使用するCPUの量は、次の式で説明できます。

```
CPU (core)% = 1 - WaitTime/TotalTime
```

ロボットの実行時間が20秒でも、15秒がWebサイトの待機時間である場合、実行しているのは5秒間のみであるため、20秒間の平均使用率は(CPUコアの)25%です。ロボットのステップは順に実行され、これは、実行中の1つのロボットが使用できるCPUコアは一度に1つのみであることを意味します。最新のCPUには複数のコアが搭載されているため、20秒実行しても15秒間待機するロボットは、実際にはクアッドコアCPUの約6%しか使用していません。

デフォルトで、RoboServerは、同時に最大で20のロボットを実行するように構成されています。同時ロボット数は、Management Consoleのクラスタ設定で構成します。すべてのロボットでCPUを6%使用すると、16から17のロボットを同時に実行している場合にCPUの稼働率はフルになります。これらの6%のロボットを同時に33起動すると、RoboServerは過負荷になり、使用可能なCPUの量は固定であるため、各ロボットの完了までに要する時間は2倍になります。実世界では、ロボットのCPU使用率は、ロボットのロジックおよびロボットが対話するWebサイトに応じて、5%から95%の間と推測されます。そのため、最大同時ロボット数の正確な値を推測または計算することは難しく、正しい値を確実に設定する唯一の方法は、負荷テストを実行して、RoboServerのCPU使用率、および負荷増加時のロボットの実行時間を監視することです。

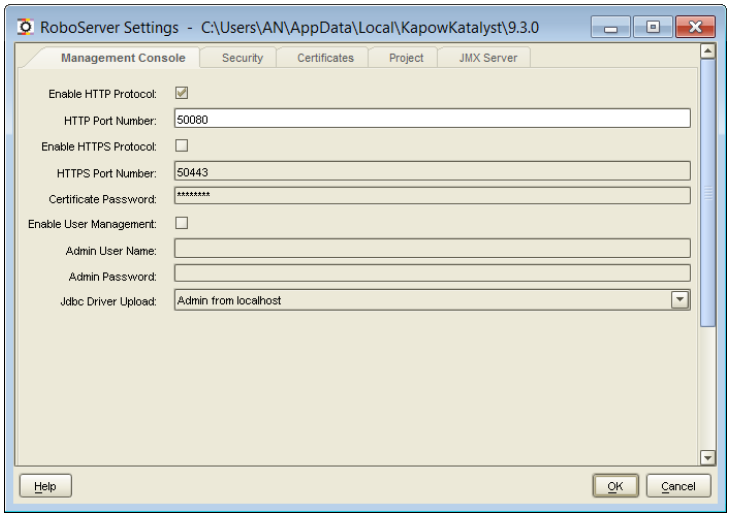
各RoboServerで処理できる同時ロボット数に影響を与える可能性があるもう1つのパラメータは、メモリー量です。ロボットによって使用されるメモリー量は、数MBから数百MBまで様々です。デフォルトで、RoboServerは1024MBのメモリーで構成されており、これは多くの場合、実際に20ロボットを同時に実行している場合は十分ではありません。「RAM割当の変更」を参照して、メモリー割当の制御方法を理解してください。RoboServerに十分なメモリーを提供しない場合は、メモリー不足エラーでクラッシュする危険性があります。適切なメモリー割当を保証する最も簡単な方法は、負荷テス

ト時にメモリー使用率を監視することです。RoboServerに2048MBのメモリーを割り当てた場合、JVMでは、事前にすべては割り当てられませんが、OSから予約しておきます(32ビットWindowsで1200MBを超える割当に失敗することが多いのはこのためです)。JVMでメモリーの使用が開始されると、OSには戻されません。最適なメモリー割当を確かめるには、通常、CPUを100%にする一連の負荷テストを実行する必要があります。各テストの完了後、予約済メモリーが実際にJVM (java.exeプロセス)によって使用された量を確認します。1024MB (デフォルト)すべてが使用された場合は、メモリーを(通常は2倍に)増やして、テストを再度実行します。JVMで予約済メモリーのすべてを使用しなかったある時点、および使用した量が、実際のメモリー要件を反映しており、この量をRoboServerの構成に使用する必要があります。

RoboServerでは、メモリー不足が発生するとクラッシュするため、この発生を防止しようとします。RoboServerでは、新しいロボットを起動する前にメモリー使用率が確認されます。これが80%を超えた場合は、ロボットを起動するかわりにキューに入れるため、メモリー割当が正しく構成されていない場合に、RoboServerがクラッシュする危険性が大幅に低減します。このメカニズムは通常、80%メモリーしきい値と呼ばれます。しきい値は、システム・プロパティ**kapow.memoryThreshold=80**を使用して構成可能です。

第18章 RoboServerの構成

この項では、RoboServer Settingsアプリケーションを使用してRoboServerを構成する方法について説明します。RoboServer Settingsは、個別のアプリケーションとして、たとえばWindowsの「スタート」メニューから、またはControl Centerから開くダイアログとして起動できます。



RoboServer Settingsのメイン・ウィンドウ

このアプリケーションを使用すると、次を構成できます(個々の項で説明します)。

- Management Console: 埋込みManagement Consoleの構成。
- セキュリティ: 認証および権限などのセキュリティ設定。
- 証明書: 証明書の使用方法。
- プロジェクト: デフォルトRoboServerプロジェクトの場所。
- JMXサーバー: JMXサーバー。

いずれかの設定の変更後は、「OK」をクリックして新しい設定を保存し、実行中のRoboServerを再起動して変更を有効にする必要があります。

RoboServerで使用できるRAMの最大量を変更する必要がある場合は、「RAM割当の変更」の項を参照してください。

埋込みManagement Consoleの構成

RoboServerには、Management Consoleを実行する埋込みWebサーバーが含まれています。WebサーバーはRoboServerの一部ですが、-MCオプションが有効な状態でRoboServerが起動された場合のみアクティブになります。デフォルトで、Webサーバーはポート50080でリスニングするため、次でManagement Console Webインタフェースを使用できます。

http://*host*:50080/

プロトコルおよびポート

Webサーバーは、別々のポートでHTTPおよびHTTPSを介してアクセスできるように構成できます。プロトコルが有効化されている場合は、そのポート番号を選択する必要があり、デフォルトはポート50080 (HTTP)およびポート50443 (HTTPS)です。

HTTPSを有効化するには、JKSフォーマットのサーバー証明書が、インストールの**Certificates/Web**フォルダ内の**tomcat.keystore**というファイルに格納されている必要があります。デフォルト

のパスワード(changeit)以外の証明書パスワードを使用する必要がある場合は、証明書パスワード・フィールドに入力できます。

管理セキュリティの有効化

Management Consoleには、同じコンピュータ(localhost)からのみでなく、別のコンピュータからもアクセスできます。Management Consoleの設定のポイントの1つは、ロボットの実行を調整することであり、このため、通常は多数のクライアントがアクセスできる必要があります。

他のマシンからManagement Consoleにアクセスできることの潜在的なセキュリティ・リスクを緩和するために、管理者パスワードを有効化できます。管理者パスワードの使用オプションを選択し、必要な管理者ユーザー名およびパスワードを入力します。これらの資格証明は、ロボットをDesign StudioからManagement Consoleに公開するとき、およびブラウザからWebインタフェースにアクセスするときに使用する必要があります。

JDBCドライバを埋込みManagement Consoleにアップロードできるユーザーを制限することもできます(JDBCドライバのアップロードの詳細はここを参照)。可能な選択肢には、いずれのユーザーもJDBCドライバをアップロードできないことを意味する、許可なし、adminユーザーがローカル・マシンからManagement Consoleにアクセスしている場合にドライバをアップロードできることを意味する、admin(localhostからのみ)、および最後に、adminユーザーが常時JDBCドライバをアップロードできることを意味する、admin(すべてのホストから)があります。

セキュリティ

RoboServer設定の「セキュリティ」タブでは、一般的なセキュリティ制限、およびRoboServerへのアクセスに認証が必要かどうかを指定します。監査ロギングもここで有効化できます。

制限

RoboServerでファイル・システム・アクセスおよびコマンドライン・アクセスが許可されるかどうかを指定できます。デフォルトでは、これは許可されません。これを有効にすると、RoboServerで実行中のロボットはファイル・システムにアクセスでき、コマンドラインの実行ステップを使用して、RoboServerを実行しているマシンで任意のコマンドを実行できます。警告: ファイル・システム・アクセスおよびコマンドライン・アクセスの有効化はセキュリティ・リスクであり、必要かどうかを慎重に検討する必要があります。有効にする場合は、マシンがローカル・ネットワーク外からアクセスできないこと、またはユーザー認証が必要であること(あるいはその両方)を確認する必要があります。ファイル・システム・アクセスおよびコマンドライン・アクセスが可能なRoboServerがインターネットからアクセス可能なマシン上で実行されていて認証を必要としない場合、RoboServerは基本的にそのマシンを外部に公開しており、たとえば誰でも、RoboServerを実行しているユーザーのアクセス権限に相当する方法でファイル・システムを変更できます。

Management ConsoleからのJDBCドライバの受入れを無効化することもできます。RoboServerをアクティブにすると、Management Consoleによってこれらに設定も送信されます。デフォルトで、これにはManagement Consoleにアップロードされているすべてのjdbcドライバが含まれます。悪意のあるユーザーがManagement Consoleに対する管理者アクセス権を入手した場合、同様に悪意のあるjarファイルをアップロードでき、それがRoboServerに送信されることになります。admin Management Consoleユーザーのみが、localhostからのJDBCドライバのアップロードを許可されている場合、前述の状況が発生するのは、攻撃者が実際にManagement Consoleが実行されているマシンの前にいるか、またはたとえばVPNに対するアクセス権を入手した場合(この場合は、より大きい問題と考えられます)のみであるため、一般的にはJDBCドライバの受入れを無効化する必要はありません。ただし、ここで説明されているように、インストール・フォルダの**lib/jdbc**ディレクトリに手動でJDBCドライバを格納することによって、JDBCドライバをRoboServerで使用可能にできます。

必須認証

無許可アクセスに対してRoboServerを保護する場合は、認証をオンにできます。これは、Kapow Katalystインストールから実行するすべてのRoboServer、つまりサービスとして起動するRoboServer、およびコマンドラインから起動するRoboServerの両方に影響を与えます。

認証をオンにするには、RoboServer認証が必要のチェック・ボックスを選択します。認証がオンのRoboServer上でロボットを実行できるようにするには、何人かのユーザーを追加する必要があります。これを行うには、追加ボタンをクリックします。新しい無名ユーザーが挿入されます。その後は、ユーザーのリストに表示されるユーザー名など、ユーザーに関する情報を入力できます。

ユーザーは、次のプロパティを使用して構成します。

表18.1 ユーザーのプロパティ

プロパティ	説明
ユーザー名	RoboServerにアクセスする際にユーザーが使用するユーザー名です。
パスワード	RoboServerにアクセスする際にユーザーが使用するパスワードです。
コメント	ここに、ユーザーに関するコメントを記述できます。
ロボットの起動	ユーザーがRoboServer上のロボットを起動できるようにします。
ロボットの停止	ユーザーがRoboServer上のロボットを停止できるようにします。
実行の表示	ユーザーがControlCenterアプリケーションからRoboServerに接続できるようにします。
設定の表示/編集	ユーザーがControlCenterアプリケーションからKapow Katalystの構成を表示および編集できるようにします。
RoboServerのシャットダウン	ユーザーがControlCenterアプリケーションからRoboServerをシャットダウンできるようにします。

認証ありで構成されているRoboServerでロボットを実行するには、呼出し側が適切な資格証明を提供する必要があります。Management Consoleでは、これは設定で行われます。Java APIを介してロボットを実行する場合、資格証明は「実行パラメータ」で説明されているとおりに提供されます。

監査ロギングの構成

RoboServerからのすべてのHTTPおよびFTPリクエストが自動的に記録されるようにするには、HTTP/FTPトラフィックを記録オプションを選択します。この結果、**log4j.logger.kapow.auditlog**で指定したLog4Jロガーを使用して、HTTPおよびFTPリクエストが記録されます。Log4Jは、アプリケーション・データ・フォルダの**Configuration**フォルダにある**log4j.properties**ファイルで構成します。

監査ログでは、Webページとそのすべてのリソースの両方に対するリクエストがすべて記録されます。次は、Googleのフロント・ページのロードによって生成されたログの抜粋です。

```
02-29 13:51:21
INFO kapow.auditlog - google google.com http://google.com/
301 0 02-29 13:51:21
INFO kapow.auditlog - google www.google.com http://www.google.com/
200 81688 02-29 13:51:22
INFO kapow.auditlog - google
www.google.com http://www.google.com/extern_js/f/CgJkYRICZGsrMEUjtw.js
200 328960 02-29 13:51:22
INFO kapow.auditlog - google
www.google.com http://www.google.com/extern_chrome/d9924a47b8c72e1a.js
200 43796 02-29 13:51:23
INFO kapow.auditlog - google ssl.gstatic.com
http://ssl.gstatic.com/gb/js/sem_6501b4b3093bbedb61d2e.js
200 31642
```

ログには次の情報が含まれています。

- タイムスタンプ
- ログ・レベル(INFO)
- ロガー(kapow.auditlog)
- ロボット名
- ホスト名
- リクエストURL
- HTTP/FTPレスポンス・コード
- レスポンス本体からロードされたバイト数

証明書

セキュアな通信の確立の主要な問題は、正しいパーティと通信していることを確認することです。他のパーティからのアイデンティティ情報をリクエストするのみでは十分ではなく、この情報を信頼するには情報を検証する方法も必要です。証明書は、パーティのアイデンティティ情報(パーティのホスト名と公開鍵を含む)と、アイデンティティ情報が正しいことを保証する信頼できるサード・パーティの署名の両方を含めることで、これに対するソリューションを提供します。証明書を信頼するのは、次の場合のみにする必要があります。

- 証明書に記載されているホスト名が、証明書の発行元サイトのホスト名と一致している場合(それ以外の場合は、他の誰かのアイデンティティの記録であり、誤った資格証明を使用していることになります)。
- および信頼しているサード・パーティによって証明書が署名されている場合。
- (さらに、証明書の有効期限日などを確認する必要がありますが、これらの詳細はここでは説明しません。)

署名プロセスの技術的な詳細については、公開鍵/秘密鍵暗号化に基づいているということ以外は詳しく説明しません。証明書の署名は、証明書の内容と署名者の秘密鍵の両方を含む複雑な計算の(かなり簡潔な)結果であり、これは、その同じ秘密鍵がないと再生成できません。署名は、証明書の内容、署名および署名者の公開鍵を含む別の計算をすることで検証できます。この計算によって、証明書が署名と一致するかどうか、したがって正しいかどうかがわかります。公開鍵では署名の検証のみが可能で、署名の生成はできないことに注意してください。したがって、公開鍵によって誰かが証明書を偽造することはできません。

署名パーティは、秘密鍵を決して誰かに与えてはいけませんが、公開鍵はできるかぎり広範に配布します。ただし、署名を信頼するにはまだ1つ問題があり、署名パーティの公開鍵の本物のコピーを所有していることを確認する必要があります。VeriSign社などの既知の署名機関の公開鍵は、あらゆるブラウザおよびJava Runtimeで配布されており、署名(したがって証明書)を信頼することは、実際にはブラウザまたはJava Runtimeがインストールされた方法を信頼することに基づきます。

公開鍵/秘密鍵のペアを作成して公開鍵を配布することによって、独自の署名機関を作成できます。これを行うには、証明書に公開鍵を埋め込みます(いわゆる自己署名証明書です)。もちろん、このような証明書を受け取るパーティは、その署名に基づいて証明書を信頼するのではなく、発行者と、証明書の通信方法を信頼しているためです。

実際の実装でこの仕組みをより柔軟にするには、署名する権限を委任できます。つまり、証明書の署名は実際には、信頼しているサード・パーティの署名ではなく、信頼する証明書を表示できる別の

パーティの署名の場合があります。この結果、レベルはいくつでも拡張可能であり、信頼のチェーンとなります。検証を効果的に行うために、署名した証明書には、信頼チェーンに関連付けられたすべての証明書(最後の証明書が自己署名ルート証明書)のコピーが含まれます。このチェーン内の証明書の少なくとも1つを、以前に信頼している必要があります。

RoboServerでは、証明書は、証明書タブの4つのプロパティに対応する4つの異なる方法で使用されます。この中の2つは、ロボットがその実行の一部としてWebサーバーにアクセスする方法に関係します。

HTTPS証明書の検証: ロボットは、アクセス先のWebサーバーのアイデンティティを(HTTPS経由で)検証する必要がある場合があります。このような検証は、フィッシング攻撃を検出するために、通常のブラウザによって常に(およびわからない方法で)行われています。ただし、目的のWebサイト用に作成されたロボットはそれらの既知のWebサイトのみにアクセスするため、ロボットによる情報の収集時には通常、検証は必要ありません。したがって、検証はデフォルトでは有効化されていません。

有効化すると、検証はブラウザと同じ方法で行われます。Webサーバーの証明書は、ブラウザで構成可能なものと同じ信頼できるHTTPS証明書のインストール済セットに基づいて確認されます。

HTTPSクライアント証明書: これは、逆方向であること以外は前述とほぼ同じです。ロボットは、アクセス元のクライアント(ロボット)のアイデンティティを検証する必要があるWebサーバーにアクセスする必要がある場合があります。おそらくWebサーバーには、証明済のアイデンティティがあるクライアントにのみ渡される資格証明または商用データが含まれています。このアイデンティティは、HTTPSクライアント証明書によって表されます。

他の2つのプロパティは、RoboServerと、ロボットがRoboServerで実行されるようにするAPIクライアントとの間の通信における認証に関係します。これらのプロパティは、クライアントがセキュアなソケット・ベースのRQLプロトコル経由でRoboServerに接続する場合にのみ適用されます。セキュアなプロトコルの主な目的は通信の暗号化ですが、少し構成すると、認証(アイデンティティ検証)も提供されます。

APIクライアント証明書の検証: RoboServerは、ロボットを実行するためにRoboServerに接続するクライアントのアイデンティティを検証できます。この検証はデフォルトで無効化されています。

有効化した場合、目的はかなり違いますが、検証の方法はHTTPS証明書の場合と同じです。接続中のクライアントの証明書は、信頼できるAPIクライアント証明書のインストール済セットに基づいて確認されます。

APIサーバー証明書: RoboServerには、接続中のクライアントに提示するサーバー証明書もあります。この証明書には二重の目的があり、1つはSSLの暗号化側が機能するようにすること(このために、RoboServerにはデフォルトの自己署名証明書が付属しています)、もう1つはこの特定のRoboServerが本物であることをクライアントに示すことです。

デフォルトのAPIサーバー証明書は、すべてのRoboServerに対して同じであるため、識別には十分ではありません。クライアントで接続先のRoboServerのアイデンティティを検証する必要がある場合は、「SSL」の説明に従って、RoboServerごとに一意のAPIサーバー証明書を作成してインストールする必要があります。

HTTPS証明書

ロボットがHTTPS経由でWebサイトにアクセスする場合は、サイトの証明書を検証します(HTTPS証明書の検証チェック・ボックスが選択されている場合)。検証は、信頼できる証明書の2つのセットである、ルート証明書のセットと追加のサーバー証明書のセットに基づいて行われます。

ルート証明書は、ブラウザとともにインストールされる場合と同様に、Kapow Katalystとともにインストールされます。これらは、アプリケーション・データ・フォルダの**Certificates/Root**フォルダにあります。

一部のHTTPSサイトでは、デフォルトで組み込まれていない認証局を使用している場合があります。この場合、Kapow Katalystでこれらのサイトからロードするには、適切な証明書をインストールする必要があります。多くの場合、これらは、アプリケーション・データ・フォルダの**Certificates/Server**フォルダにインストールされています。

実際には、HTTPSサイトの処理が目的の場合、証明書をルート証明書のセットとサーバー証明書のセットのいずれに追加するかは問題ではありません。ただし、ルート証明書はAPIクライアント証明書の確認時にも使用されるため、その範囲はより広範囲であることに注意してください。

証明書をインストールするには、PKCS#7証明書チェーン、Netscape証明書チェーンまたはDERでエンコードされた証明書として証明書を取得する必要があります。証明書をインストールするには、前述の2つのフォルダのいずれかに証明書をコピーします。証明書を格納するファイルの名前は問題ではありません。

次の例は、Webサイトhttps://www.foo.comのサーバー証明書をインストールする方法について説明しています。例はInternet Explorerに基づいています。

- Internet Explorerでhttps://www.foo.comを開きます。
- 「ファイル」メニューの下の「プロパティ」を選択し、次に「証明書」を選択します。「証明書」ダイアログが開きます。
- 「証明書のインストール」をクリックします。
- 「次へ」を2回クリックし、「完了」を1回クリックすると証明書ダイアログが終了します。これで、ブラウザに証明書がインストールされました。次のステップは、それをファイルにエクスポートすることです。
- 「ツール」メニューの「インターネット オプション」をクリックします。「インターネット オプション」ダイアログが開きます。
- 「コンテンツ」タブに移動し、「証明書」ボタンをクリックします。
- ここで、インストールした証明書を探す必要があります。多くの場合、「他の人」というタブにあります。
- 証明書を選択して「エクスポート」を押します。
- 「次へ」を2回押します。
- 証明書を**foo.cer**としてアプリケーション・データ・フォルダの**Certificates/Server**フォルダに保存します。
- 実行中のRoboServersをすべて再起動します。

注記

特定のHTTPSサイトからロードする必要があるすべてのインストールに、証明書をインストールする必要があります。

HTTPSクライアント証明書

ロボットがHTTPSサイトにアクセスするとき、Webサーバーに対してそれ自身の証明書を提供する必要がある場合があります。これは、ロボットのデフォルト・オプション、またはステップ固有のオプションで設定します。証明書を提供する1つの方法は、Kapow Katalystインストールに構成されている証明書の1つを参照することです。

新しいクライアント証明書を追加するには、RoboServer Settingsの証明書タブで、HTTPSクライアント証明書のリストの下にある  アイコンをクリックします。証明書ファイル(PKCS12フォーマットであることが必要です)の選択を求めるプロンプトが表示され、ファイルの暗号化に使用するパスワードを入力する必要があります。証明書が初めてリストに入力されると、一意のIDが付与され、これは後で証明書を選択するためにロボットで使用されます。IDを変更するには、  アイコンをクリックします。証明書は、ロボットを実行する必要があるすべてのKapow Katalystインストールに構成する必要があることに注意してください。

APIクライアント証明書

APIクライアントがSSL経由でRoboServerに接続する場合、RoboServerは、提示された証明書を検証します(APIクライアント証明書の検証チェック・ボックスが選択されている場合)。検証は、検証に失敗したクライアントからの接続をRoboServerが拒否することを意味し、信頼できる証明書の2つのセットである、ルート証明書のセットと追加のAPIクライアント証明書のセットに基づいて行われます。

ルート証明書は、ブラウザとともにインストールされる場合と同様に、Kapow Katalystとともにインストールされます。これらは、アプリケーション・データ・フォルダの**Certificates/Root**フォルダにあります。これらは、HTTPS証明書の確認に使用される同じルート証明書ですが、APIクライアント検証時はおそらく、ルート証明書の役割が大幅に縮小されます。

これは、ほとんどの場合、正式な署名機関によって発行される(高価な)証明書を使用するかわりに、独自の自己署名APIクライアント証明書を作成するためです。APIクライアント証明書は、RoboServerで認識されるように、アプリケーション・データ・フォルダの**Certificates/API/TrustedClients**にインストールする必要があります。

技術的には、APIクライアントの接続の検証が目的の場合、APIクライアント証明書をルート証明書のセットとAPIクライアント証明書のセットのいずれに追加するかは問題ではありません。ただし、前述のガイドラインは、ルート証明書がHTTPS証明書の確認時にも(どちらかというと主に)使用されるという事実から発生する問題の回避に役立ちます。

APIクライアント用の自己署名証明書は、次のようにJava **keytool**コマンドを使用して生成できます。

```
keytool -genkey -keystore client.p12 -alias client -keyalg
        RSA -storetype "PKCS12"
```

名前(ドメイン)、組織単位名、組織、市区町村、都道府県、国およびパスワードの各情報の入力を求められます。パスワードは、忘れた場合に取得する方法がないため忘れないでください。この**keytool**の呼出しでは、証明書がキーストア**client.p12**に入れられます。次に、それを別のファイルに抽出する必要があります。

```
keytool -export -keystore client.p12 -alias client -storetype
        "PKCS12" -file client.pub.cer
```

証明書が生成されたときに使用したパスワードの入力を求められます。出力ファイル**client.pub.cer**は、アプリケーション・データ・フォルダの**Certificates/API/TrustedClients**フォルダにコピーする必要があります。

APIサーバー証明書

技術的な理由で、RoboServerには、SSLを使用してAPIクライアントに接続する際に、APIクライアントに提示できるサーバー証明書が必要です。インストール時に、デフォルトの自己署名証明書がインストールされます。この証明書は、すべてのRoboServerに対して同じであるため識別目的には十分ではなく、APIクライアントでRoboServerのアイデンティティを検証する必要がある場合は置換する必要があります。

VeriSign社などの署名機関によって発行される証明書を使用する場合は、RoboServer Settingsの証明書タブにある「変更」ボタンをクリックしてインポートできます。ファイル選択ダイアログが開き、証明書を特定すると、パスワードの入力を求められます。パスワードが正しい場合は、証明書がインポートされ、証明書の発行先、発行元および有効期限の各プロパティが表示されます。

RoboServer用の新しい自己署名証明書は、次のようにJava **keytool**コマンドを使用して生成できます。

```
keytool -genkey -keystore server.p12 -alias server -keyalg
        RSA -storetype "PKCS12"
```

名前(ドメイン)、組織単位名、組織、市区町村、都道府県、国およびパスワードの各情報の入力を求められます。パスワードは、忘れた場合に取得する方法がないため忘れないでください。証明書は、キーストア・ファイル**server.p12**に保存され、これを説明に従ってインポートできます。

デフォルトRoboServerプロジェクトの設定

RoboServer Settingsの「プロジェクト」タブで、デフォルトのRoboServerプロジェクト・フォルダの場所を設定できます。フォルダはデフォルトで、インストール・プロセスによってインストール・フォルダに作成されたデフォルトのロボット・プロジェクトに設定されます。ロボット・プロジェクトの詳細は、「Design Studioユーザーズ・ガイド」を参照してください。

RoboServerデフォルト・プロジェクトはAPIでのみ使用されます。APIからロボットを実行する際に、タイプ、スニペットまたはその他のリソースに対する参照は、デフォルト・プロジェクトを探すことによって解決されます。

JMXサーバーの構成

埋込みJMXサーバーは、JConsoleなどのツールを介して、実行中のRoboServerの監視に使用できます。有効にするには、RoboServerコマンドラインで引数を指定します(詳細は「RoboServerユーザーズ・ガイド」を参照)。

センシティブなロボット入力の非表示

入力の表示オプションでは、ロボットの入力パラメータを管理インタフェースに表示するかどうかを制御します。これによって、パスワードなどのセキュリティ上の機密情報を非表示にできます。

JMXサーバー・アクセス

デフォルトで、JMXサーバーには、サーバー上の正しいポートにアクセスできるすべてのクライアントからアクセスできます。パスワードの使用オプションを選択すると、接続時に、選択したユーザー名とパスワードを指定する必要があります。

ハートビート通知

0を超える間隔(秒)を指定すると、RoboServerが実行中で問合せに応答している間は、JMXサーバーによって特定の間隔でハートビート通知が送信されます。

RAM割当の変更

インストール時に、各Kapow Katalystアプリケーションは、使用する可能性がある最大RAM量を指定して構成されます。この量は一般に、通常の作業には十分ですが、RoboServer上で多数のロボットを並列で実行する場合、または一部のロボットで大量のRAMを使用する場合は、割当を増やす必要がある場合があります。

いずれかのアプリケーションに対する割当を変更するには、インストールの**bin**サブフォルダにある、その**.conf**ファイルを編集します。RoboServerの場合、編集するファイルの名前は次のとおりです。

bin/RoboServer.conf

次の行の追加が必要です。

wrapper.java.maxmemory=1024

たとえば、RoboServerでRAMを4GBまで使用できるようにするには、次の行を追加します。

wrapper.java.maxmemory=4096

(この高い割当は、64ビット・バージョンのKapow Katalystでのみ可能です)。

.confファイルを編集できるのは、Kapow Katalystをインストールしたユーザー(たとえば、通常はWindowsの管理者)のみであることに注意してください。

必要な場合は、他のKapow Katalystアプリケーションに対する最大RAM割当を同様に変更できます。

第19章 Control Centerユーザーズ・ガイド

Control Centerは、RoboServerを管理するためのレガシー・アプリケーションです。RoboServerおよびロボットのほとんどの監視および管理は、現在はManagement Consoleを使用して実行されます。Management Consoleを利用しない古い設定については、Control Centerを使用できます。Control Centerでは、Javaまたは.NETクライアントからRoboServerへの直接通信によって開始されたロボット実行の監視も可能であり、ロボット実行の進捗情報が提供されます。

構成

ガイドは、次の項に分かれています。

- 基本概念。
- ユーザー・インタフェース。
- RoboServerを管理するためのControl Centerの使用方法。
- ロボットを管理するためのControl Centerの使用方法。
- 多岐にわたるその他のControl Centerの機能。

Control Centerの基本

この項では、Control Centerで使用される基本概念について説明します。

Control Centerの主なタスクは次のとおりです。

- RoboServerの監視およびシャットダウン。
- RoboServer上のロボットの監視および停止。
- 様々な管理タスク。

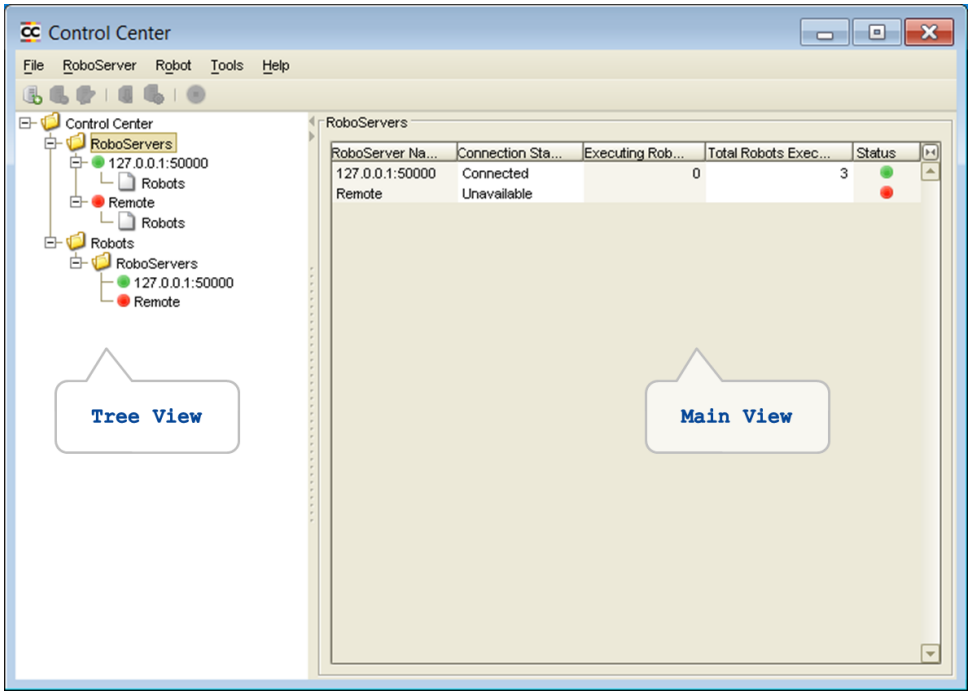
Control CenterでRoboServerを管理するには、RoboServerをControl Centerに登録する必要があります。RoboServerの登録とは単に、Control Centerで管理できるRoboServerのリストにRoboServerを追加することです。

Control Centerのユーザー・インタフェース

この項では、アプリケーションのユーザー・インタフェースを紹介して、Control Centerの使用を開始できるようにします。

メイン・ウィンドウ

Control Centerのメイン・ウィンドウを次に示します。



Control Centerのメイン・ウィンドウ

ウィンドウの左側はツリー・ビューです。ツリーのルート(Control Centerというラベル)には2つのメイン・カテゴリである2つの子があります。

- RoboServerカテゴリ(前述の選択状態のカテゴリ)には、現在Control Centerに登録されているRoboServerが表示されます。
- 「ロボット」カテゴリには、登録されているすべてのRoboServer上のロボットが表示されます。

前述のツリー・ビューからわかるように、「ロボット」カテゴリにはRoboServerというサブカテゴリがあり、これには個々のRoboServer上のロボットが表示されます。

ウィンドウの右側はメイン・ビューです。メイン・ビューには、ツリー・ビューの現在の選択に関する詳細が表示されます。その中に、登録済RoboServerの概要が表示されます(ツリー・ビューで選択されているカテゴリRoboServerに対応)。

最上部には、メニュー行とツール・バーが表示されます。メニュー行とツール・バーには、ツリーおよびメイン・ビューの現在の選択に関連するアクションが含まれます。

Control Centerのナビゲート

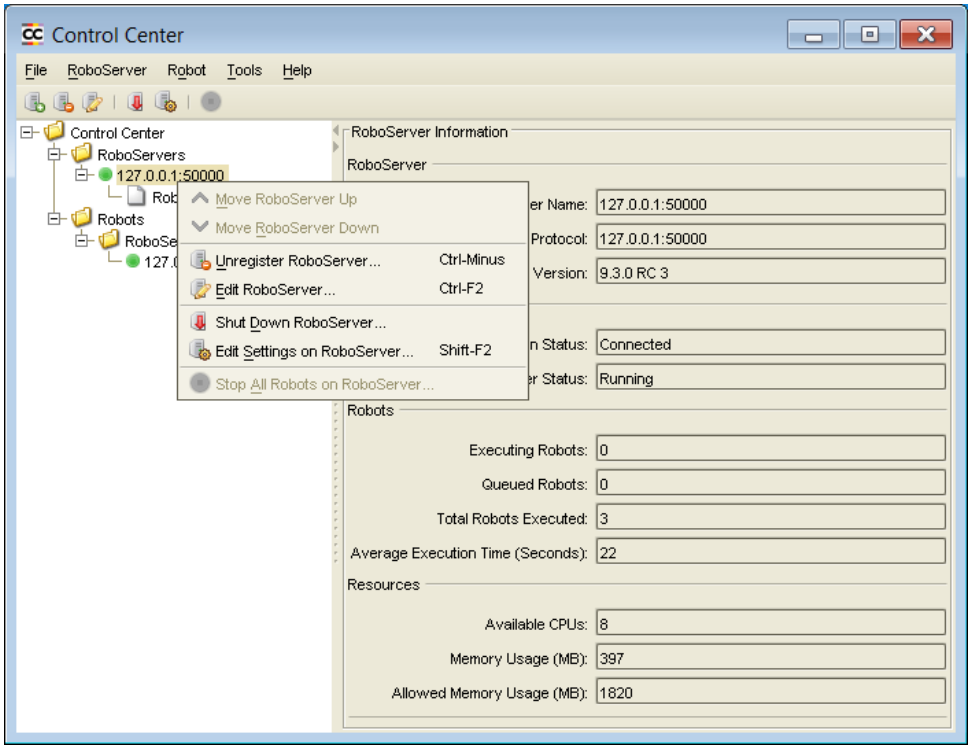
選択の変更

Control Centerを使用する場合は、ツリー・ビューの選択で現在表示される内容が決まります。マウスの左ボタンを使用してツリー・ビュー内の選択を行います。選択を変更すると、それに応じて、メイン・ビューおよびメニュー行とツール・バーは、現在の選択に関連する内容を表示するように変わります。

アクションおよびポップアップ・メニュー

前述のように、メニュー行とツール・バーには、現在の選択に関連するアクションが含まれます。これらのアクションには、キーボード・ショートカットが設定されているものもあります(たとえば、[Ctrl]-[F2]を押すとRoboServerを編集できます)。

ツリー・ビュー内またはメイン・ビューの表内を右クリックすると、ポップアップ・メニューが開きます。ポップアップ・メニューには、クリックしたアイテムに適用されるアクションが含まれます。



ポップアップ・メニュー

通常、アクションを実行する方法は複数あります。

- ツール・バーでアクションを選択します。
- 適切なメニューでアクションを選択します。
- 右クリック(ツリー・ビューまたはメイン・ビュー内)して、ポップアップ・メニューのアクションを選択します。
- キーボード・ショートカットを押します(ある場合)。

ツリー・ビューおよびメイン・ビュー内の選択に応じて、メニューには常に、適用可能なすべてのアクションが含まれますが、ツール・バーには最もよく使用されるアクションが含まれます。

メイン・ビューの表

メイン・ビューの表内のアイテムをダブルクリックすると、クリックしたアイテムに関する詳細を表示するように選択内容が変わります。たとえば、RoboServerの表を現在表示していて、表内のRoboServerをダブルクリックすると、このRoboServerがツリー・ビュー内で選択され、メイン・ビューには、このRoboServerに関する詳細が表示されます。

メイン・ビューの表の列ヘッダーを左クリックすると、クリックした列で表がソートされます。

メイン・ビューの表のヘッダーを右クリックすると、表示する列を選択できます。この選択は、現在のビューにのみ適用されることに注意してください。

RoboServerの移動

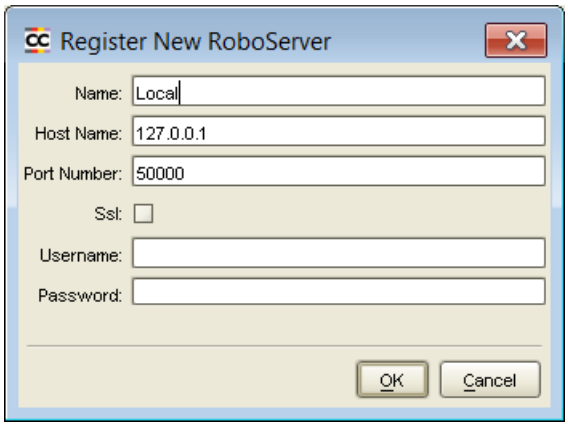
ポップアップ・メニューの例で示したように、ツリー・ビュー内のRoboServerを右クリックして、RoboServerを上へ移動またはRoboServerを下へ移動を選択すると、RoboServerを移動できます。RoboServerは、RoboServerカテゴリ内、および「ロボット」カテゴリのRoboServerサブカテゴリ内で移動します。

RoboServerの管理

この項では、Control Centerを使用してRoboServerを管理する方法について説明します。

RoboServerの登録

RoboServerを登録するには、  アイコンをクリックしますが、クリックすると、次に示すRoboServer登録ダイアログが表示されます。



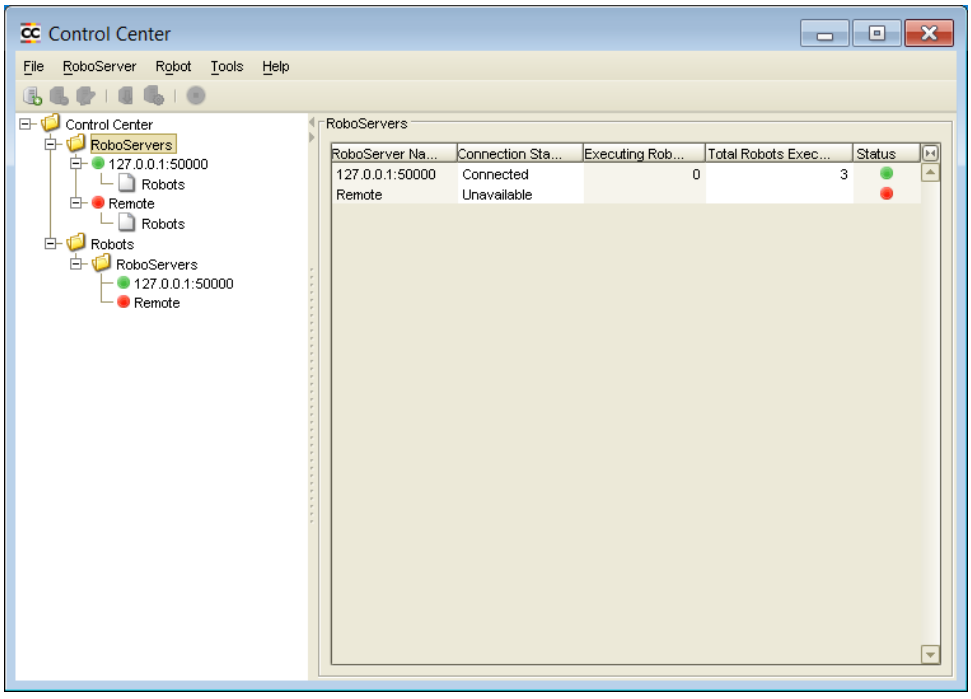
RoboServer登録ダイアログ

ここで、RoboServerに、ツリー・ビューで表示される名前を設定します。必要に応じて、登録するRoboServerのプロトコルの選択と構成、およびRoboServerの資格証明の追加も必要です。完了後、「OK」を押してRoboServerを登録します。

すでに登録されているRoboServerは登録できないことに注意してください。

RoboServerの監視

登録したRoboServerを監視するには、ツリー・ビュー内のRoboServerカテゴリを選択します。この結果、次に示すように、メイン・ビューに登録済RoboServerの概要が表示されます。



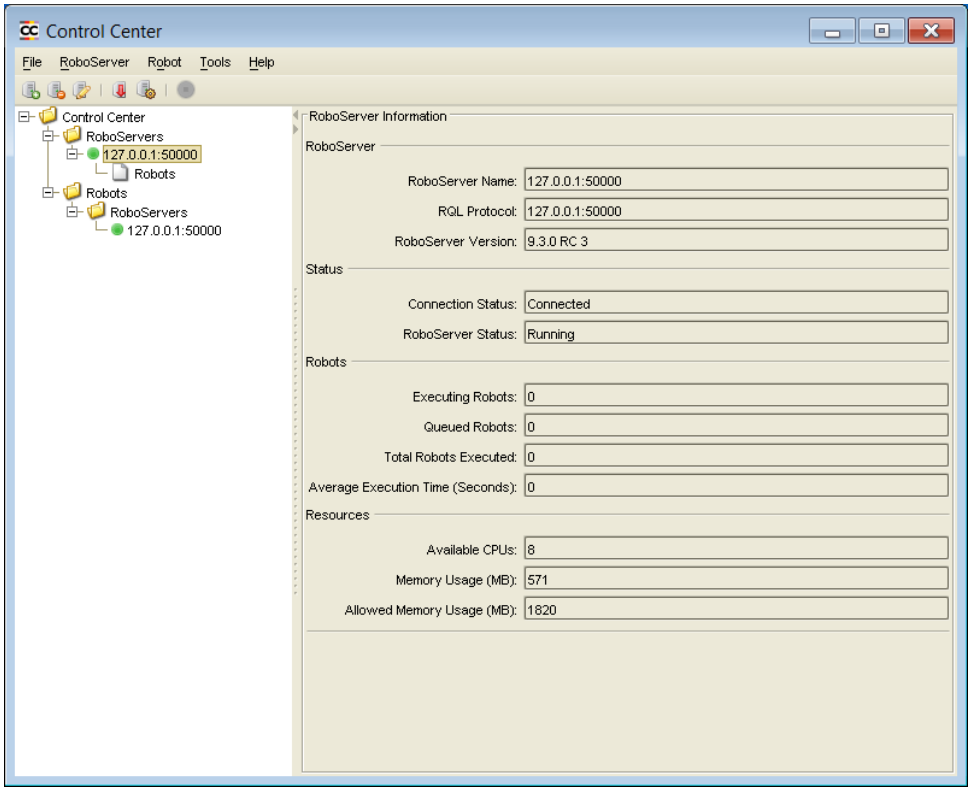
RoboServerの概要

RoboServerのステータスがツリー・ビュー内およびメイン・ビューの表内にアイコンとして表示されます。

ステータスは次のいずれかです。

- 接続済および実行中 – アイコンで示されます。
- 接続済およびシャットダウン中 – アイコンで示されます。
- 使用不可 – アイコンで示されます。
- 不明 – アイコンで示されます。

RoboServerをツリー・ビューで選択した場合(または表内のRoboServerをダブルクリックした場合)は、メイン・ビューに、この特定のRoboServerに関する詳細が表示されます。



RoboServerの表示

このビューには、次の情報が表示されます。

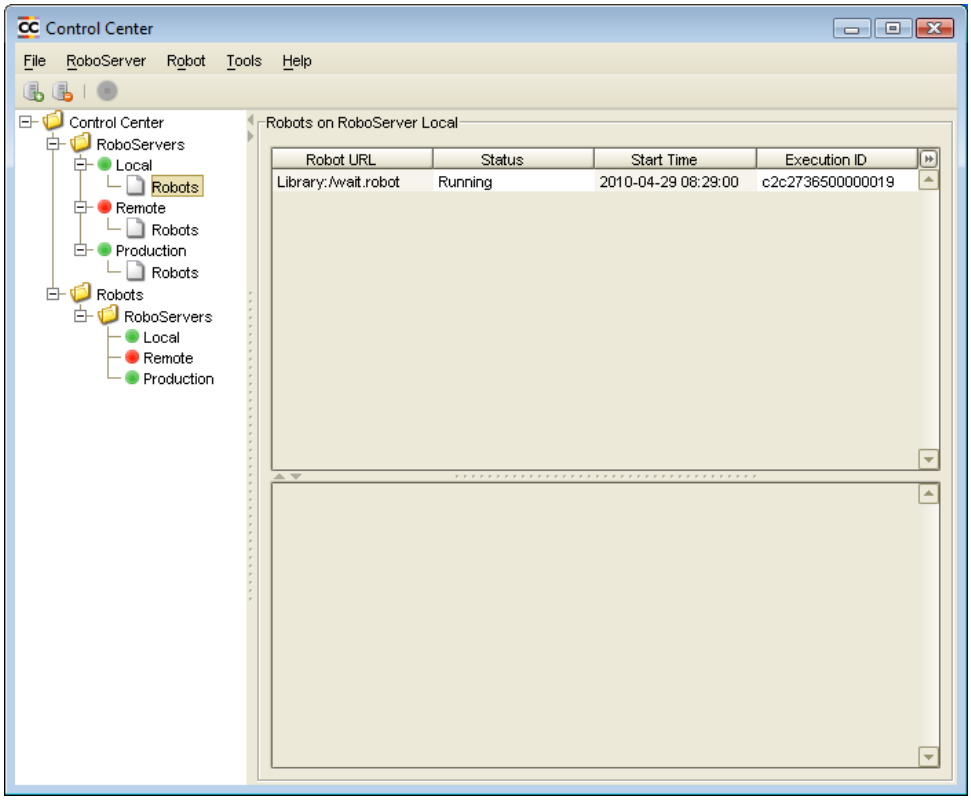
RoboServer: RoboServerの名前およびプロトコル。

ステータス: 接続ステータス(接続済、使用不可または不明)、および接続済の場合はRoboServerのステータス(実行中またはシャットダウン中)。

ロボット: 現在実行中のロボットの数、キュー待機中のロボットの数、実行されたロボットの合計数、および平均実行時間。

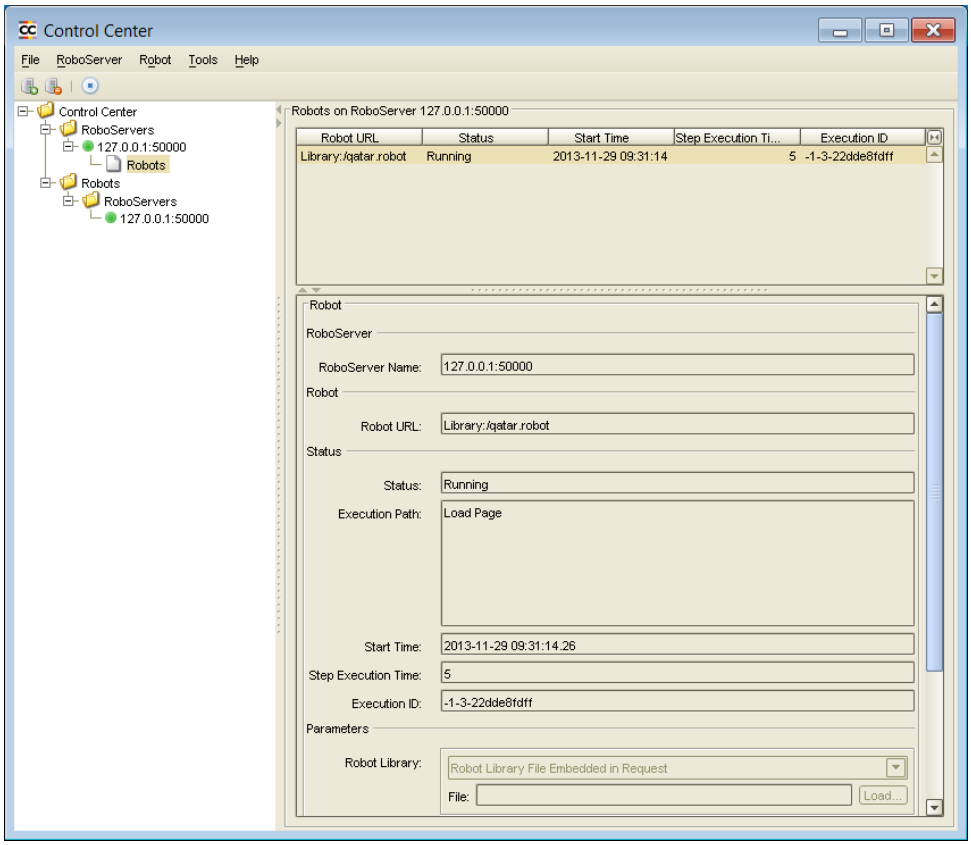
リソース: RoboServerの使用可能リソースおよび使用されているリソースに関する情報。

「ロボット」サブカテゴリを選択すると、メイン・ビューには、この特定のRoboServer上のロボットの概要が表示されます。



RoboServerのロボットの表示

メイン・ビューの表内のロボットを選択すると、この特定のロボットに関する詳細が表の下に表示されます。



ロボットの詳細の表示

このビューには、次の情報が表示されます。

RoboServer: RoboServerの名前。

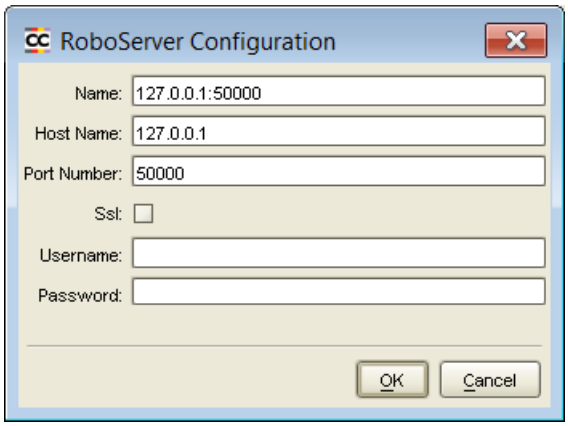
ロボット: ロボットのURL。

ステータス: ロボットのステータス(実行中またはキュー待機中)、開始時間および実行ID。

パラメータ: ロボットのライブラリおよび環境。

RoboServerの編集

RoboServerの名前およびプロトコルを編集するには、RoboServerを選択して  アイコンをクリックします。次に示すようなダイアログが表示されます。

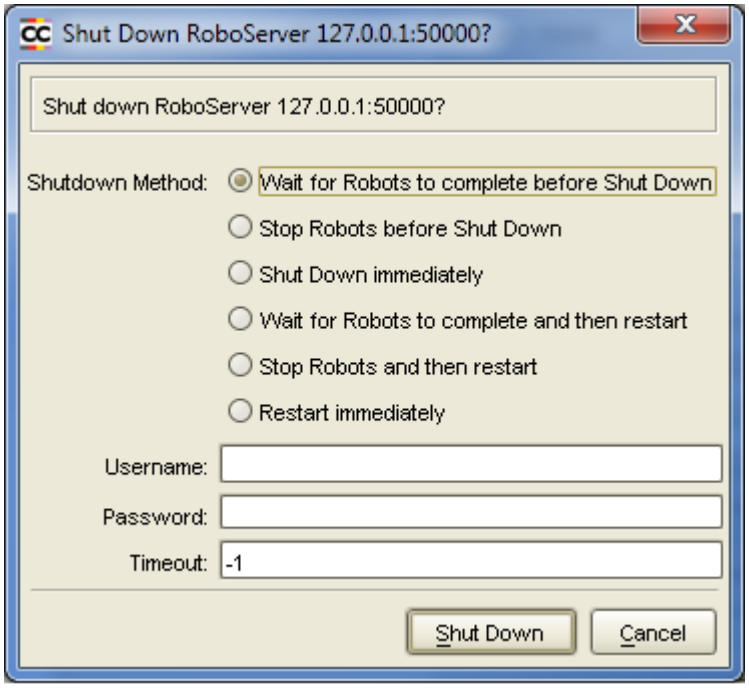


RoboServerの編集

RoboServerの名前は任意に設定できますが、プロトコルは、Control Centerに登録された別のRoboServerですでに使用されているプロトコルには変更できません。

RoboServerのシャットダウン

RoboServerをシャットダウンするには、接続済のRoboServerを選択して  アイコンをクリックします。次に示すようなダイアログが表示されます。



RoboServerのシャットダウン

RoboServerをシャットダウンする場合は、シャットダウン方法を選択する必要があります。シャットダウン方法は6つあります。

シャットダウン前にロボットの完了を待機:	RoboServerは、RoboServer上のすべてのロボット(ある場合)が完了するまでシャットダウンされません。
シャットダウン前にロボットを停止:	RoboServer上のすべてのロボットがすぐに停止され、停止後すぐにRoboServerがシャットダウンされます。
すぐにシャットダウン:	RoboServerは、最初にロボットを停止することなくすぐにシャットダウンされます。
ロボットの完了を待機した後再起動:	RoboServerは、RoboServer上のすべてのロボット(ある場合)が完了するまで再起動されません。
ロボットを停止して再起動:	RoboServer上のすべてのロボットがすぐに停止され、停止後すぐにRoboServerが再起動されます。
すぐに再起動:	RoboServerは、最初にロボットを停止することなくすぐに再起動されます。

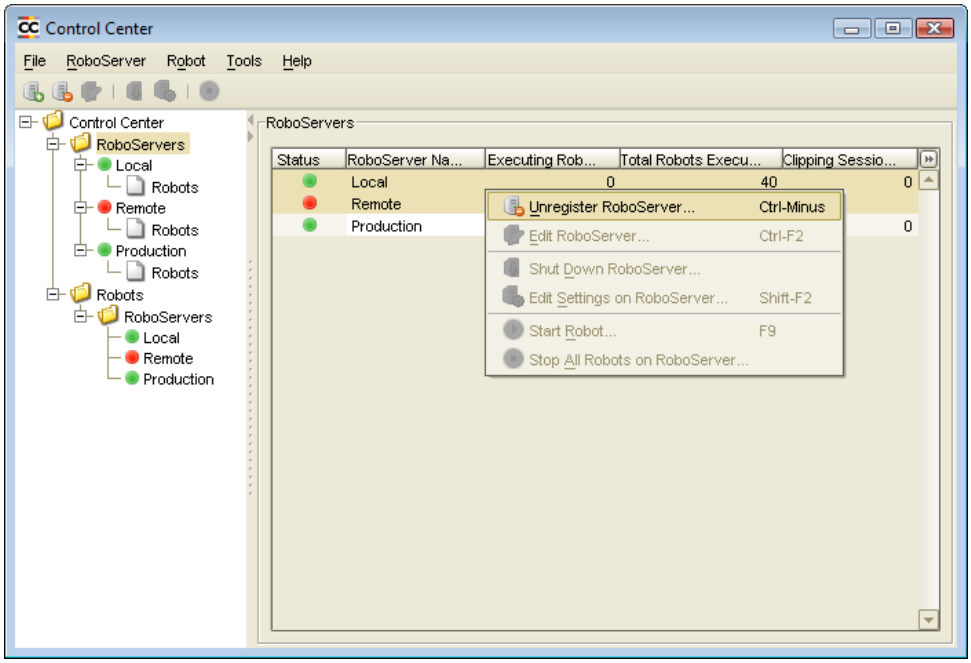
RoboServerに認証が必要な場合は、ユーザー名とパスワードを入力する必要があります。

さらに、タイムアウト(秒)を指定できます。このタイムアウトが経過すると、サーバーは、すべてのロボットが完了しているかどうかに関係なく、(選択内容に応じて)シャットダウンまたは再起動されます。タイムアウトの使用を無効化するには、フィールド値を-1 (デフォルト)に設定します。

RoboServerの登録解除

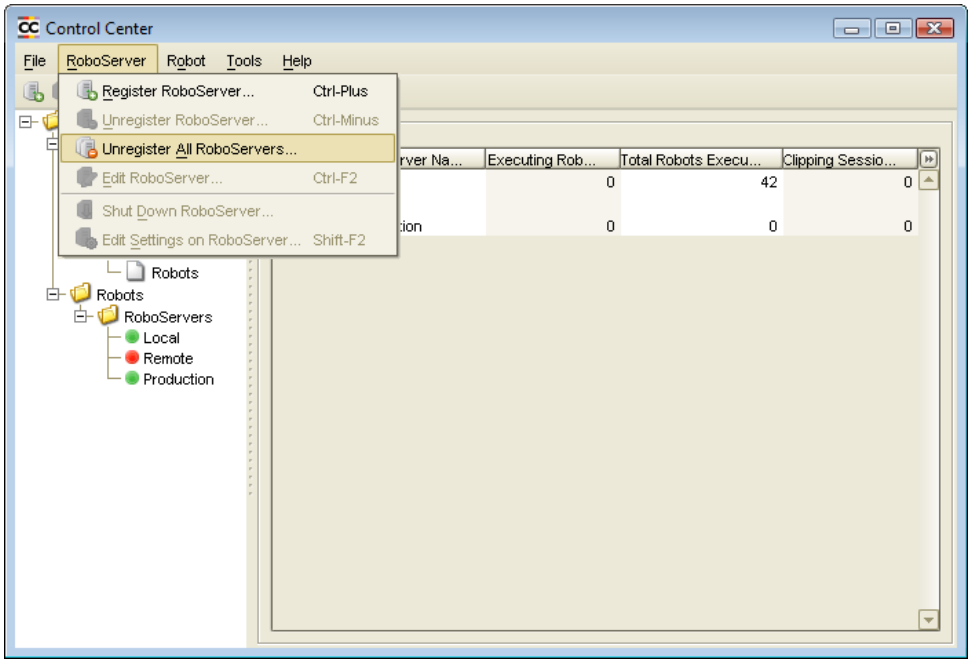
RoboServerの登録を解除するには、RoboServerを選択して  アイコンをクリックします。登録解除の確認を求めるダイアログが表示されます。

複数のRoboServerをシャットダウンするには、メイン・ビューの表で複数のRoboServerを選択し、前述のとおりに実行するか、または次に示すように、選択項目を右クリックしてRoboServerの登録解除... を選択します。



複数のRoboServerの登録解除

すべてのRoboServerの登録を解除するには、次に示すように、RoboServerメニューを開いてすべてのRoboServersの登録解除... を選択します。



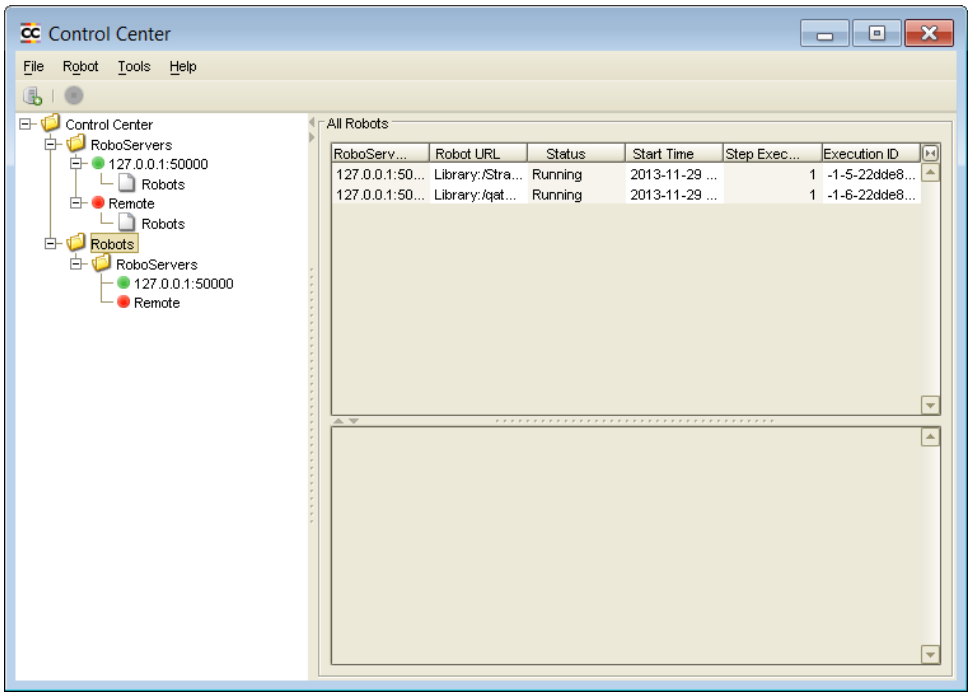
すべてのRoboServerの登録解除

ロボットの管理

この項では、Control Centerを使用してロボットを管理する方法について説明します。

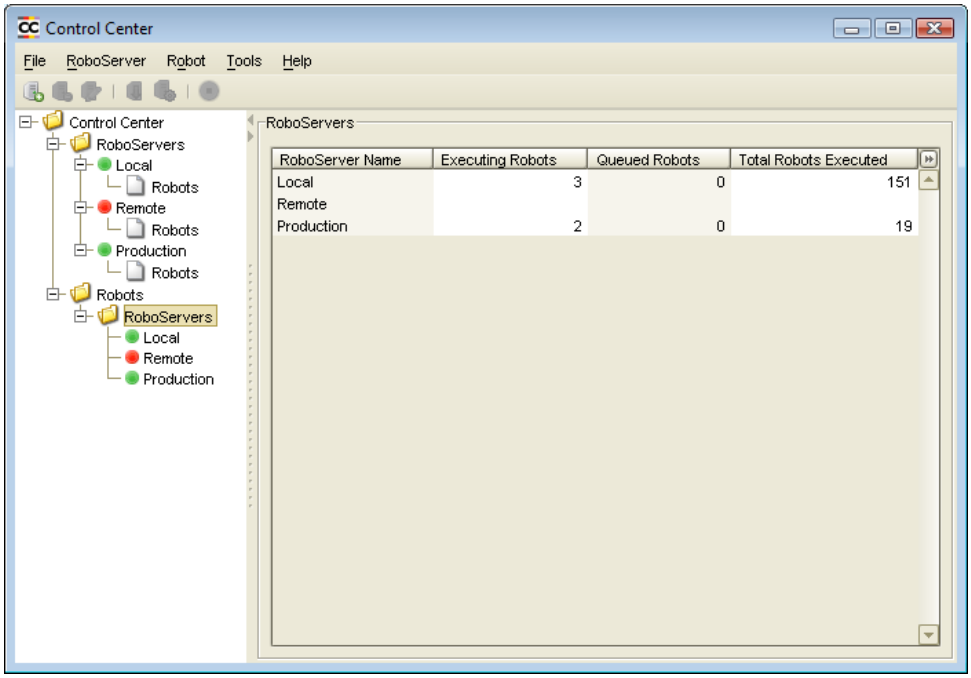
ロボットの監視

ロボットを監視するには、ツリー・ビュー内の「ロボット」カテゴリを選択します。この結果、次に示すように、メイン・ビューにすべての登録済RoboServer上のロボットの概要が表示されます。



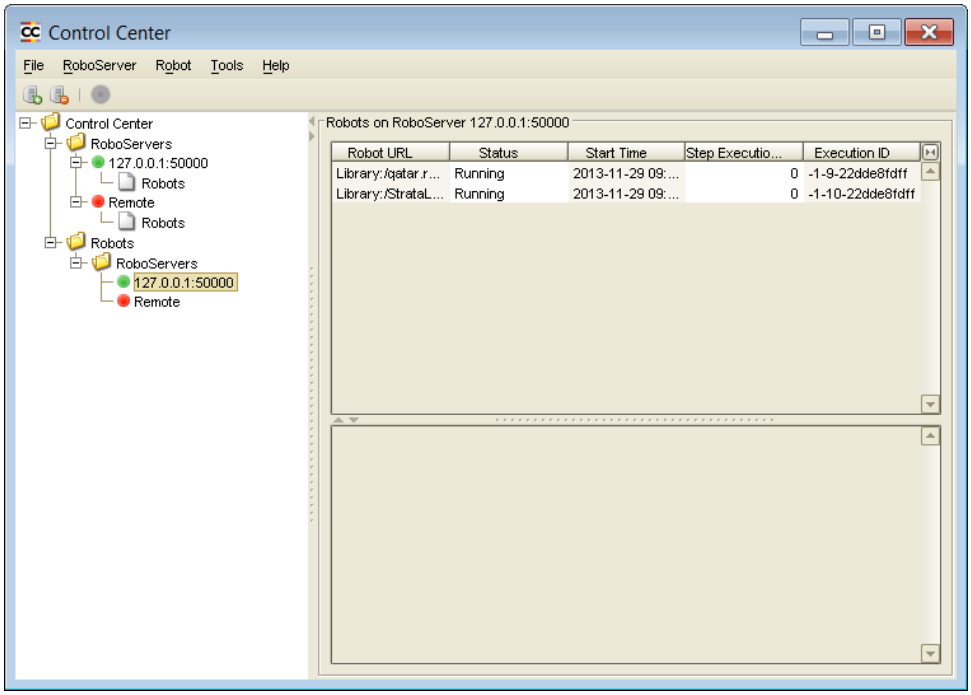
すべてのロボットの監視

RoboServerサブカテゴリを選択すると、メイン・ビューには、個々のRoboServer上のロボットの概要が表示されます。



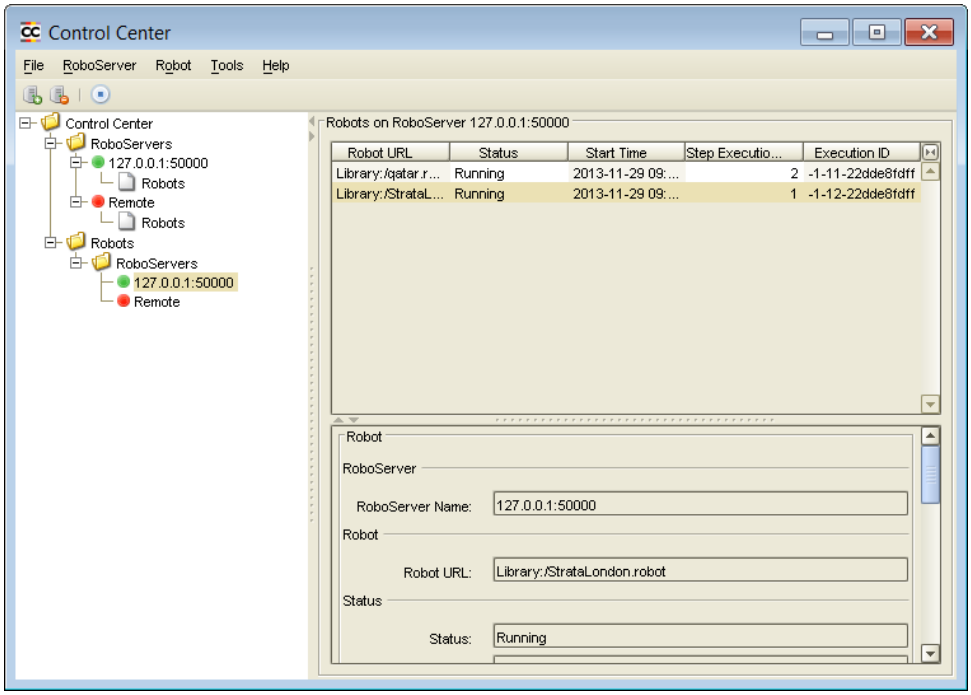
RoboServer上のロボットの監視

ツリー・ビュー内の1つのRoboServerを選択した場合(または表内のRoboServerをダブルクリックした場合)は、メイン・ビューに、この特定のRoboServer上にあるロボットの概要が表示されます。



RoboServer上のロボットの監視

メイン・ビューの表内のロボットを選択すると、この特定のロボットに関する詳細が表の下に表示されます。



ロボットの監視

ロボットの停止

ロボットを停止するには、ロボットを選択して アイコンをクリックします。ロボットの停止の確認を求めるダイアログが表示されます。これを実行する必要があるのは、ロボットがそれ自身で停止しないようにする場合のみであることに注意してください。

複数のロボットを停止するには、メイン・ビューの表でロボットを選択し、前述のとおりに実行するか、または選択したロボットを右クリックしてロボットの停止... を選択します。

特定のRoboServer上のすべてのロボットを停止するには、RoboServerを選択して アイコンをクリックします。

すべてのRoboServer上のすべてのロボットを停止するには、RoboServerカテゴリ、「ロボット」カテゴリまたはRoboServerサブカテゴリを選択して アイコンをクリックします。

その他のControl Centerの機能

この項では、RoboServerおよびロボットの管理に直接関連していないControl Centerの機能を使用する方法について説明します。

RoboServerリストのエクスポートとインポート

Control Centerをシャットダウンするときは必ず、登録済RoboServerのリストが自動的に保存され、Control Centerを再度起動したときに、そのリストが再度ロードされます。

ただし、RoboServerリストを手動でエクスポートおよびインポートすることもできます。RoboServerリストをファイルにエクスポートすると、他のControl Centerインストールにインポートできます。この結果、各インストールでRoboServerを登録する必要がなくなり、かわりに設定を簡単にエクスポートおよびインポートできます。

RoboServerリストのエクスポート

現在のRoboServerリストをエクスポートするには、「ファイル」メニューを開いてRoboServerのエクスポートを選択します。

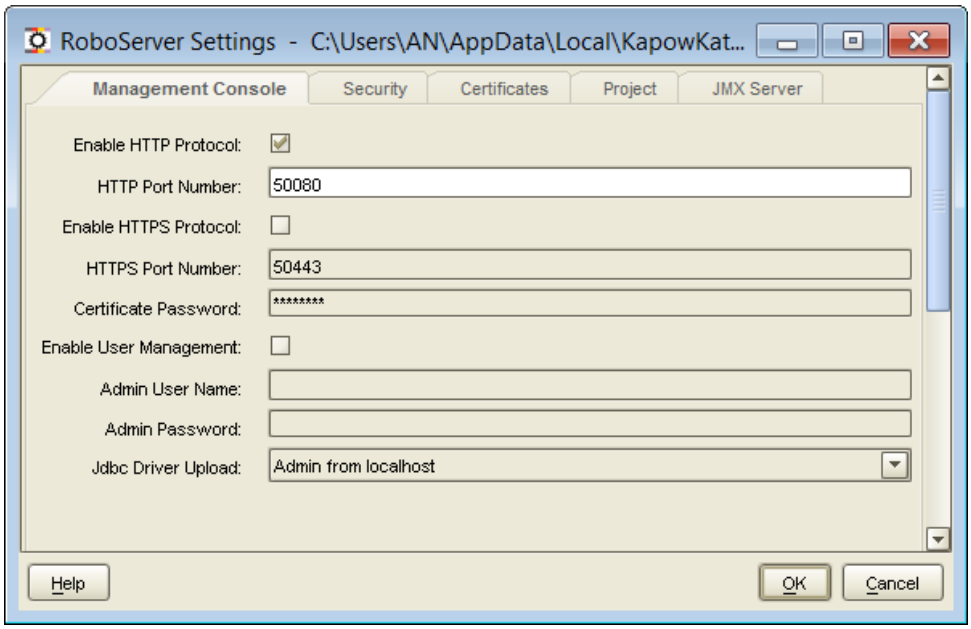
RoboServerリストのインポート

RoboServerリストをインポートするには、「ファイル」メニューを開いてRoboServerのインポートを選択します。

RoboServerリストをインポートする場合は、リスト内でまだ登録されていないすべてのRoboServerを登録します。すでに(おそらく別の名前で)登録されているRoboServerと同じプロトコルを使用するRoboServerは、再度登録はされません。

リモートのRoboServerの設定の編集

Control Centerから、接続済RoboServerの設定を編集するには、  アイコンをクリックします。次に示すようなダイアログが表示されます。



設定の編集

設定の変更後は、変更を有効にするためにRoboServerを再起動する必要があります。

第20章 TomcatでのManagement Console

デフォルトでは、Management Consoleは、RoboServer内部の埋込みコンポーネントとして実行され、簡単にインストールできます。代替手段として、スタンドアロンのTomcat 5.5、6または7のWebサーバーに、通常のWebアプリケーションとしてデプロイできます。次の表に、機能セットの違いをリストします。

表20.1 Management Consoleの機能および構成

機能	埋込み	スタンドアロンJ2SE Webコンテナ
認証	Settingsに定義されたシングル・ユーザー	プロジェクト別のロール・ベース・セキュリティ。ロールはLDAPまたは他のプロバイダを介して取得されます。
Management Consoleのデータ・ストア	埋込みDerbyデータベース	コンテナ管理のデータベース・ソース(サポートされているプラットフォーム)

埋込みManagement Consoleの構成に関する説明は、「RoboServerの構成」の項にあります。埋込みManagement Consoleは、「Management Consoleの起動」の説明に従って開始します。

Tomcat WebコンテナへのManagement Consoleのデプロイに関する詳細は、次の各ページを参照してください。

アップグレード時の注意

Management Consoleの以前のバージョンを実行していた場合は、この項で9.3.0へのアップグレード方法について説明します。

アップグレードには次のステップが含まれます。

- Management Consoleの以前のバージョンからのデータのバックアップの作成
- Management Consoleのインストール
- Management Console構成の更新
- バックアップの復元

バックアップの作成

以前のバージョンのManagement Consoleでオンライン・ドキュメントを確認し、バックアップの作成方法を学習します。 9.2 9.1 9.0 8.2 8.1 8.0 7.2 7.1 7.0

Management Consoleのインストール

このガイドの次の各ステップに従って、Management Consoleのインストール方法を学習します。8.1以降からアップグレードする場合は、古い**Configuration.xml**を再利用できます(次の項を参照)。

Management Console構成のアップグレード

バージョン8.0以前からアップグレードする場合は、新しいManagement Consoleを最初から再構成する必要があります。バージョン8.1より前は、一部の構成が**web.xml**で行われていましたが、これはすでに当てはまりません。**web.xml**をコピーまたは変更できない可能性があります。

ほとんどの構成は、**Configuration.xml**で行われます。このファイルは、以前のバージョンからコピーできます。以前のバージョンの**Configuration.xml**を**WEB-INF/**にコピーして、すでにあるバージョンを上書きします。Management Consoleを起動すると、古いバージョンに基づいて新

しい**Configuration.xml**が作成されるため、その後、アップグレードされたバージョンを**WEB-INF/**にコピーして、古い構成を上書きする必要があります。Management Console Webインタフェースにアクセスすると、指示が表示されます。

バージョン8.2現在、LDAP統合はすでにコンテナ管理ではありません。以前のバージョンで認証にLDAPを使用していた場合は、Tomcatの**conf/Catalina/localhost/**にある**ManagementConsole.xml**で、**Realm**定義を削除してください。「LDAP統合」で説明されているように、LDAP統合は現在**WEB-INF/login.xml**で構成されます。

バックアップの復元

バックアップの復元方法の詳細は、「Management Consoleユーザーズ・ガイド」を参照してください。

Tomcatへのデプロイ

ここでは、Management ConsoleをスタンドアロンJ2SE Webコンテナ上にインストールする方法について説明します。Tomcatを採用したこのガイドでは、Tomcat 5.5、6.0および7.0で配布をテストしました。J2SE Webコンテナには、Java 1.6ランタイム以降を使用する必要があります。

ManagementConsole.warの構成

Management Consoleアプリケーションは、**ManagementConsole.war**という名前のWeb Application Archive (WARファイル)の形式で提供され、Kapow Katalyst インストール・フォルダの**/WebApps**にあります。

Kapow Katalystに付属しているManagementConsole.warのバージョンは、RoboServerに埋め込まれた状態で実行するように構成されているため、Tomcatのスタンドアロン・アプリケーションとしてデプロイするには、環境にあわせて再構成する必要があります。

WARファイルはzipを使用して圧縮されているため、構成ファイルにアクセスするには取り出す(解凍する)必要があります。構成ファイルを更新した後は、ManagementConsole.warを再度zipしてTomcatサーバーにデプロイします。

次の表に、解凍したManagementConsole.warのルートから相対的に構成ファイルのリストを示します。

表20.2 構成ファイル

ファイル	構成	注意
WEB-INF/ Configuration. xml	クラスタリング、パスワード暗号化、RESTプラグイン	ファイルのバージョンを8.1からコピーした場合は、Management Consoleを起動したときにファイルが自動的にアップグレードされます。
WEB-INF/login.xml	管理者およびユーザー(これをLDAPと統合します)	
WEB-INF/roles.xml	様々なアプリケーション・ロールの権限	
WEB-INF/ classes/ log4j. properties	アプリケーション・ロギング	

Spring構成ファイル

Configuration.xml、**login.xml**および**roles.xml**はすべてSpring構成ファイル(www.springsource.org)で、ここに記載する同じ一般構文を共有します。

Springは、一連のBeanを介して構成され、各Beanにはアプリケーション内のコード部分を構成する多数のプロパティがあります。一般構文は、次のとおりです。

```
<bean id="id" class="SomeClass">
<property name="myName" value="myValue"/>
</bean>
```

表20.3 パート

パート	構成	
id="id"	BeanのIDは内部ハンドルで、アプリケーションがBeanを参照する際に使用します。Bean名とも呼ばれます。	
class= "SomeClass"	クラスは、Beanが構成するコード・コンポーネントを識別します。	
<property name="myName" value= "myValue"/>	名前**myName**と値**myValue**でプロパティを定義します。クラス属性によって定義されたコード・コンポーネントのプロパティを構成します。	

たとえば、**login.xml**に定義されている管理ユーザーを変更する場合、変更する必要があるSpring Beanの唯一のパートは、**<property name="myName" value="myValue"/>**の値属性です。

```
<bean class="com.kapowtech.mc.config.HardCodedUser">
<property name="userName" value="admin"/>
<property name="password" value="admin"/>
<property name="groups" value="ADMINISTRATOR"/>
</bean>
```

userName、passwordおよびgroupsプロパティの値を変更し、異なるユーザー名、パスワードおよびグループ関連付けでユーザーを定義できます。

login.xml内のハードコードされたユーザーのリストのように、Beanのプロパティ値が、ネストされたBeanのリストになることもあり、その場合は次のように定義されます。

```
<bean id="hardCoded" class="com.kapowtech.mc.config.HardCodedLogin"
lazy-init="true">
<property name="users">
<list>
<bean class="com.kapowtech.mc.config.HardCodedUser">
<property name="userName" value="admin"/>
<property name="password" value="admin"/>
<property name="groups" value="ADMINISTRATOR"/>
</bean>
<bean class="com.kapowtech.mc.config.HardCodedUser">
<property name="userName" value="dev"/>
<property name="password" value="dev"/>
<property name="groups" value="DEVELOPER"/>
</bean>
<-- More beans may be here -->
</list>
</property>
</bean>
```

ハードコードされたユーザーがさらに必要な場合は、リスト内の既存の**HardCodedUser** Beanをコピーして、userName、passwordおよびgroupsプロパティの値を変更することで追加できます。ネストされたBeanにはIDがないことがよくあります。リストには、**login.xml**の**adminGroups** Beanのような単純な値も指定できます。

```
<bean id="adminGroups" class="com.kapowtech.mc.config.AdminGroups">
<property name="adminGroups">
<list>
<value>ADMINISTRATOR</value>
</list>
</property>
</bean>
```

これは、各値がテキスト文字列のみの単純なリストを定義します。

トラブルシューティング

インストール時に問題が発生した場合は、Tomcatインストールの**/logs**フォルダにあるTomcatログを確認してください。構成プロセスでは、後でコマンドライン・ウィンドウでエラー・メッセージを直接印刷することになるため、多くの場合はコマンドラインからTomcatを実行するほうが簡単です。

新しいデータベースの作成

Management Consoleで使用される表のために、新しいデータベースを作成することをお薦めします。データベースに対する要件は2つあります。

- Unicodeサポート
- 大/小文字を区別する比較

デンマーク語のÆ、ドイツ語のß、キリル文字の#などのASCII以外の文字がロボットへの入力として指定される可能性があるため、Unicodeサポートが必要です。この入力はデータベースに格納されるため、Unicodeサポートがないと、これらの文字が誤って格納される可能性があります。

a.robotというロボットと**A.robot**という別のロボットをアップロードできるため、大/小文字を区別する比較が必要です。大/小文字を区別する比較がない場合は、後者をアップロードすると前者が上書きされます。

データベース・サーバーでは、Unicodeと大/小文字を区別する比較に、大きく異なる方法で対応します。次のリストに、サポートされるデータベース・システムに対する推奨事項を示します。

表20.4 Unicodeサポートおよび大/小文字を区別する比較に対する推奨事項

データベース	推奨事項
IBM DB2	**CODESET UTF-8**を使用してデータベースを作成します。
MySQL 5.x	**utf8_bin**照合付きでデータベースを作成します。 **CREATE DATABASE KAPOW_MC COLLATE utf8_bin**
Oracle	Unicodeに**NVARCHAR2**、**NCLOB**タイプを使用します。大/小文字を区別して比較するには、**NLS_COMP**を**BINARY**に設定します。
Microsoft SQL Server 2005/2008	Unicodeに**NVARCAR**、**NTEXT**タイプを使用します。大/小文字を区別して比較するには、**SQL_Latin1_General_CP1_CS_AS**などの大/小文字を区別する照合付きでデータベースを作成します。 **CREATE DATABASE KAPOW_MC COLLATE SQL_Latin1_General_CP1_CS_AS**
Sybase Adaptive	Unicodeに**NVARCAR**、**NTEXT**タイプを使用します。大/小文字を区別して比較するには、ソート順をバイナリにします(**sp_helpsort**を参照)。

データベース	推奨事項
Server Enterprise	NVARCHAR(200) NOT NULL UNIQUEを使用するようにスクリプトを変更(または4/8/16Kページ・サーバーにインストール)しないと、索引制限によって2Kページ・サイズのASEでは機能しないため、後述のスクリプトではNVARCHAR(255) NOT NULL UNIQUEが使用されます。

Management Consoleで使用される表は、プラットフォーム表、ロギング表およびデータ・ビュー表の3つのカテゴリに分類されます。プラットフォーム表にはアップロードされたロボットやそのスケジュール情報などのManagement Console専用の情報が格納され、ロギングおよびデータ・ビューの表はRoboServerと共有されます。

ユーザー権限

Management Consoleを起動すると、必要なプラットフォーム表およびロギング表の作成が自動的に試行されます(RoboServerによって作成されていない場合)。これは、データベースのアクセスに使用するユーザー・アカウントに、CREATE TABLEおよびALTER TABLE権限が必要であることを意味します。Oracleユーザーには、CREATE SEQUENCE権限も必要です。作成できない場合は、次のスクリプトを使用して表を作成するようにデータベース管理者に依頼します。

さらに、正常に機能するためには、システムに対するSELECT、INSERT、UPDATE、DELETEがユーザーに許可されている必要があります。

Management Console表に対するSQLスクリプト

表20.5 Management Console表に対するSQLスクリプト(右クリックして「名前を付けて保存」を選択)

データベース	表を作成するスクリプト	表を削除するスクリプト
IBM DB2	create	drop
MySQL 5.x	create	drop
Oracle	create	drop
Microsoft SQL Server 2005/2008	create	drop
Sybase Adaptive Server	create	drop

Management Consoleでは、Quartzというサード・パーティ・スケジューリング・コンポーネントを使用します。また、Quartzは複数の表を必要とし、それらの表は他のプラットフォーム表に存在する必要があります。これらの表は、Management Consoleの起動時に自動的に作成されますが、次のスクリプトを使用して手動で作成することもできます。

Quartz表に対するSQLスクリプト

表20.6 Quartz表に対するSQLスクリプト(右クリックして「名前を付けて保存」を選択)

データベース	表を作成するスクリプト	表を削除するスクリプト
IBM DB2	create	drop
MySQL 5.x	create	drop
Oracle	create	drop
Microsoft SQL Server 2005/2008	create	drop
Sybase Adaptive Server	create	drop

Tomcatコンテキスト・ファイルの作成

エンタープライズ環境では、データ・ソースを介してデータベースがアクセスされることがよくあります。このガイドはで、Tomcatの構成方法について、ローカルMySQLデータベース・サーバーに接続するデータ・ソースの構成方法とともに示します。

Tomcatでは、データ・ソースは、アプリケーション・コンテキスト内で定義します。コンテキストは、アプリケーションへの埋込みまたはアプリケーション外部のいずれかで宣言できます。コンテキストが埋込みの場合は、**context.xml**ファイルに定義し、そのファイルを**META-INF**フォルダのWARファイル内に格納する必要があります。外部的に宣言する場合は、ファイルをTomcatの**/conf/Catalina/localhost**フォルダに配置し、ファイル名をManagementConsole.xml（デプロイされるWARファイルと同じ名前)にする必要があります。Tomcatでは、単一のデプロイメント単位を提供しているため、埋込みコンテキストを使用したデプロイを奨励していますが、このガイドでは、ファイルの変更が容易であるため、外部コンテキスト定義を使用します。構成を修正した後は、コンテキスト・ファイルを埋め込み、Warファイルを本番環境にデプロイします。

プラットフォーム・データ・ソースの追加

Tomcatの**/conf/Catalina/localhost**フォルダに**ManagementConsole.xml**ファイルを作成し、次の内容を追加します。

```
<Context>
<!-- Default set of monitored resources -->
<WatchedResource>WEB-INF/web.xml</WatchedResource>
<Resource name="jdbc/kapow/platform" auth="Container"
type="javax.sql.DataSource" maxActive="100" maxIdle="30" maxWait="-1"
validationQuery="/* ping */" testOnBorrow="true" username="MyUser"
password="MyPassword" driverClassName="com.mysql.jdbc.Driver"
url="jdbc:mysql://localhost:3306/KAPOW_MC?useUnicode=
yes&amp;characterEncoding=UTF-8&rewriteBatchedStatements=true"/>
</Context>
```

前述の**url**パラメータはJDBC URLです。**username**および**password**属性は、データベースへの接続時に使用する接続プールを作成するために、Tomcatによって使用されます。

データ・ソースは、他のデータベースとは別に定義します。たとえば、Microsoft SQL Server 2005/2008を使用している場合は、前述の関連する3行のかわりに、次のように指定してください。

```
username="MyUser" password="MyPassword" driverClassName=
"com.microsoft.sqlserver.jdbc.SQLServerDriver"
validationQuery="SELECT 1" testOnBorrow="true"
url="jdbc:sqlserver://localhost:1433;DatabaseName=MyDbName"/>
```

Microsoft SQL Serverを使用している場合は、混合モード認証を使用するように構成する必要があります。一般に、データ・ソースで使用する値を識別するには、JDBCのドキュメントを参照してください。

URL **jdbc:mysql://localhost:3306/KAPOW_MC?useUnicode=yes&characterEncoding=UTF-8**では、ローカルMySQLの**KAPOW_MC**というデータベースが参照されます。MySQLの場合は、すべての接続文字列に**?useUnicode=yes&characterEncoding=UTF-8**を追加することをお薦めします。そうしないと、JDBCドライバで中国語、日本語またはその他UTF-8の3バイト文字が正しく処理されません。これは、コンテキストxmlファイル内では&をそのまま使用できないため、&としてエンコードする必要があることに起因しています。

rewriteBatchedStatements=trueは、MySQL JDBCドライバに対してバッチ挿入/更新を指示し、これによってkappletロボットの挿入パフォーマンスが向上します。

driverClassNameパラメータは、使用するJDBCドライバを制御します。各データベース・ベンダーからそのデータベースに対応するJDBCドライバが提供され、使用するドライバをダウンロードする必要があります。JDBCドライバ(通常は単一の.jarファイル)は、Tomcat 6/7の場合は/libフォルダに、Tomcat 5.5の場合はcommons/libフォルダにコピーする必要があります。

validationQueryは、接続プールから取得した接続が(データベース・サーバーが接続をクローズした可能性があるため)引き続き有効かどうかを検証するために、Tomcatによって使用されます。検証の問合せは軽量で、データベース・サーバーのリソースをほとんど使用しません。次のリストに、サポートされるデータベースに対する検証の問合せを示します。

表20.7 検証の問合せ

データベース	問合せ
MySQL	/* ping */
Microsoft SQL Server 2005/2008	SELECT 1
Sybase Adaptive Server Enterprise	SELECT 1
IBM DB2	VALUES(1)
Oracle	SELECT 1 FROM DUAL

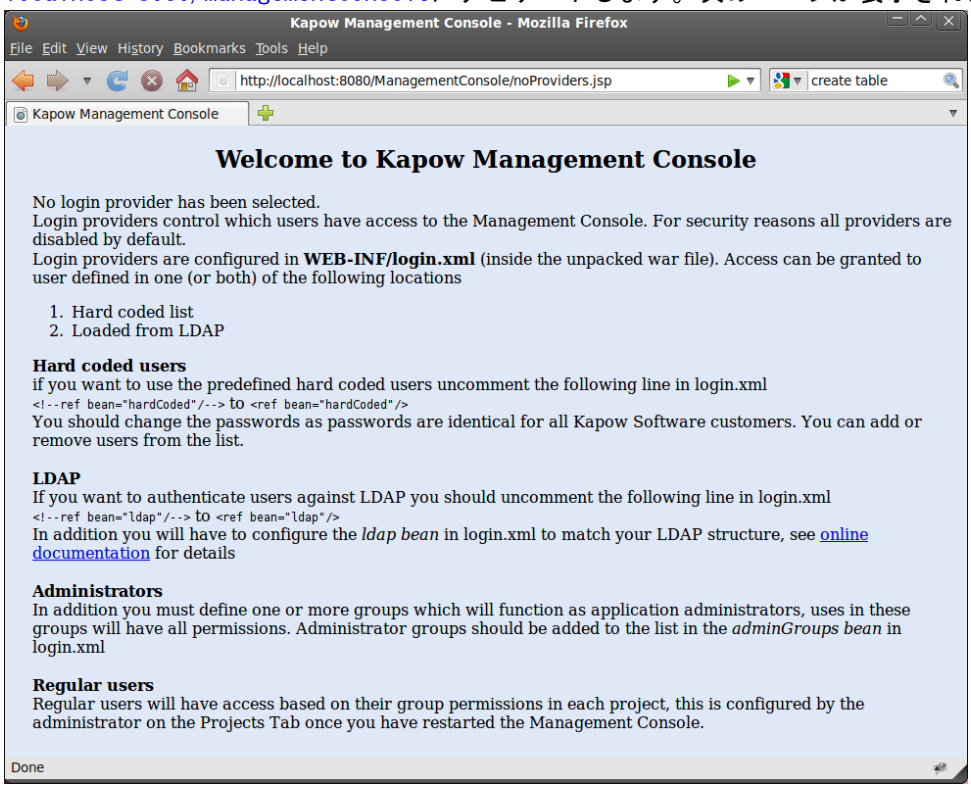
MySQL JDBCドライバには、特に軽量な**/* ping */**リクエストがサポートされています(詳細は、JConnectorマニュアルの第6.1項を参照)。

コンテキスト構成およびデータ・ソースの詳細は、『JNDI Resources HOW-TO』および『JNDI data source HOW-TO』を参照してください。

Tomcatサーバーを起動する準備が整いました。

Tomcatの起動

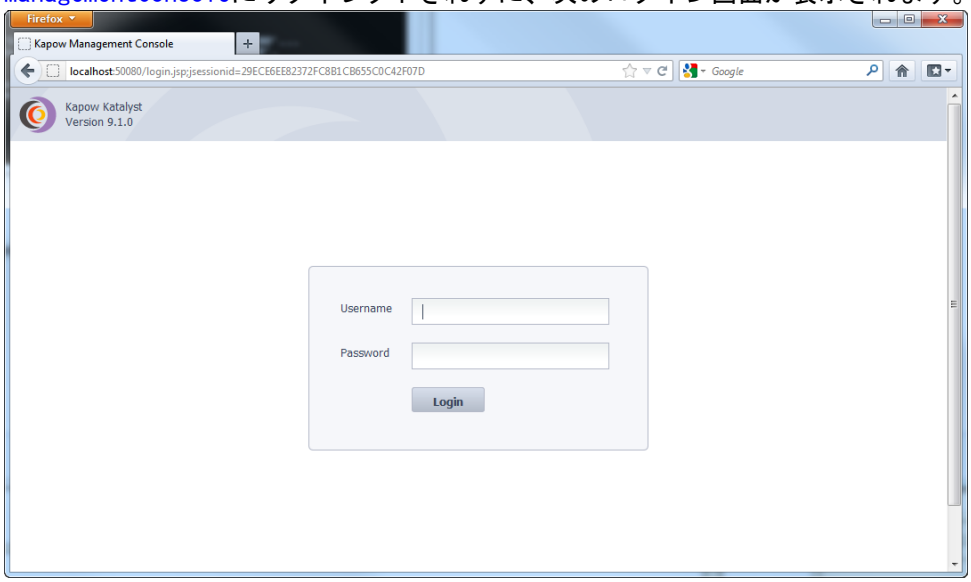
Tomcatサーバーを起動し、アプリケーションがデプロイされるまで数秒間待機して、http://localhost:8080/ManagementConsoleにナビゲートします。次のページが表示されます。



プロバイダなし

このガイドでは、事前定義のハードコードされたユーザーを使用します。**WEB-INF/login.xml**に移動して、**<!--ref bean="hardCoded"/-->**のコメントを消去します。その後、Tomcatを再起動します。

再起動後、[F5]を押してブラウザをリフレッシュすると、http://localhost:8080/ManagementConsoleにリダイレクトされずに、次のログイン画面が表示されます。

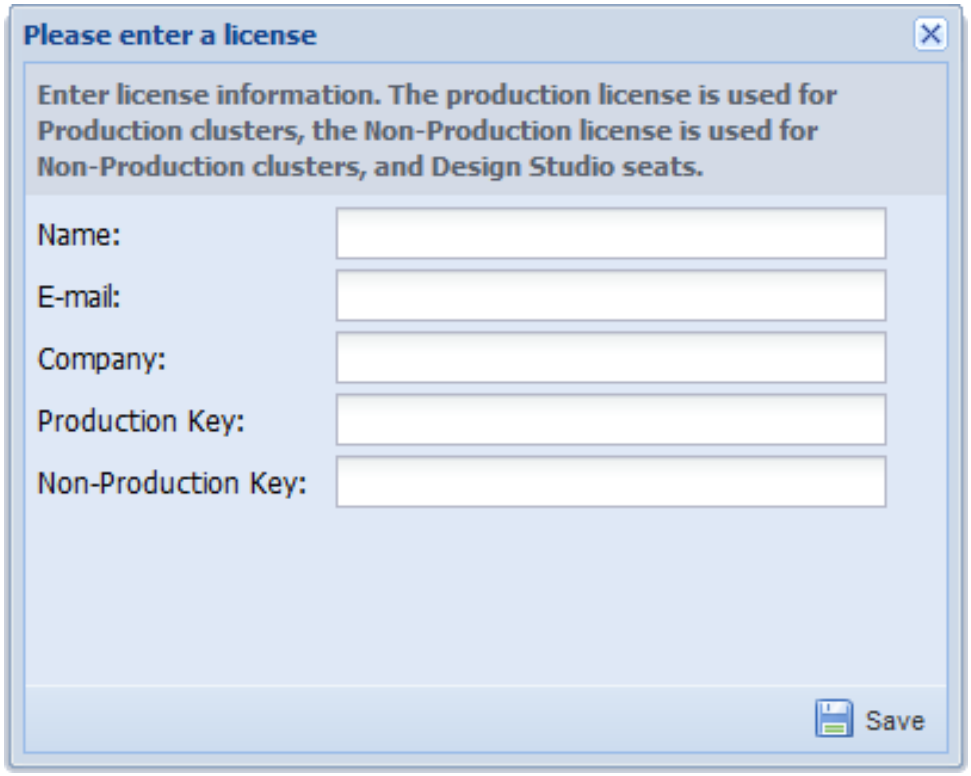


ログイン・ページ

ユーザー名とパスワードにadminと入力し、「ログイン」ボタンをクリックします。

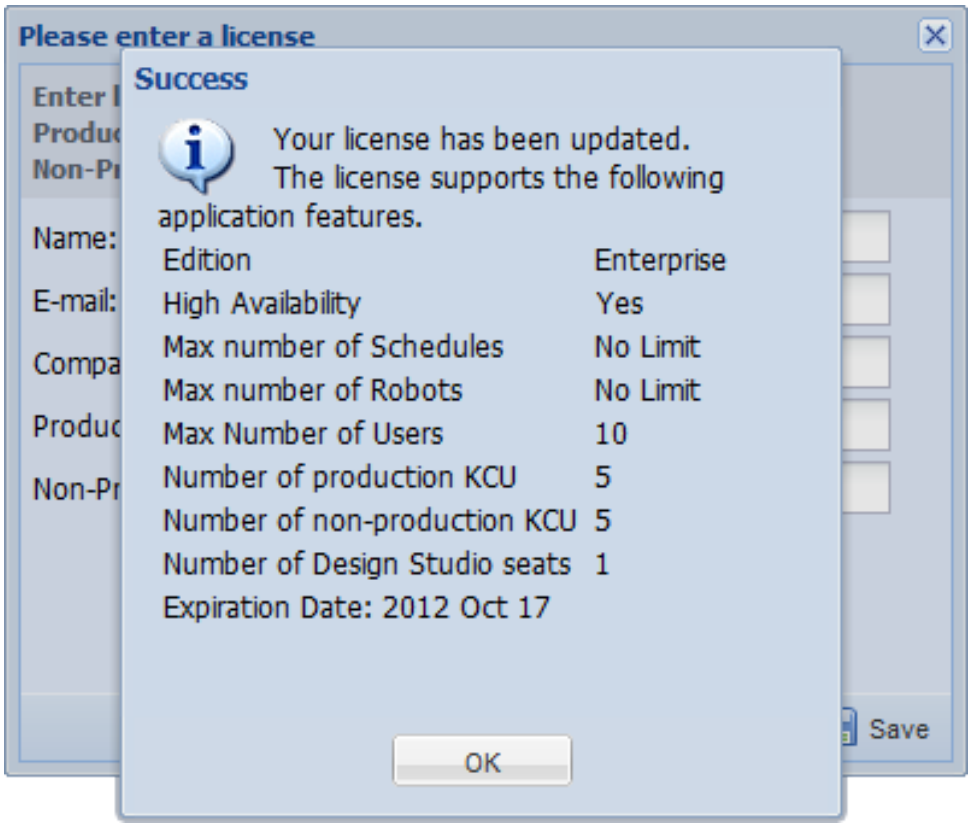
ライセンスの入力

ログインすると、次のダイアログが表示されます。



ライセンスなし

Kapow Katalystのライセンス情報を入力し、「保存」ボタンをクリックします。次のポップアップが開き、ライセンス・キーで使用可能な機能が表示されます。



更新されたライセンス

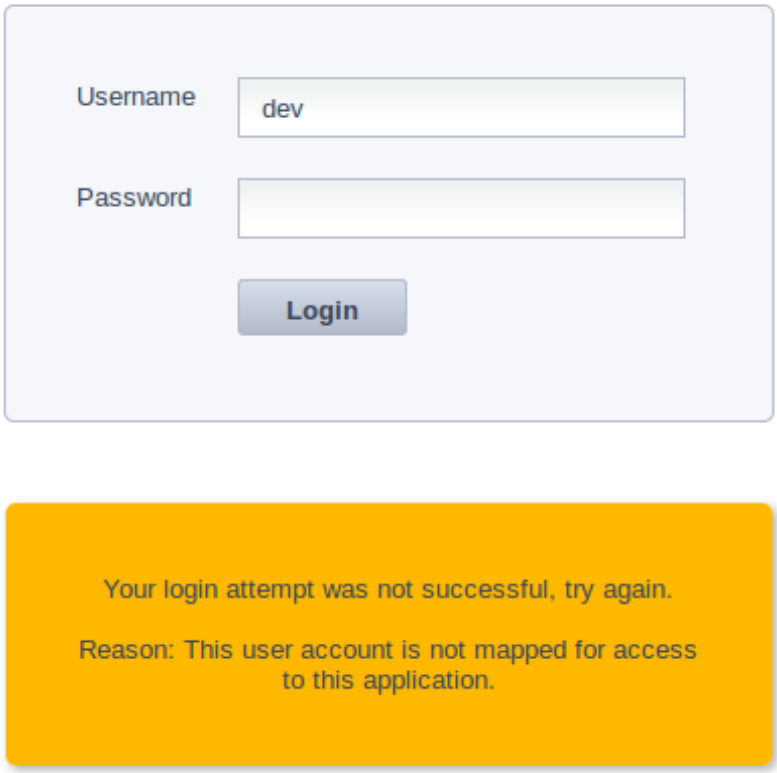
プロジェクトの権限

ログインしたadminユーザーは、標準的なユーザーに適用される通常のプロジェクト権限を無視します。adminユーザーは、ADMINISTRATORというグループ名のメンバーです。このグループ名は、**adminGroups**のリストの値の1つとしてリストされています。

```
<bean id="adminGroups" class="com.kapowtech.mc.config.AdminGroups">
<property name="adminGroups">
<list>
<value>ADMINISTRATOR</value>
</list>
</property>
</bean>
```

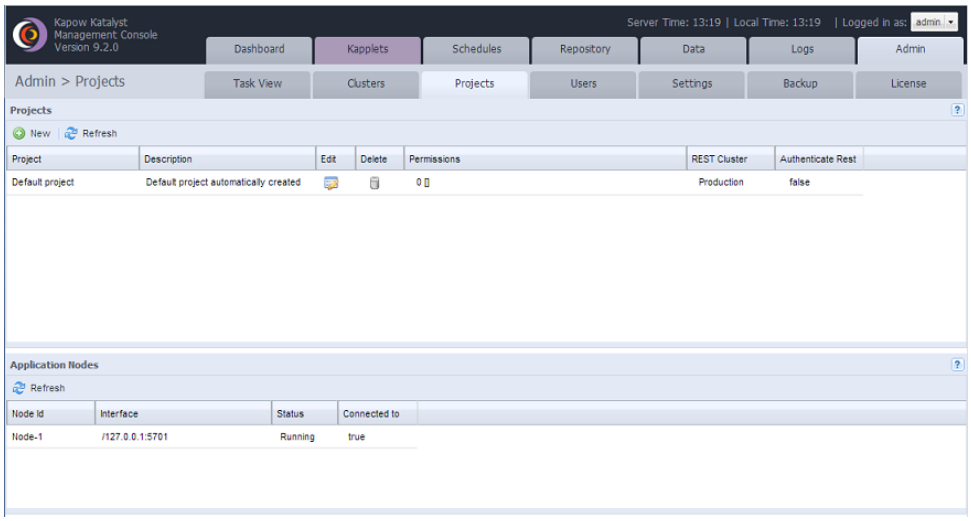
ここにリストされているグループは管理者グループとみなされ、これらのグループのメンバーは常にログイン可能で、すべてのアクションを実行する権限があります。バックアップのエクスポートと復元、およびライセンス・キーの変更を実行できる唯一のユーザーです。

devユーザー(パスワードdev)でログインした場合は、次のページが表示されます。



マップされていないユーザー
管理者でないユーザーの場合、権限はユーザーのグループ・メンバーシップに基づいています。devユーザーはDEVELOPERグループのメンバーで、DEVELOPERグループのメンバーにはプロジェクトにアクセスする権限が付与されていないため、ログインできません。

devユーザーにログインを許可するには、管理者としてログインし、「管理」タブのサブタブである「プロジェクト」タブに移動する必要があります。次のように表示されます。

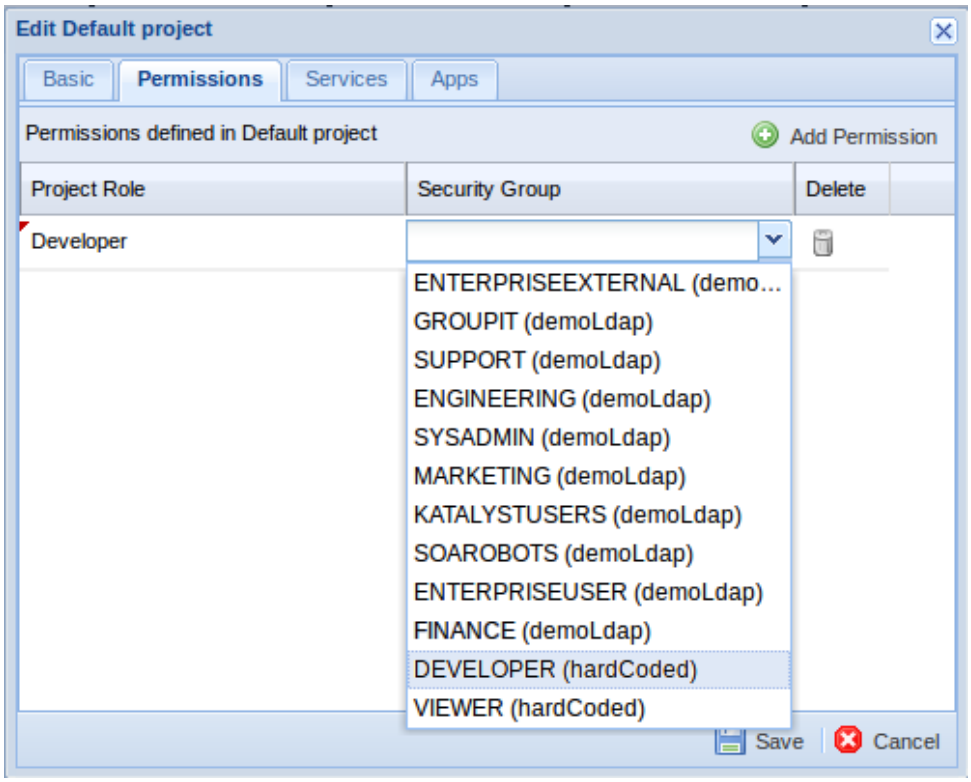


当初のプロジェクト権限
「権限」列に表示されているように、0の権限マッピングがあります。

「編集」をクリックして「プロジェクトの編集」ダイアログを開き、「権限」タブに移動します。グリッドにプロジェクト・ロールおよびセキュリティ・グループ列が表示されます。プロジェクト・ロールによって、ロボットのアップロード、スケジュールの作成、ログの表示など(この詳細は後述を参照)、Management Console内で実行できる一連のアクションが決まります。プロジェクトの範囲内で、プロジェクト・ロールをセキュリティ・グループに割り当てます。このようにして、選択した

セキュリティ・グループのすべてのユーザーは、割り当てられたプロジェクト・ロールで許可されるアクションを実行できるようになります。

権限の追加ボタンをクリックして、このプロジェクトでの権限を追加します。これによって、グリッドに新しい行が追加され、プロジェクト・ロールを選択できるドロップダウンが挿入され、「開発者」プロジェクト・ロールが選択されます。セキュリティ・グループ列をダブルクリックすると、ドロップダウンでDEVELOPERセキュリティ・グループ(devユーザーがメンバー)を選択できるようになります。次のように表示されます。



プロジェクト権限の追加

「保存」をクリックします。

DEVELOPERグループのすべてのメンバーは、開発者ロールに許可されているアクションを実行できます。すべてのロールは**WEB-INF/roles.xml**ファイルで定義します。roles.xmlで定義したロールは、各タブの下位レベル権限を定義します。次に、開発者ロールの例を示します。

```
<bean class="com.kapowtech.scheduler.client.auth.Role">
<property name="roleName" value="Developer"/>
<property name="description"
value="A user than can edit/delete schedules, robots, types,
snippets and resources, but only view logs and data"/>
<property name="permissions">
<bean class="com.kapowtech.scheduler.client.auth.Permissions">
<property name="dashBoardPermissions">
<bean class=
"com.kapowtech.scheduler.client.auth.DashBoardPermissions">
<property name="viewDashboard" value="true"/>
</bean></property>

<property name="schedulesTabPermissions">
<bean class=
"com.kapowtech.scheduler.client.auth.SchedulesTabPermissions">
<property name="viewSchedules" value="true"/>
<property name="editSchedules" value="true"/>
```

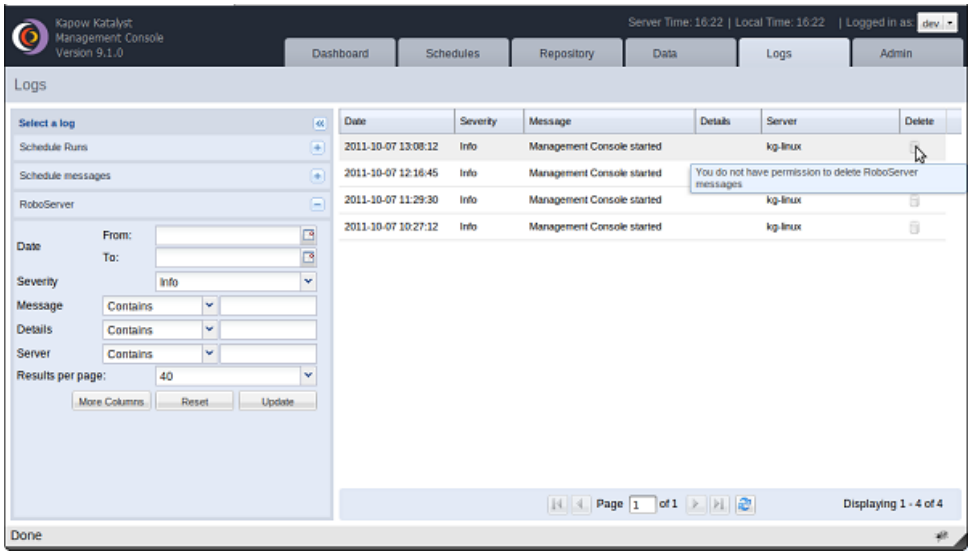
```
<property name="deleteSchedule" value="true"/>
<property name="startSchedules" value="true"/>
<property name="stopSchedules" value="true"/>
</bean></property>

<property name="robotsTabPermissions">
<bean class=
"com.kapowtech.scheduler.client.auth.RobotsTabPermissions">
<property name="viewRobots" value="true"/>
<property name="addRobot" value="true"/>
<property name="deleteRobot" value="true"/>
<property name="downloadRobot" value="true"/>
<property name="runRobot" value="true"/>
<property name="generateAPICode" value="true"/>
</bean></property>

<property name="typesTabPermissions">
<bean class=
"com.kapowtech.scheduler.client.auth.TypesTabPermissions">
<property name="viewTypes" value="true"/>
<property name="addType" value="true"/>
<property name="deleteType" value="true"/>
<property name="downloadType" value="true"/>
</bean></property>

<property name="snippetsTabPermissions">
<bean class=
"com.kapowtech.scheduler.client.auth.SnippetsTabPermissions">
<property name="viewSnippets" value="true"/>
<property name="addSnippet" value="true"/>
<property name="deleteSnippet" value="true"/>
<property name="downloadSnippet" value="true"/>
</bean></property>

<property name="resourcesTabPermissions">
<bean class=
"com.kapowtech.scheduler.client.auth.ResourcesTabPermissions">
<property name="viewResources" value="true"/>
<property name="addResource" value="true"/>
<property name="deleteResource" value="true"/>
<property name="downloadResource" value="true"/>
</bean></property>

<property name="OAuthTabPermissions">
<bean class=
"com.kapowtech.scheduler.client.auth.OAuthTabPermissions">
<property name="OAuthTab" value="true"/>
</bean></property>

<property name="dataViewPermissions">
<bean class=
"com.kapowtech.scheduler.client.auth.DataViewPermissions">
<property name="viewData" value="true"/>
<property name="deleteData" value="false"/>
<property name="exportData" value="false"/>
</bean></property>

<property name="logTabPermissions">
```

```
<bean class=
"com.kapowtech.scheduler.client.auth.LogTabPermissions">
<property name="viewScheduleRunLog" value="true"/>
<property name="deleteScheduleRunLog" value="false"/>
<property name="viewScheduleMessageLog" value="true"/>
<property name="deleteScheduleMessageLog" value="false"/>
<property name="viewServerLog" value="true"/>
<property name="deleteServerMessage" value="false"/>
<property name="viewRobotRunLog" value="true"/>
<property name="deleteRobotRun" value="false"/>
<property name="viewRobotMessageLog" value="true"/>
<property name="deleteRobotMessage" value="false"/>
<property name="viewRobotsLog" value="true"/>
<-- May view robot_run/robot_messages that are started by the API, and
therefore have no project name or the project name is
one that doesn't exists in MC -->
<property name="viewOrphanProjects" value="false"/>
</bean></property>

<property name="taskViewTabPermissions">
<bean class=
"com.kapowtech.scheduler.client.auth.TaskViewTabPermissions">
<property name="viewTasks" value="true"/>
<property name="stopTask" value="false"/>
</bean></property>

<property name="clustersTabPermissions">
<bean class=
"com.kapowtech.scheduler.client.auth.ClustersTabPermissions">
<property name="viewClusters" value="true"/>
<property name="addCluster" value="false"/>
<property name="renameCluster" value="false"/>
<property name="deleteCluster" value="false"/>
<property name="addServer" value="false"/>
<property name="deleteServer" value="false"/>
<property name="setClusterMode" value="false"/>
<property name="changeClusterSettings" value="false"/>
</bean></property>

<property name="projectsTabPermissions">
<bean class=
"com.kapowtech.scheduler.client.auth.ProjectsTabPermissions">
<property name="viewProjects" value="false"/>
<property name="editProject" value="false"/>
<property name="deleteProject" value="false"/>
</bean></property>

<property name="appPermissions">
<bean class=
"com.kapowtech.scheduler.client.auth.AppPermissions">
<!-- user may install/schedule/customize kapplets.
NOTE: user also need the 'runRobot' permission to actually execute
the kapplet -->
<property name="kappletUser" value="true"/>
<!-- user may build/delete/edit (master) kapplets -->
<property name="kappletAdministrator" value="true"/>
</bean></property></bean>
```

```
</property></bean>
```

すべての権限は、単純にtrueまたはfalseで定義します。プロパティ値がtrueの場合は、そのロールに属するユーザーは、プロパティ名で表されたアクションを実行できます。**logTabPermissions**に注目すると、開発者は、すべてのログのデータを表示できますが、ログ・メッセージを削除できないことがわかります。

devユーザーとしてログインし、Management Consoleで権限が反映されている様子を確認してください。右上隅のメニュー・ボタンをクリックしてログアウトし、次にdevユーザーとしてログインします。ログ・タブに移動して、左側のリボン・タブでRoboServerログを選択します。「削除」ボタンが使用不可で、マウス・ポインタを重ねるとツールチップ・メッセージが表示される様子に注目してください。



ログの表示とログの削除不可

複数のロールを同じセキュリティ・グループに割り当てることも、同じロールを複数のセキュリティ・グループに割り当てることもできます。複数のロールを保持しているユーザーは、複数のロールの少なくとも1つで許可されている場合は実行できます。Management Consoleに複数のプロジェクトがある場合は、グループをプロジェクト固有のロールに割り当てることで、異なるプロジェクトのユーザーを完全に分離できます。

事前定義のロールが提示されますが、別のロールをいくつでも追加でき、既存のロールをニーズに応じて変更できます。

「設定」、バックアップおよび「ライセンス」タブ(「管理」のサブタブ)で実行可能なアクションは、**adminGroup**のメンバーであるユーザーのみが使用できます。

セキュリティ

Configuration.xmlファイルでは、Management Consoleに対して、いくつかの追加のセキュリティ設定を構成できます。

ファイルのセキュリティ構成セクションは、次のようなものです。

```
<bean id="securityConfiguration"
class="com.kapowtech.mc.config.SecurityConfiguration">
<property name="allowScriptExecution" value="false"/>
<property name="jdbcDriverUpload" value="LOCALHOST"/>
</bean>
```

次に示す2つのセキュリティ設定があります。

スケジュールの一環として、ユーザーは前後処理(たとえば、スケジュールのロボットの実行前後)に使用されるスクリプトを構成できます。これらのスクリプトは、Management Consoleコンピュータのファイル・システムに配置する必要があります。セキュリティ上の理由で、外部スクリプトの実行は、デフォルトでは使用不可になります。外部スクリプトの実行を許可するには、allowScriptExecutionプロパティの値をtrueに変更してください。

スケジュールで前後処理スクリプトを使用し、スクリプトが管理者によって使用不可にされている場合、ロボットは実行されずに、スクリプトは管理者によって使用不可にされています。というメッセージで即時に失敗となります。機能が使用不可の場合も、ユーザーはスケジュールを前後処理スクリプトとともに保存できます。注意: 外部スクリプトが使用不可の場合も、前後処理の一環として擬似スクリプト・コマンド「runrobot: test.robot」を引き続き使用できます。

デフォルトでは、localhostからManagement Consoleにアクセスしている管理ユーザーのみが、JDBCドライバをアップロードできます。この動作を変更するには、jdbcDriverUploadプロパティを変更します。プロパティに使用可能な値は、次のとおりです。

- NONE: どのユーザーにもJDBCドライバのアップロードを許可しない
- LOCALHOST: (デフォルト値) localhostからManagement Consoleにアクセスしている場合は、管理ユーザーにドライバのアップロードを許可する
- ANY_HOST: どのホストからでも管理ユーザーにドライバのアップロードを許可する

デプロイメント・チェックリスト

この表は、デプロイメント・チェックリストを示しています。左側のツール・バーで印刷アイコンをクリックして印刷できます。

*マークの付いた項目はオプションです。

**マークの付いた項目はオプションの場合があります。

表20.8 デプロイメント・チェックリスト

チェック	項目	説明/注意
	新しいデータベースの作成	Management Console表をホスティングするための個別のデータベース(スキーマ)。
	JDBCドライバのインストール	ホスティング・データベースのJDBCドライバをWebコンテナの**lib**フォルダに配置します。
	**SQLスクリプトの実行	データベースへのアクセスに使用される資格証明にCREATEおよびALTER TABLE権限がない場合にのみ必須。
	データ・ソースの構成	**jdbc/Kapow/platform**のホスティング・データベースに対してJNDIデータ・ソースを構成します。

チェック	項目	説明/注意
	ログイン・プロバイダの有効化	**login.xml**にログイン・プロバイダを構成します。
	*デフォルト資格証明の置換	パスワードの暗号化/復号化に使用される組込み証明書を置換することをお薦めします。 「パスワード暗号化」を参照してください。

拡張構成

この項では、有用な複数の拡張構成オプションを示します。

LDAP統合

インストール・ガイドにはハードコードされた資格証明のリストに対してユーザーを認証する方法が記載されており、この項では、LDAPに対してユーザーを認証する方法について説明します。

login.xmlには、次の定義があります。

```
<bean id="ldap" class="com.kapowtech.mc.config.LdapLogin" lazy-init="true">
<property name="ldapServerURL" value="ldap://change-to-ldapHost:389"/>
    <property name="userDn"
    value="CN=LDAP test,CN=Users,DC=kapowdemo,DC=local"/>
        <property name="password" value="change-to-passowrd"/>
        <property name="userSearchBase"
        value="OU=Users,OU=TheEnterprise,DC=kapowdemo,DC=local"/>
        <property name="userSearchFilter"
        value="(userPrincipalName={0}@kapowdemo.local)"/>
        <property name="userSearchSubtree" value="true"/>
        <property name="groupSearchBase" value="OU=Security
        Groups,OU=TheEnterprise,DC=kapowdemo,DC=local"/>
        <property name="groupSearchFilter" value="(member={0})"/>
        <property name="groupRoleAttribute" value="cn"/>
        <property name="groupSearchSubtree" value="true"/>
<property name="convertToUpperCase" value="true"/>
</bean>
```

これは、**ldap**という名前の**LdapLogin** Beanを定義します。Beanには、LDAP統合を制御する多数のプロパティを定義します。TomcatをLDAPと統合する方法を理解している場合は、すでに馴染みがあることです。

表20.9 LDAPのプロパティ

プロパティ	説明
ldapServerURL	LDAPサーバーへのURL。これには**ldap://**プロトコルを使用します。

プロパティ	説明
userDn	他のユーザーを認証する際のLDAPへのログインに使用するDN(識別名)。
password	**userDN**アカウントのパスワード。パスワードはこのファイルにクリア・テキストで格納されるため、読取り専用アクセスのアカウントを使用してください。
userSearchBase	LDAPツリーでユーザーを検索するサブディレクトリ。
userSearchFilter	ユーザーの検索に適用するフィルタ。
userSearchSubtree	**userSearchBase**のサブディレクトリにユーザーが配置されている可能性がある場合はtrueに設定します。
groupSearchBase	LDAPツリーでグループを検索するサブディレクトリ。
groupSearchFilter	このグループのユーザーを識別するために適用するフィルタ。
groupRoleAttribute	グループ名を保持する属性。
groupSearchSubtree	**groupSearchBase**のサブディレクトリにグループが配置されている可能性がある場合はtrueに設定します。
convertToUpperCase	グループ名を大文字に変換します(デフォルトはtrue)。
allGroupsFilter	オプション。プロジェクト権限の作成時に表示するグループを制御します(後述を参照)。

「プロジェクトの権限」で説明されているように、Management Consoleの管理にLDAPアカウントを使用する場合は、**login.xml**の**adminGroups** Beanのメンバーとして属しているグループの1つを追加する必要があります。**adminGroups**にリストされたグループのどのメンバーでもManagement Console管理者に指定できるため、この目的のために新しいLDAPグループを作成できます。**convertToUpperCase**がtrueの場合は、大文字のグループ名を使用することに注意してください。

プロジェクト権限を選択する場合は、すべてのグループ名がLDAPからドロップダウンに移入されて表示されます。**groupRoleAttribute**を使用してすべてのグループをフェッチするフィルタを作成することで、グループが検出されます。ここにすべてのLDAPグループを表示する必要がない場合もあり、その際は追加のプロパティを**LdapLogin**に追加して独自のフィルタを提供することで、この動作を上書きします。

```
<property name="allGroupsFilter" value="(cn=*)"/>
```

これは、グループ名が**cn**属性にある場合(これがデフォルト)は、すべてのグループ名が検索されます。eの文字で開始するグループのみが必要な場合は、次のようにします。

```
<property name="allGroupsFilter" value="(cn=E*)"/>
```

フィルタには基本的なLDAP問合せを使用しているため、さらに複雑な問合せについては、ドキュメントの他の部分に記載されています。

高可用性

高可用性(フェイルオーバー)が必要な場合は、1つのクラスタとして連携する複数のManagement Consoleインスタンスを構成できます。完全なフェイルオーバーを実現するには、4つのコンポーネントをクラスタ化する必要があります。

表20.10 デプロイメント・チェックリスト

コンポーネント	説明
ロード・バランサ	HTTPロード・バランサは、複数のTomcatサーバー間でリクエストを分散するために必要です。
クラスタ化されたプラットフォーム・データベース	Management Consoleでは、スケジュール、ロボットなどがプラットフォーム・データベースに格納されます。フェイルオーバー・シナリオでは、単一障害点を回避するために、クラスタ化されたDBMSでプラットフォーム・データベースを実行します。
Tomcatセッション・レプリケーション	Management Consoleでは、ユーザーのセッション(インポート/エクスポート時を除く)にデータは直接格納されませんが、セッションにはユーザーの認証情報が保持されます。 セッション・レプリケーションが使用可能でない場合は、現在接続しているTomcatがクラッシュすると、ユーザーは再度ログインする必要があります。
Hazelcast	Hazelcast (www.hazelcast.com)は、複数のJVMでデータ構造をクラスタ化する際に使用します。Management Console内では、これによって重要なデータ構造のクラスタリングおよびアプリケーション・インスタンス間の相互通信が提供されます。 ここで例を示します。RoboServerでロボットを実行する場合は、RoboServerから返されるステータス・メッセージを処理するためにスレッドが必要です。このスレッドは明確なTomcatインスタンス内で実行されます。クラスタ化された環境では、ロボットを停止しようとするユーザーは、実際には、ロボットを実行しているインスタンス以外の別のTomcatインスタンスに対して停止リクエストを生成する可能性があります。この場合、停止リクエストはHazelcast経由ですべてのインスタンスにブロードキャストされ、ロボットを実行しているインスタンスがそのリクエストを受信してロボットを停止することになります。

Management Consoleの複数インスタンスのデプロイ

2つ以上の同一のTomcatインストールが必要で、そのすべてに同じバージョンの**ManagementConsole.war**をデプロイします。すべてのインスタンスで**web.xml**、**Configuration.xml**、**login.xml**および**roles.xml**ファイルが同じになるようにします。

ロード・バランサ構成

ここではロード・バランサ構成の詳細は説明しません。知っていると役に立つことは、アプリケーションが正常に起動したかどうかを判断する方法です。

ManagementConsole.xml (コンテキスト構成)または**web.xml**ファイルが無効な場合、アプリケーションはTomcatにデプロイされず、通常は404 (**/ManagementConsole/**にデプロイされていないTomcatのROOTアプリケーションへのアクセス)が返されます。

アプリケーションの起動中に検出されるその他のエラーは、アプリケーションのロード時にユーザーに表示されます。このように、アプリケーションが正常にロードされなかった理由を確認するために、常にログをチェックする必要があるとはかぎりません。ただし、起動時にエラーになった場合もアプリケーションから**200 OK**が返されるため、これはあまり実用的ではありません。また、認証が使用可能な場合、エラー・メッセージを確認するには、事前にログインしておく必要があります。

アプリケーションが正しく起動されたかどうかをロード・バランサで簡単に確認するために、URL **/ManagementConsole/Ping**に対するリクエストを作成できます。アプリケーションが正常にロードされた場合は**200**のHTTPステータス・コードが返され、エラーのスタック・トレースがある場合は**500**が返されます。

Tomcatセッション・レプリケーション

セッション・レプリケーションは**/conf/server.xml**で構成します。次に、インスタンスの検出にマルチキャストを使用する例を示します(Tomcat 5.5)。

```
<Cluster className="org.apache.catalina.cluster.tcp.SimpleTcpCluster"
managerClassName="org.apache.catalina.cluster.session.DeltaManager"
expireSessionsOnShutdown="false" useDirtyFlag="true"
notifyListenersOnReplication="true" printToScreen="true">

<Membership className="org.apache.catalina.cluster.mcast.McastService"
mcastAddr="228.0.0.4" mcastPort="45564" mcastFrequency="500"
mcastDropTime="3000"/>

<Receiver className="org.apache.catalina.cluster.tcp.ReplicationListener"
tcpListenAddress="auto" tcpListenPort="4002" tcpSelectorTimeout="100"
tcpThreadCount="6"/>

<Sender className="org.apache.catalina.cluster.tcp.ReplicationTransmitter"
replicationMode="pooled" ackTimeout="150000" waitForAck="true"/>
<Valve className="org.apache.catalina.cluster.tcp.ReplicationValve"
filter=".*\.gif;.*\.js;.*\.jpg;.*\.png;.*\.htm;.*\.html;.*\.css;.*\.txt;"/>
<Deployer className="org.apache.catalina.cluster.deploy.FarmWarDeployer"
tempDir="/tmp/war-temp/" deployDir="/tmp/war-deploy/"
watchDir="/tmp/war-listen/" watchEnabled="false"/>
<ClusterListener
className="org.apache.catalina.cluster.session.ClusterSessionListener"/>
</Cluster>
```

また、次のようにserver.xmlの**<Engine>**要素に**jvmRoute**属性を設定する必要があります。

```
<Engine jvmRoute="tomcat2" name="Catalina" defaultHost="MyHost">
```

. 注意: 廉価版のロード・バランサとして**mod_jk**を使用している場合、**jvmRoute**の値は、**mod_jk**構成に従って**workers.properties**ファイル参照にリストされている名前と一致する必要があります。

詳細は、Tomcatのドキュメントを参照してください。

Hazelcast構成

最も基本的なHazelcast設定は**Configuration.xml**で編集できますが、SSL暗号化などの高度な設定は**/WEB-INF/Hazelcast.xml**で構成する必要があります。

Management Consoleを起動すると、Hazelcastノードがポート5701（または次に使用可能なポート)で作成されます。デフォルトでは、このHazelcastノードはIPアドレス**127.0.0.1**にバインドします。クラスタ構成に加わるには、事前に、パブリックIP/ホスト名に対するバインド・アドレスを変更しておく必要があります。そのためには、**Configuration.xml**で**cluster** Beanの**interface**プロパティを変更します。次のように表示されます。

```
<bean id="cluster" class="com.kapowtech.mc.config.ClusterConfig" >
<property name="port" value="26000"/>
<property name="interface" value="10.0.0.*"/>
```

```
.......
</bean>
```

*はワイルドカードとして使用され、この場合、アプリケーションは、IPアドレスが**10.0.0**で開始する最初のインタフェースに対してバインドを試みます。これは可能ですが、結局は**127.0.0.1**または別の仮想インタフェースにバインドすることになるため、***.*.*.***を使用することはお薦めしません。

Management Consoleの追加のインスタンスを起動すると、そのHazelcastインスタンスによって既存のHazelcastノードの検出とクラスタへの結合が試行されます。この検出は、マルチキャストまたはTCP/IPを介して実行されます。

マルチキャストによる検出を使用するには、**Configuration.xml**で**cluster** Beanを変更する必要があります。これには、次の行のコメントを解除します。

```
<property name="joinConfig" ref="multicastCluster"/>
```

multicastClusterは、マルチキャスト・グループおよびポートを定義する**multicastCluster** Beanへの参照です。ネットワーク・トポロジに応じて適切に変更します。

マルチキャストを使用できないネットワークの場合は、**tcpCluster**を使用する必要があります。そのためには、この行のコメントをかわりに解除します。

```
<property name="joinConfig" ref="tcpCluster"/>
```

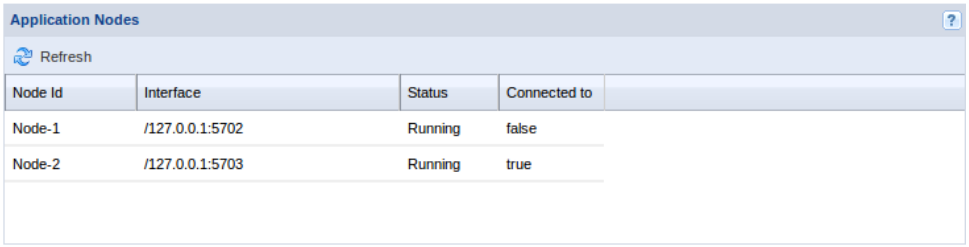
tcpCluster Beanには、**TcpPeer**（それぞれ他のHazelcastノードに対して1つ)のリストが含まれています。すべてのHazelcastノードに対して同じTCPポートを使用する場合は、ポート番号を指定する必要はありません(各ノードは、そのピアが同じポートで実行されていると仮定します)。1つのTCPクラスタに2つのノードが構成されている場合は、次のようになります。

```
<bean id="tcpCluster" class="com.kapowtech.mc.config.TcpJoinConfig">
<property name="peers">
<list>
<bean class="com.kapowtech.mc.config.TcpPeer">
<property name="host" value="10.0.0.25"/>
</bean>
<bean class="com.kapowtech.mc.config.TcpPeer">
<property name="host" value="10.0.0.26"/>
</bean>
</list>
</property>
</bean>
```

両方のノードがリストにあることに注意してください。これは、どのノードが最初に起動するかに関係なく、そのピアを検出できることを意味します。両方のアプリケーションで同じ**Configuration.xml**ファイルを使用できることもできます。また、TCPポート番号が定義されていないため、各ピアでは、それ自体のリスニング・ポートと同じポートで別のピアに対する接続が試行されます。

アプリケーション・ノード

「プロジェクト」タブに移動して、ページ下部のアプリケーション・ノード・セクションを確認することで、アプリケーションが適切にクラスタ化されているかどうかを検証できます。ここには、次のように表示されます。



これは、クラスタが現在2つのノードで構成されていることを意味します。インタフェース列には、Hazelcastがクラスタ間通信に使用しているIP/ホストおよびポートがリストされます。接続列は、現在2つのノードのどちらに接続しているかを示します。現在接続しているサーバーをシャットダウンすると、ロード・バランサによって別のライブ・インスタンスに自動的に再ルーティングされます。

アプリケーション・ノードを右クリックするとコンテキスト・メニューが表示され、ここで、あらゆるノードからのスレッド・ダンプをリクエストでき、デバッグに役立てることが可能です。

URIエンコーディング

デンマーク語のÆØÅやドイツ語のßなどの非ASCII文字が使用されている名前でロボットをリポジトリにアップロードする場合は、Webコンテナに対するURIエンコーディングをUTF-8に構成する必要があります。

Tomcatでは、**/conf**フォルダ内の**server.xml**ファイルにある**<connector>**定義でこれを操作します。次のように、属性**URIEncoding="UTF-8"**を追加します。

```
<Connector port="8080" URIEncoding="UTF-8"...../>
```

パスワード暗号化

バージョン8.2現在、Management Consoleでは、パスワードを格納する際、証明書ベース(公開鍵、秘密鍵)の暗号化を使用します。以前のバージョンからインポートすると、新しい証明書ベースのアルゴリズムを使用してパスワードが自動的に再暗号化されます。

証明書および照合する秘密鍵はJavaキーストアに格納され、Management Consoleにデフォルトの証明書および秘密鍵が格納されたキーストアが付属して出荷されます。すべての顧客が同じキーストアを入手するため、独自のキーストアを作成することをお薦めします。そうしないと、エクスポートしたものを誰かがロードしてパスワードが取得される可能性があります。

独自のキーストアの作成

Management Consoleをすでに起動した場合は、証明書をアップグレードする必要があります。

キーストアはpkcs12フォーマットである必要があり、Java SDK (Oracle.comからダウンロード可能で、現在はここから入手可能)に付属の**keytool**アプリケーションを使用して作成できます。次のコマンドは、有効期間が365日の証明書で、新しいpkcs12キーストアを作成します。

```
keytool -genkey -alias mc -keyalg
RSA -validity 3650 -keystore mc.p12 -storetype pkcs12
```

パスワードおよびX.509秘密鍵に格納する情報について入力を求めるプロンプトが表示されます。ファイル**mc.p12** (-keystore引数からの値)が現在のディレクトリに作成されます。-validity 3650は、証明書の有効期間が10年であることを意味します。

> 認証局(CA)で発行された証明書は、pkcs12に秘密鍵と公開証明書の両方が格納され、秘密鍵に対するパスワードがアプリケーション構成の一部としてクリア・テキストで記述されるため、使用はお薦めしません。

新しい証明書の使用をManagement Consoleに指定するには、**Configuration.xml**ファイルを変更します。ファイルは、**ManagementConsole.war** Webアーカイブ内にあり、解凍する必要があります(詳細は、「Tomcatへのデプロイ」を参照)。**Configuration.xml**には、次のエントリがあります。

```
<bean id="keyStore" class="com.kapowtech.mc.config.KeyStoreConfig">
<property name="location" value="/WEB-INF/mc.p12"/>
<property name="password" value="changeit"/>
<property name="alias" value="mc"/>
</bean>
```

ここに、キーストアの位置、パスワードおよび別名を指定する必要があります。キーストアを**ManagementConsole.war**にコピーする場合、位置はアプリケーションのルートから相対的にする必要があります。ファイル・システムに格納されたキーストアを参照する場合、位置は**file://**で開始し、キーストアの位置への絶対参照である必要があります。

キーストアのアップグレード

Management Consoleを初めて起動すると、キーストアから秘密鍵を使用してチェックサムが作成され、これによって、キーストアが置換された場合の検出、および提供された証明書を使用してパスワードを実際に復号化できるかどうかの検証が可能になります。独自のキーストアをインストールする前にManagement Consoleをすでに起動した場合は、パスワード変換を実行するために、Management Consoleを構成する必要があります。

最初に、現在のキーストア・ファイルをユーザーのホーム・フォルダなどの新しい位置にコピーして、次に**Configuration.xml**を変更し、古いキーストアに関して次のようなパスワード・コンバータを作成します。

```
<bean id="oldKeyStore" class="com.kapowtech.mc.config.KeyStoreConfig">
<property name="location" value="file:///home/roboserver/mc.p12"/>
<property name="password" value="changeit"/>
<property name="alias" value="mc"/>
</bean>
<bean id="passwordConverter"
class="com.kapowtech.scheduler.server.service.PasswordConverter">
<constructor-arg ref="oldKeyStore"/>
</bean>
```

以前の証明書を使用して既存のパスワードとチェックサムを復号化するパスワード・コンバータが構成され(古いキーストアの正確な位置、別名およびパスワードを指定する必要があります)、新しい秘密鍵(先ほど構成した)を使用してパスワードが再暗号化され、新しいチェックサムが作成されます。変換はManagement Consoleの次回起動時に実行され、スケジュールが数多くある場合は、アプリケーションの起動時に変換が発生し、しばらく時間がかかる可能性があります。パスワード変換はチェックサムとキーストアが同期していない場合にのみトリガーされ、変換後にチェックサムが新しいキーストアと一致するようになるため、**Configuration.xml**から**oldKeyStore** Beanおよび**passwordConverter** Beanを削除する必要はありません。

第21章 サポート

問題が発生した場合は、次のリソースが役立ちます。

カスタマ・サポート

Kapow Katalystの使用に関してなんらかの問題がある場合は、遠慮なく支援を依頼してください。その方法として、Kapow Softwareカスタマ・サポートに電子メールでご連絡ください (support@kapowsoftware.com)#. 電話サポートも利用できますので、Kapowライセンス・キーとともに提供された連絡先電話番号に問い合せてください。

プレミア・サポートが有効なお客様は、優先度1の問題に年中無休で対応するフリーダイアルも使用できます。プレミア・サポートのお客様で、この番号を紛失した場合は、support@kapowsoftware.com に問い合せてください。

Kapow Katalystアプリケーションの多くは、アプリケーション内から不具合レポートを送信することもできます。その場合は、「ヘルプ」メニューの不具合のレポートを選択します。不具合について可能なかぎり多くの情報と、不具合が発生する直前の状況を提供してください。

セルフサービス・ポータルとナレッジ・ベース

保守が有効なKapowのお客様にもセルフサービス・オンライン・サポート・ポータルへのアクセス権を取得する権利があり、このポータルには、一般的に検出される問題に対する解決に加え、実装のヒントおよび秘訣、セルフヘルプ・ビデオなどが含まれているナレッジ・ベースがあります。

ユーザーIDがなく、ポータルへのログインを希望する場合は、 support@kapowsoftware.com に連絡して手配を依頼してください。

サポート・ポリシー

リリース9.3では、次のサポート・ポリシーを発表しています。

- Kapow Katalyst 9.2は、2015年4月1日までサポートします。
- Kapow Katalyst 9.1は、2014年9月1日までサポートします。
- Kapow Katalyst 9.0は、2014年6月1日までサポートします。
- Kapow Katalyst 8.3は、2014年5月1日までサポートします。
- Kapow Katalyst 8.2は、2013年11月1日までサポートします。
- Kapow Katalyst 8.1は、2013年5月1日までサポートします。
- Kapow Katalyst 8.0は、2012年11月1日までサポートします。
- Kapow Web Data Server 7.2は、2012年5月1日までサポートします。

第22章 ライセンス

ライセンス情報

Kapow Katalystバージョン9.3.0の諸条件については、次を参照してください。

- Kapow Katalyst エンド・ユーザー・ライセンス契約
- サード・パーティおよびオープン・ソース・コード - ライセンス条項
- 製品プライバシ・ポリシー

エンド・ユーザー・ライセンス契約

最終更新日: 2011年3月29日

PLEASE READ THIS END USER LICENSE AGREEMENT ("EULA") CAREFULLY BEFORE DOWNLOADING, INSTALLING OR OTHERWISE USING THE SOFTWARE OR ANY ACCOMPANYING DOCUMENTATION. BY CLICKING "I AGREE" BELOW OR BY DOWNLOADING, INSTALLING OR USING THE SOFTWARE OR DOCUMENTATION, YOU AGREE TO BE BOUND BY THE TERMS OF THIS EULA. IF YOU DO NOT AGREE TO THE TERMS OF THIS EULA, YOU ARE NOT AUTHORIZED TO DOWNLOAD, INSTALL OR USE THE SOFTWARE OR DOCUMENTATION.

1. CONDITIONS OF USE. You (the individual and/or entity on whose behalf you are using the Software) are permitted to download, install and use the Software and Documentation only if: (a) you have a signed, written license agreement ("License Agreement") with Kapow Technologies, Inc. or its subsidiary ("Kapow") or its authorized reseller, or (b) you are using the Software on a limited trial basis.

Any other use is strictly prohibited. If applicable, the License Agreement remains in full force and effect, and if this EULA is inconsistent with the License Agreement then the License Agreement controls and governs. You are responsible for reviewing and understanding the terms of the License Agreement to ensure your compliance.

2. LICENSE GRANT. Subject to the terms and conditions of this EULA, Kapow grants you a non-exclusive, non-transferable, royalty-free, limited license to internally use the Software for the time period authorized by Kapow in the license key ("Evaluation Period") solely to test and evaluate the Software to determine whether you desire to purchase a commercial license to the Software.

3. RESTRICTIONS. You are not otherwise permitted to use the Software and, in particular, are not permitted to use the Software for your own or a third party's commercial benefit or in the normal course of your business operations. You may not copy the Software or the Documentation except solely as necessary for your authorized use and provided that all proprietary notices in the original are retained. You may not, nor may you assist another to, modify, translate, convert to another programming language, decompile, reverse engineer, disassemble or otherwise attempt to discern the source code of the Software.

4. THIRD PARTY AND OPEN SOURCE CODE. The Software includes certain third party and other code, including, but not limited to, free and open source software (collectively, "Other Code") covered by other licenses ("Third Party Licenses"), as identified in the Third Party and Open Source Code License Terms available at http://help.kapowsoftware.com, and the Oracle License below, all as may be revised by Kapow from time to time. You should check the Third Party and Open Source Code License Terms periodically for changes. Your license to the Other Code is subject to the applicable Third Party License, even if contrary to this EULA. The terms and conditions of the Third Party Licenses do not apply to nor govern other parts of the Software or the Software as a whole. As required under the applicable Third Party License, Kapow makes available the source code for the applicable

Other Code as described in the Documentation. If the Third Party and Open Source Code License Terms require you to have separately obtained rights from a third party in order to use any other Other Code, you represent and warrant that you have all such rights and agree to indemnify and hold harmless Kapow for your failure to do so.

5. OWNERSHIP. All right, title and interest, including all intellectual property rights, in and to the Software and Documentation are owned and reserved by Kapow and its licensors. No license or other right of any kind is granted to you except as expressly provided in this EULA or in the Third Party Licenses. Nothing in this EULA shall be construed as conferring any license or right with respect to any of Kapow's or its licensors' trademarks, trade names or brand names in any way.

6. CONFIDENTIALITY. You acknowledge and agree that Kapow is providing the Software under this EULA as a convenience to you, and that the Software, Documentation and information relating to the Software and Documentation are confidential and/or proprietary information of Kapow. You agree that you will not disclose to any third party, the Software or Documentation or any information regarding the Software's performance or your evaluation thereof, including any results of benchmarking, without Kapow's prior written consent. The Software, Documentation and information relating to these may be used only for the purposes authorized under this EULA.

7. TERM; TERMINATION. This EULA is effective from the time you accept this EULA until the Evaluation Period expires unless earlier terminated. Either party may terminate this EULA at any time by providing written notice to the other party. If you fail to comply with any term of this EULA, Kapow may immediately terminate this EULA upon notice to you. Upon any termination of this EULA, you shall cease all use of the Software and Documentation, destroy all copies of the Software and Documentation in your possession or under your control, and provide written notice of such destruction to Kapow. All provisions, other than Section 2, will survive any termination of this EULA.

8. NO ASSIGNMENT. You may not assign or otherwise transfer this EULA, the Software, Documentation or any of your rights in the foregoing, whether by operation or law or otherwise. Any unauthorized assignment or other transfer of any copy of the Software or Documentation is void and automatically terminates this EULA.

9. DISCLAIMER OF WARRANTIES. THE SOFTWARE, THE OTHER CODE, AND THE DOCUMENTATION ARE PROVIDED "AS IS." KAPOW AND ITS LICENSORS DISCLAIM ALL WAR-RANTIES OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, WARRAN-TIES OF PERFORMANCE OR MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE OR WARRANTY AGAINST INFRINGEMENT. YOU BEAR ALL RISK RELATING TO YOUR USE OF THE SOFTWARE, THE OTHER CODE, AND THE QUALITY AND PERFORMANCE OF THE SOFTWARE, THE OTHER CODE AND DOCUMENTATION. Kapow does not warrant that the Software or the Other Code will meet your requirements, operate without interruption or be error free.

10. LIMITATIONS OF LIABILITY. KAPOW AND ITS LICENSORS SHALL NOT BE LIABLE TO YOU OR ANY THIRD PARTY FOR ANY DAMAGES WHATSOEVER, INCLUDING WITHOUT LIMITATION, DIRECT, LOSS-OF-PROFIT, INDIRECT, SPECIAL, INCIDENTAL, EXEMPLARY OR CONSEQUENTIAL DAMAGES, ARISING OUT OF OR RELATING TO THIS AGREEMENT OR CAUSED BY THE USE OF OR INABILITY TO USE THE SOFTWARE, THE OTHER CODE, OR FOR ANY ERROR OR DEFECT IN THE SOFTWARE, THE OTHER CODE, OR DOCUMENTATION, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

11. GOVERNING LAW; JURISDICTION. This EULA will be governed by and construed under the laws of the State of California, without regard to its conflicts of law principles. The state and federal courts in Santa Clara County, California have exclusive jurisdiction and venue over any dispute arising out of or relating to this EULA. Each party consents to the personal jurisdiction and venue of these courts.

12. GENERAL PROVISIONS. This is the entire agreement between you and Kapow with respect to its subject matter and supersedes any prior or contemporaneous agreements regarding the same (except, as applicable, the License Agreement). If any provision of this EULA is held

to be void, invalid, unenforceable or illegal, the other provisions shall continue in full force and effect.No waiver or modification of this EULA will be binding upon either party unless made in a writing signed by both parties and no failure or delay in enforcing any right will be deemed a waiver.All notices in connection with this EULA shall be deemed given (i) five days after being deposited in the mail, postage pre-paid, certified or registered, return receipt requested, (ii) one day after being sent by overnight courier, charges prepaid, with a confirming fax, (iii) or upon personal delivery; and addressed as set forth in this EULA or to such other address as the party to receive the notice so designates by written notice to the other.Notices to Kapow will be sent to the address below and Notices to you will be sent to the address you provided to Kapow as part of the download process.

13. EXPORT.You may not export the Software or Documentation outside of the United States without Kapow's prior and express written consent.You agree to comply fully with all laws and regula-tions of the United States and other countries ("Export Laws") to assure that neither the Software nor Documentation are exported, directly or indirectly, or used in violation of Export Laws.

14. U.S. GOVERNMENT RIGHTS.If you are the U.S. Government or a contractor or subcontractor (at any tier) of the U.S. Government and are licensing the Software for use by the U.S. Government or in connection with any contract or other transaction with the U.S. Government, you acknowledge that by accepting delivery of the Software and Documentation, the Software qualifies as commercial computer software and commercial computer software documentation within the meaning of the acquisition regulations and contract clauses applicable to this procurement.The terms and conditions of this EULA are fully applicable to the Government's use and disclosure of the Software and Documentation and supersede any conflicting terms or conditions.

15. PRIVACY.By accepting this EULA, you acknowledge and agree that you have reviewed and agree to Kapow's Privacy Policy located at http://help.kapowsoftware.com.

16. PATENTS.One or more patents or patents pending owned by Kapow may apply to the Software and/or its features and functionality.These patents or patents pending include the following: US7904369, US Appl. No. 12/987,371, EP1342171, US7698277, US2009055727, EP1949262, HK1128839, US20090265420, EP2018757, EP1269347, US Appl. No. 10/240,463.

Copyright (C) 2001-2011 Kapow Technologies, Inc., a Delaware corporation and its subsidiaries, including Kapow Technologies A/S, a Danish corporation, 260 Sheridan Avenue, Suite 420, Palo Alto, CA 94306

ORACLE LICENSE

The Other Code includes the JDBC driver for Oracle ("Oracle Program"), which is subject to the following terms ("Oracle License"), on behalf of Oracle America, Inc. and its subsidiaries and affiliates under common control (collectively, "Oracle"):

License Rights: You are granted a nonexclusive, nontransferable, limited license to use the Oracle Program to run the Software for your own business operations.Oracle may audit your use of the Oracle Program.

Ownership and Restrictions: Oracle retains all ownership and intellectual property rights in the Oracle Program.You may make a sufficient number of copies of the Oracle Program for the licensed use and one copy of the Oracle Program for backup purposes.You may not: (i) use the Oracle Program for any purpose other than as provided in the "License Rights" above; (ii) distribute the Oracle Program, assign this Oracle License or give the Oracle Program, program access or an interest in the Oracle Program to any individual or entity; (iii) remove or modify any Oracle Program marking or any notice of Oracle's proprietary rights; (iv) use the Oracle Program to provide third party training on the content and/

or the functionality of the Oracle Program, except for training your licensed users; (v) assign this Oracle License or give the Oracle Program, access to the Oracle Program or an interest in the Oracle Program to any individual or entity except as provided in the Oracle License; (vi) cause or permit reverse engineering (unless required by law for interoperability), disassembly or decompilation of the Oracle Program; or (vii) disclose results of any Oracle Program benchmark tests without Oracle's prior consent.

Export: You agree that U.S. export control laws and other applicable export and import laws govern your use of the Oracle Program, including technical data; additional information can be found on Oracle's Global Trade Compliance web site located at http://www.oracle.com/products/export/index.html?content.html. You agree that neither the Oracle Program nor any direct product thereof will be exported, directly, or indirectly, in violation of these laws, or will be used for any purpose prohibited by these laws including, without limitation, nuclear, chemical, or biological weapons proliferation.

Disclaimer of Warranty and Exclusive Remedies: THE ORACLE PROGRAM IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND. ORACLE AND KAPOW FURTHER DISCLAIM ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION, ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IN NO EVENT SHALL ORACLE OR KAPOW BE LIABLE FOR ANY INDIRECT, INCIDENTAL, SPECIAL, PUNITIVE OR CONSEQUENTIAL DAMAGES, OR DAMAGES FOR LOSS OF PROFITS, REVENUE, DATA OR DATA USE, INCURRED BY YOU OR ANY THIRD PARTY, WHETHER IN AN ACTION IN CONTRACT OR TORT, EVEN IF WE HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. ORACLE'S AND KAPOW'S ENTIRE LIABILITY FOR DAMAGES HEREUNDER SHALL IN NO EVENT EXCEED ONE THOUSAND DOLLARS (U.S. $1,000).

No Technical Support: Oracle's technical support organization will not provide technical support, phone support, or updates to you for the Oracle Program.

End of Agreement: You may terminate this Oracle License by destroying all copies of the Oracle Program. Oracle or Kapow has the right to terminate your right to use the Oracle Program if you fail to comply with any of the terms of this Oracle License, in which case you shall destroy all copies of the Oracle Program.

Relationship between the Parties: The relationship between you and Oracle is that of licensor/licensee. Neither party will represent that it has any authority to assume or create any obligation, express or implied, on behalf of the other party, nor to represent the other party as agent, employee or franchises, or in any other capacity. Nothing in this Oracle License shall be construed to limit either party's right to independently develop or distribute software that is functionally similar to the other party's products, so long as proprietary information of the other party is not included in such software.

Open Source: "Open Source" software -software available without charge for use, modification and distribution - is often licensed under terms that require the user to make the user's modification to the Open Source software or any software that the user 'combines' with the Open Source software freely available in source code form. If you use Open Source software in conjunction with the Oracle Program, you must ensure that your use does not: (i) create, or purport to create, obligations of us with respect to the Oracle Program, or (ii) grant, or purport to grant, to any third party any rights to or immunities under our intellectual property or proprietary rights in the Oracle Program. For example, you may not develop a software program using an Oracle Program and an Open Source program where such use results in a program file(s) that contains code from both the Oracle Program and the Open Source program (including without limitation libraries) if the Open Source program is licensed under a license that requires any "modifications" be made freely available. You also may not combine the Oracle Program with programs licensed under the GNU General Public License ("GPL") in any manner that could cause, or could be interpreted or asserted to cause, the Oracle Program or any modifications thereto to become subject to the terms of the GPL.

Third Party Beneficiary: Oracle is a third party beneficiary of the Oracle License.

製品プライバシ・ポリシー

法定開示

概要

当社の目標はすべての面で最高水準の倫理と職業主義を守ることであり、この目標の重要な要素は、お客様のプライバシに対して敬意を払うことです。この書面では、お客様から収集した個人情報の使用に関するポリシーと実践について詳細を述べます。お客様から収集する個人情報は職業的なビジネス関係に即した内容で、ソフトウェアの使用と顧客関係の効率的な管理に使用できる情報に限定されます。お客様の同意なしに個人情報を収集することはなく、勧誘の目的で、Kapowの組織または直接のビジネス・パートナ以外に個人情報を販売、貸与または再配布することはありません。

詳細

収集の対象となる情報

RoboServerとManagement Consoleに組み込まれているcall home機能を使用し、次の情報をソフトウェアの起動時と日次ベースで、お客様から収集して追跡します。会社名、電子メール・アドレス、アプリケーション名、アプリケーションのバージョン番号とエディション番号、IPアドレス、ライセンス・キー、実行されたステップ数、HTTPリクエスト数、ロードされたバイト数、実行されたロボット数、返された値数、格納された値数、HBaseプット数、アクセスされたドメイン数(ドメインの名前ではなく)、レガシー環境の使用状況、Management Console内のスケジュール、Kapplets、ロボット、プロジェクト、クラスタ、RoboServer、OAuthユーザー、OAuthアプリケーションの数、Management Consoleでのスクジュールの実行回数とKappletの実行回数、RoboServerおよびサーバーのオペレーティング・システムが実行されているCPUの数。これは、受注契約またはソフトウェア・ライセンス契約(あるいはその両方)の締結によって、お客様が自発的に提供する情報であり、当社がお客様のソフトウェア使用状況を管理し、お客様の組織とのエンタープライズ・ソリューション・ビジネス関係を維持するために必要な情報です。

さらに、多数のソフトウェア使用パラメータと統計を収集して、ソフトウェアの改善と開発に役立てるための情報を取得します。これらには、製品の特定機能の使用カウンタとインジケータが含まれます。これは、特定機能の重要性の確認に役立ち、レガシー機能の場合は、機能を中止して製品から削除できるかどうかの確認に役立ちます。

情報の利用方法

情報は、統計を作成するために使用され、お客様のソフトウェア使用状況、およびKapowに対する契約義務への準拠に関する監視/管理に役立てます。収集した統計は、お客様の使用パターンを理解するためのベースとしても使用され、製品の継続的な改善に使用されます。

個人情報の共有

お客様のプライバシを尊重するために、個人情報や会社固有の情報を販売、貸与、またはいかなる方法でも関連のない外部のパーティ(業界誌、リスト・ブローカーなど)と共有することはありません。当社に提供された情報は、連絡や準拠の管理目的で当社組織内の様々な部門間で必要に応じて共有し、当社と緊密な関係にあるビジネス・パートナ(つまり、当社ソフトウェアに関する販売とサポートの権限を有するサード・パーティ)と、同じ使用目的で共有する場合があります。また、裁判所命令に準拠する必要性から、指定した個人と共有する場合があります。顧客ベースについて集計したデモグラフィック・データ(たとえば、「顧客の20%は機械または資本設備の製造者」)を、外部のパーティと適宜に共有する場合があり、お客様の同意を得た特別な目的のために、個人情報を業界誌や業界分析者などの特定外部パーティと共有する場合があります(たとえば、エンタープライズ・ソリューションの使用状況に関する記事の執筆について、編集者がお客様に連絡する場合など)。また、お客様の会社名を明らかにし、選択した引用を出典とともに会社のプレゼンテーションなどの資料で使用する場合があります。

不法侵入と盗難からの保護

一部の個人情報や会社固有の情報は、当社のお客様およびこれらの情報を自発的に提供する他の関連パーティから必然的に収集されます。このデータを安全に保護する当社に対するお客様の信頼に感謝いたします。オンラインやオフラインで収集した情報が、電子的な不法侵入や盗難のために危険さらされることはないと絶対的に保証することはできませんが、当社では、お客様のデータを次の方法で保護しています。

· セキュリティに関するベスト・プラクティスの追随

· 不法侵入の監視ツールと予防ツール(ファイアウォールなど)の利用

· 業種別セキュリティ・アラートの監視、および

· オペレーティング・システムとアプリケーションに対するベンダー・セキュリティ・パッチの適用

情報に対するアクセスの制限

Kapowによる情報の取得をブロックするためのファイアウォールの実装に制限はありません。収集予定の情報がファイアウォールによってブロックされているために、Kapowのtracker.kapowtech.comシステムで受信されない場合、当該お客様のRoboServerソフトウェアのパフォーマンスに対する負の影響や低下はありません。

このポリシーに対する変更

このプライバシ・ポリシーは、通知なしで適宜に変更される場合があります。したがって、このポリシーを定期的にレビューして、収集されている個人情報とその使用状況を確認してください。

連絡方法

このプライバシ・ポリシーや関連事項に関する質問やコメントは、Kapow Technologies, Inc., 260 Sheridan Avenue, Suite 420, Palo Alto, CA 94306宛にお送りください。