Oracle® Endeca Web Acquisition Toolkit

Usage Guide

Version 3.1.1 • December 2013

ORACLE

Copyright and disclaimer

Kapow Katalyst 9.2

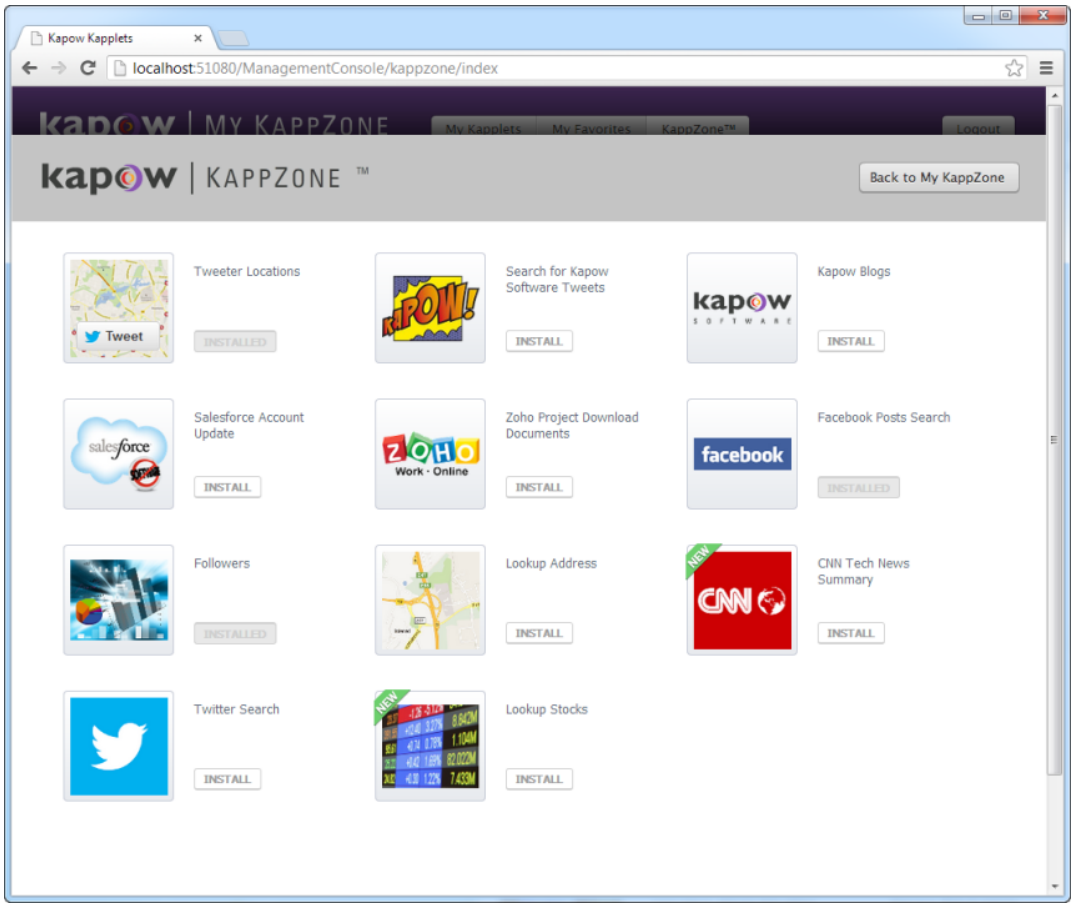
What's New in 9.2

Highlights of new features in Kapow Katalyst 9.2:

- Kapow KappZone™
- Creating Kapplets
- Customizing Kapplet branding
- Introducing a New Dedicated Spreadsheet View
- Making Virtualization and Cloud Deployment of Kapow Katalyst™ Even Easier
- Other Nice Features in 9.2
- Support end-of-life notices

Kapow KappZone™

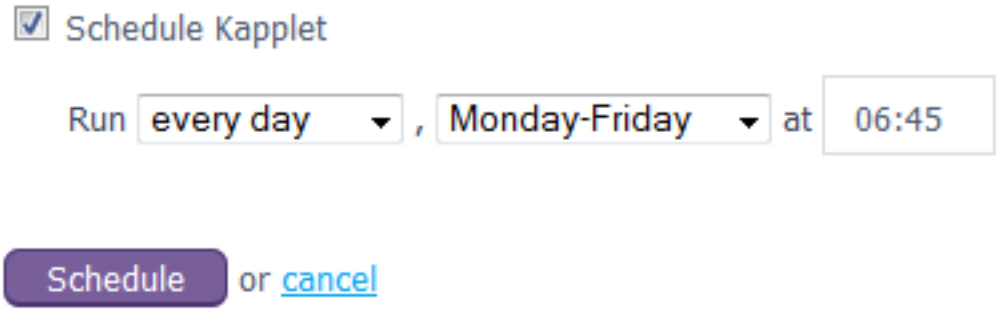
Kapow KappZone™ is a new role-based distribution and discovery container for Kapplets. The users access KappZone to discover Kapplets that are available to them based on their assigned user role access.



They run Kapplets from their personal "My KappZone" where they can access and run all their installed Kapplets.

Running Kapplets on a schedule

The users can schedule their installed Kapplets to run at specific times using the new scheduling feature in Kapplets.



Creating Kapplets

In 9.2 the Kapplets-tab has a Repository-tab for managing Kapplet design. Adding or editing a Kapplet brings you into the new advanced Kapplet Designer, with a lot of new configuration options for Kapplets. Including support for drag and drop of image files for setting the Kapplet icon image.

Multiple Robots in a Kapplet

A major new feature is that you can add multiple robots to a Kapplet. The robots will be executed concurrently when running the Kapplet.

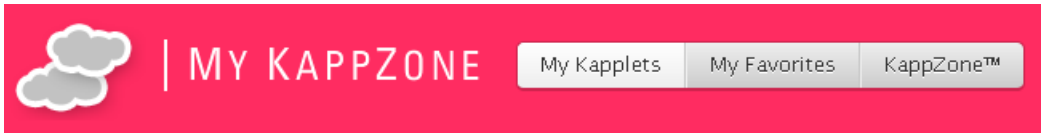
The input variables for each robot will be added to Kapplet input. Input variables of the same type can be joined, so multiple robots can share the same input values.

The input types and fields can be rearranged by the Kapplet designer by drag and drop with the mouse.

Kapplet output is organized by type, so multiple robots returning the same type will show their results in a single output table. Robots can return multiple different types, and as for the inputs, the output types and fields can be rearranged by drag and drop by the Kapplet designer.

Customizing Kapplet branding

As of version 9.2.2, you can customize the branding of KappZone and My KappZone to comply with your corporate color scheme, as well as to replace the Kapow logo with your own.



See the user's guide for details.

Introducing a New Dedicated Spreadsheet View

As part of our ongoing effort to provide support for specific data types we are expanding the use of dedicated data views in Design Studio. In 9.0 we introduced the XML view in addition to our Browser

View, and now with 9.2 we are introducing the Spreadsheet View providing much more advanced capabilities for extracting data from Microsoft Excel™ files and a spreadsheet-like interaction for selection of cells.

When using the new Spreadsheet View Excel files are no longer converted to HTML for extracting data and we have added dedicated steps and cell-finders to support data extraction from spreadsheets.

You can loop over the available sheets in the file, select ranges in the spreadsheets to loop through, and select between formatted values, raw data, or formulas for view and extraction.

The new Spreadsheet View significantly reduces the memory footprint of spreadsheet files in Design Studio making it easy to work with very large spreadsheets.

A special spreadsheet metadata tab in the Spreadsheet View allows extraction of file metadata such as file creation and modification dates, author and other document properties, along with the actual spreadsheet content.

Making Virtualization and Cloud Deployment of Kapow Katalyst™ Even Easier

With 9.1 we moved from the CPU based licensing introducing KCUs to support running in dynamic virtualized environment with fail-over support, dynamic allocation of KCU capacity and load balancing. With 9.2 we are further improving this by providing automated push of RoboServer configuration settings when a new RoboServer is added to a cluster, making it very easy to add new RoboServers to you Kapow Katalyst™ production environment.

RoboServer Configuration in the Management Console

The RoboServer operational configuration settings are from 9.2 defined on the cluster in the Management Console. This means that when a RoboServer is added to a cluster in the Management Console, it automatically gets the settings configured on the cluster. This includes database settings, proxy configuration, logging, profiling, etc. It also includes automated distribution of the JDBC jar-files needed for RoboServer to connect to the database. The jar-file can simply be uploaded to the Management Console and when needed, it will be distributed to the connected RoboServers.

To ensure that robots run with well-defined settings, configuration changes are pushed in a managed fashion by setting the cluster into Maintenance Mode allowing it to wait for running robots to finish before pushing out the new configuration.

The configuration settings in the Management Console are backed up with a full Management Console backup and restored with the restore function, so this also makes future upgrades a breeze. No need to apply settings and add JDBC jar-files to individual RoboServers, simply install and start the new version, restore your backup to the Management Console and your RoboServers are configured as before.

Other Nice Features in 9.2

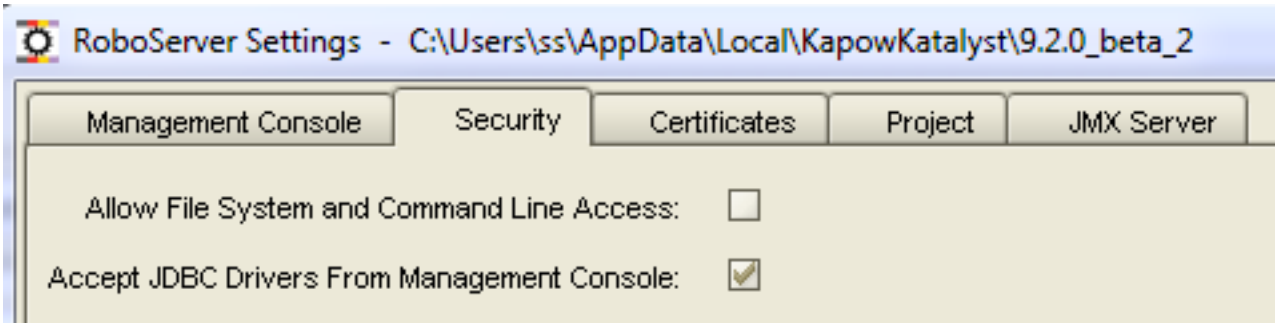
9.2 Includes a number of other improvements and additions that will make robot building and system management easier and more efficient. Explore the ones below to get even more benefit from your Kapow product.

Design Studio

- Inserting new step menu now includes selection of the step to insert to always make it a one click process to insert a new step in the robot.
- Tag Finder Patterns include support for "Does Not Match.." to make it easier and more efficient to make more specific and robust tag-finders.
- PDF Extraction is updated with a minor update.
- Design Studio can get Database Configuration from a Management Console cluster, making it easier to build robots requiring data from databases to operate.
- A new "Test Page Type" Step allows you to verify the type of the page view the robot is seeing the possible values are: HTML, XML, Excel, and Binary.

RoboServer Security Settings

With 9.2 we have added new security settings for managing file system access on RoboServers, if necessary preventing robot developer access to production servers' local file system and commands. These particular settings have to be managed locally on the RoboServer either in the configuration files or with the new RoboServer Settings application.



Scheduler

The pre- and post-processes in a schedule can now run robots.

The default security settings in the Management Console now prevent running scripts on the Management Console server by default. This is configurable in the Management Console configuration file.

License Management

There is a new License Tab in Management Console for entering license keys and this tab also provides access to monitor Design Studio Seats currently in use. This allows an administrator to see the connected Design Studio seats and identify connected users and workstations.

Support end-of-life notices

With 9.2 we have the following support end-of-life notices.

EOL for JMS RoboServer service

The JMS RoboServer service is deprecated and will be removed in the first major release in 2014. Customers using the service should migrate to the socket based service.

EOL for old storage system

In version 7.2 (released spring 2010) we introduced new storage and logging systems in Kapow Katalyst. The old system was kept to allow customers to transition into the new and improved system.

With the first major release in 2014 the old system has reached EOL and will be removed.

The removal will include RoboManager, Database Storage Environment, File Storage Environment, Multiple Files Storage Environment, Database Message Environment, Email Message Environment, File Message Environment, Refind Object Step, Refind Key Attribute Type, and Database Output Type.

We encourage our customers that still use the old system to plan a migration. Please contact support for more info.

Getting Started

Welcome to the Kapow Katalyst product family. In this introductory part of the documentation you will find:

- An overview of the Kapow Katalyst purpose and functionality, a short introduction to its components, and reading guidelines for the rest of the documentation.
- An installation guide that also covers configuration in general terms.
- Notes for Users of Earlier Versions that highlight important changes in the product and will help you switch the new version.
- A number of introductory video tutorials that will help you get started using Kapow Katalyst.
- Advanced video tutorials that dig deeper into specific topics and techniques.
- An overview of the different kinds of Kapow Katalyst license keys.
- Kapow Compute Units KCU.

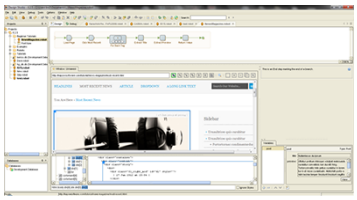
If you are upgrading from a previous version of Kapow Katalyst, it is strongly recommended that you read the Notes for Users of Earlier Versions.

If you are looking for support or other ways of getting help you might want to check out our list references to key net-based resources.

Overview

Kapow Katalyst is a platform for application integration and process automation. It can integrate applications that weren't built to be connected and automate processes across such heterogeneous systems; cloud/SaaS applications with premise systems, legacy systems with modern web applications, back office systems with partner websites.

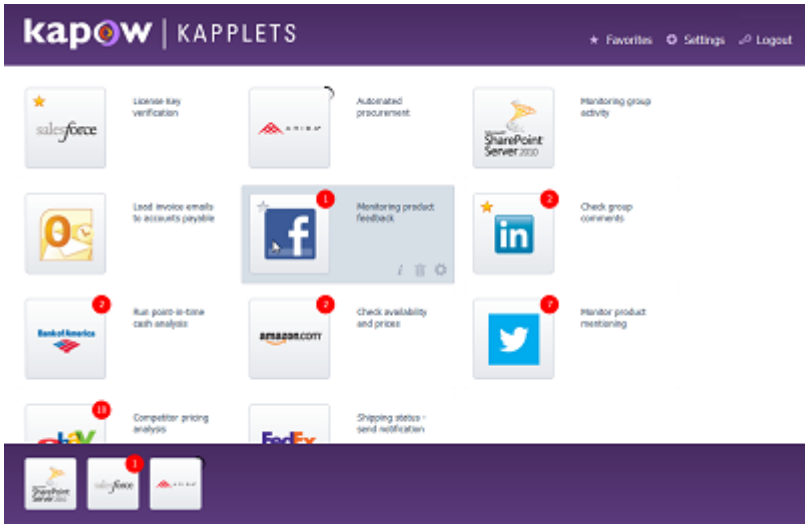
With our visual editor Design Studio (pictured), you click through the applications and data sources you want to integrate and create an automated workflow.



Kapow Katalyst Visual Editor: Design Studio

In Kapow Katalyst, these workflows are known as *robots*. As you build a robot, you are free to navigate through the applications as you integrate them. You can login to applications, extract data parts of a page, enter data into forms or search boxes, make menu selections, and scroll through multiple pages. Your robot can also access databases, files, APIs, and web services, exporting data from one application and loading it into another; transforming data as necessary along the way.

Once built, robots are uploaded to a repository in the *Management Console*. From here, they can be scheduled for batch-execution on the *RoboServer* or executed on-demand via Java or C# APIs, tailored REST services that are instantaneously available once the robots have been added to the repository, or exposed as special-purpose end-user web applications called *Kapplets*.



Kapplets

The Management Console is also responsible for load balancing, failover, monitoring of RoboServer health and management of user roles and permissions.

Next Steps

If you have not done so already, you should install Kapow Katalyst now.

We strongly recommend that you start by following the series of tutorials. The tutorials will take you through building your first robots, deploying them to the Management Console and exposing them as Kapplets.

For a more in-depth coverage, see the user's guide sections on the relevant application or topic:

- Design Studio
- Management Console
- Kapplets
- Java and C# APIs

Installation Guide

Kapow Katalyst has one unified installer, which installs *all* the software, even though different parts of it will be run on servers and on client computers. The role (server, client, etc.) of a particular installation is decided during the steps that take place after basic installation of the software.

The following subsections describe:

- How to install Kapow Katalyst, either interactively or "silently" (i.e., without user interaction).
- How to enter license information so that you can start Kapow Katalyst.
- How to configure Kapow Katalyst.
- How to set up Kapow Katalyst server applications to start automatically when the computer reboots.
- The supported platforms and system requirements.

Upgrading from Earlier Versions

In the interests of stability, different versions of Kapow Katalyst may be installed side by side on the same computer without interfering with each other (except that they must be configured to use different port numbers for Management Console if run simultaneously). This means that you can install a newer version and get acquainted with it while still doing your daily work with the older version.

This also means that important data like uploaded robots, execution schedules and so on will **not** be shared between different versions installed on the same computer. Thus at some point you will need to copy this data from the old Kapow Katalyst installation to the new one. (There is a similar but separate issue with Design Studio projects as discussed in Notes for Users of Earlier Versions.)

You copy this data from one version of Management Console to another by making a backup of the old installation and "restoring" it into the new one. This process is discussed here. Making a backup is done slightly differently in different versions of Kapow Katalyst, so check the online documentation for the relevant old version to learn how to create a backup. You will find the documentation here: 9.0 [http://help.kapowtech.com/9.0/topic/doc/sc/OptionsTab.html] 8.2 [http://help.kapowtech.com/8.2/topic/doc/sc/OptionsTab.html] 8.1 [http://help.kapowtech.com/8.1/topic/doc/sc/OptionsTab.html] 8.0 [http://help.kapowtech.com/8.0/topic/doc/sc/OptionsTab.html] 7.2 [http://help.kapowtech.com/7.2/topic/doc/sc/OptionsTab.html] 7.1 [http://help.kapowtech.com/7.1/topic/doc/sc/OptionsTab.html] 7.0 [http://help.kapowtech.com/7.0/topic/doc/sc/OptionsTab.html]

If you are running Management Console in a Tomcat web server (see below), be sure to make the backup before modifying your Tomcat configuration for the new version of Kapow Katalyst.

Management Console in Tomcat

The Management Console is a key part of Kapow Katalyst that by default is run as an embedded component inside the RoboServer application. However, it may also be installed as a regular web application on a standalone Tomcat web server. If you want to do that (or decide later to switch to doing that), you need to do a number of additional manual steps as described in the section on Management Console in Tomcat.

If you are upgrading from a previous version of Kapow Katalyst where you have installed Management Console in a Tomcat web server, be sure to make a backup of it (see above) before you install the new version into Tomcat.

Installing Kapow Katalyst

This section covers both interactive and "silent" installation on Windows as well as normal and headless installation on Linux. The section also lists the most important folders that will be created during installation.

Before installing on Windows or Linux, you must decide whether to install the 32-bit or the 64-bit version. The key issues to consider are:

- Your operating system may only support one of the versions. Generally, 64-bit operating systems support both.
- On Linux, the 32-bit version is the only one that provides Design Studio, the central tool for developing robots. However, it limits allocation of RAM to RoboServer to about 1.1 GB.
- The 64-bit version is mainly useful on servers where you want to run RoboServers that use much RAM. (You will need to configure the allowed amount of RAM after installation as described in the section Changing the RAM Allocation).

It is possible to install both the 32-bit and the 64-bit versions of Kapow Katalyst on the same computer, and they will share configuration and robots without any extra work on your side. However, every time you want to start one of the Kapow Katalyst applications, you must take care to start the right version (32-bit or 64-bit). Additionally, on Linux, make sure to select different installation folders for the 32-bit and 64-bit versions (on Windows, this is automatic).

Installing on Windows

Note

You need to have administrator rights to install on Windows.

You will have received instructions on how to download the installer. Download and save the `KapowKatalyst_9.2.0.msi` (32-bit) or `KapowKatalyst_9.2.0_x64.msi` (64-bit) file to your hard disk. After the download is completed, run the file to start the installation and follow the steps and dialogs of the installer. Then proceed to enter license information as described in Entering License Information.

Silent Installation on Windows

A silent installer is able to run without user interaction. This is convenient if, for instance, you need to automate the installation process in a script. If you wish to perform a silent installation of Kapow Katalyst, you need to run the following command with administrative rights:

```
msiexec /qn /i KapowKatalyst_9.2.0.msi
```

This will install the program to the default location. If you wish to specify another location, instead run:

```
msiexec /qn /i KapowKatalyst_9.2.0.msi INSTALLDIR="dir"
```

where `dir` is the location you wish to install to. After installation, you still need to proceed to enter license information as described in Entering License Information.

Installing on Linux

Note

You may install as an unprivileged user on Linux.

You will have received instructions on how to download the appropriate file. Download and save the installation `.tar.gz` file to your hard disk. This file is named `KapowKatalyst_9.2.0.tar.gz` for the Linux 32-bit version and `KapowKatalyst_9.2.0_x64.tar.gz` for the Linux 64-bit version. The following instructions refer to the Linux 32-bit version, but the others are the same except for the filename.

When the download is completed, the installation is done by extracting the contents of the file. In most Linux distributions, this can be done by right-clicking the file and choosing the appropriate extraction option. The file can also be extracted from the command line as follows:

```
$tar xzf KapowKatalyst_9.2.0.tar.gz
```

Alternatively, to extract the file to a specific directory, the following command can be used:

```
$tar xzf KapowKatalyst_9.2.0.tar.gz -C /destination_directory
```

When the file has been extracted, proceed to enter license information as described in Entering License Information.

Important Folders in Kapow Katalyst

There are a few folders and files in the installation that you need to know about.

Installation Folder

The installation folder is the folder where Kapow Katalyst is installed. On Windows, the installation folder defaults to:

```
C:\Program Files\Kapow Katalyst 9.2.0
```

On Linux it will be a directory named KapowKatalyst_9.2.0 in the directory where you extracted the archive.

The installation folder contains the following important folders:

`bin` contains all executable programs for Kapow Katalyst.

`API` contains files and documentation related to the Kapow Katalyst Integration APIs.

`Plugins` contains the Kapow Katalyst plugins.

`lib/jdbc` contains the installed JDBC database drivers. These are drivers that will always be available in Kapow Katalyst applications. Normally, you should manage JDBC drivers as described here, though.

Project Folder

The project folder contains your Library of robots and types, as described in the Design Studio User's Guide. The location of the project folder can be configured with the Settings application as described in Configuring Kapow Katalyst. On Windows, the default location is like one of the following, depending on your Windows version:

```
C:\Documents and Settings\username\My Documents\My Robots\9.2.0
```

```
C:\Users\username\Documents\My Robots\9.2.0
```

On Linux the default location of the project folder is:

```
~/KapowKatalyst/9.2.0
```

The project folder must contain a single folder named `Library`.

Application Data Folder

The application data folder contains files that are private to the Kapow Katalyst but which differ for different users of the same computer. On Windows, the application data folder is (depending on your Windows version):

```
C:\Documents and Settings\username\Local Settings\Application Data\KapowKataly
```

```
C:\Users\username\AppData\Local\KapowKatalyst\9.2.0
```

On Linux the application data folder is:

```
~/.KapowKatalyst/9.2.0
```

To change the location of the application data folder you can edit the `common.conf` file in the `bin` folder of the installation folder. You should add these two lines:

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

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

Make sure that the user running Kapow Katalyst has read and write access to the folder.

Under normal circumstances, you should never modify or delete files or folders within this folder directly; the GUI tools should be used instead. The application data folder contains the following important folders:

`Certificates` contains the HTTPS certificates known by Kapow Katalyst. See the Certificates section for more information.

`Configuration` contains configuration files, which are maintained as described in the section Configuring Kapow Katalyst.

`Data` contains the embedded Derby database used by the Management Console (except when installed as a web application in a Tomcat web server).

`DemoDatabase` contains the Development Database which is used in the Management Console Beginner Tutorial, and which you can also use in "toy projects" during your introduction to Kapow Katalyst.

`Logs` contains log files.

Entering License Information

The first step after basic installation is to enter license information. See the following pages for information on where to enter a license key in Management Console and Design Studio, respectively. It is not necessary to enter license information in the RoboServers, as it will automatically receive the necessary license information from Management Console.

Management Console (License Server)

Before you can enter license information into Management Console, you need to start it.

Windows Use the "Start Management Console" item in the Start menu.

Linux — Start Management Console from the command line. It is part of the RoboServer program, which is found in the `bin` directory under the installation (see Important Folders in Kapow Katalyst). It can be started with:

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

(You can also do this on Windows if you leave out the `./` part.)

Auto-start — As an alternative, if you would want later to set up auto-start of the Management Console as described in Starting Servers Automatically, you may select to do that already now instead of starting Management Console by hand.

Once the Management Console has been started (no matter how you do it), you must open its GUI in a browser. On Windows, you can do that with the aid of the "Management Console" item in the Start menu. On all platforms, you can open a browser and go to http://localhost:50080/. Accept the license terms and enter your license information including your license keys.

Design Studio

If Design Studio is not already started, do as follows:

Windows — Use the "Design Studio" item in the Start menu.

Linux — Start Design Studio from the command line by launching the `DesignStudio` program in the `bin` directory under the installation (see Important Folders in Kapow Katalyst) as follows:

```
$./DesignStudio
```

(You can also do this on Windows if you leave out the `./` part.)

Next, Design Studio needs a license and will therefore show a dialog that looks as follows:

You have three choices, depending on what information you have been provided from Kapow and/or your system administrator:

License Server — If your administrator has provided you with the URL to a central license server (Management Console) that administers the license for all Design Studio users, select License Server and simply enter the URL and credentials for that server. The license server must be running and have available Design Studio seats in order for you to use Design Studio.

Developer License — The developer license is a combined license key for both Management Console and Design Studio. You can either enter the license key in your local Management Console, or use the dialog in Design Studio. Whenever you start Design Studio, the Management Console will automatically start as well.

Trial License — The trial license works in the same way as the developer license, but is intended for short-time use for trial/demonstration purposes.

For a detailed example on how to enter license keys in Design Studio, see the video below:
Videos are not included in the PDF. Please visit open this url http://player.vimeo.com/video/60997108 in a browser to see the video.

Normally, Design Studio will display the dialog only the first time it is started. It will, however, display the dialog again if the license server cannot be contacted, e.g. because it has been moved or is not running.

Configuring Kapow Katalyst

After installing Kapow Katalyst, you should configure the installation to suit your needs. Some configuration is done using the Settings application as described in the guide on Configuring RoboServer.

Other configuration, relevant mostly for administrators of the Kapow Katalyst system, is done via Management Console, more specifically via the Servers tab, the Projects tab and the Options tab. It is especially important to set up the necessary RoboServers and clusters on the Servers tab. The following section shows how to start these RoboServers (and Management Console) automatically.

Starting Servers Automatically

If your installation includes any server functionality, you probably will want to configure it to start the necessary servers automatically. One case where this is not relevant is if you only need to run Design Studio while being connected to a Management Console that e.g. your IT department has set up.

When referring to "server functionality", we mean RoboServers and the Management Console (license server). In fact these two functionalities are provided by the same server program, `RoboServer`, depending on the arguments given to it when it is started.

The section Starting RoboServer contains a detailed description of the command-line arguments to the `RoboServer` program and we will not repeat them here. Suffice to say that the `-service` argument is what enables the RoboServer program to execute robots. Similarly, the `-MC` argument is what enables the Management Console functionality (see also Starting the Management Console).

With this in mind, we will turn to the details on how to make `RoboServer` start automatically (whatever the arguments). This is quite different on Windows and Linux.

Starting Servers on Windows

To make `RoboServer` start automatically on Windows, you need to add it as a Windows service. We will show how to add and remove Windows services using the `ServiceInstaller.exe` program that is included in the Kapow Katalyst installation.

Adding Windows Services

To run RoboServer as a service you need to install it first using the `ServiceInstaller.exe` program. The following is a general example outlining the command-line arguments to this program (although displayed on multiple lines here, this is a one-line command):

```
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:
: The account of the user that has to run RoboServer. Kapow Katalyst stores configuration in the user's directory and it is important to choose a user that has the correct configuration.

  If RoboServer has to run as a domain user you need to enter the account in the form `domain\account`

	If RoboServer has to run as a regular user you need to enter the account in the form `.\account`
wrapper.ntservice.password.prompt:	The value **true** will prompt the user for the password for the account. If you prefer to enter the password on the command line, you must instead use `wrapper.ntservice.password=whatever-the-password-is`.
wrapper.ntservice.name:	The name of the service to install. The name of the service can not contain spaces.
wrapper.ntservice.starttype:	`AUTO_START` if the service should be started automatically when the system is rebooted. `DELAY_START` if the service should be started after a short delay. (Not supported on Windows XP and 2003). `DEMAND_START` if you want to start the service manually.
wrapper.syslog.loglevel:	Redirect the console output from RoboServer to the event log.
wrapper.app.parameter.*:	The arguments for RoboServer. You can enter as few or as many as needed.

When the service is installed the user will be granted the "log on as a service" rights. If the service fails to start, check that the right is granted by opening `gpedit.msc` and (on Windows 7) navigate to `Computer Configuration -> Windows Settings-> Security Settings -> Local Policies -> User Rights Assignment -> Log on as a service` and add the user.

Example

This example installs a service named `RoboServer9.2.0` (note: no spaces) that runs as the user `bob`. When running the script, the user will be prompted for the password. The arguments for RoboServer are `-p 50000` (meaning that it will be able to execute robots):

```
ServiceInstaller.exe          -i          RoboServer.conf
wrapper.ntservice.account=.\bob
wrapper.ntservice.password.prompt=true
wrapper.ntservice.name="RoboServer9.2.0"
wrapper.ntservice.starttype=AUTO_START
wrapper.syslog.loglevel=INFO  wrapper.app.parameter.1="-p"
wrapper.app.parameter.2="50000"
```

Example

This example installs a service named `RoboServer9.2.0` (note: no spaces) that runs as the user `bob` on the domain `mydomain`. The arguments for RoboServer are `-p 50000 -MC` (meaning that it will be able to both execute robots and run Management Console):

```
ServiceInstaller.exe          -i          RoboServer.conf
wrapper.ntservice.account=mydomain\bob
wrapper.ntservice.password=secret
wrapper.ntservice.name="RoboServer9.2.0"
wrapper.ntservice.starttype=AUTO_START
wrapper.syslog.loglevel=INFO  wrapper.app.parameter.1="-p"
wrapper.app.parameter.2="50000" wrapper.app.parameter.3="-
MC"
```

Removing Windows Services

To uninstall a service you can run

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

wrapper.ntservice.name: The name of the service to remove

Example

This example removes a service named `RoboServer9.2.0`.

```
ServiceInstaller.exe -r RoboServer.conf
wrapper.ntservice.name="RoboServer9.2.0"
```

Starting Servers on Linux

The simplest way to make `RoboServer` start automatically on Linux is to use `crontab`. Use the following command to create or edit the list of scheduled jobs in Linux for the desired user (it may be easiest to do this either as `root` or as that particular user):

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

To the list of scheduled jobs add for example:

```
@reboot $HOME/KapowKatalyst_9.2.0/bin/RoboServer -p 50000
```

or

```
@reboot $HOME/KapowKatalyst_9.2.0/bin/RoboServer -p 50000
-MC
```

This way the `RoboServer` program will be started with the indicated command-line arguments upon reboot. Note that you must identify the `bin` directory under the actual installation folder.

Supported Platforms

This is the list of supported platforms. The versions listed are the ones on which Kapow Katalyst has been tested. This does not mean that it will not work on other versions. If in doubt whether it works on a version not listed contact Kapow Software to get more information.

IDE Platforms	
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 10.04 LTS (64-bit platform requires package ia32-libs)
Server Platforms	
Windows	Windows Server 2003, Windows Server 2008, Windows Server 2008 R2, Windows XP, Windows Vista, Windows 7
Linux	Red Hat Enterprise Linux 5.6 or 6.0, CentOS 5.6, Debian 5.0 (64-bit platform requires package ia32-libs)
Databases	
Oracle*	Version 11g

Databases	
IBM DB2	9.7
Microsoft SQL Server	Version 2005, 2008 and 2008 R2
Sybase*	Adaptive Server Enterprise 15.0 using the jConnect for JDBC 3.0 version 6.0.5 driver
MySQL*	Version 5.x. RoboManager is not supported on version 5.5 or later.
Postgres 9.2	If you don't want to use the 'public' schema, you must set the default schema for the user account accessing Postgres (ALTER USER <username> SET search_path to <schema>)
HBase	The step Store in HBase Table can be used with HBase Version 0.90.6.
APIs	
Java	Java 1.5 or later. Including J2EE servers such as Oracle WebLogic 11, IBM WebSphere, and JBOSS
JMX	Java 1.6
.NET	C#, .NET Version 4.0
Management Console Portals	
Tomcat	Version 5.5, 6.0 and 7.0. Minimum Java 1.6.

* Be aware that loss of data may occur when storing data in Oracle, Sybase or MySQL. On Oracle empty string is converted to null. On Sybase empty string is converted to " " (a single space). On MySQL millisecond precision is lost when storing dates. Check ObjectKey Caveats for details.

System Requirements

The tables below specify system specification for different platforms. The requirements may depend on the application so these should only be taken as guidelines and not as absolute numbers. A complex clipping solution might require much more power than a simple collection solution. The recommendations for servers are for one server. The number of servers used for a given application (the size of a cluster) is a completely different matter and should be estimated using methods described elsewhere.

IDE requirements:

	Minimum	**Recommended**
Windows	Intel Core Duo 1.8 GHz CPU (or AMD equivalent), 2GB RAM, 260MB Free Disk Space	Intel Core Duo 2.66 GHz CPU (or AMD equivalent), 4GB RAM, 260MB Free Disk Space
Linux	Intel Core Duo 1.8 GHz CPU (or AMD equivalent), 2GB RAM, 260MB Free Disk Space	Intel Core Duo 2.66 GHz CPU (or AMD equivalent), 4GB RAM, 260MB Free Disk Space

Server requirements:

	Minimum	**Recommended**
Windows (32-bit)	Intel Xeon L5520 CPU (or equivalent AMD Opteron), 1GB memory per CPU core + 512MB for OS, 500MB Free Disk Space	Intel Xeon X5680/X5677 CPU (or equivalent AMD Opteron), 1.5GB memory per CPU core + 1GB for OS, 500MB Free Disk Space

	Minimum	**Recommended**
Linux (32bit)	Intel Xeon L5520 CPU (or equivalent AMD Opteron), 1GB memory per CPU core + 512MB for OS, 500MB Free Disk Space	Intel Xeon X5680/X5677 CPU (or equivalent AMD Opteron), 1.5GB memory per CPU core + 1GB for OS, 500MB Free Disk Space
Windows (64-bit)	Intel Xeon L5520 CPU (or equivalent AMD Opteron), 1.5GB memory per CPU core + 1GB for OS, 500MB Free Disk Space	Intel Xeon X5680/X5677 CPU (or equivalent AMD Opteron), 2GB memory per CPU core + 1.5GB for OS, 500MB Free Disk Space
Linux (64-bit)	Intel Xeon L5520 CPU (or equivalent AMD Opteron), 1.5GB memory per CPU core + 1GB for OS, 500MB Free Disk Space	Intel Xeon X5680/X5677 CPU (or equivalent AMD Opteron), 2GB memory per CPU core + 1.5GB for OS, 500MB Free Disk Space

Real-time data: If you have a solution where users are waiting for results in real-time, CPU speed is normally the bottleneck, and you should buy the fastest CPU available for your hardware platform.

Dedicated hardware: For best performance we recommend that you always run RoboServer on dedicated hardware. That means that you should not run database servers and other services on the same hardware as your RoboServers.

Operating system: Kapow recommends using the Linux operating system as it is currently faster than Windows when running Java applications. In some scenarios we have measured up to 33% more throughput when using Linux.

Notes for Users of Earlier Versions

In the interests of stability, different versions of Kapow Katalyst may be installed side by side on the same computer without interfering with each other. This means that you can install a newer version and get acquainted with it while still doing your daily work with the older version. While doing so, you will probably want to open the older version's default robot project also in the Design Studio of the newer version. One thing to be aware of here is that if you *save* a robot with the newer version, you will not be able to open it using the older version of Design Studio. Thus it may be prudent instead to make a copy of the project and use the copy while you learn the new version of Design Studio.

Below we will list the most important differences between Kapow Katalyst versions that you should take into account when upgrading.

Version 9.2 of Kapow Katalyst introduces new capabilities for extracting data from Microsoft Excel™ files. You will find advice on how to use these instead of the old ones in the section on New Excel Capabilities in 9.2.

Version 9.1 of Kapow Katalyst no longer includes the RoboRunner application. You will find advice on how to use Management Console instead in the section on RoboRunner Discontinued in 9.1.

Version 8.2 of Kapow Katalyst introduced some minor changes that in rare cases may affect the execution of older robots. Details on these and other changes are found in the section on Changes in 8.2.

Version 8.1 of Kapow Katalyst replaced the concepts of "models" and "objects" with the concepts "types" and "variables". Details on this major change are found in the section on New Variables and Type System in 8.1.

Version 8.0 of Kapow Katalyst introduced new ways to display a robot's control flow and error handling. This affects all robots, even those built with previous versions, as discussed in New Control Flow and Error Handling in 8.0.

Version 7.2 of Kapow Katalyst introduced a new storage system that is much easier to use. Users upgrading from the previous versions can switch to the new storage system at their own pace; this is discussed in detail in New Storage System in 7.2.

Version 7.1 of Kapow Katalyst introduced a change in the location of certain files to comply with the installation rules of Microsoft Vista and Windows 7. Starting from this release, the files relevant to Kapow Katalyst are divided into 3 folders: The Installation Folder, the Project Folder and the Application Data Folder. You can read about these folders here. These changes mean that you no longer need to be a Windows power user (have write access to the "Program Files" folder) in order to run Design Studio.

New Excel Capabilities in 9.2

Since the way Excel is handled in 9.2 fundamentally different from the way that Excel was handled in previous versions of Design Studio it is important to know that there is no simple way to upgrade robot from earlier version to 9.2 when these involves Excel. It may involve a considerable amount of rewriting. This is because in previous versions Excel was transformed into HTML during loading and subsequent extraction etc. was then done in the same as for HTML, but in 9.2 Excel is loaded into the new Spreadsheet Page View and dedicated steps are used for data extraction from this.

The 'Convert Excel to HTML' Legacy Option

Old robots still works since there is a legacy option called 'Convert Excel to HTML' that allows this and old robots have this option turned on by default. If you want upgrade a robot to use the new Excel capabilities

you have to turn this option off. You do this under 'Robot Configuration' by clicking on the 'Configure Options' button and on the 'Legacy' tab of the Configure Option dialog you remove the check mark for the option. But changing this would, unless you don't do any extraction or looping that works on the Excel, break your robot. You will have to go through your robot and replace some of the step with the new dedicated Excel steps where ever the robot work on the data from the loaded Excel document. So unless you have compelling reasons for upgrading your old robots to use the new Excel capabilities you should probably not do it. This does not mean that you cannot upgrade your robot to use other 9.2 feature, just that you should leave the legacy option turned on.

Notice: the 'Convert Excel to HTML' option is always set in your old robot even if it did not involve any 'Extract from Excel' steps or Excel in any other way. It is important to remember this in case you load an old robot into 9.2 and use this as your starting point for building a new robot, because if you do not turn off the 'Convert Excel to HTML' option then you will not be able to take advantage of the new Excel capabilities.

The Extract from Excel Step Action is Deprecated

The 'Extract from Excel' step action has been deprecated and cannot be used in new robots instead you should use a 'Create Page' step action for this. How to do this is described in How to Load an Excel Page from a Variable.

RoboRunner Discontinued in 9.1

As previously announced, the RoboRunner application is no longer supported. This guide will show you what to do instead.

The task of scheduling robots at given intervals is now handled by the Management Console, which is a web based application. We will look at each of the different RoboRunner execution scenarios, and how these tasks are done using the Management Console. Before you proceed any further we recommend that you read the Management Console User's Guide, as we will be referring to it in great detail later.

Although the robots are scheduled by the Management Console they are in fact executed on a RoboServer. You can read more about the RoboServer here. The RoboServer can be monitored, using Control Center, or the Management Console Dashboard.

Scheduling concepts

RoboRunner had two ways of executing robots:

The -file option

If your robots were scheduled individually using RoboRunner, you will have to create a separate schedule for each robot to have it run at the desired time, as described here. Scheduling robots this way has an added benefit, as you can now parametrize your robots using input objects.

If you have multiple robots that need to run at the same time, you can create a schedule by selecting the robots on the Robots Tab in Management Console, right clicking and selecting 'Create Schedule' from the context menu.

The -dir option

If you were running multiple robots in a directory using RoboRunner's `-dir` option, use a 'robot group job'. This allows you to schedule a number of robots at a given time based on the name of the robot. This means that robots that have to run together will have to either:

- Have a common name prefix or suffix, like batch1_bestbuy.robot, batch1_newegg.robot, or

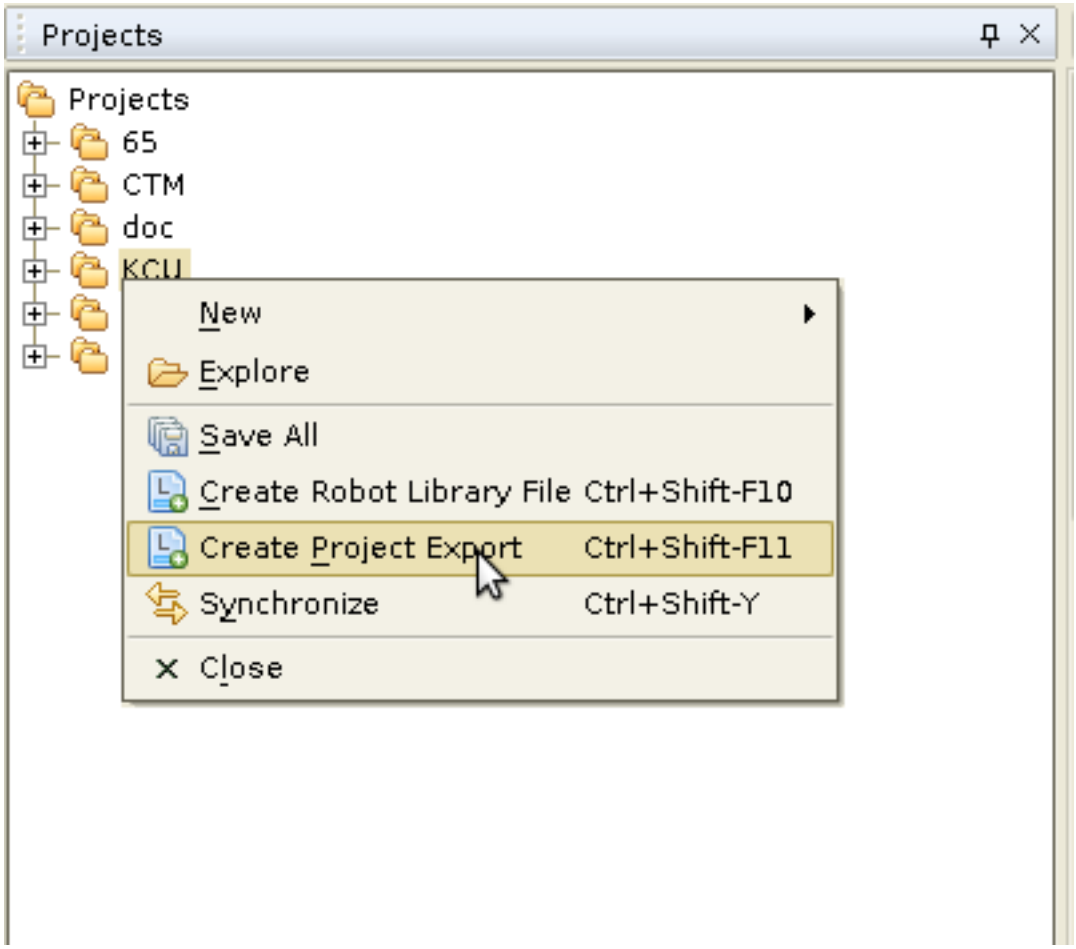
- Be in a project of their own. This allows you to use the pattern based criteria with the `.*` argument.

Before you start uploading your robots you should choose which of the two strategies you want to use, as changing this later may require a lot of work.

Important: Robots that use the FileStorageEnvironment or the FileMessageEnvironment are not supported by the Management Console. You can either rewrite the robots to use the Write File [../ref/robomaker/reference/stepaction/WriteFileStepAction.html] step action, or you can store the data in a database and write a script or program to convert the stored data into files.

Bulk deploying to Management Console

Once you have structured your robots, you are ready to deploy them to Management Console. This is done by creating a Project Export from within Design Studio. To do this, right click the project folder in the project view, and select 'Create Project Export' from the context menu.



Then select a location where you want to store the project export, and click the OK button.

If any of your robots were developed before version 8.1, you will first have to expand any `.model` files you may still have. You can read more about types and models here. Also, names of robots in a project must be unique even if they are located in different sub-folders. A dialog will inform you if there are any name conflicts.

Once you have created the project Export, open the Management Console and go to the Options Tab (Admin => Options) to perform the Project Import.

Scheduling robots

Once your robots are uploaded, you can create schedules to execute your robots.

If your robots use the old database storage system (Database Storage/Message/Info Environment), you will have to configure options on the 'Legacy' tab of the 'New Schedule Dialog'. However, we recommend that you rewrite these robots to use the new storage paradigm introduced in 7.2. You can read about the new storage system here.

The new storage system includes a new and improved database logging system, which is integrated into the Management Console, and allows you to 'jump' directly from a schedule to the errors generated by the robots. This will work in parallel with the legacy Message Environments, but again we recommend that you rewrite your robots, since the Storage/Message Environments will be removed from the product at some point in the future.

Changes in 8.2

This section is meant only for users of older versions of Kapow Katalyst. It will detail changes done in version 8.2, some of which may affect the execution of older robots:

- When all branches of a Try step fail, the situation may be logged or reported back to the caller of the robot as an API exception, depending on how the Try step is configured (this is described in detail in How to Handle Errors). When running the robot on RoboServer in versions prior to 8.2, all the details on what caused each branch to fail were logged or reported. However, it was not stated that all branches of the Try step failed, and the name or location of the Try step was not given either (this information was available only in the debugger).

  Starting with version 8.2, it will be reported or logged as a separate error that "All alternatives of the Try step failed", including a reference to the Try step. This error will be reported or logged prior to the details about the failed branches. Note that this change may affect some API clients and will affect the contents of the robot property `Robot.executionErrors` which can be used in expressions in the robot.

- If you are using database logging you will have to updated one of the log tables. The `ROBOT_MESSAGE` in the log database now has an additional column named `PROJECTNAME`. Before you can use database logging you must update the `ROBOT_MESSAGE` table. There are two option, either delete the `ROBOT_MESSAGE` table (you will loose all errors/messages from previous runs), the table will be recreated when RoboServer (or Management Console) is restarted. If you wish to keep history from previous runs, or the credentials used by RoboServer doesn't have the ALTER TABLE privilege you should have your DBA add the `PROJECTNAME` column. The type of the column should be `VARCHAR(255)` (`NVARCHAR2` on Oracle, `NVARCHAR` on Sybase and Microsoft SQL server).

New Variables and Type System in 8.1

This section is meant only for users of older versions of Kapow Katalyst. It will further the understanding of the new variables and type system in version 8.1.

As of version 8.1, the terms "object" and "model" are no longer used. A robot can now contain a number of **variables**, and each variable has a **type**. This new terminology is meant to make things more clear, as the term "object" was previously used rather loosely. Overall, the following changes have been made:

- The terminology is changed, so that what was in the model editor called objects is now called types, and what was in the robot called objects is now called variables. The Model Editor is therefore now called the **Type Editor**.
- Types and variables do not work in the same way as objects did. A type should only be viewed as a blueprint for a variable, that is, variables can be created from a certain type. The old terminology indicated that "objects" were in fact created in the model editor, and could then be added to the robot, if necessary. This is no longer the case. Therefore, it is an important difference to understand: types do NOT define variables, but only blueprints, and to make use of a type in a robot, a variable OF that type must be created.
- A type file, as opposed to an old model file, only defines a single type. Previously, model files could define any number of objects, making it difficult to maintain an overview.
- It is possible to create more than one variable of a certain type. This is opposed to previously, where an object defined in a model file could only be added to the robot once.
- A number of **simple types** exist that can be used for storing a single value. The purpose of these, among other things, is to replace the "ScratchPad" object. Also, in many cases, they make creating a robot easier as they can be used instead of having to create types simply for storing temporary data. To be able to distinguish, the types created by the type editor are referred to as **complex types**. Only variables of complex types can be used as input to the robot, or returned as output. Variables of simple types should thus be considered internal robot variables.
- A complex type is not defined as being used for either "input" or "output". Instead, it is possible on a variable to set whether it should be used as input. The "output" term is no longer used; all variables can be output (except variables of simple types which, as mentioned above, cannot be used as either input or output).
- The old model files can still be used, but can no longer be edited. The contents of the model files is now interpreted as types, and not as objects. This means that the types defined in a model file can be used to create variables - just as types defined in the new type files. In model files, objects were defined as being either input or output. This information is no longer needed, as input is defined on variables. Therefore, it is simply discarded when loading model files. As an example of when to be aware of this is when creating a variable from a type defined in an old model file. If the object from which the type is derived was previously an input object, it is important to note that the variable created from the corresponding type will NOT automatically be set to input. This must be done manually.
- Model files can be converted to type files. This creates a single type file for each type (object) defined in the model files, and these can then be edited if necessary. A Model is converted into Types by right clicking the model in the project view (in Design Studio) and selecting 'Expand Domain Model File' from the context menu.
- The old concept of global variables has been changed. Now, any variable can be set as global. Being able to make any kind of variable global provides a higher degree of freedom in how to structure a robot. When opening a robot using the old global variables, these are converted into the new kind of global variables of the simple type "Short Text".

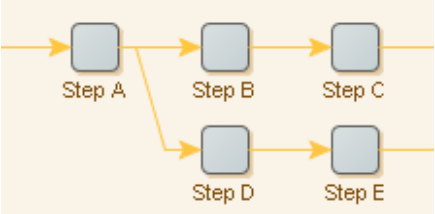
New Control Flow and Error Handling in 8.0

Version 8.0 of Kapow Katalyst introduces a new way to display a robot's control flow and contains extensive changes to the way errors are handled. This affects all robots, even those built with previous versions, meaning that although they still work in the same way, these robots will look differently when opened in version 8.0 or newer. For the benefit of users of the earlier versions of Kapow Katalyst, this chapter will give examples of the differences.

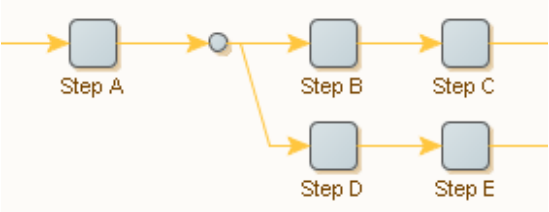
Branch Points and Try Steps

The most visible change is that ordinary steps now can have only one outgoing connection. If the step has more than one step following it, an explicit branch point (a small circle) or a Try step is inserted after it. These new steps are the only ones that can have more than one connection going out from them.

This also means that the "Branching Mode" property has disappeared from the step configuration. If a step previously had the "All Branches" branching mode, it will now be followed by a branch point as seen in the following example. First a snippet of a robot as it looked in version 7.2:

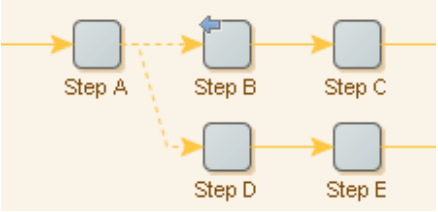


In version 8.0, the same robot now looks as follows:

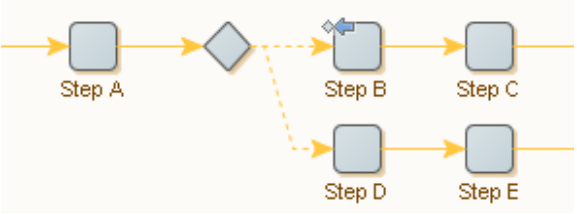


Of course a branch point is only inserted after a step if it has more than one step following it.

If the step previously had the "Until Successful Branch" branching mode, it will now be followed by a Try step rather than a branch point. This step is a core component of the new way to handle errors and is described in Error Handling; at this point we will merely show an example of what it looks like. Consider the following part of a version 7.2 robot:



When you open this robot in version 8.0, it will look this this:



End Steps

The introduction of End steps is a small but important change. An End step does nothing (in particular it does *not* end robot execution), but it provides an explicit and automatically inserted "last step" on each and

every branch in the robot. This means that if you click on an End step (in Design Mode), *all* of the branch will be executed. Previously, Do Nothing steps were often added manually to serve the same purpose. This is no longer necessary.

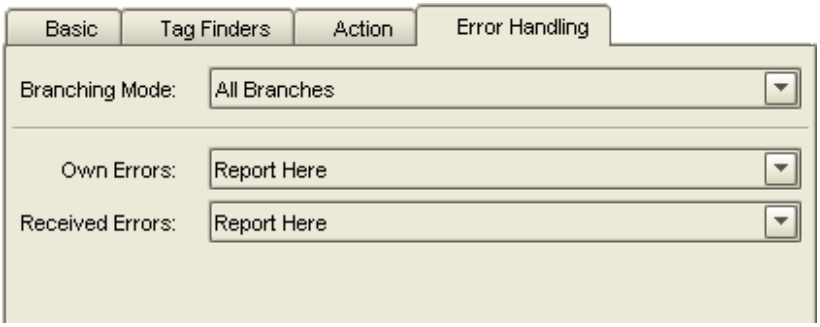
A single explicit "begin robot" marker (a right-pointing triangle) is also displayed in each robot. It is provided merely for balance, and because it makes a robot a bit more similar to a group step (another new feature).

The following complete robot shows both End steps and the marker at the beginning of the robot. In version 7.2, the robot looked like this:

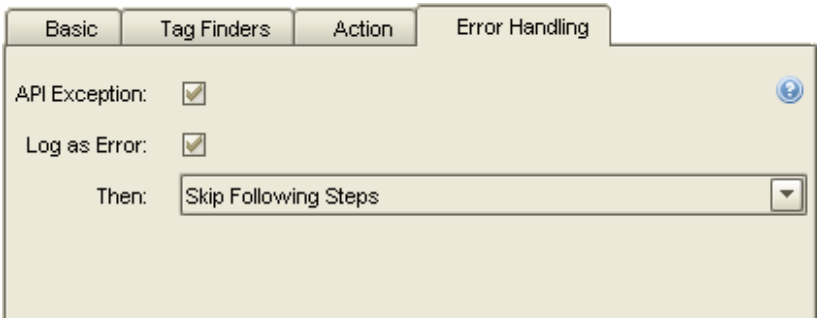
And when opened in version 8.0, the same robot looks like this:

Error Handling

The area of error handling in robots has been thoroughly revised. The section How to Handle Errors explains this in detail; in the following we will merely indicate how the older way of error handling translates into the new one. In version 7.2, error handling was configured by way of the following properties:



All of this has been replaced by the following set of properties, which are explained below:



Errors May Be Reported or Logged No Matter How They Are Handled

It used to be that the step property "Own Errors" specified both whether to "report" an error and what effect the error should have on further execution of the robot (error handling). These aspects have now been separated and can be configured independently. Instead of "Own Errors" we now have three properties: "API Exception", "Log as Error" and "Then".

No matter how the error is handled (even if it is ignored), the incident can be reported back to the caller of the robot (this is named an "API Exception'). In addition and independently of this, the error can be logged ("Log as Error"). Both API exceptions and logging can be selected or deselected for each step independently.

The error handling aspect of the former "Own Errors" property is now taken care of by the "Then" property.

Errors Are Handled Where They Occur

Previously, one very important way to handle an error was to specify that it should be "sent backwards" (to a step further to the left in the robot) for handling. The configuration of "Received Errors" and "Branching Mode" on the (possibly many) traversed steps would then decide how the error would ultimately be handled. It might end up being "reported" (causing an API exception); affect the execution of the robot; or be ignored, and it was impossible to know without considering the configuration of all traversed steps.

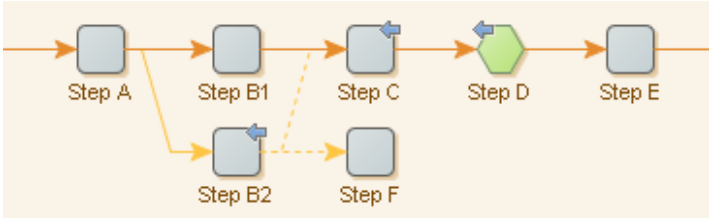
This whole notion of "sending an error backwards" has now been replaced by a more direct approach, where you always specify *on the step where the error occurs* what to do with it, without depending on the configuration of other steps. Thus the "Received Errors" property has been removed.

When an old robot is opened, an analysis is done for each step that uses "Send Backwards" in order to determine how to assign the "Then" property. Most often, the result is that instead of "Send Backwards" (displayed as on the step), a step will now use either "Try Next Alternative" () or possibly "Skip Following Steps" () in combination with "API Exception" and "Log as Error".

"Send Backwards" is still available as a way to handle errors, but is deprecated (meaning you cannot select it when you edit the "Then" property). The only reason it is still available is that it is necessary in order to execute some robots in exactly the same way as in previous versions of Kapow Katalyst. It is needed only by a small number of robots. When examining these robots, you should think of all steps as having "Received Errors" set to "Send Backwards".

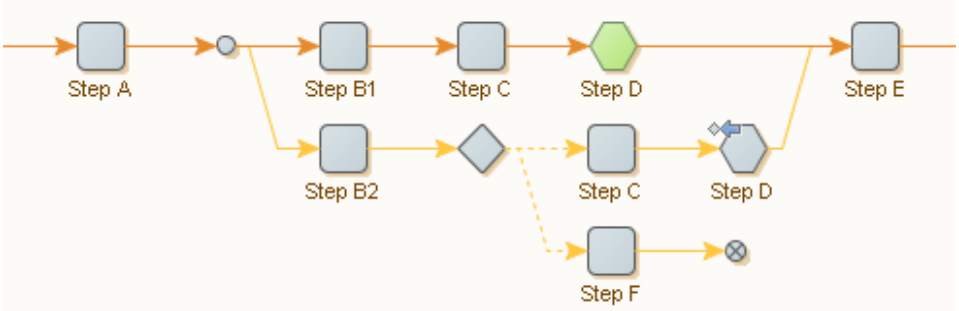
Small Changes to Robot Structure

In rare cases where branches merge, the result of the analysis described above may be that the step's "Then" property should be configured in different ways depending on the path followed to the step. In such cases, the step (and often small parts of the robot) will be duplicated to "de-merge" branches in a way so this dependency on the path disappears. Thus you may see some surprising expansion of the robot. The following robot is an example (it may seem contrived, but robots that exhibit this behavior *are* odd). The robot looked like this in version 7.2:



When step D is executed, it may report an error. If the step is reached on an execution path that goes via step B1, the error will be sent back to B1 and will be reported there. However, if step D is reached via step B2, it will not be reported. Rather, it will cause execution to proceed at step F.

When opened in version 8.0, the robot is automatically modified as follows:



We see that step D has been duplicated and that the two copies have been configured differently. The one associated with step B1 (in green) has "API Exception" and "Log as Error" turned on and uses "Skip Following Steps", while the one for B2 does no logging, but uses "Try Next Alternative". Step C has been duplicated also, in order to make it possible to connect the steps correctly.

Small Changes to the Order of Error Reports

It should be noted that in earlier Kapow Katalyst versions, "Send Backwards" made it possible to delay reporting an error as an API exception. In version 8.0, however, if an error is reported, it is reported as soon as it occurs. For some robots, this means that errors are reported back to the caller in a different order than previously, and they may be grouped into API exceptions in a different way, too.

New Storage System in 7.2

With version 7.2 of Kapow Katalyst storing data in databases has become easier than with previous versions. The new storage mechanism lowers the learning curve and eliminates tedious tasks, allowing you to spend more time writing robots.

This guide describes the differences between the old and new storage systems, and the upgrade path required to move to from the old to the new storage system.

The Old Storage System

Since the first release of Kapow Katalyst it has been possible to store data in databases. Although the system has proven its worth, a number of problems have become visible over time.

Problems with the old storage system

- Storing data in a database required a special DatabaseOutputObject.
- The DatabaseOutputObject had a refindKey field; few understood the importance of the refindKey at the time the object was constructed, and once data had been stored, changing the refindKey was a cumbersome process.
- You had to register each robot with the RoboManager database, so the robot would have a robotId.
- At run-time you had to provide 3 environments, a DatabaseStorageEnvironment, a DatabaseRobotInfoEnvironment and a DatabaseMessageEnvironment.

- The Refind Object action was confusing, and error prone to use.

These issues introduced a number of pitfalls, which even experienced users fell into. Most common is probably the duplicate robotId, where an already registered robot was used as a template, but the copy was never re-registered with RoboManager, and suddenly two robots had the same RobotId.

The New Storage System

The new storage system, introduced with version 7.2 of Kapow Katalyst, intends to fix the shortcomings of the old storage system.

Benefits of the new system

- No dedicated OutputObject for databases, just use regular output objects.
- No registration of robots.
- Data is stored using the Store in Database step action.
- No environments, logging is optional and configured using the Settings application.
- No RoboManager, the log is viewable through a web interface.
- New Find in Databases, and Delete in Database step actions.

The new system has completely removed the use of RoboManager and environments, which greatly simplifies the process of running a robot that collects data into a database.

Instead of the refindKey, you now have an ObjectKey, but in addition to configuring it when you build the model, it can also be specified inside the robot (also no fields are selected for the ObjectKey by default, forcing you to consider which fields define object identity).

When an object is stored using Store in Database, 7 additional (household) fields are stored. These are similar to the 6 hidden fields contained in the old DatabaseOutputObject, except for the following changes:

- ObjectKey replaces refindKey.
- ExecutionId replaces RobotRunId (allows you to correlate with API executions).
- RobotName replaces RobotId (so make sure the robot names are unique).
- LastUpdated is added, this allows you to see the last time this record was updated (similar to the functionality added by the old ChangeDate plug-in).

Legacy Options

After a fresh install of 7.2 the old storage system is completely hidden, this is done to prevent confusion for new users. If you need to build old Database Output Objects, or run a robot with environments in the Debugger, you have to enable Legacy Properties in Settings; this is done on the Legacy tab.

Backwards Compatibility

Since the two storage systems required different table layouts, they are not compatible. However, in most scenarios the two systems can coexist without problems, allowing you to migrate your robots over time. You should not mix the usage of the new and the old system within a single robot.

In most cases you can execute robots using both storage systems on the same RoboServer, as long as you don't provide a DatabaseRobotInfo Environment to robots which use the new storage system. This means that if you use a deployment descriptor or API code to configure the environments, you will have to make a distinction in your code between robots using the new and the old system. Alternatively you can temporarily give new robots a RobotId allowing them to be run with the RobotInfo/Message environments, and then remove the environments once all robots have been migrated.

If a new robot is executed with a Database Storage Environment, it will be ignored, as long as the robot doesn't contain any old Database Output Objects or Return Object steps.

If you execute a robot with both a Database message environment and database logging enabled for RoboServer, you will get two different logs: One you access through RoboManager, and another one you access via the Log View in the Management Console.

Migration

Migrating from the old to the new storage system usually requires four distinct migration steps. Migration of the models, the robots, the data, and any SQL scripts interacting with the data tables.

Migration of Models

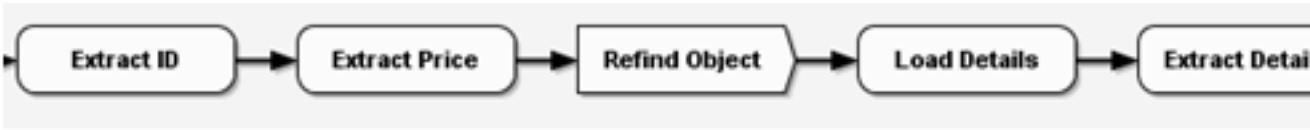
Change the model type from DatabaseOutputObject (Legacy) to OutputObject and remove the 6 hidden fields (robotId, robotRunId, refindKey, firstExtractionDate, latestExtractionDate, extractedInLatestRun). Consider which fields should be part of the ObjectKey.

Migration of Robots

The following steps must be completed to migrate your robots.

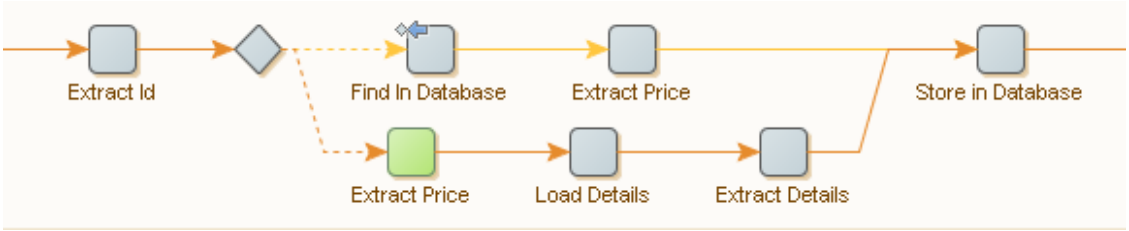
- Replace Return Object with Store in Database, and configure it appropriately.
- Replace Refind Object with Find in Database. Since Refind Object will update the object and stop if the object was found, the structure of the robot has to be changed a little.

Here is a classic scenario using Refind Object. The object Id is extracted (the only attribute used for refindKey in this example), then the price is extracted (not part of refindKey), then the object is refound. If the object exists, the price is updated (and the rest of the branch is skipped), if the object doesn't exist, the details page is loaded and the details are extracted and the object is then returned.



Refind Object Example

This should be converted to:



Using Find in Database

First we extract the id (Object Key attribute) then we use Find in Database to check if the object already exists. If the object exists the current record is loaded into our output object, and we now extract the price on-top of the previous value, and then we store the object again. If the object was not found by Find in Database an error is generated, and "Try Next Alternative" error handling sends the execution into the lower branch where we extract the price, load and extract the details, and then store the object.

The new method may require an extra Extract Price step, but it is much easier to see what happens, and there is no refind-object-magic.

If your robot has any Query Database or Execute SQL steps which query the storage tables, these may need to be updated as well.

Migrating Existing Data and Tables

If you have small amounts of data, or you collect all the data every time your robots run, the easiest is simply to delete (drop) the existing tables, and create new tables using the updated model objects.

If you have large amounts of data, or need the historic data from previous runs, you have to convert your existing data tables into the new table layout. It is possible to use tools from your database vendor to add/remove and rename the fields, but it is not recommended. The best solution is to use a robot to load the old data and store it using the new format.

To do this, create a table from you updated output object; you will have to rename the original data table unless you have given the new version of the object a different name. Then create a robot which loads the existing data using the Query Database step action, and stores the objects using Store in Database.

If you require the data history, like first and last Extracted date, you can extract these from the old row, and use the Execute SQL step to update the object after it has been stored.

If you have exposed the refindKey outside of your Kapow environment, you probably need to track which ObjectKey maps to an existing refindKey, and then provide the outside world with a mapping table, allowing them to convert existing refindKeys into the new ObjectKeys. This is important since you can't always generate an ObjectKey identical to the previous refindKey, since the robotId was often part of the refindKey calculation.

Migrating SQL Scripts

It is not possible to give a generic rule for how to migrate external SQL scripts, since the possibilities and complexity of such scripts are virtually endless. Instead we will try to give some guidelines.

Moving production data out of the harvest database

It is not a good idea to run user queries against the harvest tables (the tables the robots collect into), instead you should have a separate set of tables for the production data. These tables will contain the refindKey/ObjectKey, but not necessarily any of the other household fields. The production tables usually should be updated each time a robot has executed successfully.

Previously, the rows to move was often identified by using the robotRunId column to find the most recent run, but this column no longer exists. Instead we now have a column named LastUpdated, which contains a timestamp of when this row was last updated. This way all you need to do is find all rows which have been updated since you last moved data out of the harvest tables. If you have multiple robots harvesting data into the same table, you now need to use the robotName and not the robotId to distinguish data from different robots.

Scripts against RoboManager tables

The new database log offers superior information compared to the old RoboManager logs, some of these (like logging of Input Objects) are for internal use, but the rest are viewable in log view of the Management Console. If you have scripts containing queries against the RobotRun or Message tables in RoboManager, these will need to be rewritten to match the new logging layout.

Most significantly the robotId and robotRunId have been replaced with RobotName and ExecutionId. The advent of the executionId allows you to trace a specific run or message to an execution created using the API, something which was not previously possible. The new ROBOT_RUN table also contains a number of extra statistics, such as queue time, execution time and total runtime.

Beginner Tutorials

This section contains an overview of Kapow Katalyst followed by three tutorials that will guide you through your first project in Kapow Katalyst. Make sure to install and set up Kapow Katalyst correctly before proceeding with these videos. Click the videos, like the one below, to play them. Click the link in the bottom right corner of the page when you are ready to go to the next tutorial video.

On each page you will also find a transcript of the video.
Videos are not included in the PDF. Please visit open this url http://player.vimeo.com/video/62084944 in a browser to see the video.

This is the first of four beginner tutorials which will guide you safely through your first project with Kapow Katalyst. In this first video, you will get an overview of the workflow involved when working with Kapow Katalyst along with an introduction to the main program called Design Studio. Before watching these tutorials, make sure you have installed and set up Kapow Katalyst correctly. Please follow the relevant parts of the Installation Guide in the Getting Started section of the documentation at help.kapowsoftware.com.

Kapow Katalyst is very simply stated a platform which enables you to fully automate any process that you would be able to perform in a browser via your mouse and keyboard.

Please sit back and watch as you are taken through the general procedure from idea to automated process.

It all starts with an idea of a process you want to automate. In these Beginner Tutorials we want to automatically extract the most recent stories from a website called News Magazine.

Our first step will be to check out the website. What exactly do we want?

When we feel confident about what we want to achieve, we will open Design Studio, the program used to create the automated processes. The first time you open Design Studio you will get a welcome screen which links to this Beginner Tutorial, along with the rest of the documentation. Click OK and you will be able to see the main window of Design Studio.

On the left side we have the projects view. Right now it contains only the default project which includes a collection of example files and a Tutorials folder where samples of the files we will be creating in these Beginner Tutorials can be found.

Double clicking the file called Post.type in the projects view, opens it in the type editor, which is used to edit and create this kind of file. A type defines what kind of data can be stored in a variable of that type. If you are unfamiliar with types and variables, you can think of a variable as a bucket which can hold objects, like text or images, and the type can be thought of as the mold which produces that type of bucket.

This particular type is designed to contain the information we will be extracting from the News Magazine website. In the very last tutorial we will get into the process of creating a type.

The extraction of stories from News Magazine is performed by an automated process called a robot. Think of a robot as an automation of any process you would perform in a web browser.

Double clicking the file called NewsMagazine.robot, also from the Tutorials folder, opens up the robot editor, which is used to edit and create robot files.

The Robot View at the top of the editor displays the structure of the robot. Each step corresponds to an action performed by the robot. Going through the steps in the Robot View the robot loads News Magazine, navigates to the most recent articles and extracts a title and a preview of each story by using a loop. The robot then finally returns the collected values

Clicking the second step in the Robot View executes the Load Page action and we see that News Magazine loads in the Browser View below. As we will see later, the Browser View makes it really intuitive to build a robot.

The next tutorial will show you how to build this robot yourself.

Once we have automated the process of extraction with the robot, we will upload that robot to the Management Console. The Management Console is a web-based application for managing the operational aspects of Kapow Katalyst. From the robot we can create a Kapplet, which publishes the robot as an app for yourself and others to use.

The final product will be a Kapplet which automatically extracts the most recent stories from News Magazine and returns them for the user to view and download.

You are now ready to start building your first robot. Start the Beginner Tutorials by watching the Robot Beginner’s Tutorial.

Robot Beginner's Tutorial

What follows is a robot beginner's tutorial video, along with a transcript of the video.
Videos are not included in the PDF. Please visit open this url http://player.vimeo.com/video/61707847 in a browser to see the video.

General Introduction

This is the second of four beginner tutorials which will guide you all the way through your first project with Kapow Katalyst. It is advised to start with the Overview video before starting this tutorial.

You are about to learn how to build a robot in Kapow’s Design Studio. Please feel free to follow along on your computer.

Specific Introduction

In Kapow Katalyst, robots are used to automate processes that can be performed in a web browser. Robots can mimic and automate any set of mouse and keyboard instructions that you would otherwise have to perform manually.

In these beginner tutorials it is the goal to automate the process of extracting the most recent stories from News Magazine which is a site built specifically for these tutorials. There is a link to the site in the text associated with this video. (http://kapowsoftware.com/tutorial/news-magazine/index.html.) Under the tab Most Recent News, we find the three most recent articles on News Magazine. From these we want to extract their title and the short article preview that is given. We will design our robot to do all this.

Creating a New Robot

With Design Studio running, create a new robot by right-clicking the default project and choosing New>>Robot... A window opens, asking for the name of our new robot. Since this is our first robot we will call it MyFirstRobot.robot. Click next. Now direct the robot to the front page of News Magazine (http://kapowsoftware.com/tutorial/news-magazine/index.html) and click finish.

The robot editor opens and News Magazine loads. Notice that the new robot file has automatically been selected in the projects view on the left.

The Robot Editor

In the robot editor we have five different main views.

There's the browser view which shows us the loaded page exactly like we'd expect to see it in a browser.

Under the browser view, the html view shows the html of the loaded page.

At the very top there's the robot view where you can see the actions performed by the robot. Actions can be anything from clicking a link to storing data in a database. The active step is highlighted in green and steps to the left of the active step have been executed. Right now the End-step, which is the small round step, is the active step and the load page step has been executed. We will get back to the meaning of the End-step later on.

The step view on the right is used to configure the action performed by the active step. Since our current active step, the End-step, does not have any properties to configure, the step view contains only a description of the step.

Finally the variables view specifies any variables used by the robot for input, output or for storing data during execution.

The Browser View

Use the browser view to navigate to the page we want to extract from. A double click in the browser view corresponds to a single click in a regular browser. Double click the Most Recent News tab to get to the stories we want to extract.

Notice that a new step is created and executed in the robot view and the Most Recent News page is loaded in the browser view.

Scroll down the page and see the three news articles that we want to extract from.

If we single-click within the browser view, a green box appears around the HTML tag in which we have clicked. This green box marks the selected tag in the browser view. By selecting a tag and right clicking it, a menu appears which presents some actions that can be applied to the selected tag. We will use this in a moment.

Looping Through Tags

The next step is to make a loop which iterates through the three most recent articles. To do this we simply select any tag within the tag of the first article. Select for example the picture. Now, right-click in the selected tag and choose Loop>>For Each Tag. A For Each Tag step appears in the robot view, and in the browser view a blue box appears around the first article in our loop.

The For Each Tag step has arrows which can be used to iterate through the loop. Use these to iterate through the three articles and confirm that they are selected properly by the blue box. Clicking the arrows will not alter the robot in any way, but they can help us ensure that the loop has the expected iterations. Use the left-most arrow to return to the first article.

The function of the end step now becomes apparent. The loop will loop over every step bounded by the For Each Tag step and the End step. Note that we cannot add any steps after the End step.

Adding a Variable

Now, before we can extract anything we need to add a variable to contain the text we are going to extract. As mentioned in the Introduction tutorial I have already prepared a type called post to use with this robot.

Right click the white box in the variables view in the lower right corner and select Add Variable of Complex Type>>post. A window opens in which we may configure the variable. Let us keep the default settings and click OK.

A variable of the selected type now appears in the variables view. It contains the two attributes title and preview, which correspond to what we want to extract.

Extracting

Looking at the browser view again we are now ready to make the extractions. Click then right-click the title of the post marked with the blue box and choose Extract>>Extract Text>> title. Extracting within the blue box, called the named tag, ensures that the other iterations of the loop will extract the corresponding titles and previews from the other posts.

Now do the same for the article preview. Click and right-click the preview text and choose Extract>>Extract Text>> preview. Notice that the extracted text is now shown in the variables view. Also notice that two extraction steps have been added and executed in the robot view. Use the arrows on the For Each Tag step, while looking at the variables view, to observe that the text is extracted properly from the other posts as well.

The Return Value Step

To output the collected data we now need to insert a return value step into our robot. Right-click the end step and choose "Insert Step Before". This inserts a new step before the end step. Click the new step to ensure that it is the active step. The step action view has now changed to that of an empty step for which we can select an action.

Notice the warning symbol which appears over the step and over the step action tab. This is a warning that no action has yet been chosen for this step.

Click “Select an Action”, under the Action tab of the Step View, and choose the action Return Value. The Step View changes again to show the properties of the Return Value action. We see that the variable we added earlier has been chosen by default. The robot is now ready to be tested in the debugger.

The Debugger

Switch to Debug mode by clicking the Debug button above the Robot View.

The bottom part of the robot editor is now replaced with panels containing various tools to monitor the execution of the robot. The default tab in the main panel shows Input and Output. Run the robot by choosing Run from the Debug menu. The robot should now successfully execute and the output should be shown in the main panel. We have now successfully built a robot that extracts the three most recent articles from the News Magazine website.

If the execution fails for some reason, you can either redo the tutorial and check all the steps or check the NewsMagazine.robot, which is a robot identical to the one we have just built. The News Magazine robot can be found in the Tutorials folder in the default project.

Next step is to upload the robot to the Management Console and create a Kapplet. Move on to the Management Console tutorial or if you need help on specific topics go to help.kapowsoftware.com.

Management Console Beginner's Tutorial

What follows is the Management Console Beginner's Tutorial video, along with a transcript of the video. Videos are not included in the PDF. Please visit open this url http://player.vimeo.com/video/62068023 in a browser to see the video.

This is the third of four beginner tutorials which will guide you safely through your first project with Kapow Katalyst. It is advised to watch the overview video and complete the robot tutorial before following this tutorial.

You are about to learn how to use the Management Console to run robots as Kapplets. Please feel free to follow along on your computer.

The Management Console is a web-based application for managing the operational aspects of Kapow Katalyst. First of all, the Management Console acts as a repository for robots and types. It also enables you to create and manage Kapplets, which are robots that have been published as apps, easy to distribute and run.

This tutorial will show you how to upload the News Magazine robot to the Management Console, and publish it as a Kapplet.

Open Design Studio and MyFirstRobot, the robot we created in the Beginner Tutorial on Robots. Then ensure that the robot editor is in Design Mode by clicking the Design Button at the top left corner of the robot editor.

To upload the robot to the Management Console, select Upload to Remote Management Console from the Tools menu. Everything should already be correctly configured so just press Upload. The robot, along with any types associated with it will now be uploaded to the Management Console.

Click the link that appears to the Management Console. This should open your browser in the Repository of the Management Console.

The Repository contains any robots, types and other files uploaded to the Management Console. Observe that the News Magazine robot has been listed. Similarly click the Types tab to check that the associated type has been uploaded correctly.

To create a Kapplet from the robot, go to the Kapplets tab at the top of the page. This opens the Kapplet Repository, which is so far empty. There are also two other sub-tabs called My KappZone and KappZone. The repository is where you can add, remove and edit Master Kapplets and is only visible to administrators. The two other tabs are visible to all users and are available for you to test the user experience. My KappZone contains all Kapplets installed by the user and the KappZone the place from which the user installs Kapplets. My KappZone is therefore individual for each user.

To create a Kapplet click the plus at the top left of the repository. Call the new Kapplet `MyFirstKapplet` and click OK. The new Kapplet now appears in the repository.

We have now created an empty Kapplet which has been added to the list. Click the KappZone tab to view the Kapplet in the KappZone.

The new Kapplet is so far disabled, which is indicated by the diagonal lines. This means the Kapplet is only visible to administrators and needs configuration before it can be installed by users. Hover over the Kapplet and click the Design Kapplet icon.

A setup screen now opens for the Kapplet. It contains four sections which can be expanded one at a time. Branding is expanded by default. Here you can edit the name of the Kapplet, upload an icon for the Kapplet, and add a description.

Beneath the Branding section is the Robots section where robots can be added to the Kapplet. Click on the Robots section to expand it and click the plus. Now start typing “MyFirstRobot”. When MyFirstRobot appears in the dropdown, click it and click Add. Our robot has now been added to the Kapplet.

The Kapplet Input section is empty since our robot takes no input.

The Kapplet Output section, on the other hand, contains our post variable as expected.

Now close the Kapplet configuration window by clicking the “X” at the top left. This takes us back to the KappZone where our Kapplet now has the icon we assigned to it. The Kapplet is however still disabled, but as an administrator it is possible to test the Kapplet before enabling it to users.

Click install on the Kapplet. This will add an instance of the Kapplet to My KappZone. Click My KappZone to view it. To test that the Kapplet works, click on it and click Start. This will run the robot and present the result in a table. It is also possible to download the results as excel.

Now that we have seen that the Kapplet works, it is time to publish it to the users. First close the results, then navigate to the Kapplets Repository. From here you have to right click the Kapplet and choose to Enable Selected. This will enable the Kapplet for users.

Going back to My KappZone we can see that the diagonal lines disappeared from the Kapplet. Congratulations, you have now published your first Kapplet to the KappZone.

The next tutorial will teach you how to create a type in Design Studio.

Type Beginner's Tutorial

What follows is the Type Beginner's Tutorial video, along with a transcript of the video.
Videos are not included in the PDF. Please visit open this url http://player.vimeo.com/video/49855893 in a browser to see the video.

General Introduction

This is the fourth of four beginner tutorials which will guide you through your first project with Kapow Katalyst. It is advised to complete the other Beginner Tutorials before this tutorial.

You are about to learn how to build a type in Kapow's Design Studio. Please feel free to follow along on your computer.

Introduction to Types

In Kapow Katalyst, variables are used by robots to store data in. Robots use variables as input, output or to store temporary values during execution. These variables are categorized by types. Examples of types are text, image, PDF, number and so on. The given examples are all what we call simple types. That is, types which are predefined in Design Studio.

Alternately we can create our own Complex Types. Think of a complex type as a bucket of simple types.

Let me explain with an example.

In these beginner tutorials we have been automating the process of extracting and storing the most recent stories from News Magazine. From these three articles we extracted the title and the short piece of text that is given. The title can be held by a variable of the simple type short text and the preview by a variable of the simple type long text. We have to design our complex type to contain each one of these.

Creating a New Type

With Design Studio open, create a new complex type by right-clicking the default project and choosing New>>Type... A window opens, requesting a name for the new type. Since this is our first type we will call it MyFirstType.type. Click finish and the type editor opens. Notice that the type we just created has now been highlighted in the projects view on the left.

Adding Attributes

The most important part of the type editor is the list of attributes associated with our new type. Attributes describe the different values that a variable of our complex type can contain. Each attribute has a name, a type, and a list of other properties associated with it.

Let’s start by making the title attribute. Add a new attribute by clicking the plus sign in the lower left corner of the attribute list. A new window opens in which we can configure the new attribute. Name the attribute “title” and select for it the type “Short Text”. Short Text is a simple type which can contain text, no longer than one line. Click OK to add the attribute to our complex type.

Likewise add an attribute named preview to contain the short article preview. This attribute should be of the type Long Text which defines a text longer than one line.

What we have now made is a complex type from which we can make variables in a robot. Variables of the type MyFirstType will then be able to hold two values; title and preview. Save the type by choosing save from the file menu.

Changing the robot

MyFirstType.type is now exactly the same as the complex type post.type which we used in the Beginner Tutorial on robots. If we open MyFirstRobot.robot we can now switch the post variable for a variable with the type MyFirstType.

Do this by right clicking the post variable in the variables view and choosing edit variable. In the Edit Variable window that opens, choose MyFirstType as the type for this variable and click OK.

The last step is to save and upload the modified robot to the Management Console. It will automatically take the place of the previously uploaded version.

Congratulations you have now fully completed your first project with Kapow Katalyst.

Advanced Tutorials

This section contains tutorials on advanced topics of Kapow Katalyst.

Branches, Robot States, and Execution Flow

What follows is a video explaining the concepts of Branches, Robot State, and Execution Flow.
Videos are not included in the PDF. Please visit open this url http://player.vimeo.com/video/40930102 in a browser to see the video.

Video transcript:

This tutorial will explain how and why to use branches in you robots. In the process it will be necessary to introduce the concept Robot States and discuss robot execution flow in general.

If you have completed the beginner tutorials you will know that robot execution starts at the leftmost step and continues sequentially to the end step, where robot execution terminates. This is an example of a linear robot with no Branch Points.

Before showing you the branch point we have to introduce the concept of Robot States. At every step, the robot has various elements which make up its state. The most important elements are the currently open windows and frames and current values of variables, but the state also includes cookies, authentications and so on.

All of these elements make up the robot state.

Back in the robot view we have now introduced a branch point. It's been inserted by selecting Add Branch from the Edit menu in Design Studio.

The robot now sequentially executes each branch from top to bottom. Every time the robot reaches an end step, execution continues from the next branch.

So why go through the trouble of using multiple branches, instead of having a completely linear robot?

Well there are many answers to this question but the most important reason is that the robot reverts to its previous state every time execution goes back to a branch point. As we talked about before, state includes open pages, variable values, and so on, so every time the robot goes back to the branch point, it returns to the page it was on when it passed that branch point and forgets everything that happened in the branch.

Let me show you an example of how to use this.

I'm currently working on a robot which searches the site Momondo for travels to three different destinations but from the same departure city. The robot enters departure city into a form, then splits into three branches and enters three different destinations and clicks to search. Clicking the branch point we enter the state which the robot has when it splits into multiple branches. This is the state which the robot reverts to each time a new branch is executed. This means that the robot does not have to load the site and input the departure city three times, wasting time and CPU power. The robot simply rolls back and continues where it left off, entering a new destination into the form for each branch.

You can use this technique every time one page has to be handled in multiple different ways. We can even join these three branches again since they all use the same steps for the last part of the branch.

To redirect an arrow, first select it by holding Ctrl then clicking on it. Then drag the end of the arrow to the step to which you want to connect it.

You can also create a new arrow by dragging from the right side of one step to the left side of another.

As you may have guessed, you can make some pretty creative robot trees in this fashion, but don't panic! The execution flow is determined by one simple rule and one rule only.

Once execution reaches an end step, execution will continue from the next branch of the most recently reached branch point.

Let me repeat that for you.

Once execution reaches an end step, execution will continue from the next branch of the most recently reached branch point.

Once execution reaches an end step, execution will continue from the next branch of the most recently reached branch point.

It's pretty intuitive once you get the hang of it.

Okay I have a confession. There are other rules which govern execution flow and there is one exception to the rule mentioned before: For Each loops.

For Each loops include the For Each Tag action, the For Each Window action, the For Each URL action, etc.

If you have completed the beginner's tutorials, you have used a For Each Tag loop in your robot, and know how it works. Now we have a new way to think about For Each loops. You can think of a For Each loop step as a branch point where each iteration of the loop corresponds to a branch.

In other words: Once execution reaches an end step, execution will continue from the next branch of the most recently reached branch point or from the next iteration of the most recently reached loop step, whichever comes first.

Loops can be used very effectively in constellation with branches.

There are also other aspects which can make robot execution flow non-linear. One of the most prominent is Error Handling. When an error occurs at a specific step, the error handling of that step decides where the robot will continue execution from. Keep this in mind. To learn more about error handling click the question mark at the top right corner of the Error Handling tab.

So what if I want to keep some information from one branch to the next? Well, let's talk a little bit more about robot states, because not all elements are kept in the robot state. Global variables for example are totally linear in time throughout execution of the robot and never revert to earlier values when the robot rolls back to former states. This means that you can transfer information among branches or among iterations of a loop.

You can convert any variable to a global variable by checking the checkbox Global when adding the variable to your robot.

Also note that Try Steps look similar to Branch Points but they are not the same. Try Steps are only activated by error handling.

Looping Basics

What follows is a video showing how to implement basic looping in your robots, along with a transcript of the video.

Videos are not included in the PDF. Please visit open this url http://player.vimeo.com/video/38922811 in a browser to see the video.

Video transcript:

This video will give you an introduction to looping within your robots. In particular we will be looking at the types of loops which can be accessed directly from the Browser View.

Looping is both touched upon in the Beginner Tutorial Videos and in the video on Branches, Robot States, and Execution Flow. If you have no experience with looping, I highly recommend you to take a look at these videos before proceeding with this video. Take special notice of the way loops alter the execution flow.

The most useful robots are often those which perform a large quantity of actions, simply those which get a lot done. Often this includes performing the same operations in a number of similar cases. An example is the NewsMagazine robot used in the Beginner Tutorials. This robot uses a For Each Tag step to extract text from several blog posts. The For Each Tag step is just one of many loop steps to which the same logic applies. They all somehow let you perform the same procedure in a number of related situations.

The most basic loop steps are categorized as For Each Tag Loops, because they all somehow loop through tags in the current window. Let me go through these basic loops one at a time.

For Each Tag

The first of three loop steps we are going to discuss in this video is the loop literally called For Each Tag.

For Each Tag loops over each tag of a given name directly within the found tag. The first tag in the loop has been indicated by the blue box in this screenshot of the source view. I now overlay the screenshot with lighter blue boxes to show the following iterations of the loop.

For Each Tag is the loop step which I find myself using most often, simply because of its mixture of flexibility and ease of use. It is also the loop step which we used in the Beginner Tutorial Videos.

Whenever I need to loop over a selection of similar tags, like the products listed on this page, the first thing I try is to right click the first element that I want and select Loop >> For Each Tag. Design Studio then sets up a For Each Tag loop which loops through tags similar to the one I right clicked.

I can now iterate through the loop, using the arrows on the For Each Tag step, to view all the named tags formed by the loop. As discussed in the Beginner Tutorial Videos the blue box formed by a loop is called a named tag and is used as a point of orientation for Tag Finders of following steps, so if we insert a step which extracts the price of the first product, the following prices will be extracted likewise in subsequent iterations of the loop. This is the way all loops, considered in this video, operate.

Sometimes, however, it does cause problems to insert the loop, like here on vimeo.com. I right click the first element that I want and choose the For Each Tag loop, but the resulting loop only includes the topmost listed video. We can see this by trying to go to the next iteration. This results in a window opening to tell us that we have reached the last iteration of the loop.

To fix this I have to go directly to the configuration of the loop step where, in this case, I need to remove the specification of class to loop over. Design Studio guessed that we only wanted tags with the class top but really we want to loop over every listed item, independent of class. I delete the class specification and the loop now works as expected.

To use For Each Tag effectively you should study the different ways to configure the For Each Tag step.

For Each Table Row/Column

For the following loops it should be mentioned that they are often interchangeable and a given situation may be handled in any number of ways. I will try to teach you the basic principles of each loop type so you can be intelligent about which type you choose, but there is no single correct way of doing things.

The two next types of loop steps we will look at are both derivatives of the For Each Tag step, but they have more specific uses. They are called For Each Table Row and For Each Table Column and they respectively loop through rows and columns of a table. As with the For Each Tag step they have been conveniently implemented in the right click menu.

In this screenshot the first row is shown with the named tag marked 1 and the first column is the named tag marked 2. As you can see, combining the two types of loops will let you loop over every element in a table.

To insert a loop over table rows or columns, right click on any table element and select the appropriate action from the Loop submenu. You may either choose to include or exclude the first row or column.

I have now inserted a loop which loops through each column of the newest Ikea furniture. Notice that the name of the loop step is For Each Tag. Instead of having a unique step for looping through tables, the For Each Tag step has just been automatically configured to loop through columns in a table.

For Each Tag Path

The next type of For Each Tag loop is called For Each Tag Path. It is very similar to For Each Tag, which we just discussed.

The difference between the two is that For Each Tag Path loops over tags that are at any level inside the found tag whereas For Each Tag only loops over tags that are directly inside the found tag.

Sometimes the tags you want to loop over are not all directly inside one parent tag, or possibly the tags you want to loop over are all on different levels then you will need to use the For Each Tag Path loop. Notice in this example how the div tags looped over are all within a td and a tr tag and are therefore not directly within the found tag.

The easiest way to determine whether to use For Each Tag or For Each Tag Path is to look at the page structure in the Source View.

For Tags with Class

Just like For Each Table Row and Column were derivatives of the For Each Tag loop as is the For Tags with Class a derivative of the For Each Tag Path loop. As an example of the For Each Tag Path loop let me show you how to use this derived version.

Okay, now let us delete the table column loop we set up and take a look at the For Tags with Class. This loop iterates over all tags with the same value of their class attribute, which is often the case for tags with similar content. This time we have to be a bit more specific which tag we select before right clicking and we also have to keep an eye on the source view.

Usually using this loop goes something like this: As we click on the tags containing each product we look in the source view and notice that they all have the same class, namely productContainer. Once we realize this we can simply right click on one of the tags and choose Loops >> For Tags with Class >> productContainer. We then iterate through the loop to check that the named tags match our expectations.

Again notice that the inserted step is not called For Tags with Class but rather For Each Tag Path, which has simply been configured to perform the specific task.

For Each URL

The last loop we will take a look at is the For Each URL action, which is in a category by itself.

For Each URL simply loops through each URL inside the found tag. It is often useful if you need to extract or click on every link in a specific area of a page, regardless of the context of the link.

For Each URL is most easily inserted by selecting the tag containing the links you would like to loop over, then right clicking the selection and choosing Loop >> For Each URL.

I have now set up a loop which iterates through each URL in this article. It by default skips duplicate URLs.

Let's leave the For Each URL action at that. Just note that For Each URL has a number of configuration possibilities which can be changed in the step view.

Finally I have two notes that will help you when using loops.

Note 1

Just to spell out what I said in the video on Branches, Robot States, and Execution Flow: A For Each loop step, like any of those in this video, executes every subsequent step in the robot view for every iteration of the loop, so if you want your robot to continue execution beyond the loop, then you will have to insert a separate branch before the loop step. This branch will then be executed after the loop has finished.

Note 2

It is often nice to be able to break a loop or skip an iteration based on certain conditions. If we for example reach an iteration where one of the steps within the loop cannot be performed, it would often be logical to skip this iteration altogether.

As mentioned in the video on Branches, Robot States, and Execution Flow this can be done by adjusting the Error Handling of the step that fails. In the Step Error Handling View you can choose Next Iteration or Break Loop if an error occurs at this step.

At default this option is set to Skip Following Steps which corresponds to letting the robot hit an end step at its current execution position. In other words, if an error occurs at this step, the robot will go back and execute the next branch of the most recently reached branch point or the next iteration of the most recently reached loop step, however it will also cause an API Exception and Log an Error as indicated by the check boxes.

These were just the basics of looping. To learn more check out the Loops in Forms and Repeat-Next Loop videos. Also feel free to go to help.kapowsoftware.com to read about loops in greater detail.

Loops in Forms

What follows is a video demo showcasing some of the loop steps which apply to forms, along with a transcript of the video.
Videos are not included in the PDF. Please visit open this url http://player.vimeo.com/video/38922810 in a browser to see the video.

Video transcript:

This video will give insight into types of loops which are useful when working with forms. It is a direct continuation of the Looping Basics video, and will extend on the knowledge obtained there.

Many of the loops discussed in this video will have a form similar to the For Each Tag loops from the Looping Basics video but the form loops are all different from the For Each Tag loops in that they don't assign Named Tags for each iteration, instead they perform some other action for each element they work on.

For Each Loops in Forms

There are two For Each Loops and one other loop specifically designed to be used with forms. They are called For Each Radio Button, For Each Option and Loop Field Values.

I will be demonstrating each of the Form Loops on this library search page.

For Each Radio Button does what you would expect; it selects one radio button in a group for each iteration. The easiest way to insert the step is to right click a radio button and select For Each Radio Button in the Loop submenu. Iterating through the loop we see that each radio button is selected sequentially.

The next form specific loop is called For Each Option. It loops through options in a drop down list. Again it is easily selected from the right click menu. Once selected; a window appears asking whether you would like to skip any of the options of the drop down. This is useful for skipping any options that are not appropriate for what you want to do. In this case I will loop through all the options. Each option is now selected when clicking the arrows on the loop step.

The last loop I want to discuss in relation to forms is called Loop Field Values . This is not a For Each loop, which means that it is used slightly differently from the other loops I have described. It can be used on any text input field to loop through listed values that are then inserted into the field. One value for each iteration of the loop. Again we simply right click and select the Loop Field Values option. We can now choose which types of values to loop through. We can either choose one of the predefined options or we can compile our own list of values that we would like the robot to input. Since we are searching a library I will compile my own list consisting of Hemingway, Shakespeare, and Poe. Iterating through the loop we now observe that the stated authors are entered in to the text input field.

That concludes this video on Loops in Forms. To learn more go to help.kapowsoftware.com.

Repeat-Next

What follows is a video demo introducing the Repeat-Next loop, along with a transcript of the video. Videos are not included in the PDF. Please visit open this url http://player.vimeo.com/video/38922812 in a browser to see the video.

Video transcript:

This video will give insight into an advanced but very useful loop called a Repeat-Next loop, which works particularly well for looping through pages where each page leads to the next.

We will build upon the skills learned in the Looping Basics video, so make sure to watch that first.

Repeat-Next

This loop type is the oddball in the looping family, or maybe you could argue that it is not even part of the family because the way this loop is designed is totally different from all the other loop steps in Design Studio.

First of all the Repeat Next loop consists of two individual steps which need to be used collaboratively to have any kind of effect.

The concept is actually pretty simple: you place a Repeat step followed at some point by a Next step. When execution reaches the Next step, execution will revert back to the Repeat step and proceed execution from there, marking one iteration of the loop. If a Next step is not reached after the Repeat step then the loop will terminate.

The catch here, and also what makes the Repeat-Next loop so genius, is that while most of the robot state is reverted at the beginning of each iteration, the page reached at the Next step is actually transferred to the next iteration of the loop. So unlike the other loops we have looked at where the entire robot state is reverted at the beginning of each iteration, we can handle a different page in each iteration of the Repeat-Next loop.

Demo

A typical example of using the Repeat Next loop is when looping through multiple pages of search results. We insert a Repeat step … followed by a step which clicks to get to the next page … and then a Next step … It's as simple as that to loop through all the pages. We can now use the arrows on the Repeat step to iterate through the loop and observe that a new page is loaded for every iteration.

Of course we still need a way to terminate the loop. This can be done by setting the error handling of the Click step to “break loop”. If the click step does not find a link to the next page, we assume that we must have reached the last page and the loop breaks.

If we then want to perform some steps on each page, we can add a branch step after the Repeat step … and add a new top branch. In this new branch we can add loops and other steps without them being influenced by the Click and Next actions in the other branch. I can for example add a loop over all the search results on each page … extract information from each result … and return that collected information. In total I get a robot which extracts every result from every page of the search. We can get an idea about the execution flow by single-stepping through the robot in Debug Mode. I have sped up the recording so you can clearly see how all search results are extracted, one page at a time.

Remember the form in which the Repeat-Next loop is used here. A top branch which executes the steps you want to perform on each page and a lower branch which loads the next page and calls the next iteration of the loop. This constellation is useful and very common when using this type of loop.

Note that setting the error handling of a step to “Next Iteration” does not work with Repeat-Next loops. In order to proceed to the next iteration, a Next step must have been executed.

Try Step

This is a video explaining how to use the try step to set up conditions and error handling in your robots. Note that this video is quite advanced.
Videos are not included in the PDF. Please visit open this url http://player.vimeo.com/video/63734136 in a browser to see the video.

Video transcript:

This video will demonstrate how to use try steps when building robots in Design Studio. Applications of the try step are crucial for building robust robots acting on a dynamic web environment where changes in structure are commonplace. Use cases include trying multiple different strategies for interacting with or extracting from a website, and setting up conditional statements in a robot. We will go through two examples in this video, demonstrating the two use cases, starting with the latter.

As a first example we'll be using ExtractFromTable.robot, which is a basic robot that extracts data from a table on the News Magazine website. It can be found under Robots in the Examples folder of the default project. I open the robot and hide the projects view.

Once open, you will notice that the robot has only one path to follow in the robot view until it reaches this diamond shaped step called a try step. What a try step does is set up multiple alternatives for the robot to try. A try step can have any number of alternatives which are shown as dashed arrows emerging from the right side of the try step. If an alternative succeeds the robot continues as usual, ignoring any other alternatives from that try step. If an alternative does not succeed, however, the robot goes back to the try step and tries the next alternative.

To use try steps it is therefore important to understand what I means for an alternative to "succeed" or "not succeed"? Let me try to demonstrate with this example robot.

ExtractFromTable is a simple robot that extracts person information from a table by using a loop. Clicking the “for each tag” step and scrolling down reveals the table which the robot extracts from. Inside the loop, which loops through the rows in the table, the four pieces of information for each person are assigned to different attributes of the person variable. The try step is used to assign either true or false to the isMale attribute, based on the value of the last column in the table, called sex. The column can have either one of two values, “Male” or “Female” and we want to assign true or false to the isMale attribute respectively.

To test the value in the sex column, a Test Tag step has been inserted in the first alternative. Test Tag is a step which can conduct an action based on whether the found tag matches a specified pattern. Clicking the Test Tag step we see that the found tag is the cell in the sex column of the row from which we are extracting in the current iteration, and the pattern to match is simply set to "male" and only matches against text, ignoring case. If the pattern matches, the step then does what is specified in the Error Handling Tab. Under Error Handling the step has been set to "Try Next Alternative", which is also indicated by the small icon in the upper left corner of the step. This means that if the sex is male, and the pattern is thereby matched, the Test Tag step will do what is specified under Error Handling and try the next alternative. The second alternative therefore, has a step that assigns "true" to the isMale attribute. If the pattern is however not matched, the Test Tag step does nothing and execution proceeds normally to the next step, which assigns false to the isMale attribute.

In other words, an alternative can be said to succeed if no errors which specify to try next alternative are encountered, and can be said to not succeed if such an error is encountered. As soon as one alternative succeeds, all other alternatives are skipped.

In the example I just went through, the error was generated by the Test Tag step, but usually errors are caused by steps which are unable to perform their actions. This is most often because steps cannot find a certain tag or loading of certain content times out, but it could be anything that stops a step from performing its action.

Executing the example robot in debug mode you will see that the correct values are assigned to the isMale attribute. True values are indicated by check marks and false values are indicated by the absence thereof.

To give you an even better understanding of how the try step works, let me show a more complex, yet common, use case. In Design Studio I have this robot which needs to log into a site and extract some data, which is only available upon login. The page used by the robot is of course the familiar News Magazine page. The robot is called Login and is available from the examples folder. If you have closed the Projects View, it can be reopened from the Window menu.

Look at the robot view to get an overview: In the login process, the robot uses save and restore session in such a manner that the robot logs into the site and saves the session on the first run. Consecutive runs will then load the saved session instead of performing the login sequence again. At some point the session expires and the robot will have to log in and save a new session. Most robot executions will thus be able to restore a session and be able to skip the entire login procedure.

So… we have three different scenarios to test for: either it is the first time the robot executes and there is no saved session, or there is a saved session which is already logged in, or there is a saved session but it has expired.

Looking at the robot view, there are two alternative paths for the robot to take, depending on whether it is able to use the stored session or not. To show each case, let’s try simulating them by clicking through the robot.

The first time the robot is executed there is no saved session. The robot will first try the topmost alternative from the try step, but if I try executing the Restore Session step it will fail, since there is no session to restore. When this step fails, it will cause an error, triggering error handling which is set to “Try Next Alternative”. The robot will therefore have to execute the second alternative, which follows this lowermost

branch, loading News Magazine, clicking to get to the login page, entering username and password, clicking to login, saving the session for future executions of the robot, clicking again to get to the data we want to extract, and finally extracting the specific text. We have now simulated the first run of the robot.

Going back to the first step in the robot, let's simulate the second run. In all executions following the first, the session should restore successfully, so following the topmost alternative the session is restored, the robot clicks to open the page that we want to extract data from, and finally the data is extracted – simple.

We now want to replicate what happens if the robot is to execute after the login session has expired. To do this, open Logout.robot from the examples folder. The logout robot restores the saved session and clicks logout on the News Magazine page. In this way we are emulating expiration of the login session, rather than actually waiting the 10 minutes it takes for the session to timeout and expire on this specific page. Click the end step in the logout robot to execute all steps.

Going back to the login robot click the step named "Click site data" to make it the active step and click refresh in the toolbar to re-execute all steps up until the active step. After all the steps have executed again, notice that the page shown in the browser view looks as though we are still logged into News Magazine. This is because the saved session also contains the entire robot state, including the html of the page we were on when the session was saved. Restoring the session therefore does not properly refresh the page to show us its current state. Clicking on the extract step to execute the click step will however refresh the browser to show us that we are no longer logged in. The extract step therefore fails to extract the data we want, causes an error and tries the next alternative from the try step, which logs in and saves a new session for future use.

Thus the login robot handles all three cases and thereby ensures that the robot always logs in before extracting the data which is otherwise not available.

To summarize on what we have looked at in this video, from a try step spring multiple alternative routes for the robot to take. An alternative succeeds unless it has a step that causes an error and specifies to "Try Next Alternative" in the Error Handling Tab. Only the first alternative that succeeds is executed, the others are completely ignored. If an alternative does not succeed, the robot goes back and tries the next alternative.

Here are a couple of tips when using try steps:

As we saw in the first example, any condition steps, meaning steps with "Test" in their names, work well together with the try step.

If no alternatives of a try step succeed then the try step itself generates an error. Specify what action to take in the error handling tab of the try step itself.

You can give a try step a name by double clicking below it in the robot view.

When specifying to "try next alternative" under error handling for any step, you can also choose which try step to go back to based on their names. This means you can have nested try steps allowing for complex robot structures.

That concludes this video on the topic of try steps.

For more information on this topic, do a search on help.kapowsoftware.com

The Spreadsheet View

This is a video explaining the features of the Spreadsheet view. Below is a transcript of the video.
Videos are not included in the PDF. Please visit open this url http://player.vimeo.com/video/61006912 in a browser to see the video.

Video transcript:

As of Kapow Katalyst 9.2, robots have a whole new way of handling Excel documents. Instead of being converted into html-pages, Excel is now shown as a spreadsheet directly in the Page View in Design Studio. In this video I will give an overview of the features of the Spreadsheet View and take you through the process of building a robot which extracts from an Excel document. Feel free to follow along on your own machine.

An example of a robot which loads an Excel document can be found in the examples folder of the default project, and has the name excel.robot.

Go ahead and open the robot. Excel documents can be loaded in the same way as regular pages are loaded, either from a URL or from a file on your machine or server. In this particular robot it is loaded by clicking a link, which is performed by the step named Click PersonData.xlsx.

Click the Loop Rows step to view the document. The spreadsheet now appears in the page view, with rows and columns named and arranged like you are used to it from Excel. This particular document contains tables with person data.

There are two different sheets with 100 entries in each. It is possible to switch between the two sheets in the lower left corner of the Page View. It is also possible to see the document information by clicking the leftmost tab. In this video however, I will focus on Sheet1.

From the dropdown menu in the lower right corner of the page view it is possible to look at the unformatted values of the document or the raw formulas. Changing this view will also affect the extracted values when inserting a new step, so I am going to change it back to showing formatted values.

A cell is selected by clicking on it and it is even possible to select a range of cells by clicking and dragging, or clicking on a row or column name to select the entire row or column. The upper left corner selects the entire spreadsheet.

I will now delete the loop step and all the extract steps from the robot and demonstrate how easy it is to extract from an Excel document. I drag to select the steps, then hit the delete button on my keyboard.

To the new Spreadsheet View belongs a whole family of new step actions, which enable you to loop, extract, and test cells. Just like in the browser view these functions are available from the right click menu as I will show you in a moment.

First we want to insert a loop step which loops through all the rows of the table. First I click the upper left corner of the Spreadsheet View to select the entire spreadsheet. Then I right click inside the selected area – meaning anywhere inside the Spreadsheet View– and select Loop>>Loop Table Row>>Exclude First Row. I am excluding the first row since we are not interested in the header values.

The Loop in Excel step now sets the first row to loop over as the Named Range. Clicking the arrows on the loop step will show how the other rows are selected. It is now possible to extract from the Named Range and, because of the loop, corresponding values will then be extracted from all the other rows.

To extract first the ID, I right click the ID value and select Extract>>Extract Number>>ID. I click OK in the wizard that appears, which is already properly configured.

Likewise I can now extract the first name as a text into the name variable, the age as a number into the age variable and the gender as a Boolean into the isMale variable. The gender value is either true for male or false for female.

To show that the entire table can now be extracted, I switch to debug mode by clicking the icon in the upper left corner and clicking run in the toolbar. All 100 values from the Excel document now appear in the list of results.

Data Conversion

What follows is a video describing how to perform data conversion in robots.
Videos are not included in the PDF. Please visit open this url http://player.vimeo.com/video/31897133 in a browser to see the video.

Video transcript:

Sometimes when designing a robot we need to convert some text or numbers using simple or complex conversion rules. The question is: how do we do this easily in Design Studio?

In this video we will be talking about this field.

It may not look like much but it is actually a very powerful tool when building robots and it pops up everywhere in the Design Studio interface.

The field is called a Data Converter List, and for now let's regard it as a black box. It may take an input in the form of a value from, for example, a variable or extracted text; but input is not a strict necessity. The input, if present, is given by the placement of the Data Converter List and is always stated somewhere above it.

The Data Converter List has exactly one output, which is either stored in a variable or handled otherwise below.

In short, a Data Converter List is used to manipulate text and numbers within robots, but although it is possible to perform both text and number manipulation with the Data Converter List, input and output is strictly speaking always interpreted as text.

Some use examples include extracting the name from an email address,..

..multiplying a number by two,..

or adding three days and two hours to a given date.

Let's take a look at the interior.

As the name Data Converter List implies, the field contains a list which you can add a variety of different converters to by clicking the plus icon. These converters all manipulate the input in some way.

Here is an example where we have the Kapow Software website URL as the input. We start with no converters in the converter list so the output is the same as the input. By adding the Extract converter we can extract the middle part of the URL. Don't worry about how the converters work for now. We will get back to that.

Then we can add a converter which converts the text to upper case.

The two converters are then chained together in series so the output of the first flows into the input of the second.

The order in which the converters are listed represents the order in which they are performed. The order can be changed by clicking the arrows. In this case it makes no difference to the end result.

Converters can be deleted by clicking the minus.

Clicking the pen and paper icon opens a window used to configure the converter. This window also opens when the converter is first added to the list.

The Converter configuration window always contains the two important fields Test Input and Test Output which, as the names imply, demonstrate the input and output from the given converter.

Learn how to use the different converters by clicking the question mark in the configuration window. Clicking it opens the documentation. Here you can also read about Expressions and Patterns. Knowing how to use these is an invaluable skill when working with data converters.

For example if you need to use mathematical operations to manipulate a number, expressions are needed. I can multiply the input by two by using the converter called evaluate expression. Let's take a closer look at the expression used to perform this operation. It converts the input to a number and then multiplies it by two. Remember that all input and output from converters is text so we need to convert the input to a number before applying any mathematical operations to it.

Here are some examples of the use of data converter lists in Design Studio.

The data converter list you will find the most useful is probably the one located in the Extract step action. It allows you to convert any extracted text before storing it in a variable.

Let me show you an example of how to use this.

I'm in the process of creating a robot which extracts job offerings from LinkedIn. For now it simply loads the page and loops through the tags which contain job descriptions. I would like to extract the location of each job but it's stated in the same tag as two other pieces of information, namely company name and date. The three pieces of information are only separated by hyphens.

I start by extracting the entire text into a variable called location. I then select the Extraction step in the robot view and add an extract converter to the data converter list of the extraction step. The extract converter uses a pattern to decide which part of the input should be extracted and used as output. In this case we want to extract everything within the first and the second hyphen.

After finishing the pattern we can see that the Test Output field reflects the text we wanted to extract.

Selecting the end step and iterating through the loop we can see that the location has successfully been extracted from each of the job descriptions.

As you can see, converters are quickly implemented and extremely effective.

Lastly I'll give you a couple of tips on the use of data converters.

Tip 1: Many fields in Design Studio can be changed to data converter lists. This may be done by choosing Converters from the Value Selector located at the right side of the field.

Tip 2: One of the most useful converters is the Replace Pattern converter. It combines expressions and patterns to provide powerful text manipulation.

Tip 3: In addition to the restricted one input of data converter lists, additional variables can be fetched by using expressions. This allows us to combine data from several variables.

Thank you for watching this short introduction to data converters.

For more information on the available converters, consult the documentation at help.kapowsoftware.com under References >> Design Studio>>Data Converters.

Snippets

What follows is a video describing how to use snippets to share steps among robots.

Videos are not included in the PDF. Please visit open this url http://player.vimeo.com/video/32911272 in a browser to see the video.

Video transcript:

Creating complex robots can quickly become a messy affair which hogs precious screen real-estate. Putting steps into groups is an easy way to clean up such a robot and simplify the steps it takes to reach the end result.

To group steps you simply select the steps you want to group then press the group button in the tool bar above the robot editor.

Furthermore you can give each group a name when you create it. This makes it easier to keep track of the specific function of the group.

Collecting independent parts of a complex robot into simple groups makes the robot much easier to understand for yourself and others.

Naming groups to remember their insides is essential. We could for example have a group that logs into a site, performs a complex conversion, stores values in a database, or looks something up online.

Thinking about it, I actually have a login group in one of my robots which I would also like to use in some of my other robots. Now I could of course start copying and pasting this group into some of my other robots. That would solve the problem for now but if I needed to change the login procedure in the future then I would have to copy the modified group step into all my robots again.

Let me introduce the snippet concept. A snippet is created and edited as a group step but is stored in a separate file and can be used as a custom step in as many robots as you want.

Having the step information stored in a separate file from the robots, means that changing the snippet in one robot will change it in all of the other robots as well. This can save a lot of time and greatly simplify your robots.

…In addition to containing steps, snippets can also include a list of variables used by the snippet.

It could for example be a login variable containing username and password. This variable is then automatically available in every robot using this snippet.

CREATING A SNIPPET

Let me show you how this all works in Design Studio.

Here is a robot which logs into Zoho CRM, looks up some contact information, extracts it and returns it. The login procedure of this robot has been grouped and would be perfect to have handy for other robots which are also to perform tasks on Zoho.

Creating a snippet from the steps that perform the login sequence couldn't be easier. I simply select the group I already made. Then I select Convert Group to Snippet from the tool bar above the robot editor.

Design Studio now prompts for a name and I choose the name `LoginZoho` which was already the name of the group.

The snippet is then created which is visualized by the small snippet icon in the lower left corner of the box.

Furthermore the newly created snippet file has been selected in the projects view and the snippet has also been opened in the snippet editor in a new tab. The snippet editor will open every time we modify our snippet in a robot.

I switch to the `LoginZoho` snippet editor. The snippet editor looks very much like the robot editor. It has a snippet view showing the steps of the snippet, a step view showing the action performed by the active step in the snippet view and a variables view as we know it from the robot editor. Instead of the browser view, the snippet editor has a configuration view where we can write a description of the snippet.

Note that in the snippet editor we can only make modifications to the configuration view and the variables view. If we want to make modifications to the steps in the snippet we will need to do that in the context of a robot editor.

As you may have noticed when looking at the snippet view, the two steps which use the Login variable have been marked with error indicators, pointing out that the variable is not yet present in the snippet.

Accordingly, the last thing we have to do to complete our snippet is to add the Login variable to the snippet. This will ensure that the login credentials will be available to any robot which uses this snippet.

I add it by simply right clicking the variables view and choosing the appropriate type, exactly as I would have done in a robot. In the variable configuration window that now appears I have to ensure that I configure the variable to match the one in the robot precisely. The variable name is correct so I just need to check Use as Input and add default values to the attributes… then I click OK.

Now we have a fully functional `LoginZoho` snippet.

USING THE SNIPPET

Let me show you how to add the snippet to another robot. Here's a new robot which I also want to perform some task on Zoho. A snippet step is inserted by choosing Insert Snippet Step Before from the tool bar.

An undefined snippet step now appears in the robot. I then choose the `LoginZoho` snippet from the dropdown in the step view of the snippet step.

The snippet has now been inserted into the robot and can be executed by clicking the end step…

As seen in the variables view, the login variable from the snippet is now available for the robot to use. The small snippet icon to the left of the variable indicates that this variable belongs to the snippet. That means that we cannot edit the variable directly in the robot, but only in the snippet editor. It also means that removing the Login snippet from the robot will remove the variable as well.

If we want the variable to be in the robot permanently then we have to add it manually to the robot, giving it the same name, type and configuration as the login variable from the snippet. This new variable will then take the place of the snippet variable.

Now, if I make a change to the snippet in any of its instances, all the other instances will also be changed. If I for example switch the places of the Enter Username and Enter Password steps and then go back to the first robot, we see that the order of the two steps has been changed here as well.

TIPS

To round off this video let me give you some tips when working with snippets.

TIP 1

Often you can end up with robots having huge variables containing everything needed for that robot. When creating snippets you should ultimately split these variables such that only the attributes needed in the snippet will be included in the snippet variable.

In general you should try to create your types based on function rather than making them robot specific.

TIP 2

To make the snippet as independent from the rest of the robot as possible, make sure to put any non-default robot configuration directly into the steps in the snippet where necessary.

In other words if you have clicked here, then here and made any changes that are important to the steps in the snippet, then make sure to click here in step view for the steps of the snippet and make the same changes.

TIP 3

Always document the context of the snippet by writing a description in the snippet configuration view.

That is all for this video. If you want to learn even more about snippets you can consult the documentation at help.kapowsoftware.com.

Date Extraction - Simple Case

What follows is a video showing how to modify the News Magazine robot to extract article dates, along with a transcript of the video.
Videos are not included in the PDF. Please visit open this url http://player.vimeo.com/video/40930103 in a browser to see the video.

Video transcript:

Thank you for using Kapow Katalyst.

This tutorial will show you how to easily extract dates from any website using the Date Extraction step in your robot. The tutorial consists of two parts. This first part is a follow-along tutorial with a simple example, the second part is a real case scenario showing a trickier example.

Please feel free to follow along on your computer for this first part.

If you have completed the beginner tutorials you might remember the News Magazine robot. It extracts the most recent stories from News Magazine which is a site made for tutorial purposes. We want to modify the robot to also extract the date and time from the most recent articles.

Start Design Studio and open the Type called post from the Beginner Tutorials folder in the default project. Add a new attribute called date and give it the type Date. Now save the Type, close the Type Editor and open the NewsMagazine robot from the same folder.

Click the Return Value step in the robot view. The robot will now execute to this step. I am going to close the projects view and the source view to get some more space to work with.

Scroll down in the browser view and locate the date and time given above each picture. We are going to add a step which extracts the date and time from each of the three articles.

To do this we right click the tag containing time and date and select Extract >> Extract Date >> then choose the variable post.date.

A new window opens which will help us extract the date into the standard date format which is the only format accepted by the simple type Date and thus by our post.date variable.

Let's look at the important parts of Extract Date Configuration window. The field Test Input shows the raw text as extracted. In the field called Pattern we will write the pattern of the date exactly as it is stated

in the extracted text. Right now the pattern is "dd MM yyyy" which means that the date consists of date, month and year, separated by spaces.

We see that this corresponds to the format of the date in the extracted text. Design Studio has deduced the pattern for us so if we just want to extract the date we don't have to modify anything. The field Test Output shows us the date in the standard date format as extracted from the Test Input.

Let's say that we also want to extract the exact time that each article was published. This means that we have to expand our pattern to also capture this information. Keeping an eye on the Test Input, add " * hh:mm" to the Pattern. Notice that the Test Output changes to incorporate the time of day.

Let me explain the pattern we added. Spaces and the colon correspond to their respective characters in the Test Input. Asterisk corresponds to "at" in the Test Input but can in general be used to represent any number of non-whitespace characters. "hh" means hours and "mm" means minutes. To get a full list of things to put in the Pattern field you can click the question mark at the top right of the window.

Click OK. We have now successfully extracted the time and date from each article. Test the robot in Debug Mode to confirm this.

Date Extraction - Tricky Case

What follows is a video demo showing how to extract dates when date information is spread into multiple places, along with a transcript of the video.
Videos are not included in the PDF. Please visit open this url http://player.vimeo.com/video/27240628 in a browser to see the video.

Video transcript:

This video will show you how to extract the date and time on a site that has date information spread into multiple places. It is recommended to complete the tutorial on Simple Date Extraction before watching this video.

I have made this robot to extract my Skype call history. It logs into Skype and loops through a table which contains my history. The only problem is that the year is not stated directly in the date and time column but only in the blue bar above. Somehow I have to combine the two pieces of information. Luckily this is easily done using converters.

Before entering the loop I add a step which extracts the text in the top bar into the variable called Year. Inside the loop I insert a new step which I choose to have the Extract action. Then I select the date tag from the first row of the table and use the yellow square button to the right of the Address Bar to use it in the Extract step.

Now, to combine this extraction with the Year variable, I add a converter under the Action tab of the extract step. I choose the converter called Evaluate Expression which allows me to append the value from the variable Year to the extracted date.

An Evaluate Expression Configuration window opens. The Test Input field shows the date as extracted. In the Expression field I simply write "INPUT" with capital letters, which is the extracted date. I follow this by a plus and a space in quotation marks, this adds a space after the date. Then I add another plus followed by "Year", representing the variable containing the year which we extracted earlier.

Looking at the Test Output I verify that the two extractions have been combined into one text. To learn more about expressions I can click the question mark next to the Expression field. Click OK.

I now add another converter, the same as used in the Simple Date Extraction Tutorial, called Extract Date. The Output of the previous converter is used as input for this converter.

The configuration window opens for the Extract Date converter and I insert a new Format pattern and delete the default one given by Design Studio. Now, just as in the Simple Date Extraction Tutorial, I add the pattern to extract the date from the test input and let the converter convert the date to the standard date format. "MM dd hh:mm MM yyyy". The month is given two times but that is not a problem. The first occurrence will simply be ignored.

I click OK, choose to extract into the Date variable and check that the date and time has been extracted successfully.

That concludes this tutorial and demo on extraction of dates.

Kapow Compute Units (KCUs)

9.1 introduces a major change in the licensing for Kapow Katalyst, with the move away from CPU core based pricing to a capacity based pricing based on Kapow Compute Units (KCUs), completely independent of the chosen hardware configuration.

What are KCUs?

A KCU is a Kapow Compute Unit and is defined as a unit of measure for how many operations (or steps) a Kapow Katalyst RoboServer can perform in one second (and is unrelated to underlying server capacity).

A step is the smallest unit of action which can be performed within a RoboServer. Examples of steps are, loading a web page, writing a data record to a database, or performing a transformation on a data element.

One KCU represents a total of 5000 KCU points per second. The number of Kapow steps that make up one (1) KCU depends on the type of Kapow steps involved, as each step type consumes a different amount of the KCU. The steps are divided into groups the most important groups are listed here:

1. Steps that both do I/O and execute JavaScript, 10000 KCU points
   E.g.: 2 page loads per second with 4 KCUs

2. Steps that do either I/O or execute JavaScript (but not both), 1000 KCU points
   E.g.: 20 Call REST Web Service Steps per second with 4 KCUs

3. Extraction and transformation steps 1 KCU point
   E.g. 40,000 extract or assign steps per second with 4 KCUs

(The complete list is available in Design Studio by clicking the Help->KCU Information menu item)

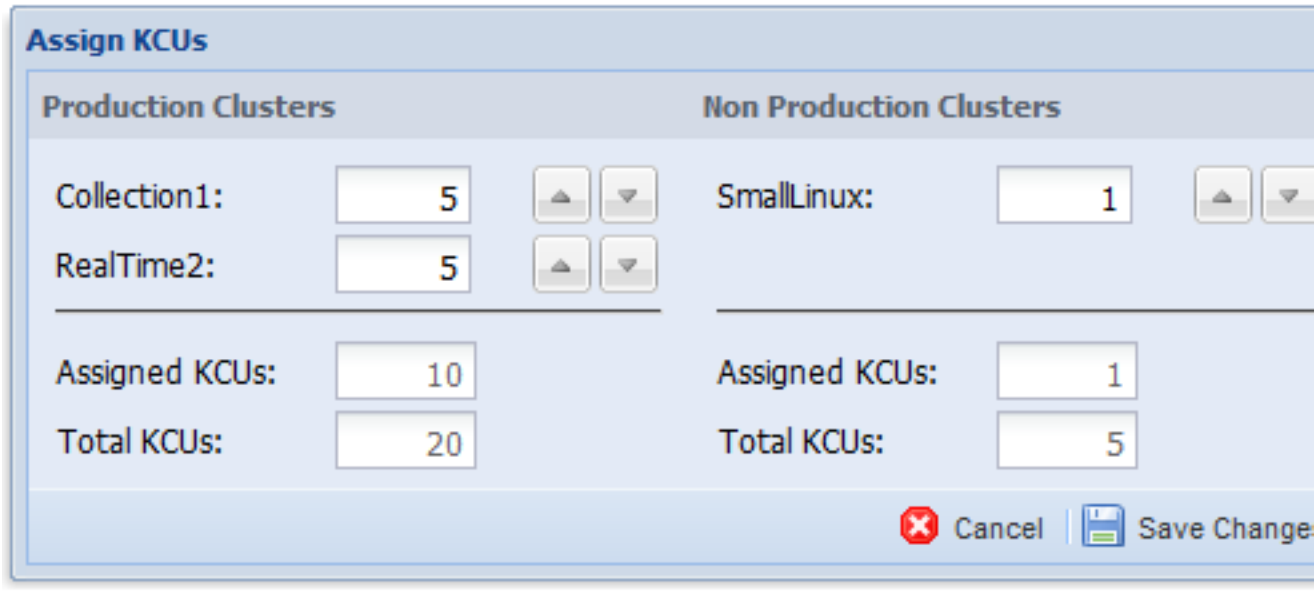
The above is provided that you have enough CPU power and sufficiently low response time from the source server. (Note: We have empirically measured the average page load time on a powerful CPU over the 23000 most visited web sites to 6.7 seconds).

The total number of KCUs a robot uses can be seen in the Design Studio Debugger Summary information after running the robot.

Deploying and Assigning KCUs

With KCUs we can deliver the needed computing capacity independent of the target hardware environment and physical CPUs. This allows the flexibility of choosing, cloud-based or on-premise, virtualized or physical CPU environments without the need to control the number of allocated CPUs.

The available KCUs are assigned to the RoboServer Clusters in the Management Console and automatically distributed between the RoboServers available within the Cluster.


Assign KCUs
Production Clusters
Non Production Clusters
Collection1:
5
RealTime2:
5
SmallLinux:
1
Assigned KCUs:
10
Total KCUs:
20
Assigned KCUs:
1
Total KCUs:
5
Cancel
Save Change

License Keys

Before you can begin using your Kapow Katalyst installation, you must enter your license key(s) into Management Console as described in Management Console (License Server). You will have received these keys from Kapow.

There are three different kinds of license key:

Production key	Permits production use of the Kapow Katalyst system.
Non-Production key	Permits use of the Kapow Katalyst system for non-production use like test and staging.
Developer Seat key	A special kind of non-production key that will allow you to run all the Kapow Katalyst programs on your own computer as part of the same installation. There are, however, some performance limitations, as this it intended to be used only for development or as a trial installation.

A license key (no matter which kind) also describes which Kapow Katalyst features you have, and how many KCUs you have bought. A Developer Seat key always contains 1 KCU, while the two other kinds may contain more.

If you have both a Production key and a Non-Production key, you may install them into the same Kapow Katalyst system (that is, in the same Management Console). You may also choose to set up one system for each key. In both cases, you will need to set up at least two different clusters (either on the single system or one cluster on each of the two systems), configured as Production and Non-Production, respectively. The KCUs in your licenses can then be assigned to these clusters.

In no case should a key be entered into more than one Management Console.

Design Studio User's Guide

Design Studio is the application for creating robots and types. In Design Studio, you can also debug your robots and create database tables for types that need to be stored in databases.

Design Studio is an integrated development environment (IDE) for robot development. This means that Design Studio is all you need for designing robots and types.

Robots are programmed in an easy-to-understand visual programming language with its own syntax (structure) and semantics (meaning). To support you in the construction of robots, Design Studio provides you with powerful programming features including interactive visual programming, full debugging capabilities, an overview of the program state, and easy access to context-sensitive on-line help.

Design Studio also lets you create the types that are used by robot variables for data extraction and input. With Design Studio's Type Editor, you are able to design types that are *modeled* after real-world data. In the most common case, a type is designed to hold the data that a robot extracts from a data source.

Organization

The Guide is structured as follows: First, you are introduced to the essential concepts of Design Studio. Then you are taken on a tour of the user interface and provided with an overview of the core building blocks of any robot. With the basics firmly in place, we get to the tutorials that show you how to use Design Studio to create robots that do something useful. The tutorials get gradually more advanced until, finally, you are ready to create robots that perform the tasks that you decide. The tutorials are the meat and bone of this User's Guide and it is important that you master them before proceeding.

The rest of the Guide is divided into various topics ("How To..."). You should skim through them to get an idea of what they cover. Then you can return later when you need more information or help on one of the topics.

Before You Read On

Before you proceed, it is recommended that you read the Getting Started section, which will introduce you to Design Studio and the context it is used in, help you get Kapow Katalyst installed, and take you through a couple of basic tutorials.

Also, before proceeding, make sure that you fulfill the following reader requirements:

- A basic understanding of what programming is.
- A basic understanding of HTML.
- A basic understanding of JavaScript.

Other Resources

Additional information on Design Studio is available in the reference documentation [../ref/robomaker/RoboMaker.html], which is easily accessible from the Help menu in Design Studio.

Design Studio Concepts

Design Studio is a programming environment for creating robots and designing types. Robots are created using a special-purpose programming language with its own syntax and semantics. Like other

programming environments, Design Studio uses several concepts that you, as a robot designer, must understand in order to fully comprehend the workings of Design Studio. The purpose of this section to introduce the most important concepts and it is recommended that you refer back to this section whenever you encounter a concept you do not understand. Please be aware that you may not understand all the Design Studio concepts described below right away; they will become clearer as you explore Design Studio and start creating robots.

Robots

The most important concept in Design Studio is a **robot**. A robot is a program designed to accomplish some task involving a data source, usually a web site, but it could also be an Excel document or a database. Typically, one robot is written per task per data source. For example, you would create one robot for extracting news from http://cnn.com, another robot for extracting news from http://yahoo.com and yet another robot for extracting product information from an online product catalog.

Basically, a robot can be programmed to do (automatically) everything you can do in a browser, as well as extract data from a database or an Excel document and combine this with storing data in a database, file, etc.

The Robot State

When a robot is executed, it works on a **robot state**. The robot state consists mainly of four elements:

- windows
- variables
- cookies
- authentications

The **windows** element is the currently open windows, each containing a page. This page could be an HTML page, a spreadsheet, an XML page etc. and we say that the page has a given *Page Type* depending on what type of page is loaded into the window and the look of the Page View and the steps you can insert in your robot will depend on this type. At least one window is always open, and one window is marked as the current window. The **variables** element contains the current values of the variables. The **cookies** and **authentications** elements are the HTTP cookies and authentications, respectively, received during communication with a web server.

Steps

A robot is made up of **steps**. A step is a building block in a robot program. There are four types of steps: Action steps, Try steps, Group steps and End steps.

A Step

A step works on a robot state and processes it according to the configuration of the step. A step thus has an input robot state and generates an output robot state. The only exception to this is the End step. End steps mark the end of a branch in a robot and not as such the end of a robot, i.e. the the robot does not necessarily stop execution after an end step. End steps are the only steps in a robot that does not have outgoing connections.

An End Step

Steps may have properties, such as a step name, a list of tag finders, a step action and an error handling. While Action steps have all these properties, other types of steps only have some of these.

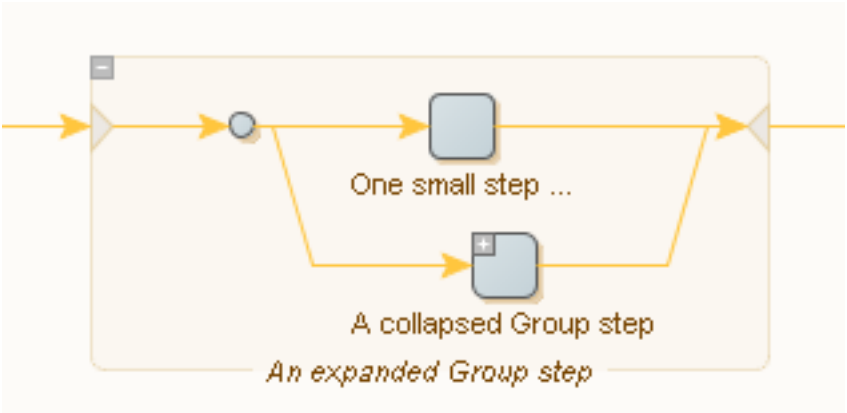
The **step name** provides a symbolic name for the step, such as "Extract Headline" and "Load Search Page". In the above robot, the step name is "MyStep".

The **finders** find the elements (e.g. HTML tags or Excel cells) in the page that the step action (see below) should work on. Some step actions require a single element, whereas others can handle more than one element. Some step actions accept no elements at all. There are two kinds of finders: **Tag Finders** that finds tag in HTML or XML pages and **Range Finders** that finds cells in Excel pages.

The **step action** is the action that the step performs. The action is the "heart and brain" of the step, and it is the selection of the right step action that is the challenge of robot writing. For example, an Extract action might extract the text from a tag in an HTML page and store it in a variable. And a Click action might load the URL residing in an <a>-tag and replace the page of the current window in the robot state with the newly loaded HTML page. An action usually changes the robot state. For example, the Extract action changes the variables, and the Click action may change the pages/windows, the cookies and the authentications.

A step can be **executed**. A step that is executed accepts a robot state as input and, by applying the finders and step action in turn, produces an output robot state. The output robot state is then passed on to the following step and becomes its input robot state. Some step actions are termed **loop actions** (and steps having such actions are called **loop steps**). A loop step may generate zero or more output robot states, and causes the following steps to be executed once for each of them.

Steps can be grouped together in Group Steps. Group steps can be expanded or collapsed. The figure below shows an example of an expanded Group step with a collapsed Group step inside it. The icons and in the top left corner of Group steps is used to expand or collapse these.



Group Steps

A step is **valid** if it has been properly configured so that execution can be attempted. For example, if a step has no action, it is invalid since execution cannot be attempted.

A step definition also specifies error handling for the step, but this is more complex and is described separately in a following subsection.

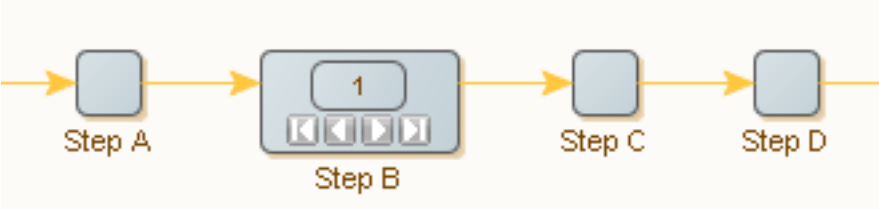
Connections and Execution Flow

Steps can be connected to other steps via **connections** that affect how execution flows between steps. Consider the simple robot below:

This robot consists of three steps named "step A", "step B", and "step C". Assuming that no errors occur, and that each step generates exactly one output robot state, the robot is executed as follows: An initial robot state will be generated and used as input to step A (being the first step). Step A will produce an output robot state. This output robot state will be the input robot state of step B. Similarly, step B will produce a robot state and this will be the input robot state of step C. Once step C has executed and produced an output robot state, execution completes. In short, the execution of steps can be described as follows: "A, B, C".

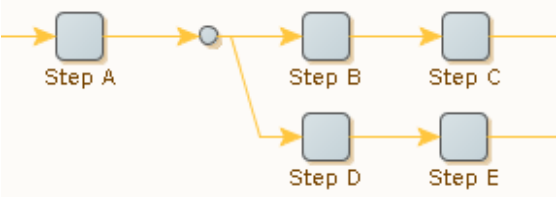
Sometimes, a step generates no output robot state when executed. This happens when an error or a **test step** causes execution to continue somewhere else in the robot, and it is described in the section on Conditions and Error Handling.

Steps containing a loop action may process the input state several times, each time outputting a distinct robot state. Consider the robot below where step B contains a loop action:



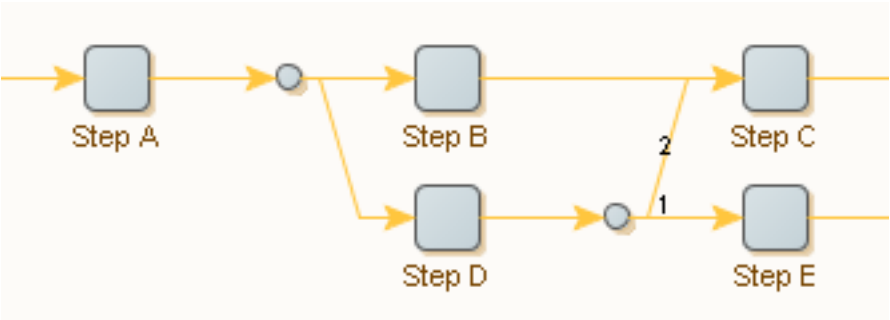
Assuming that there are no errors or test steps, that step B outputs three robot states, and that all other steps output exactly one robot state, then the steps will be executed in the following order: "A, B[1], C, D, B[2], C, D, B[3], C, D", where B[*N*] refers to the *N*th iteration of the loop action contained in step B. Note that the output robot states by step B will be different robot states — that is, each iteration will output a new robot state. Hence, step C will receive a new input robot state each time it is executed.
Videos are not included in the PDF. Please visit open this url http://player.vimeo.com/video/40930102 in a browser to see the video.
Video Tutorial on Branches, Robot States and Execution Paths

A step can connect to more than one step. This is called **branching**. Consider the robot below:



In this robot, step A is followed by a **branch point**, where the connection splits out in two **branches**. One branch consists of step B and step C, and another consists of step D and step E. All of the branches coming out of a branch point are executed, one after another. Therefore, assuming that no errors or test steps change the control flow and that each step generates exactly one output robot state, then the robot above will be executed as follows: A, B, C, D, E. However, it is important to note that step B and step D will each receive a copy of the same output robot state produced by step A.

Branches can merge, and in complicated ways. Consider the following robot:



This robot illustrates how connections can be explicitly ordered. In this robot the branches of step D are executed in the order specified by the numbers, that is, step E is executed before step C. If an order is not specified (by numbers), connections are executed top-down. Thus, assuming that there are no test steps, that no errors occur, and that each step generates exactly one output robot state, the robot is executed as follows: A, B, C, D, E, C. The first time step C is executed it will receive the output robot state produced by step B; the second time step C is executed it will receive the output robot state produced by step D.

Sometimes you want to select (i.e. execute) only one of several branches, depending on circumstances. The following section shows how to do this.

Conditions and Error Handling

A robot may have to use different approaches in different cases. The cases may be distinguished either based on explicit tests, that is, by evaluation of conditions, or because errors occur and need to be handled.

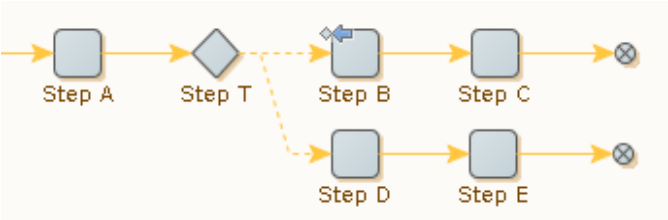
Conditions change the flow of execution based on the content of the input robot state (e.g. the presence of a particular tag in an HTML page). Error handling is about changing the flow of execution when particular errors occur (e.g. some anchor tag is not found on the HTML page as expected and thus cannot be clicked). Often, however, a situation can be seen both ways: An anchor tag should be clicked if found (this is a condition), or the robot can just try and click it and handle the error (if it is not found). In some cases, what is commonly thought of as a condition is too complex to be written up as such, for example a condition saying "if this particular page can be loaded without error". In such a case, there is nothing to do but try and load the page and treat any error merely as an indication that the condition failed.

Other errors are signs of genuine problems with the robot or the web site being accessed. For example, the web site may be down and cause a page loading error, or a tag finder might fail to find a needed tag due to a dramatic page layout change of an HTML page. To sum up, a particular error may be considered a failed condition in some circumstances, and a genuine error in other circumstances. The interpretation depends on the robot.

Because of this blurred boundary between conditional execution and error handling, Design Studio provides both features in a unified way. For every step, it is configurable what to do in case an error occurs. Furthermore, steps with a **test action** (based on a condition of some sort) reuse the same approach, meaning that if the condition is not met, the (default) action is to do as if an error occurred.

For each step in the robot, you can configure the desired reaction to errors. We will describe two of the most useful error handling options here; see How to Handle Errors for information on the other options. The first to be described is closely linked to the Try step.

The **Try step** is similar to a branch point because it may have several branches going out from it. It differs from a branch point because branches beyond the first one are executed only if a step on the preceding branch encounters an error which it then handles by means of the option "Try Next Alternative". Consider the robot below and assume that each ordinary step is expected to output exactly one robot state:



The icon indicates that step B is configured to handle errors by "Trying Next Alternative".

If step B executes successfully, step execution is as follows: "A, T, B, C". Because the first branch going out from T executes without error, the second branch is not executed at all.

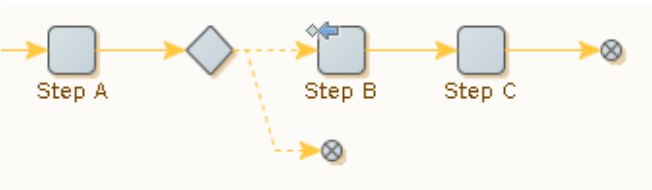
If, on the other hand, step B encounters an error, then the execution of steps is as follows: "A, T, B, T, D, E". After the error in step B is handled, execution will not continue at the following step, but instead at the beginning of the next branch going out from the Try step.

Each branch going out from a Try step represents one possible way to proceed from that point. Steps near the beginning of each branch probe if execution along the branch is a viable approach (and otherwise effect a "Try Next Alternative"), while later steps do the actual work when the branch turns out to be the right one for the case at hand. The probing steps near the beginning of a branch may be either test steps, or just any kind of steps that — if they encounter an error — indicate that this branch is not the way to proceed. There may be any number of such branches going out from a Try step.

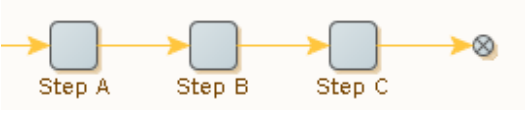
Readers familiar with ordinary programming languages like Java, JavaScript, C# or similar will note how the above robot is similar to an "if-then-else" construct: The first branch after the Try step contains the condition (the "if" part) and the "then" part, while the last branch contains the "else" part. Should there be more than two branches, then the ones between the first and the last ones are like "else-if" parts.

If the first branch attempts to do some action that may error, the example can also be likened to a "try-catch" construct: The first branch is the "try" part, while the second branch is like the "catch" part.

Another error handling option, "Skip Following Steps" provides a more compact way of expressing a common special case, which is exemplified by the following robot. The step that can encounter an error is the first one on the first branch, and the second branch does nothing:



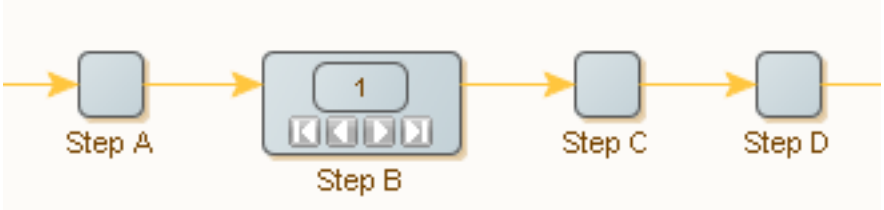
The effect of this is to simply skip execution of the steps after step B if it encounters an error. The same effect can be achieved without the Try step by using the error handling option "Skip Following Steps" (which is the default), in the following way:



Location and Location Code

When an error is handled, it is possible to report it back to the caller of the robot, or to have it logged. In both cases, a message is included that briefly describes the error, together with a location and location code for the step that encountered the error.

The **location** of the step that encountered the error is the list of steps (including iteration numbers) one needs to execute in order to reach that step from the first step. Consider the robot below:



If step C reports an error on the second iteration of step B, then the location is written as: "step A - step B[2] - step C". Note that the location contains the step names and iteration numbers, separated by hyphens. Branch points are omitted.

The **location code** is similar to the location, but the name of each step is replaced by a unique identifier for that step, thereby avoiding name clashes. For the location example above, the location code may be: "{a-i1-a}". You can use the location code in Design Studio to go directly to the step that reported the error (using "Go To Location" in the "Edit" menu).

You should be aware that the iteration number in the location and location code is 0 indexed, so the first iteration will be "{a-i**0**-a}

Snippets

A snippet is a group of steps that can be reused in several robots. A snippet is maintained in a file separate from the robot. Whenever the contents of a snippet is changed in one robot, it is automatically updated in other robots that uses the same snippet. A snippet is inserted into a robot using the Snippet step, and edited in-line. Snippets contents cannot be edited without being inserted into a robot. Watch the video below or read on to learn more.
Videos are not included in the PDF. Please visit open this url http://player.vimeo.com/video/32911272 in a browser to see the video.

A video introduction to the concept of snippets.

The Snippet step inside a robot is in many ways similar to a Group step. However, whereas the steps inside a Group step are part of the robot, the steps inside a Snippet step are maintained in a separate file and can be reused in other robots inside the same project. A robot is incomplete and cannot execute if a snippet that it references is not present in the project.

A groupable selection of steps can easily be converted into a reusable snippet 'Create snippet from selection'  icon after selecting the steps to convert. If only a single group step is selected the 'Convert snippet to group'  icon will turn that group into a reusable snippet. A snippet can be easily embedded into a robot by clicking the 'Convert snippet to group'  icon after selecting a Snippet step.

A snippet can also define a set of variables, that will be included in the set of variables of any robot that uses the snippet.

A snippet can have a *description*. This is edited in the Snippet Editor and is shown on every occurrence of that snippet in robots.

Variables and Types

Two important concepts in Design Studio are those of **variables** and **types**. When creating a variable, it must be chosen which type it should have. There are two kinds of types: complex types and simple types. A **complex type** defines a set of **attributes**. This expresses that each variable of a complex type denotes

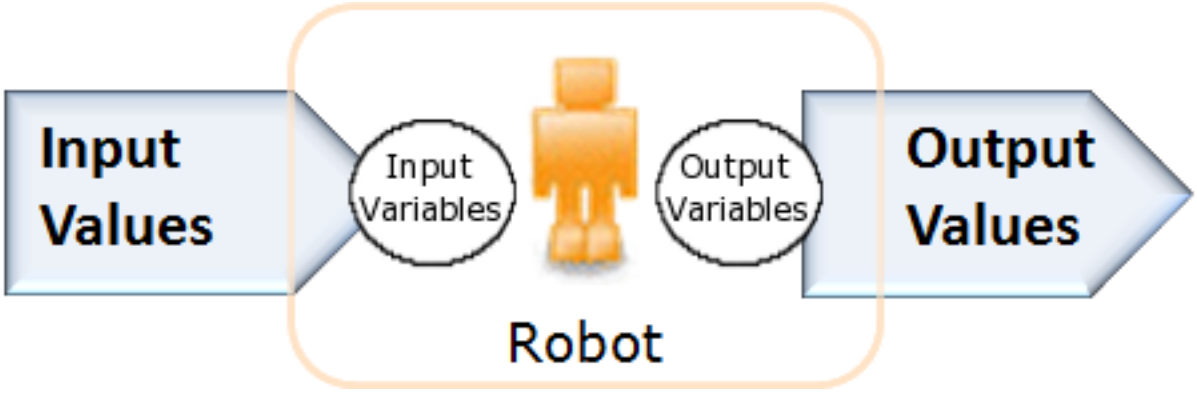
several (named) values. Note that we generally refer to each attribute, e.g. 'title', of a variable of complex type, e.g. 'Book', as a separate variable and denote its value using the fully qualified **attribute name**, i.e. 'Book.title'. Complex types can be created from within Design Studio to fit any need one might have. A simple type does not define any attributes, but only represents the type of a single value. Thus, a variable of a **simple type** contains a single value, for example a text string, and is referred to only by its variable name, e.g. 'Username'. Simple types are built-in and cannot be edited, nor can they be created.

Every variable can be associated with a default initial value that it retains unless the robot explicitly reassigns it, which it often will as values are extracted and manipulated during the execution. Most robots output the values of variables, e.g. by returning them to the caller or inserting them in a database. Robots might also take **input values** which are then assigned to specific variables that have been marked as receiving their values from input. These are simply called **input variables**.

An important difference between variables of complex and simple types is that variables of simple types cannot be used as input variables, and their values cannot be outputted. However, they are useful, for instance, for extracting temporary data or as global counters. Generally, variables of simple types should be viewed as temporary variables, internal to the robot.

Values from variables of complex types can be outputted in various ways. For example, a robot extracting news from some web site might output the values of news variables; each news variable would then have a complex type with attributes such as 'headline', 'bodyText', 'date', 'author', etc., and each outputted news value then comprises a (possibly) unique (sub-)value for each of the named attributes.

For robots containing input variables, it must be specified as part of the input to the robot to which input variable a given input value has to be assigned. For example, a shopping robot that orders books at http://amazon.com might depend on input values containing user and book information. These might be assigned to two input variables in the robot called "user" and "bookInfo" of type "User" and "BookInfo". The below figure shows how a robot accepts input values and generates output values.



The figure shows robot input-output. Input values are assigned to input variables, and the values of some variables are outputted. Only variables of complex types can be assigned from input or have their values outputted.

Libraries and Projects

Robots and types are organized in **libraries**. A library is a collection of robot definitions, type definitions and other files needed to execute the contained robots. A library serves as the deployment unit for robots. This means that a library is how you bundle robots and their required files when you want to distribute and deploy the robots in a runtime environment, such as RoboServer.

When you are working in Design Studio you are working on at least one **robot project**, but you may have many robot projects open in Design Studio at any time. The purpose of a robot project is to develop a robot library. A robot project contains the robot library that you are developing a given set of robots in, as well as other files that are useful for your work on the robot library. Files placed in the library may

also be accessed by robots using a special library protocol [../ref/robomaker/reference/stepaction/support/LibraryProtocol.html].

Thus, a robot project is what you work on when you are developing robots, and a robot library is how you distribute and deploy your work.

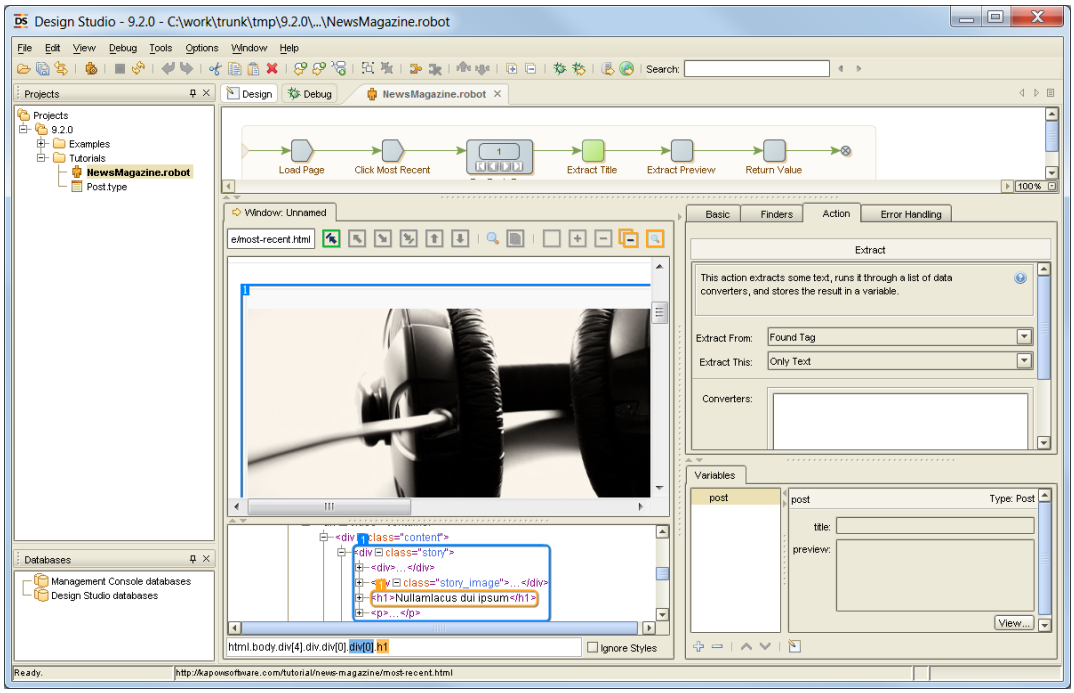
Getting Started with Design Studio

This section gets you started with Design Studio. It introduces you to the Design Studio user interface, including selected menus and functions, the Robot Editor and the Type Editor. Then, the core building blocks of any robot, namely the step actions and data converters, are described. Finally, there are sections on patterns and expressions.

When reading this section, it is recommended that you startup Design Studio and explore the Design Studio user interface as you follow the tour. Note, however, that the tour explores the Design Studio user interface as it appears at startup if you do not create or load a robot. This means that the user interface will be quite empty, and many functions, such as debugging, will not make much sense. Don't worry about this; there will be plenty of opportunities to see many of these functions in action when you start on the tutorials following this section.

A Tour of the Design Studio User Interface

When a valid license has been provided (see Installation guide for details on how to enter license information), the **Design Studio Main Window** appears, as shown below.

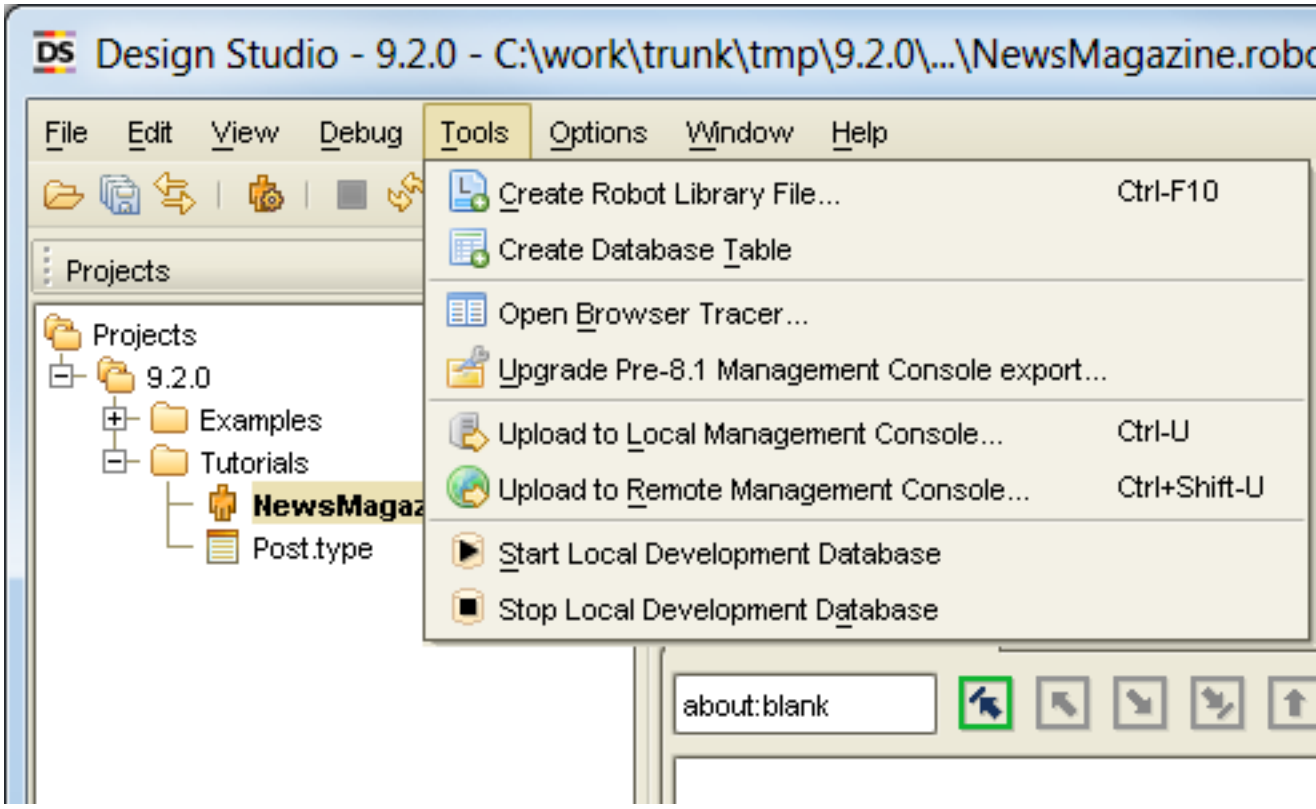


The Design Studio Main Window

At the top of Design Studio, you see the **menu bar** and the **toolbar**. To the left is the **Projects View** and to the right of this the **Editor View**. In the following sections we will describe each of them in more detail.

The Menu Bar

The menu bar is located at the very top of the Design Studio window



Example of the Design Studio Menu Bar

The available menus and the items in these depends on which kind of file is open in the Editor View. The following menus will, however, always be available even if no file has been opened (some items may be disabled):

- The *File* menu is a classical file menu, that has items for manipulating files, projects etc.
- The *Options* menu allows you to change some of the default settings in Design Studio as well as define proxy servers and database connections for use in Design Studio.
- The *Window* menu has items to change the layout of the user interface, e.g. Reset Layout.
- The *Help* menu contains links to the online reference help, documentation and help on how to get support.

As soon as you open a file, e.g. a robot, the edit menu will be added to the available menus:

- The *Edit* menu offers a range of edit actions that you can perform on the opened file. The available action depend on the type of the file, but will always contain the Undo and Redo actions.

If you open a type or a robot file then one more menu will become available:

- The *Tools* menu offers items that lets you perform various tasks related to the type of the file, e.g. generation of database table (for types), or deployment of robots to the Management Console (for robots).

Finally, if you have opened a robot file, three more menus may be available for you:

- The *View* menu that offers items that lets you perform various actions on the view or open additional views that are not open as default.

- The *Debug* menu that contains actions related to the debugger, e.g. start the debugger at the current location of the robot.
- The *Breakpoints* menu that contains actions related to the breakpoints in the debugger, e.g. adding and removing breakpoints. This menu is only available when the Robot Editor is in the debug mode.

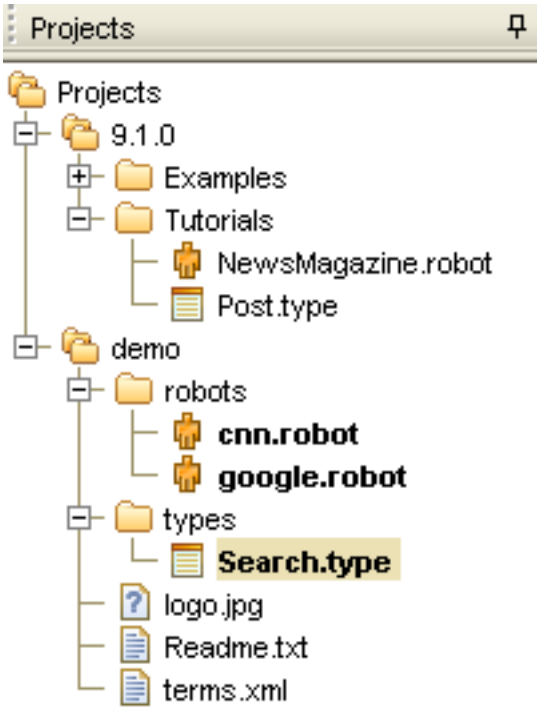
Many of the menu items have keyboard shortcuts that lets you perform the same action using the keyboard, e.g. Ctrl-D for switching between Design and Debug mode in the Robot Editor. The keyboard shortcuts will be shown to the right of the name of the items in the menus.

The Tool Bar

Example of the Design Studio Tool Bar

The Tool Bar contains buttons that let you perform many of the same actions as you can do with menus. The available buttons also change depending on which Editor is the active one in the Editors View. Many buttons have keyboard shortcuts, e.g. Ctrl-D for switching between Design and Debug mode in the Robot Editor. You can learn about these by letting the mouse hover for a short while over a given button until a tool tip will appear. If there is a keyboard shortcut to the given button, then this will be shown in the tool tip.

The Projects View

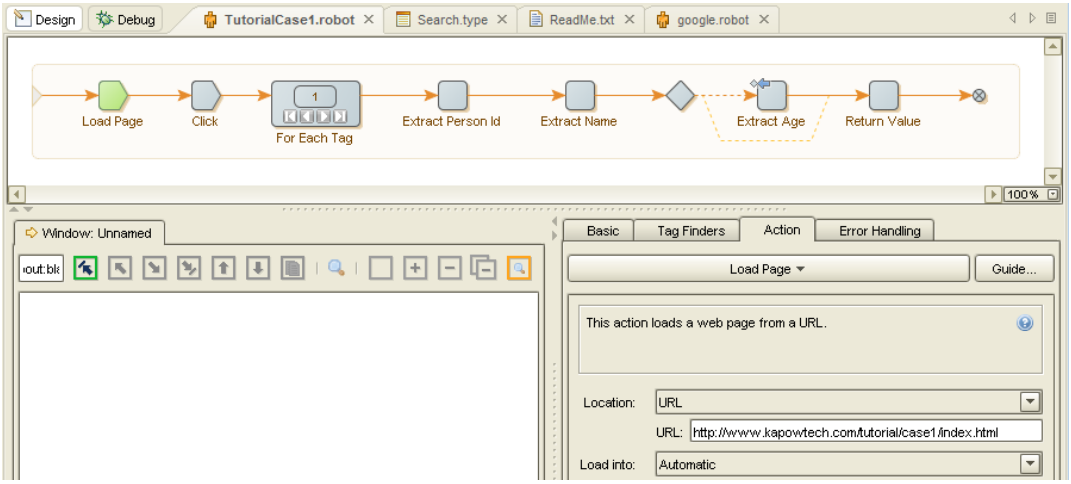


The Projects View

The **Projects View** is located to the left below the toolbar icons in the Design Studio Main Window. The Projects View shows an expanding/collapsing tree structure representing the robot projects that are opened

in Design Studio. By clicking on a + or - in this tree you can expand or collapse the corresponding subtree. The Projects View may contain as many opened robot projects as you like. The Projects View is equipped with a popup menu that allows you to perform various actions, e.g. create a new robot in a folder or open a previously saved robot.

The Editors View



An Example of a Robot Opened in the Editors View

The **Editors View** is where you edit your robots and types. You can have many editors open at the same time, but only one editor will be shown. The editors are shown as tabs at the top of the Editors View and you can click on these to switch to another editor. There are three kinds of editors:

- The Robot Editor in which you edit a robot.
- The Type Editor in which you edit a complex type comprising one or more attributes.
- The Text Editor in which you may edit plain text file.

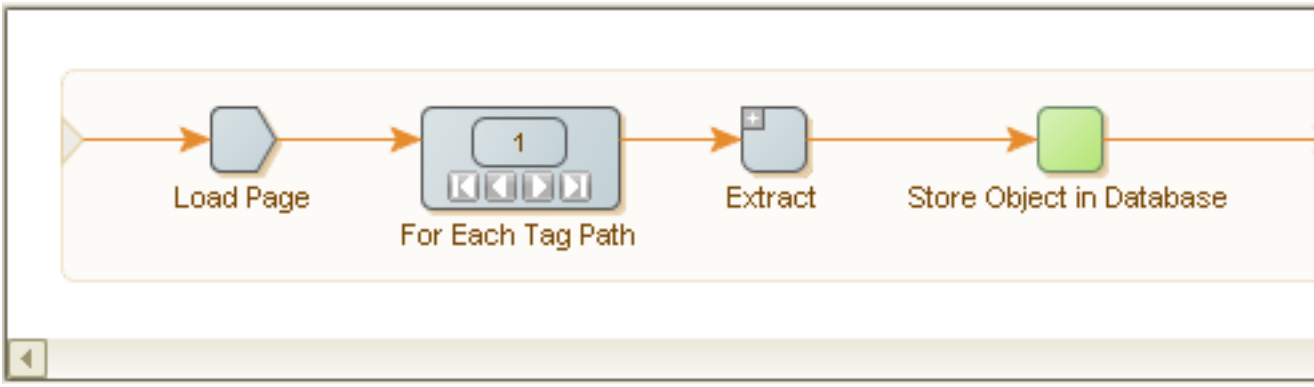
The Robot Editor

The **Robot Editor** is, as the name suggests, the editor you use when editing robots. When you open a robot it will be opened in a new Robot Editor that will be placed in a new tab on the Editors View. The Robot Editor has two modes: design and debug and the editor will always open in design mode. You select which mode the editor should be in by clicking one of the two mode buttons placed in the left corner of the Robot Editor. Depending on which mode you select, the view will change and the options available to you will change, e.g. in toolbars and menus.

In each mode there are several subviews that make up the view of the Robot Editor. For the design mode these are: the **Robot View** where the robot is shown, the **Windows View** that shows the page before the current step of the robot, the **Step View** that shows the configuration of the current step, and the **Variables View** that shows the current values of the variables used by the robot.

In the following section we will describe the views of the design mode. The debug mode will be described separately in a section below.

The Robot View



The Robot View With a Robot

The **Robot View** is located just at the top of the Robot Editor just below the tabs. The Robot View shows you the robot program — that is, the steps and connections that make up the robot. It is in the Robot View that we navigate in our robot (e.g. select steps) and edit its structure (e.g. delete, move or connect steps). In the following, we will give a short description of how you do this.

Current Step

Current Step

In the Robot View there is a notion called a current step. The basic idea is that the partial robot that you are building is actually executed while you are building it. The current step marks the position in this execution and the Studio shows the state for this in the Page View and Variables View. The current step is marked with a green color. If you left-click a step, the robot is executed up to that step, if possible, such that the step becomes the current step. While the robot is executing, the step you left-clicked will be shown in yellow and when execution reaches the step it becomes green and will be the new current step. If execution cannot reach the clicked step (e.g. because a HTML page could not be loaded) then execution stops at the last step it could reach on its way to the new step, and this step will become the new current step. If you click on a step that the robot has already executed to then no execution will take place, but the new step will immediately become the new current step. It is always the current step that you configure in the Step View.

Current Execution Path

Another notion is that of a Current Execution Path. This is the path in the robot that the execution went through to get to the current step and the path that the robot would continue along until it would reach the end of some branch in the robot. The current execution path is marked by a darker color on the connections. You can change the current execution path by clicking on a connection which will result in this connection being included in the path.

Selection

Selected Step

You can select more than one step or connection by holding down the Ctrl key and left-clicking the steps or connections. You can also hold down the left mouse button and drag it over the steps to mark. To deselect the currently selected steps and connections just click somewhere outside of the robot.

When steps or connections are selected, you can apply actions to them. For example, you can insert a new step following a selected step by first selecting the step after which you want the new step to be inserted and then clicking the icon in the toolbar. You can also right-click on a step or connection to bring up a pop-up menu. If you do this on a step or connection that is not already selected, then it automatically becomes selected.

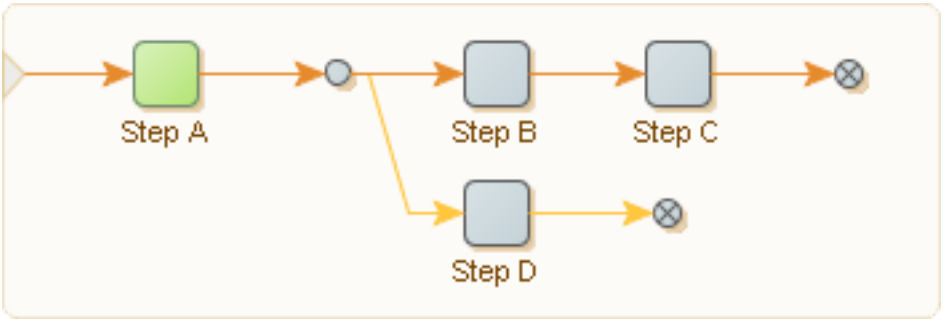
Edit Actions

The Robot Editor lets you perform a long range of actions on steps and connections. These include standard editor actions like copy, paste, cut and delete, but also actions that affect the execution of the robot in the design view, e.g. changing the iteration of a loop. You can perform actions on either the current step (if no other step is selected), on selected step(s) or on selected connection(s). An action may be performed by clicking the corresponding toolbar button or by popup menus on the selected elements. There are also keyboard shortcuts for most actions. These are shown in the tool tips of the toolbar buttons or popup menu items.

You may configure another step than the current by selecting it (by either Ctrl-clicking on it or dragging a selection box around it) and pressing the F2 key or by choosing "Configure Step..." from the popup menu. This will open a Step Configuration dialog that gives you the same options as the Step View.

Grouping and Ungrouping Steps

Two edit actions that we will describe in more detail are the actions for grouping and ungrouping steps. You group steps by selecting the steps to group and then use the Group action () from either the toolbar or the popup menu on the steps. Some selections cannot be grouped. A group step must have exactly one ingoing connection and exactly one outgoing connection, so this must also hold for the selection of steps that you want to group. The only exception to this is that if your selection of steps does not have any outgoing connection then the selection can still be grouped, but in this case the topmost End step is connected to the end of the group and the created group will have a connection to a new End step outside the group. Take a look at the example robot below.



An Example Robot

In this robot, the following are some examples of steps you can group:

- all the steps
- any single action step alone, e.g. Step A, Step B, etc.
- the branch point, step B, step C and the end step after step C

and the following are some examples of steps you cannot group:

- the branch point and Step B (more than one outgoing connection)
- steps B, C, D and the two End steps (more than one ingoing connection)

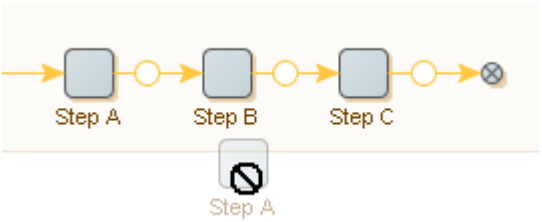
An expanded group step may be select by either clicking (while holding down the Ctrl key) close to the edge or by including it in a drag selection.

You ungroup a Group step (or steps) by selecting it (or them) and use the Ungroup action () from either the toolbar or the popup menu on the steps. The Group and Ungroup actions are inverse, so if you group a selection of steps and immediately ungroup them again, the structure of the robot is unchanged.

The Robot View also contains actions to expand or collapse more than one group simultaneously. The toolbar actions and expand or collapse all Group steps in the robot. The similar actions and found in the popup menu on steps will do the same, but restricted to the Group steps in the selection.

Drag and Drop

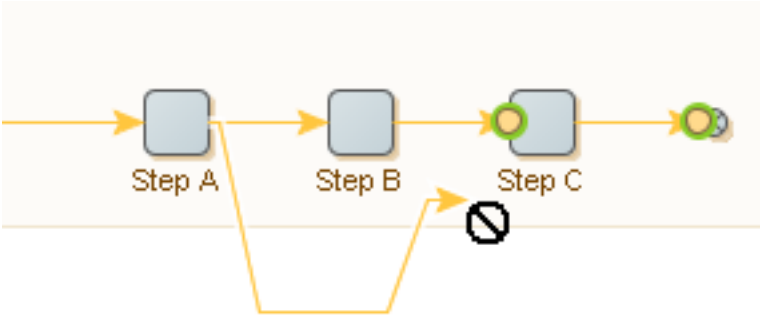
Another way of editing a robot, apart from using actions, is by using the mouse to manipulate elements directly. You can directly move a step by clicking on it and dragging it. As soon as you drag the step, special indicators will appear showing where you may drop it. You can also select several steps and move these together. You can also move the endpoint of a connection. You do this by first selecting the connection. Then you move the mouse to one of the handles at the end of the connection, click on the handle and move this to a new location. As soon as you click on a handle special indicators will appear showing where you may connect the edge. To abort a drag and drop action, move the mouse outside the robot and let go of the mouse button as shown below:



Drag and Drop a Step

Adding a New Connection

You can also create new connections using the mouse. Place the cursor near the end of a step so that an indicator appears (an orange circle with a green halo). Click on the indicator and a new arrow will appear. Keep the left mouse button pressed; move the mouse and a new connection will follow your mouse when you move this around (see image below). New indicators will appear and you can then move the mouse to one of these and drop the new connection end point there by letting go of the left mouse button.



Adding a New Connection

Undoing and Redoing Changes

Everything you do when you edit a robot you can undo and redo. So you don't have to fear accidentally dropping a step at a wrong position or deleting a connection. You can always just press Ctrl-Z (or click the icon) one or more times to undo what you just did and you can press Ctrl-Y (or click the icon) to redo (if you happened to undo too much).

Step Validation

While you edit your robot, the Robot View will validate your steps and any invalid steps are underlined in red. If you move the mouse to an invalid step, an explanation of why the step is invalid is shown.

The Windows View

The **Windows View** is located below the Robot View in the Robot Editor. In the Windows View, you view part of the **current robot state** — that is, the part of the robot state that has to do with the loaded pages. The state that is shown is the input state to the **current step**.

In the Windows View, you see the **Page Views** of the windows in the current robot state. When loading from a URL, several windows may be opened, each containing a page. The current window is marked with an arrow. For each window, the Page View is split into several sub views depending on the type of the page, e.g. if the loaded page is an HTML page the Page View has sub views that reflects this. There are four types of pages, but essentially only two different types of Page View. The page types HTML, XML and Binary use the same view and the Excel page type has its own specialized Page View. The following two sections describes these two Page Views.

If you want to see the **Cookies View** of the state of the current step you can open the Cookies Dialog from the View menu. Cookies are added to this list as the robot loads web pages that use cookies.

Similarly, you can open the **Authentications Dialog** from the View menu to see the authentications of the current state.

The HTML Page View

The HTML Page View is used when the page type of a Window is HTML, XML or Binary. The view is split into several sub views, as shown below: the **Address Bar**, the **Tag Path View**, the **Browser View** and the **HTML View**.

The HTML Page View

The Address Bar show the URL of the document in the Page View. You can also (like in a browser) enter a new URL in this and if you press Enter a new Load Page step will be inserted in the robot before the current step.

In the Tag Path View, you see the path from the root tag of the page to the selected tag. In the Browser View, you see the page as it appears in a browser. In the HTML View you see the HTML of the page - as it looks like after JavaScript has been applied and any asynchronous data has been fetched.

You can select a tag in the Browser View by left-clicking in any of the views. The currently selected tag is shown with a green box in the Browser View and the HTML View, and with a green background in the Tag Path View. You can hold down Alt while clicking inside the currently selected tag to move the selection one level out, i.e. select the tag that encloses the selected tag. You can also hold down Alt and Shift while clicking inside the selected tag. This will move the selection one level in towards the tag that you clicked.

You can also change the current selection using the buttons to the right of the Address Bar. The and icons move the selection one level out or in. The icon selects the root tag of the page, while the icon selects the innermost tag inside the selected tag. The and icons select the tag above or below the selected tag. You can also search for a tag by clicking the icon.

The Browser View also shows the tags found by the Tag Finders of the current step. These tags are called **found tags** and will be shown with an orange box in the Browser View and the HTML View, and with an orange background in the Tag Path View. If you edit the Tag Finders, you can click the icon to show the new tags found. You can also configure the Tag Finders to use only the currently selected tag by clicking the icon, to use the currently selected tag as well as any other tags found by clicking the icon, to not use the currently selected tag by clicking the icon, or to not use any tags at all by clicking the icon.

Furthermore, the Page Views shows the **named tags**. Named tags are marker tags that are used as reference when finding other tags. Named tags can be set by step actions — for example, some loop actions use a named tag to mark the result of the current iteration of the loop. You can also set named tags manually. Named tags are shown with a blue box in the Browser View and the HTML View, and with a blue background in the Tag Path View.

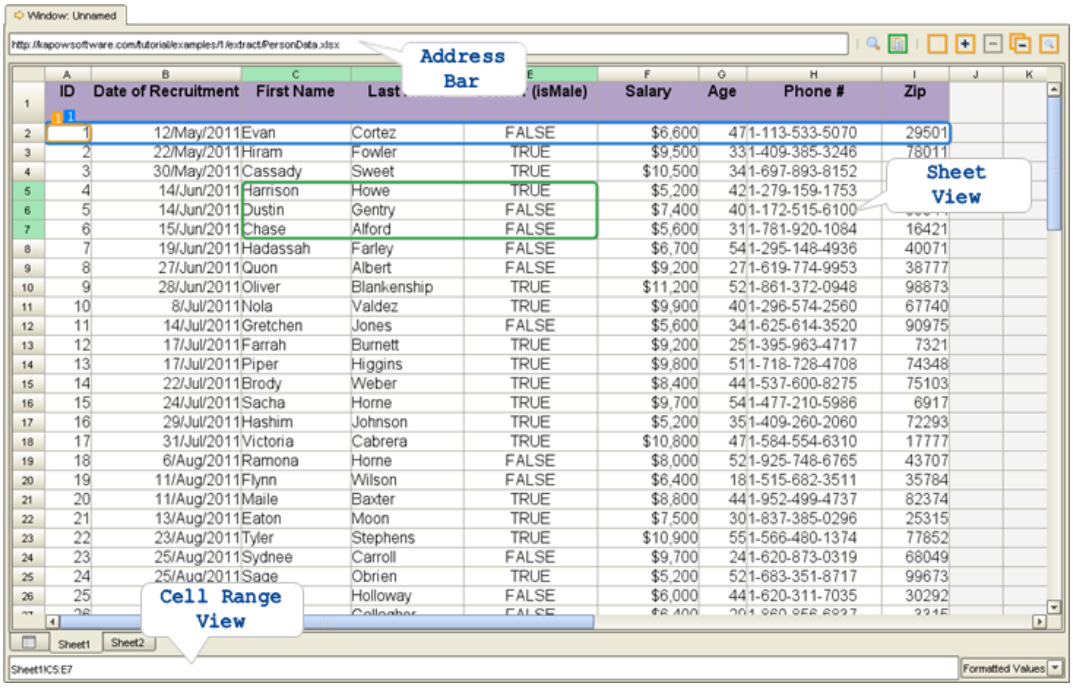
In all Page views you can right-click a tag to open a pop-up menu that allows you to configure the current step or insert a new step. This is very useful and will probably become your preferred way of configuring the current step. From the menu, you can choose "Use only this Tag" or "Use this Tag" to configure the Tag Finders to find the tag that you clicked. You can also choose an action such as Enter Text from the menu. This will configure the current step to use the corresponding action, in this case Enter Text, on the tag that you clicked. The available option in the popup menu depends on the tag you click and typically one option is shown in bold text. This is the default option and can also be selected directly (without using the popup menu) by double-clicking on the tag in the Browser View. For example, if you want to enter a text in an input tag you just double-click on the input field in the Browser View. This will bring up a dialog from which you can enter the text and when you subsequently click on the OK button of this dialog a new step will be inserted in your robot before the current step and this will have an Enter Text action configured with your text.

To copy text or HTML from the HTML View, right-click the tag of interest and select an option from the menu that appears.

The Spreadsheet View

The Spreadsheet View is used when the document loaded into the window is an Excel document.

The Spreadsheet View shown below is split into several sub views: the **Address Bar**, the **Sheets View** and the **Cell Range View**.



The Spreadsheet View

The **Address Bar** in the Spreadsheet View is the same as the one on the HTML Page View and its functionality is identical to this.

The **Sheets View** shows the content of the spreadsheet in the usual manner. It contains one tab for each individual sheet in the spreadsheet document as well as an extra tab showing the **document properties** of this document (shown in Excel under File > Info). This document properties tab is the first tab in the Sheets View and is identified by the icon. Document properties include details such as title, author name, subject, last modified data, etc. The view on the document properties tab is organized just as a regular sheet from the document and its content can be extracted and looped over in exactly the same manner as these sheets.

Elements of the Sheets View may be **selected** in various different ways. You can select a single cell of the spreadsheet by clicking with the mouse on the cell in the Sheets View. If you hold down the left mouse button and drag the mouse you can select more than one cell. Once you have a current selection you may use the keyboard to move the selection around, e.g. by using either the arrow keys, or one of the keys: Home, End, Page Up or Page Down.

If you click on a column or row header then the entire column or row will be selected and if you click on top left corner of the Sheets View then the entire sheet will be selected. When ever an entire column, row or sheet is selected then this is an open ended selection in the sense that if the column, row or sheet changes in size then the selection still selects the the entire column, row or sheet and not just a column, row or sheet of the specific size it had when you selected it. For example, if you are writing a robot to extract all the rows of a sheet (using a loop step) and the sheet on the day that you create your robot has 42 rows it will not fail the next day when the sheet has more or less rows than 42 rows. If you instead had used a selection like Sheet1!A1:I42 for the table then the robot would always try to extract 42 rows (and 9 columns) regardless of how many rows (and columns) the sheet actually contained.

If you hold down the Shift key while selecting either with the mouse or the keyboard the current selection will be extended. This extension works by selecting the range that contains the current *lead selection* and the new added cell. The current lead selection is the first cell that you selected when you created the current selection, e.g. if you start dragging with the mouse from cell C5 to E7 in Sheet1 you will get a selection "Sheet1!C5:E7" and your lead selection cell will be C5, if you on the other hand were dragging from E7 to C5 then your lead selection cell will be E7.

The **Cell Range View** show the current selection as a Excel cell range, e.g. Sheet1!C5:E7. If you enter a new cell range in the view and press Enter, then this becomes the new current selection. The syntax of cell ranges is described in the reference documentation on Cell Ranges [../ref/robomaker/reference/userinterface/CellRanges.html].

The Sheets View also shows the cell or cells found by the Range Finders of the current step. These cells are called **found cells** and will be shown with an orange box in the Sheets View. If you edit the Range Finders, you can click the icon to show the new cells found. You can also configure the Range Finders to use only the currently selected cell or cells by clicking the icon, to use the currently selected cell or cells as well as any other cells found by clicking the icon, to not use the currently selected cell or cells by clicking the icon, or to not use any cells at all by clicking the icon.

Just as for the HTML Page View you can also search specific content within the Spreadsheet View by clicking the icon and the cell found by your search will be the new current selection. You can also copy the content of the current selection to the Clipboard using the icon. If you copy a range that contains more than one cell the content of the Clipboard may be pasted directly into Excel and the values of each cell will be placed in the separate cells in Excel.

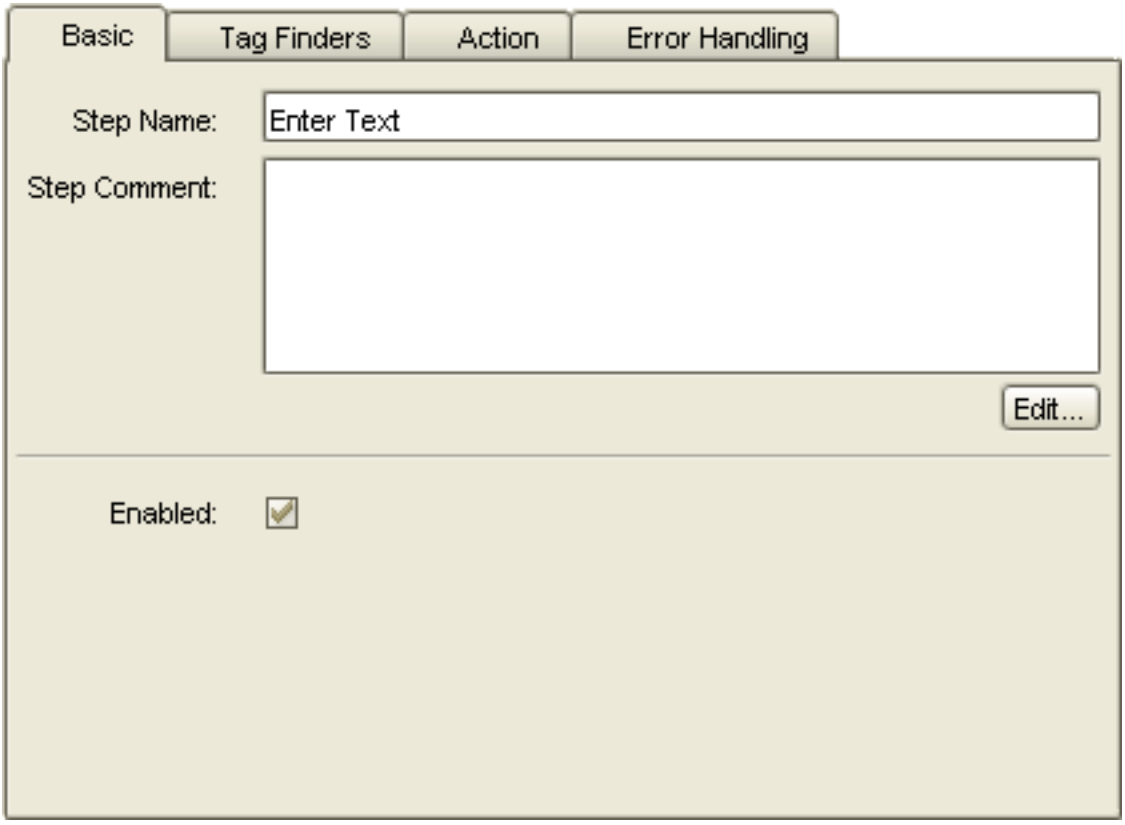
The **View Mode Selector** is a combo box situated in the lower right corner of the Spreadsheet View. This determines what is shown in the cells of the Sheets View. The possible options are:

Formatted Values:	The cells contains what you see in Excel, e.g. dates are shown formatted and numbers may be shown with less decimals than the actual values of the cells.
Plain Values:	The cells contains the actual values that Excel would show if the values of the cells were not formatted, e.g. without rounding of decimals.

Formulas: If a cell contains a formula then this is shown and otherwise the same value as for the Plain Values mode is shown.

The Step View

The **Step View** is located to the right of the Page View. The Step View is separated from the Page View using a divider, which you can drag either left or right to make more room for one of the views.



The Step View for an Action Step

The Step View shows the configuration of the current step. You can view and edit the different properties of the step by clicking one of the tabs: "Basic", "Finders", "Action", and "Error Handling". For some step types, e.g. Try or Group steps, some or all of these properties are not relevant and the corresponding tabs are therefore not shown. The actual view of the Step View will therefore depend on the current step.

In the "Basic" tab, you find the name of the step, as well as the comment attached to it. Steps that have an attached comment are shown with a name in bold in the Robot View. If you rest the mouse pointer on a step, the comment will be displayed as rollover text.

In the "Finders" tab, you can view and configure the list of finders of the step. You normally configure the finders by right-clicking on an element in the Page View. See How to Use the Tag Finders for more information.

If the step is an action step you can in the "Action" tab view and configure the action for the step. For a description of the actions available, see Step Actions and Data Converters below.

In the "Error Handling" tab in the Step Window, you can see how the current step handles errors. See How to Handle Errors for more information.

The Variables View

The **Variables View** is located below the Step View. The Variables View is separated from the Step View using a divider, which you can drag either up or down to make more room for one of the views.

The Variables View is divided into a left part and a right part with a similar divider. The left part shows the variables in a list. When you select a variable in this list, the details of that particular variable is shown in the right part of the view. The view shows the variable's values at the current step of robot execution, and these cannot be edited.

You can add or remove variables by right-clicking the list of variables. This displays a menu of variable types that can be added to the list. The menu also makes it possible to remove the selected variable.

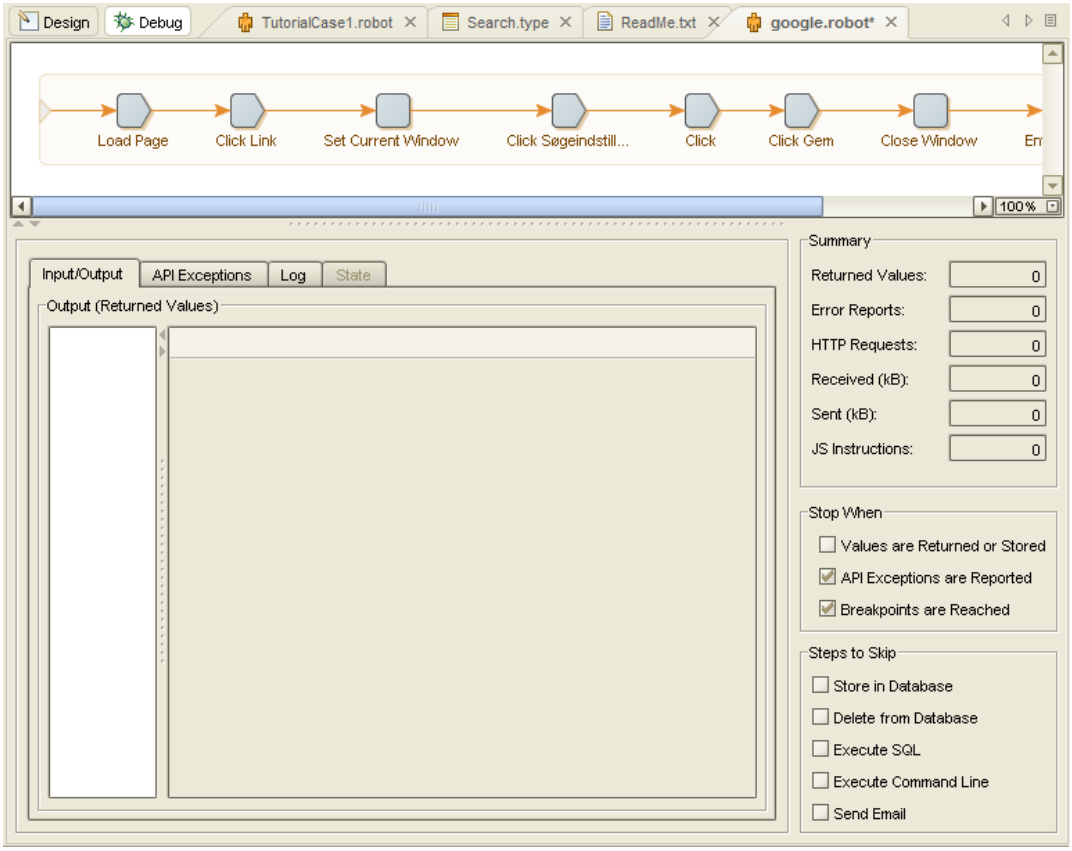
You can edit the initial values of the variables by pressing the "Edit" button or double-clicking the variable list. This will open a dialog which looks very much like the Variables View and works in the same way. The important difference is that the dialog displays the values of the variables before any step has been executed, and these values can be edited.

Initial values for input variables are used when writing and testing the robot. When a robot is run in production, the input variables will be initialized to values determined by the application running the robot (if the application does not provide values, the robot run will fail).

Initial values for variables are the values that they will have at the start of the robot, i.e. at the first step. The values apply both when you are writing, testing, and running the robot in production.

The Debug Mode

The Robot Editor contains a specialized mode for debugging robots. You can switch between design and debug mode by clicking on one of the two mode buttons placed in the left corner of the Robot Editor. You can also switch to the debug mode by clicking the icon in the toolbar in Design Studio Main Window. Alternatively, if you want to debug from the current step in Design Studio, you can click the icon. When you switch to the debug mode the Robot Editor will change appearance, and look as shown below.



The Debug Mode

The top of the Robot Editor in debug mode also contains a **Robot View**, similar to that of the design mode. Note, however, that the Robot View in debug mode has a current step only when you are actually debugging the robot. This current step is *not* always the same as the current step in the Robot View in the design mode.

Below the Robot View is a large panel divided into four sub-panels, the main panel and three panels named "Summary", "Stop When" and "Steps to Skip".

In the main panel, you see the results of the debugging process divided into four tabs. In the "Input/Output" tab, you see a list of all used variables, if any, and a list of all values that have been returned so far during the debugging process. In the "API Exceptions" tab, you see a list of the API exceptions reported so far during the debugging process. In the "Log" tab, you can see whatever has been written to the log so far during the debugging process. (Some actions, particularly those that take a while to execute, such as the Loop Form action, write status information to this log. Step errors will also be logged if the step is so configured.) Whenever the debugging process has been temporarily stopped, the "State" tab shows the robot state that is input to the current step. The "State" tab contains seven sub-tabs. The "Variables" sub-tab shows the list of variables. The "Windows", "Cookies", and "Authentications" sub-tabs show the state, in much the same way as the State View in Design Studio with its associated dialogs. The "Local Storage" and "Session Storage" sub-tabs shows the HTML5 objects that have been persisted locally. The "API Exception" sub-tab contains the API Exception generated at the current step, if any. For all API Exceptions (and the errors that form part of them), you can click the Goto button to go directly to the step that generated the error — that is, the step that generated the error will become the current step in Design Studio.

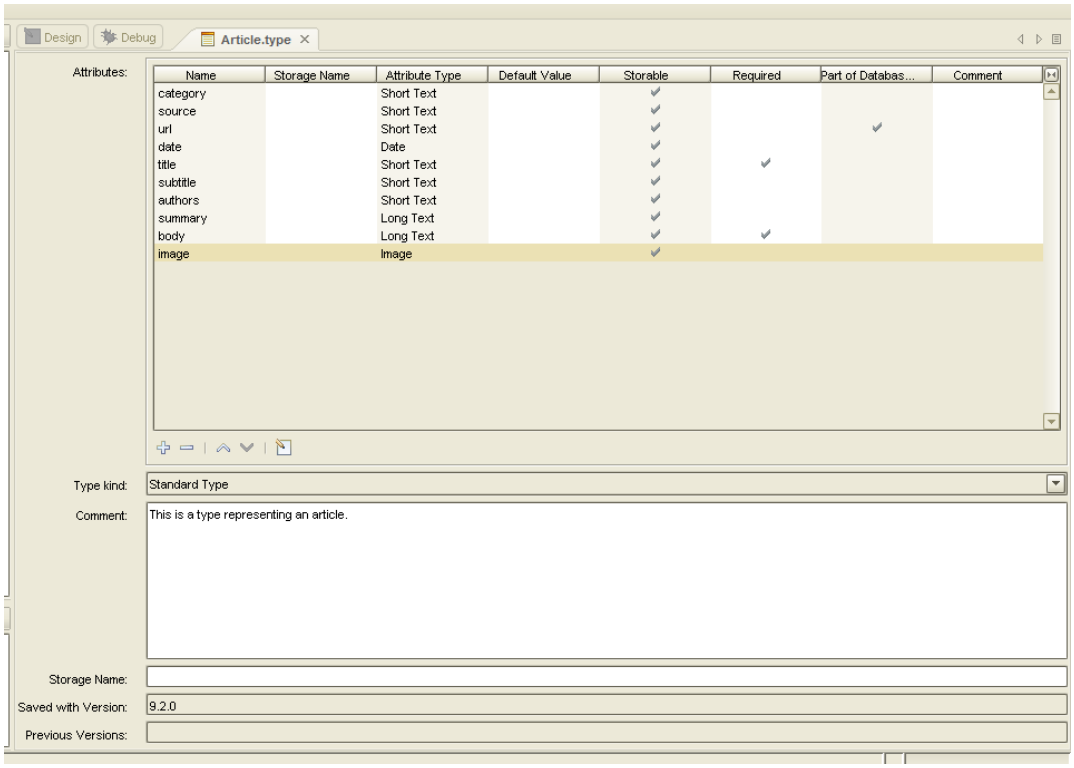
In the "Summary" panel, you see an overview of the number of variables returned or written to a database and generated API exceptions so far during the debugging process.

In the "Stop When" panel, you can specify the criteria for when the debugging process should temporarily stop (besides ending normally).

For more on debugging robots, see How to Debug a Robot.

The Type Editor

This section gets you started with the Type Editor by introducing you to the editors user interface. The view of this editor is shown below:



The Type Editor

The currently edited type is configured in the main window. Among other things, you can configure the attributes of the type. This is done in the **Attribute Table**. You can add new attributes, remove attributes, change their order, and configure attributes using the buttons below the Attribute Table.

The Text Editor

The Text Editor is a simple editor for plain text files (.txt). This is only included as a convenient way to edit simple text files such as "readme" files etc. The allowed extensions that the editor can open are: txt, java, jsp, js, log, html, xml and csv, but the editor does not use the information that these extensions imply about file content. All files are treated as plain text files (no syntax highlighting).

General Editing

This section gives a few general hints related to editing robots in Design Studio. We have already talked about this when editing robots in the Robot View, but the hints presented here are more general than that. They also apply to when you make changes to a robot in the Step View, to a type in the Type Editor or to a text in the Text Editor.

Undoing and Redoing Changes

Essentially everything you do when editing something in Design Studio, you can undo by pressing Ctrl-Z or clicking the icon. Similarly, an undone change can be redone by pressing Ctrl-Y or clicking the icon.

Cutting, Copying, and Pasting

In Design Studio, most items (e.g. steps, data converters and finders) can be cut, copied, and pasted using the shortcuts Ctrl-X, Ctrl-C, and Ctrl-V, respectively. In most lists, e.g. the list of finders for a step, all items can be copied using the shortcut Ctrl-Shift-C.

Types

When working with a type, it is important that you configure all of the relevant properties. Otherwise, the type will be invalid, or it will not behave as expected when used in Design Studio.

All types must have a valid name. The type names must start with a letter or an underscore and must contain only letters, digits and underscores. Two types in the same project must not have the same name. The name of a type is simply its file name without the extension. E.g. a type with the file name 'ExampleType.type' will be available in Design Studio as 'ExampleType'.

Also, all types must have a type kind which indicates how the type will be used. You can choose the type kind using the "Type kind" drop-down box below the Attribute Table. Normally, it will not be necessary to select anything here, as the type kind "Standard Type" should always be used unless one has need for the legacy type kind "Database Output Type". See the reference documentation on Design Studio [../ref/robomaker/reference/typeeditor/TypeConfiguration.html] for more information.

Configuring Attributes

The attributes within a type must also be correctly added and configured in order for the type to be valid. You must specify both a name and a type for each attribute. The available attribute types are:

Attribute Type	**Description**
Integer	An integer, e.g. 12. The possible range is from -9223372036854775808 to 9223372036854775807, both inclusive.
Number	A number, e.g. 12.345. The possible range is from $\pm 2.2 \times 10^{\#308}$ to $\pm 1.8 \times 10^{308}$ with slightly more than 15 digits of accuracy.
Boolean	A boolean value, i.e. either "true" or "false".
Character	A single character, e.g. "A".
Short Text	A short text. Will be displayed in a one-line text field.
Long Text	A long text. Will be displayed in a multi-line text box.
Password	A password. Will be displayed in a password field that shows asterisks instead of the characters in the password.
HTML	An HTML clip. This is the same as a Long Text, except that you can preview the clip in a browser window.
XML	An XML document. This is the same as a Long Text, except that only well-formed XML documents are allowed.
Date	

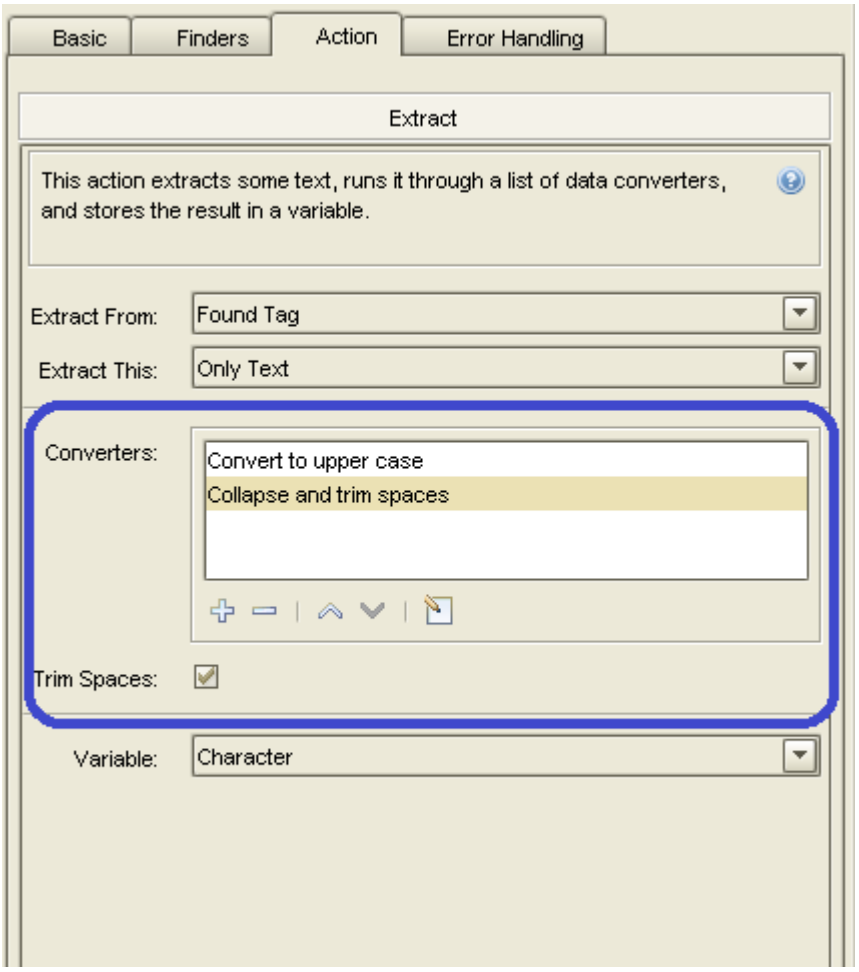
Attribute Type	Description
	A date. The date must be of the form yyyy-mm-dd hh:mm:ss.n, e.g. "1992-04-25 10:33:06.0".
Binary	Binary data, i.e. any sequence of bytes.
Image	An image. This is the same as Binary Data, except that you can preview the image.
PDF	A PDF document. This is the same as Binary Data, except that you can preview the PDF document.
Session	A session (containing cookies, authentications, etc.).
Currency	A currency code, as defined by the ISO-4217 standard, e.g. "EUR" for Euro.
Country	A country code, as defined by the ISO-3166 standard, e.g. "DE" for Germany.
Language	A language code, as defined by the ISO-639 standard, e.g. "de" for German.

An attribute has a number of other configurable properties, such as whether or not the attribute is visible from within Design Studio, and whether or not the attribute is required to contain data before a variable value of the type can be stored. Refer to the reference documentation on Design Studio [../ref/robomaker/reference/typeeditor/AttributeConfiguration.html] for more information.

Step Actions and Data Converters

This section contains general information about the step actions and data converters available in Design Studio. In Design Studio, a short description will be shown together with each action and data converter, and you can get more information about the action or data converter by clicking the "More..." button associated with the description. Moreover, in the Design Studio Help menu, you can find a help entry on every step action and data converter.

Several of the actions, e.g. Extract, include the possibility of running some text content through a list of data converters and then storing the result in some variable.



The Spreadsheet View

As mentioned earlier, a data converter processes some text, e.g. the Extract Number data converter accepts an input text containing a number in some format and outputs a text containing the same number in a standardized format. Because a data converter takes a text as input and outputs another text, data converters can be **chained** so that the output of one data converter becomes the input to the next data converter. The final output is the text outputted by the last data converter in the list of data converters. For example, if the list of data converters consists of a Convert to Upper Case data converter followed by a Remove Spaces data converter, and the input text to the list is `"R  oboMa ker"`, then the output text will be `"ROBOMAKER"`.

Patterns

A **pattern** is a formal way of describing a text. For example, the text "32" can be described as a text containing two digits. However, other texts also contain two digits, e.g. "12" and "00" and can therefore also be described as a text containing two digits. We can express this by the pattern \d\d which is a formal way of expressing that a text must contain two and only two digits (\d is the symbol for a digit). We say that these texts **match** this pattern. Design Studio patterns follow the Perl5 syntax.

A pattern is composed of normal characters and **special symbols**. Each special symbol carries its own special meaning. For example, the special symbol "." (dot) means any *single* character and matches all single characters, e.g. "a", "b", "1", "2", ...

The table below provides an overview of the most commonly used special symbols. For a complete overview of all the special symbols available, see the reference documentation on patterns [../ref/robomaker/reference/regularexpressions/RegularExpressions.html].

Table 1. The Most Commonly Used Pattern Special Symbols

Special symbol	Meaning
.	Any single character, e.g. "a", "1", "/", "?", ".", etc.
\d	Any decimal digit, e.g. "0", "1", ..., "9".
\D	Any non-digit, i.e. same as ".", but excluding "0", "1", ..., "9".
\s	Any white space character, e.g. " " and line break.
\S	Any non-white space character, i.e. same as ".", but excluding white space (such as " " and line break).
\w	Any word (alphanumeric) character, e.g. "a", ..., "z", "A", ..., "Z", "0", ..., "9".
\W	Any non-word (alphanumeric) character, i.e. same as ".", but excluding "a", ..., "z", "A", ..., "Z", "0", ..., "9".

Example: The pattern ".an" matches all texts of length three ending with "an", e.g. "can" and "man" but not "mcan".

Example: The pattern "\d\d\s\d\d" matches all texts of length five starting with two digits followed by a white space and ending with two digits, e.g. "01 23" and "72 13" but not "01 2s".

If you want a special character, such as "." or "\", to act as a normal character, you can **escape** it by adding a "\" (backslash) in front of it. So, if you wish to match exactly the "." character, instead of any single character, you should write "\.".

Example: The pattern "m\.n\\o" only matches the text "m.n\o".

You can organize a pattern into **subpatterns** by the use of parentheses: "(" and ")".

Example: The pattern "abc" can be organized as "(a)(bc)".

All single characters are considered subpatterns.

Example: In the pattern "abc", each single character "a", "b", and "c" is considered a subpattern.

Subpatterns are useful when applying **pattern operators**. The table below provides an overview of the pattern operators available.

Table 2. The Pattern Operators

Operator	Meaning
?	Matches the preceding subpattern, or the empty text.
*	Matches any number of repetitions of the preceding subpattern, or the empty text.
+	Matches one or more repetitions of the preceding subpattern.
{m}	Matches exactly *m* repetitions of the preceding subpattern.
{m,n}	Matches between *m* and *n* repetitions (inclusive) of the preceding subpattern.

Operator	Meaning
{m,}	Matches *m* or more repetitions of the preceding subpattern.
a|b	Matches whatever the expression *a* would match, or whatever the expression *b* would match.

Example: ".*" matches any text, e.g. "Design Studio", "1213" and "" (the empty text).

Example: "(abc)*" matches any number of repetitions of the text "abc", e.g. "", "abc", "abcabc", and "abcabcabc", but not "abca".

Example: "(\d\d){1,2}" matches either two or four digits, e.g. "12" and "6789", but not "123".

Example: "(good)?bye" matches "goodbye" and "bye".

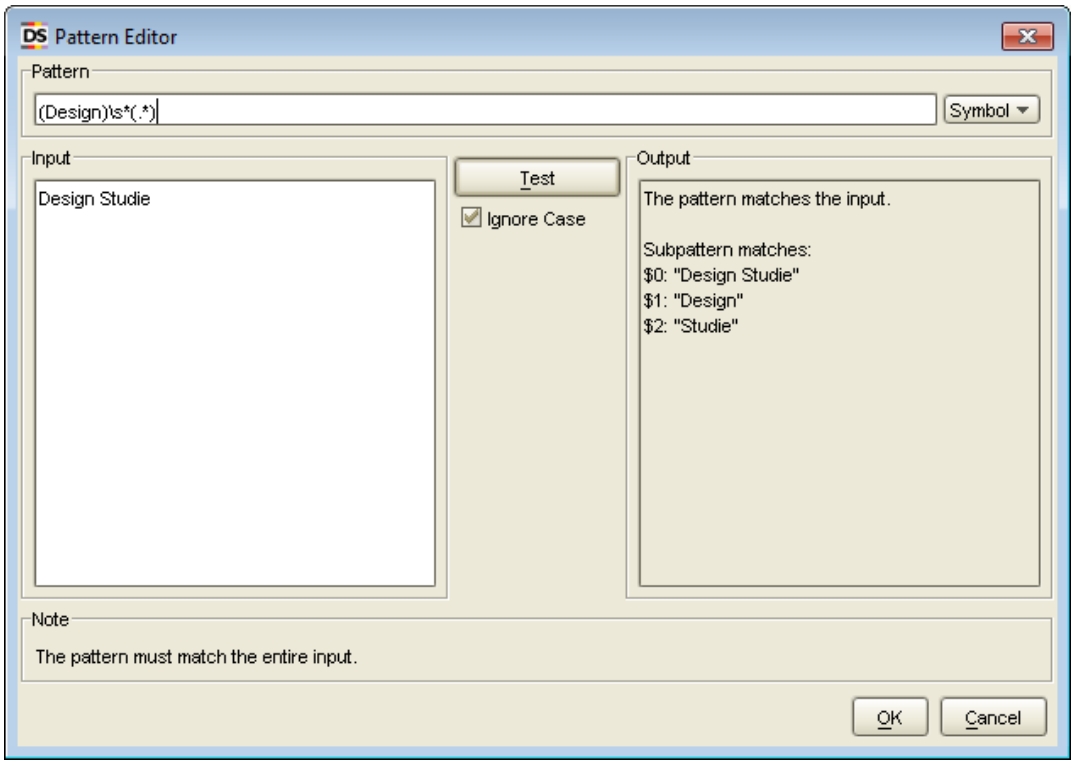
Example: "(good)|(bye)" matches "good" and "bye".

As with other special characters, you can escape the special characters that appear in pattern operators by adding a "\" backslash in front of the character.

Subpatterns are useful when you want to extract specific text pieces from a text. When you make a subpattern using parentheses, you can extract the part of the text that is matched by that subpattern. For example, consider the pattern "abc (.*) def (.*) ghi". This pattern has two subpatterns that are made by means of parentheses. If the pattern is matched against the text "abc 123 def 456 ghi", the first of those subpatterns will match the text "123", and the second subpattern will match the text "456". In an expression (see Expressions), you can refer to these subpattern matches by writing "$1" and "$2". For example, the expression "X" + $1 + "Y"+ $2 + "Z" will produce the result "X123Y456Z". This is a very important extraction technique in Design Studio.

By default, the repetition pattern operators (*, +, {...}) will match as **many** repetitions of the preceding pattern as possible. You can put a "?" after the operator to turn it into an operator that matches as **few** repetitions as possible. For example, consider the pattern ".*(\d\d\d).*". If the pattern is matched against the text "abc 123 def 456 ghi", the subpattern "(\d\d\d)" will match the second number in the text ("456"), since the first *-operator will match as many repetitions as possible. If you put a "?" after the *-operator, so that the pattern becomes ".*?(\d\d\d).*", the subpattern "(\d\d\d)" will match the first number in the text ("123"), since the *?-operator will match as few repetitions as possible.

It is recommended that you experiment with patterns on your own. The best way to do this is to launch Design Studio and find a place where you can enter a pattern, such as in the Test Tag action. Then, click the "Edit..." button to the right of the pattern field, to open the **Pattern Editor Window** shown below.



The Pattern Editor Window

In the Pattern Editor Window, you can enter a pattern and test whether it matches the test input text in the "Input" panel. When you open the window, Design Studio will usually have set the test input text to the text that the pattern will be matched against if the given step is executed on the current input robot state. However, you can also edit the test input text yourself, to try the pattern on other inputs. To test the pattern, click the "Test" button. The result of the matching will then be shown in the "Output" panel.

The "Symbol" button is very useful when you want to enter a special symbol in the pattern. When you click it, a pop-up menu will be shown, from which you can choose the symbol to insert in the pattern. This way, you don't have to memorize all the special symbols and their meanings.

For more on patterns, consult the reference documentation on patterns [../ref/robomaker/reference/regularexpressions/RegularExpressions.html].

Expressions

An **expression** typically evaluates to a text. For example, the expression

```
"The author of the book " + Book.title + " is " + Book.author + "."
```

evaluates to the text "The author of the book Pet Sematary is Stephen King.", if the variables **Book.title** and **Book.author** contain the texts "Pet Sematary" and "Stephen King", respectively.

You can also do numeric calculations within the expression. For example, if the variable **Book.price** contains the price of a book, you can multiply this with 100 using the following expression:

```
Book.price * 100
```

The table below provides an overview of the most commonly used sub-expression types. For a complete overview of all sub-expression types available, see the reference documentation on expressions [../ref/robomaker/reference/expression/Expression.html].

Table 3. The Most Commonly Used Sub-Expression Types

Sub-Expression Type	Notation	Meaning
Text Constant	"text" or >>text<<	Evaluates to the specified text, e.g. "Stephen King", or >>Stephen King<<.
Variables	variablename.attributename	Evaluates to the value of the specified variable, e.g. "Book.author" might evaluate to "Stephen King".
Current URL	URL	Evaluates to the URL of the current page.
Subpattern Match	$n	Evaluates to the text matched by subpattern *n* in an associated pattern (if any). For example, this is used in the Advanced Extract data converter, as shown below. $0 evaluates to the text matched by the entire pattern.
Function	func(args)	Evaluates the specified function by passing it the specified arguments and converting its result to a text.

Note that you can specify a text constant using either the quote notation or the >>text<< notation, for example "Stephen King" or >>Stephen King<<. If you use the quote notation, and you want a quote character to appear inside the text, you have to write it as two quote characters. For example, write "This is some ""quoted"" text" to get the text "This is some "quoted" text". If you use the >>text<< notation, anything can appear inside the text, except ">>" and "<<". Thus, you can write quotes directly, as in >>This is some "quoted" text<<. The >>text<< notation is useful for long texts that contain many quote characters, such as HTML.

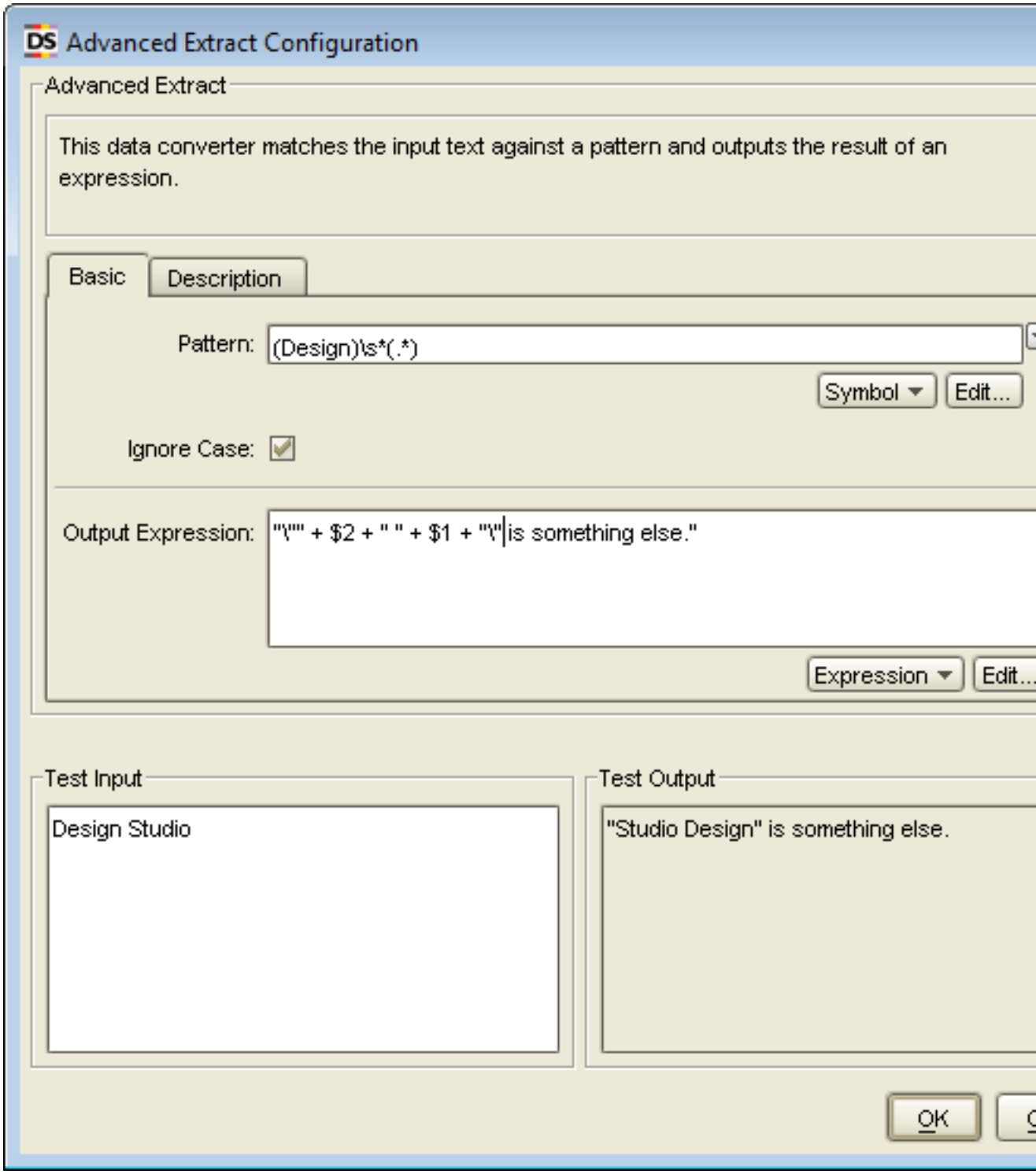
The following table shows the most commonly used functions in expressions. For a complete overview of all functions available, see the reference documentation on expressions [../ref/robomaker/reference/expression/Expression.html].

Table 4. The Most Commonly Used Sub-Expression Functions

Function	Meaning
toLowerCase(arg)	Converts the argument to lower case.
round(arg)	Rounds the argument to the nearest integer

Example: The expression "The discount is " + round((Item.oldPrice - Item.newPrice) / Item.oldPrice) + "%." evaluates to "The discount is 10%." when the item's old price is $99.95 and the new price is $89.95.

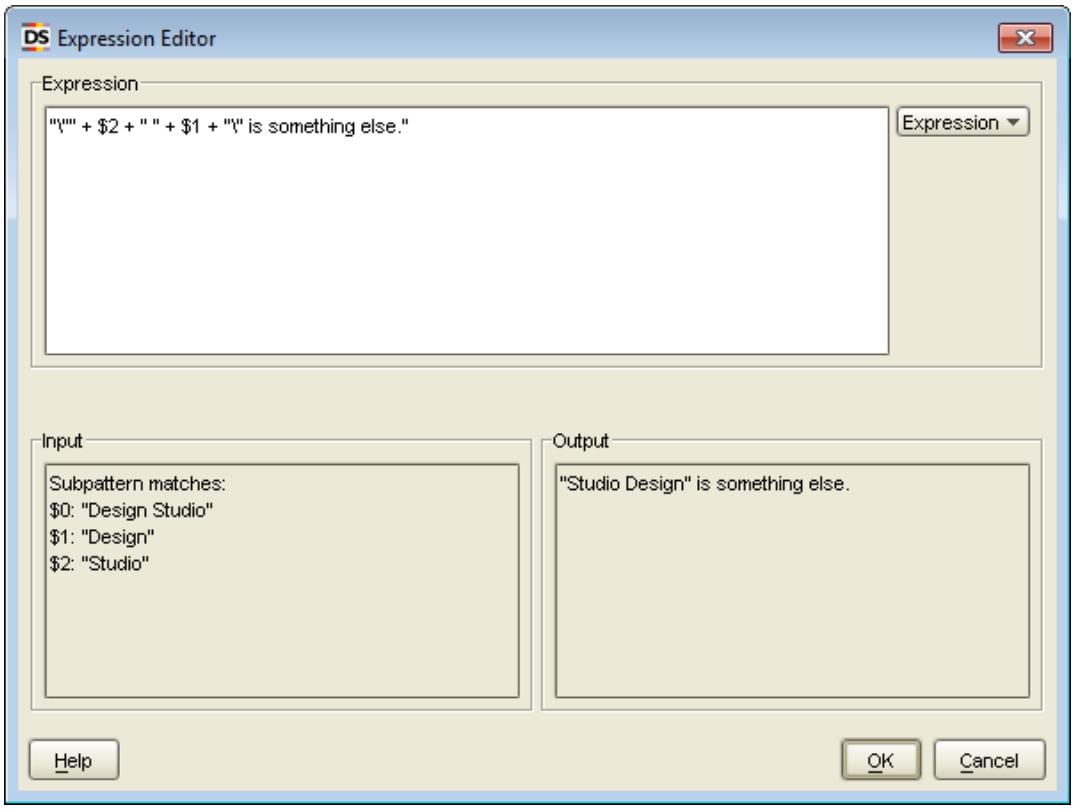
It is recommended that you experiment with expressions on your own. The best way to experiment with expressions is to launch Design Studio, select the Extract action for the current step, and then add an Advanced Extract data converter. Click the icon to configure the data converter. This opens the **Advanced Extract Configuration Window** shown below.



The Advanced Extract Configuration Window

In the example shown, note the use of the $n notation to extract parts of the input text. Try to type your own input text into the text area to the left, your own pattern into the "Pattern" property, and your own expression into the "Output Expression" property. You can follow the result in the right area, while typing the expression.

Also, try to click the "Edit..." button to the right of the expression field. This opens the **Expression Editor Window** shown below.



The Expression Editor Window

In the Expression Editor Window, you can enter an expression and test what it evaluates to. If the expression is associated with a pattern, as in the Advanced Extract data converter, the result of matching the pattern against the current input text will be shown in the "Input" panel. You can see whether the pattern matches, and if so, what subpattern matches your expression can refer to using the $n notation. Note that the testing functionality is not available everywhere in Design Studio.

Click the "Expression" button to open a useful pop-up menu, from which you can choose among the available sub-expression types and functions. This way, you don't have to memorize all of these.

For more on expressions, consult the reference documentation [../ref/robomaker/reference/expression/Expression.html].

Working with Projects and Libraries

When you are working in Design Studio you are always working with projects and you may have any number of projects open at any time. The purpose of a project is to develop a library containing a collection of robots and the files required by these robots. Typically, you create a project for each separate usage of robots, for example one project for each application in your company that uses robots. Two projects cannot share files, e.g. a type always belongs to one project or in other words the scope of a type is the project it belongs to.

A project is a folder located anywhere in the file system. The project folder can have any name you want, but must contain the following sub-folder:

`Library` — this folder contains the library of the project

In the `Library` folder, you should place all robot files, type files, and other files used by the robots, such as files that are loaded from the robot library. You can organize the files in the `Library` folder in any way you want, using sub-folders as appropriate.

The example below shows a project folder named `NewsAndStocksProject` for a project that develops a robot library for extracting news from news sites and stock quotes from stock sites.

Table 5. An Example of a Project for News Extraction

```
NewsAndStockProject/
   Library/
      News/
         CNN.robot
         Reuters.robot
         News.type
      Stocks/
         Nasdaq.robot
         NYSE.robot
         Stocks.type
```

As you can see, the project has a `Library` folder with robot and type files divided into News and Stocks sub-folders.

When you close Design Studio it will remember which projects and files were open and the next time you open Design Studio it will open these again.

Current Project

In Design Studio you can work with many projects, but the other applications in Kapow Katalyst, e.g. RoboServer, always works on a specific project, referred to as the **current project**. When you install Kapow Katalyst, a default project will be created for you and selected as current. If you open Design Studio the first time this current project will be the only opened project. If you close all projects before you close Design Studio then the next time you open Design Studio again it will open the selected current project.

You can change the current project selection by opening the Settings application, specifying the path to your new project folder in the Current Project Folder property in the "Project" tab, and then clicking OK to close Settings.

Manipulating Robot Projects

To create a new project you simply choose the "New Project..." from the File menu. This will open the New Project dialog as shown below. Here you enter a name and location for your new project. When you press Finish on the dialog a new project will be created in the location you specified where the name of the project folder will be the name you gave for the project. If you entered the name `MyProject` and the location: `C:/KapowProjects` then the following folders will be created:

```
c:/KapowProjects/MyProject
c:/KapowProjects/MyProject/Library
```

To open an existing choose the "Open Project..." from the File menu and choose the project folder of your existing project in the Open Project dialog that opens up.

To close a project again right click on the project in the Projects View and choose Close from the popup menu. You can also close all projects from the File menu.

Robot Library Files

When you want to distribute and deploy your robot library in a runtime environment, such as RoboServer, you can pack the robot library into a single file called a **robot library file**. You can do this by choosing Create Robot Library File from the Tools menu in Design Studio. This will pack together all files contained in the robot library of the file in the current editor and save the result as a single file having a name that you choose. Note that you should save all your open files, such as robots and types, before doing this, to get the latest changes into the robot library file. You can then make the robot library file available to RoboServer and execute robots from the robot library. See the RoboServer User's Guide for more information on how to do this.

As mentioned, a robot library may contain files used by the robots. You can load a page from a file in the robot library that the robot belongs to. This is done using the special non-standard protocol named `library`. For example, if the file `MyPage.html` is located in the folder `MyFolder` in the robot library folder, you can load from that file using this URL:

```
library:/MyFolder/MyPage.html
```

This will work no matter whether the robot library is represented as a folder or has been packed into a robot library file.

Interacting with Databases

Interaction with databases is often required. The following subsections explains when and how that is done from Design Studio

Database Mappings

Robots may need to access databases through various database accessing steps (e.g. "Store In Database"), and in order for them to do so, a reference to a named database must be provided for these steps. The named databases used by a robot must be accessible from the RoboServers in order for the robot to be executed successfully on RoboServers.

While designing robots in Design Studio, however, it is often convenient to use local databases that will not be available from the RoboServers. Rather than having to remember to change the named databases on the various database accessing steps before deploying a robot, Design Studio has an extra layer of abstraction to help overcome this problem; the database mapping. A database mapping maps a named database in a database accessing step of a robot to a Design Studio database. As long as the robot is executed from within Design Studio the named databases of the database accessing steps are mapped to the Design Studio databases specified by the mappings. The user of Design Studio can thus use local databases while designing and testing robots without having to change the referenced named databases of the database accessing steps before deploying the robots.

Using database mappings also makes it easy for the user of Design Studio to make the robot store values in a different database - it is simply a matter of reconfiguring the mapping to make it point to a different database.

Generally, working with database mappings in Design Studio is quite simple, but there are a few things which are nice to know.

A database mapping is a small configuration file defining simply which database to map to as well as whether Design Studio should display various warnings helping the user correctly configure the mapping and the referenced database. The name of the mapping is the file name of the configuration file. This means

that if you create a mapping with the file name "objectdb", the database that the mapping points to will be accessible under the name "objectdb" in robots.

You can create a database mapping in several ways:

- In the File menu, select "New Database Mapping". This will open a wizard which prompts for a name for the mapping file and displays the mapping configuration options. When the wizard is finished, the mapping will be created in the selected project and folder and will now be usable in robots.

- In the database view, right-click the database that you wish to associate with a project, select "Add to Project" and simply select the project to add the database to. This will prompt you for a name to use for the database mapping (the mapping file name and the name that the database will be accessible under). Notice that a name is suggested. This is the default database name, and the name used to access this database in other Kapow Katalyst applications than Design Studio.

- Through the database warning system. When, in Design Studio, you open a robot using a database that you do not have a mapping for, a warning will be displayed informing you of this and asking whether you want to create a mapping with the name of the database referenced in the robot. This allows you to, for instance quickly run robots sent to you from other developers who have other databases defined - without modifying the robots.

Types and Databases

If your robot outputs the values of variables to a database, the types of these variables need to define which attributes must be part of the key that is used when storing the values in the database. The Database Key for the value is calculated as a secure hash of the attributes that are marked to be part of the Database Key.

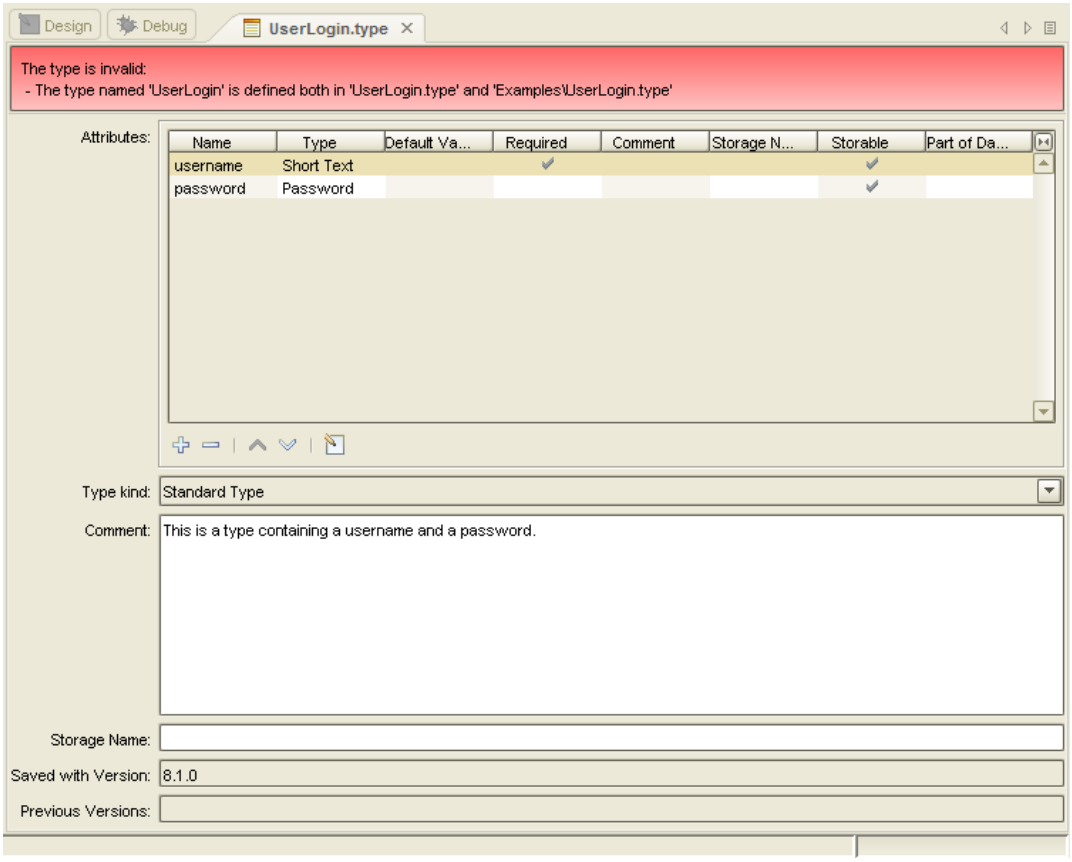
You may also specify a Storage Name as part of your attribute definition. This is an optional different name to use when storing the attribute.

When saving values to database storage using the Store in Database action, it is necessary that the appropriate database table exists in an available database. This means that it is a requirement that the table contains columns matching the attributes of the type.

See Creating and Deleting Database Tables for more information on how Design Studio can assist you in setting up the appropriate database tables. Consult the Settings in Design Studio section for more information on setting up database connections in Design Studio.

Database Warnings

Database warnings help you configure the database mappings, the robots and the referenced databases correctly. The warning system automatically monitors for potential problems such as type validation issues, missing tables, or missing database mappings and if an issue arises, a warning message is displayed in a status bar (see figure below).



Invalid Type in the Type Editor

Also, the warning system will monitor which databases names are used in robots and assist in creating any missing mappings. The system also perform a shallow monitoring of the databases which means that it does not constantly ping the databases to see if they are online, but rather updates the information when it needs it. This means that the system caches the table structures of the relevant database tables and uses this cache to compute any warnings in order to prevent an excessive number of database queries. The cache will be re-created when Design Studio knows something has happened that requires it. Any external modifications of the database tables or the database availability are not monitored. There is, however, the option of re-building the database cache for a single database or for all databases. This option is available through the database view, and through some warnings when relevant. For instance, to re-create the cache for a database, right-click it in the Database View and select "Refresh".

Creating and Deleting Database Tables

If you want to store your extracted variable values in a database, you should create matching tables in the database. Design Studio can assist you in creating these tables. It does so by examining the types you have created, and generating SQL that is appropriate for creating a new database table. Thus, when storing the value from a variable of some type, tables representing that type must be present in the database.

In the Tools menu, select Create Database Table () to open a window where you can choose the name of your database, the type of database, and the type(s) that you want to create the table(s) for. By clicking Generate SQL, you will be shown a suggestion for an SQL statement for creating the table(s), which you then have the option to modify as well as execute or save.

The SQL shown is a recommended suggestion — you can change the statement to fit your needs, if required. For example, you might change the column type for a Short Text attribute from

"VARCHAR(255)" to "VARCHAR(50)" in order to conserve database space, or you could add an auto-incrementing primary key. However, under normal circumstances, you should not modify the table name or any of the column names, nor remove any of the columns.

Note: If the database table already exists, it will be dropped from the database when executing the SQL (because of the "DROP TABLE" statement at the beginning).

Storing Data in Databases

This section explains how the database storage of Kapow Katalyst works.

ObjectKey

The tables you create for a type in a database will have a column for each of the attributes in your type, plus an additional 7 house-hold fields, named: ObjectKey, RobotName, ExecutionId, FirstExtracted, LastExtracted, ExtractedInLastRun, and LastUpdated. The most important of these is the ObjectKey, as it is the primary key for the table.

Note: The reason for the name 'ObjectKey' is to be found in the terminology previously used in Kapow Katalyst. Previously, types and variables were called 'objects'. Refer to this upgrade guide for more information. To adhere to the new terminology, 'ObjectKey' should be called 'ValueKey'. Renaming it would cause quite a lot of backward compatibility problems, though, and therefore it has been allowed to keep its old name.

The ObjectKey for a type is what uniquely identifies values extracted from variables of that type when stored in a database. You therefore have to figure out what uniquely identifies values of the type. If you are building a car repository, the VIN number may be enough to provide unique identification of each car. If you are collecting baseball results, you may need *the year, the team names, the ballpark, the date*, to uniquely identify each match.

As you build the type you can select how the ObjectKey is going to be calculated. This is done by checking the "*part of database key*" option when creating a new attribute. For our car example the VIN number would be the only attribute marked *part of the database key*, for the baseball match example, the attributes *year, team names, ballpark, date*, would all be marked as *part of database key*.

The robot developer may also specify the Key directly on the Store in Database action, if he wishes to override the default algorithm defined on the type.

The attributes which are not *Part of Database Key* are sometimes referred to as non-key fields. The car could for example have a price attribute, but even if the price changed we would still consider it the same car.

Store in Database

Kapow Katalyst provides 3 actions for managing values in a database: Store in Database [../ref/robomaker/reference/stepaction/StoreInDatabaseStepAction.html], Find in Database [../ref/robomaker/reference/stepaction/StoreInDatabaseStepAction.html], Delete from Database [../ref/robomaker/reference/stepaction/DeleteFromDatabaseStepAction.html]. The Find and Delete actions are rather trivial, but Store in Database does more than just store the value.

Store in Database may either insert a new value (of some type) into the table, or update an existing value which was previously stored. Here is a list of exactly what happens.

1. When storing the value of some variable, the ObjectKey is calculated based on the variable's values of the attributes which in the variable's type are marked *Part of Database Key*, or, alternatively, if the robot developer specifies a key on the action, this key is used instead.

2. Using the calculated key, a check is made to see if the value already exists in the database.

3. If the value does **not exist**, a new row is inserted into the database (under this ObjectKey).

4. If the value already **exists**, it is updated, that is, all the non-key attributes are written to the table (under this ObjectKey).

House-hold fields

Whenever a value is inserted all the 7 house-hold fields are updated. On update, only some of the fields change. The table below provides an overview.

Table 6. The 7 house-hold fields, and when they change

Field	Description	Changed on
ObjectKey	The primary key for this value	Insert
RobotName	The name of the robot which stored this value	Insert and Update
ExecutionId	The execution id for the robot execution which stored this value	Insert and Update
FirstExtracted	The first time the value was stored.	Insert
LastExtracted	The last time the value was stored.	Insert and Update
LastUpdated	The date when the value was last updated.	Update*
ExtractedInLastRun	If the value was extracted in the latest run (uses 'y' and 'n').	Insert and Update*

After each robot execution (in which the robot used Store in Database), all values that have previously been collected by this robot, but have not been stored during this run will have ExtractedInLastRun set to 'n' and LastUpdated set to now, indicating that the value was not found on the website during the latest run.

Note: If a value was found in the previous run, but no non-key fields have changed, then LastUpdated will not be updated. However, if the value was not found in the previous run, but in a run prior to that, LastUpdated will be updated even if the non-key fields have not changed, as this means that the value was deleted from the site and then reappeared again later.

Harvest Tables

The tables created by Kapow Katalyst are often referred to as harvest tables, as the robots are harvesting data into these tables.

To find out what information was available on a website the last time the robot was run you can use SQL like:

```
SELECT * FROM table WHERE ExtractedInLastRun = 'y'
```

However if you are running queries against a table at the same time as a robot is storing data into the table, the result will be comprised of data from the previous run, mixed with whatever data the executing robot has stored so far. Kapow therefore recommends that you copy the data out of the harvest tables, and into a different set of production tables, so you can run your queries against a stable data set.

There are many solution where robots are used to store data in a database, but most of them fall under one of the 3 scenarios listed below.

Table 7. Copy harvest data to production tables

Scenario	Description
Repository matching website (small data sets)	The idea is to have a repository that matches the items on a website 1-to-1. The easiest way to accomplish this is have a production table, that is truncated (deleting all rows) every time the robot is done executing, and then copy every record from the harvest table where ExtractedInLastRun='y' into this table. This works well for small data sets.
Repository matching website (large data sets)	Same as above, but the data set is too large to copy all data after every robot execution, instead we want to update the production table after each robot execution based on the changes which have occurred. This is where the LastUpdated field comes in handy. All values which have been updated, will have a LastUpdated field value larger than the start time of the robot. You can get the start time from the database logging tables, or you can have the robot store it somewhere. You can detect deleted values like this `SELECT * FROM table WHERE LastUpdated > 'StartTime' AND ExtractedInLastRun = 'n'` You can detect new values like this `SELECT * FROM table WHERE LastUpdated > 'StartTime' AND ExtractedInLastRun = 'y' AND FirstExtracted > 'StartTime'` You can detect updated values like this `SELECT * FROM table WHERE LastUpdated > 'StartTime' AND ExtractedInLastRun = 'y' AND FirstExtracted < 'StartTime'` Then update your Production table accordingly.
Historic data	The default setup allows you to see when a value was first extracted and when it was last updated, but you cannot see if it was found in the 5th, 7th and 10th run of the robot, but not in the 6th and 8th run. In this case, you should copy all the data from your harvest table into another table after the robot run, but in your new table the ObjectKey should not be a primary key. Instead create an extra column called RUN_ID and use it together with the ObjectKey to create a compound primary key. If you don't need a RUN_ID you could simply create an auto incremented column and use that as the primary key of your secondary table. Truncate the harvest table before each run.

You don't have to copy all the house-hold fields to your production table, only the ObjectKey is required for you to update your production tables.

Concurrency considerations

If you have multiple robots storing values of the same type to the same database there are a few things you should be aware of.

- Every time a value is stored, the RobotName column is updated. If you have two robots storing the 'same' value (as identified by ObjectKey), only the last one will show after the robots are done executing.
- If two robots store the same value at 'exactly' the same time, you will get an error. They will both find that the value is not in the table, and try to insert it, but only one of them will succeed. In most cases the error can simply be ignored as it is in fact precisely the same value.

- If you run the same robot twice at the same time and the robot stores data in a database, you break the way the ExtractedInLastRun column is used!

  When the first robot is done executing, it will update the ExtractedInLastRun to 'n' for all values it has not stored. This includes all values stored by the second robot so far. Later when the second robot finishes, it will set ExtractedInLastRun to 'n' for all values stored by the first robot, completely negating the first run.

Value Relations

The storage system does not provide an automated way of managing relations between values. If you have a value of type Person and one of type Address, and you want to link them, you will have to maintain this link.

The easiest way to create a link is to have the ObjectKey of then Person value be a foreign key in the Address value which should be linked to this person.

If the ObjectKey is calculated automatically from the type, you can use the Calculate ObjectKey action, to generate the key and assign it to each of the address values before you store them.

You should be careful when building robots with relations between stored values. If an error occurs when you store the Person value, you should make sure that no Address values are stored.

ObjectKey Caveats

If you are using MySQL, Oracle or Sybase, you should be aware of the following with regards to ObjectKeys.

- On Oracle empty string is stored as null.
- On Sybase empty string is store as " " (a string with a single space).
- MySQL doesn't have millisecond precision on timestamps.

These three cases all result in a potential loss of data when the data is stored in the database. The ObjectKey is calculate inside the robot based on the data in the given variable, if you later load the value from the database and try to recalculate the ObjectKey, the ObjectKey will be different if data loss occurred in any of the attributes marked as part of database key.

Putting It All Together

By now, you have been introduced to the major concepts in Design Studio, you have taken a tour of the Design Studio user interface, you know the concepts of robots, types, step actions and data converters, and you have a feel of what patterns and expressions are.

Let us now put all this Design Studio-knowledge into use and make some robots. However, first we need an overview of how robots are normally built — that is, their structure.

Robots mimic human behavior — that is, they do (more or less) what you do when you are looking for content on the Internet using a browser: You start by searching for the content. Once found, you read and process it. Similarly, most robots can be roughly divided into two parts: A navigation part and an extraction part.

Navigation is concerned with "getting to where the content is." Navigation mainly includes loading pages and submitting forms. When navigating in Design Studio, you typically use the Click action to navigate to and between web pages.

Extraction is concerned with "getting the right content." Extraction mainly includes selecting, copying, and normalizing content from a web page that you have navigated to. When extracting in Design Studio, you typically use the Test Tag action to skip uninteresting ("noisy") content, the Extract action to copy content into variables, and the data converters for normalizing the content so that it gets the format you want, e.g. the right date and number format. Once extracted, you output the value with the "Store in Database" or "Return Value" action.

So, the typical robot starts with one or more steps, each containing a Load Page or Click action, in order navigate to the interesting content on some web site. It proceeds with one or more steps, each containing an Extract action, and ends with a step storing or returning the extracted value.

Note that in many robots the navigation and extraction parts overlap because the content to extract is located on several pages. Again, this is similar to when you look for content yourself; often, you have to visit several pages to get the content you want.

Most robots include other actions than the ones mentioned above, e.g. a For Each Tag action for loading several similar looking pages or extracting values from several similar looking table rows. Because robots have different tasks, they have different needs. For this reason, we have included a considerable number of step actions and data converters in Design Studio. Start with familiarizing yourself with the basic and most commonly used step actions and data converters, and then begin to explore. Experience shows that one can create most robots using only a handful of step actions and data converters. So, find your own favorite step actions and data converters and stick to them until you feel a need to explore others.

How to Write Well-Structured Robots

A robot is a program so writing well-structured robots is important for the same reasons as it is for writing programs in other programming languages. Writing unstructured robots is like writing books with no chapters and no table of contents. So writing well-structured robots is important because:

- it helps document the robots
- it makes it easier to maintain the robots
- it makes it easier to find your way around the robots

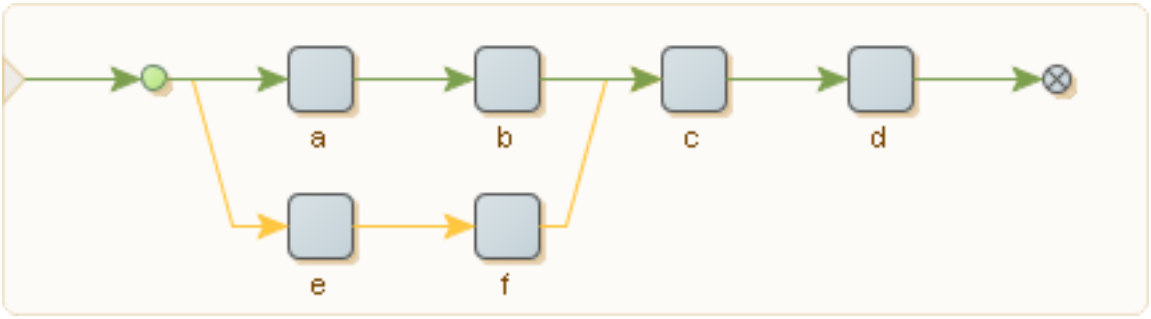
A side-effect of writing well-structured robots is that it often also makes robots load faster in Design Studio and generally be more responsive when edited, e.g. when updating the Robot View.

The two main 'tools' that you have at your disposal to write well-structured robots are: *Snippet steps* and *Group steps*. Both these two types of steps are used as a way to take a part of a robot, give it a descriptive name and let you pack it up in a single step. In this way you can forget what the part of the robot does in detail and concentrate on other issues of you robot. This is very similar to other concepts found in other programming languages, e.g. methods, function, procedures, etc.

You use a group step to pack up and hide steps that performs a well-defined task that you can give a descriptive name, e.g. Login to site X, Report error. It is important that you can give a relatively short descriptive name to the group step that describes what the steps inside the group does. If you can't then it is probably because they do not perform a well-defined task. Introducing a group step you help document your robot, because the name describes what that part of the robot does.

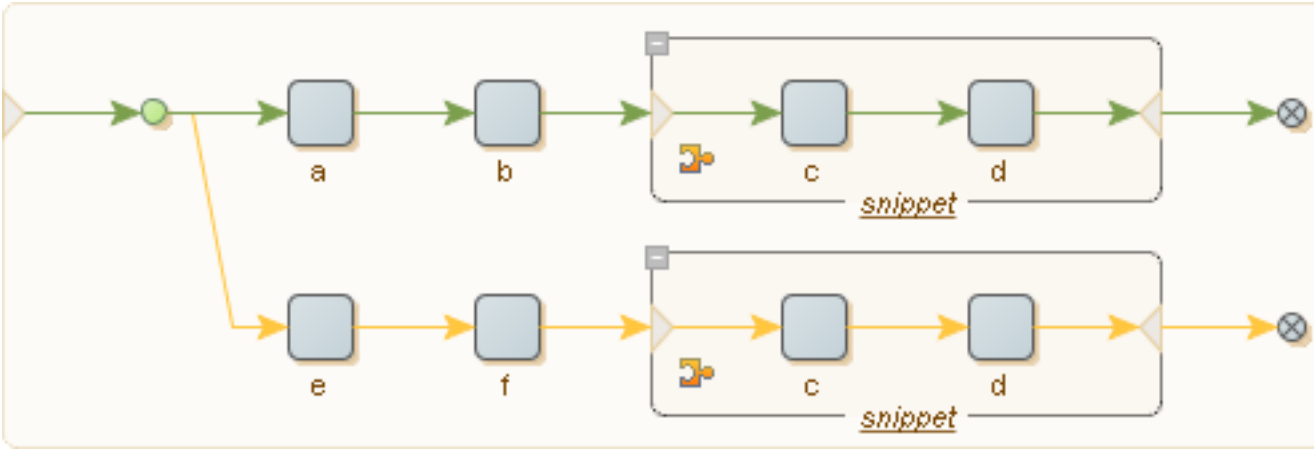
Although snippet is mainly introduced to share functionality between robots they can also be used inside a single robot to help structure this. If you have a collection of steps in robot that is used in several branches, e.g. by having edges from different part of the robot joining at the start of the steps, you can replace this kind of sharing of steps by introducing a snippet containing the steps.

Let us look at a simple example of one way one may structure a robot using snippets and groups instead of joining edges. Assume that our robot that looks like this:



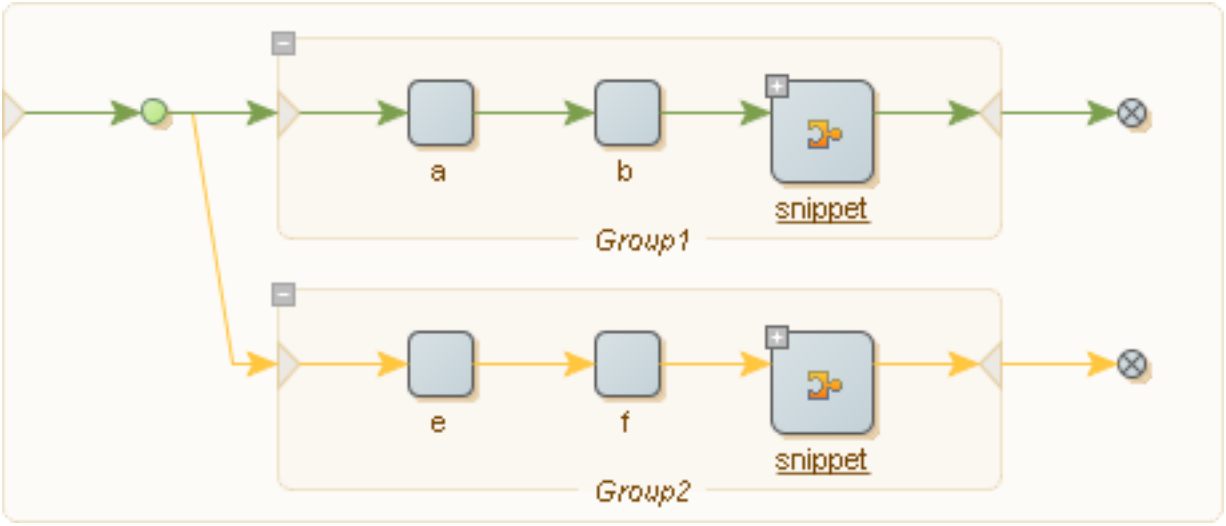
Sharing steps using joining edges

The last two steps c an d is shared by the two branches starting with the steps a and c. In real life you probably have a much larger robot and more than two branches sharing steps in this way and the steps involved may be far apart. This may result in the robot being very hard get an overview of. As a first step towards getting a better structured robot we introduce a snippet step containing the steps c and d. This looks like this:



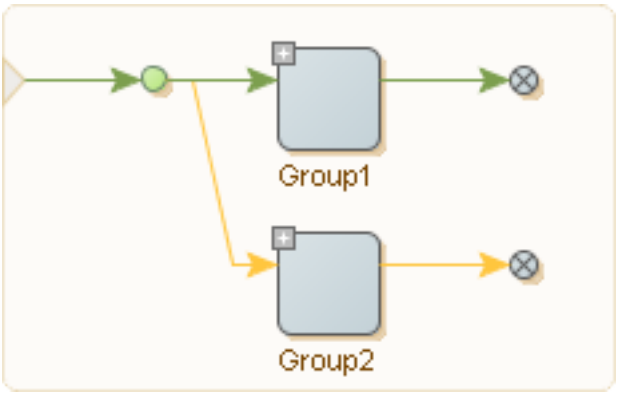
Sharing steps using a snippet

We can now edit the steps inside the snippet steps and still be sure that our changes are shared in the two branches. We can also further structure our robot by putting each of two branches into a group step:



Grouping the branches steps

Finally, if we close the two group steps we get the following simple robot:



The final result of the restructuring

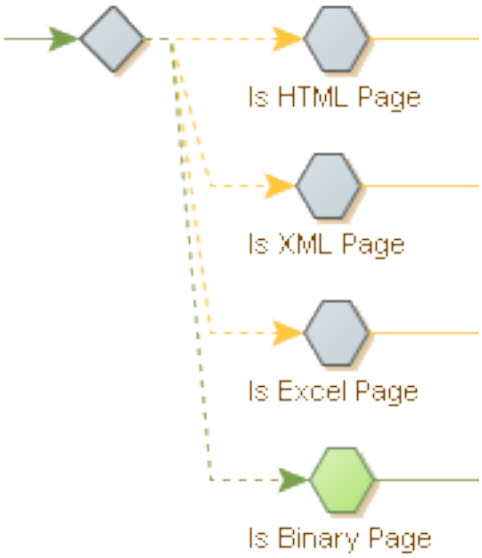
This resulting robot does two task, one performed by Group1 and the other performed by Group2. By giving these two groups descriptive names the robot has a more logical structure than the original robot.

Admittedly this is a very simple example, but when robots gets beyond a certain size and contains edges crisscrossing the Robot View it is very easy to lose overview of the robot giving a whole new meaning to the word *spaghetti code*. Restructuring the robot in the manner describe above may help regain that overview.

How to Determine the Type of a Page

When loading a page e.g. by clicking on a link in an HTML page you cannot be sure that the page that gets loaded is another HTML page. The loaded page could be an XML page, an Excel page or a page of some content type that Design Studio does not support. So how do you in a robot test what the page type of a loaded page is such that you can add branches to handle each page type separately? You do this using the Test Page Type step.

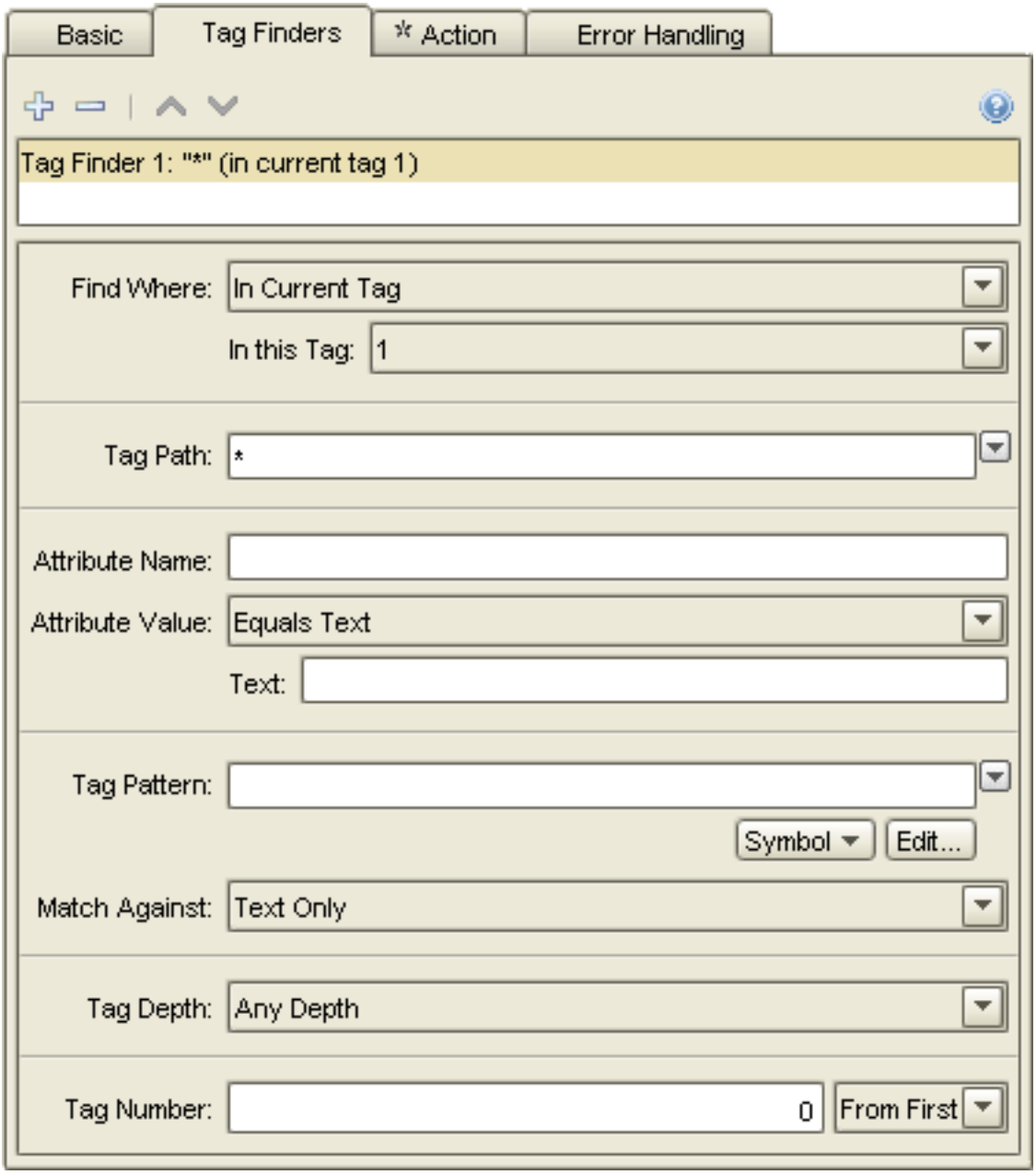
The Test Page Type step tests the type of the page in the current window and there are currently four page types: HTML, XML, Excel and Binary. The figure below shows a piece of a robot that tests for the four different page types. This consists of a Try step followed by four branches where each branch starts with a Test Page Type step configured to test for a each their page type. The names of the Test Page Type steps have been change to indicate what their page type is.



Steps That Test the Page Type

How to Use the Tag Finders

A Tag Finder is used to find a tag on a page. The most common use of a Tag Finder is in a step, where the Tag Finder is used to find a tag on which the selected action should be applied. The list of Tag Finders of the current step is located in the "Tag Finders" tab in the Step View, and is shown below.



The Finders Tab in the Step View

Understanding Tag Paths

To understand the Tag Finder, the concept of a **tag path** is important. A tag path is a compact text representation of where some tag is located on a page. Consider this tag path:

```
html.body.div.a
```

This tag path refers to an <a>-tag inside a -tag inside a <body>-tag inside an <html>-tag.

A tag path can match more than one tag on the same page. For example, the tag path above will match all of the <a>-tags on this page, except the third one:

```
<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>
```

You can use indexes to refer to specific tags among tags of the same type at that level. Consider this tag path:

```
html.body.div[1].a[0]
```

This tag path refers to the first <a>-tag in the second -tag in a <body>-tag inside an <html>-tag. So, on the page above, this tag path would only match the "Link 4" <a>-tag. Note that indexes in tag paths start from 0. If no index is specified for a given tag on a tag path, the path matches any tag of that type at that level, as we saw in the first tag path above. If the index is negative, the matching tags are counted backwards, i.e. starting with the last matching tag which corresponds to index -1. Consider this tag path:

```
html.body.div[-1].a[-2]
```

This tag path refers to the second-to-last <a>-tag in the last -tag in a <body>-tag inside an <html>-tag. So, on the page above, this tag path would only match the "Link 5" <a>-tag.

You can use an asterisk (‘*') to mean any number of tags of any type. For example, the tag path

```
html.*.table.*.a
```

refers to an <a>-tag located anywhere inside a <table>-tag, which itself can be located anywhere inside an <html>-tag. There is an implicit asterisk in front of any tag path, so you can simply write "table" instead of "*.table" to refer to any table tag on the page. The only exception is tag paths starting with a punctuation mark (‘.'), which means that there is no implicit asterisk in front of the tag path, so the tag path must match from the first (i.e. top-level) tag of the page.

With asterisks, you can create tag paths that are more robust against changes in the page, since you can leave out insignificant tags that are liable to change over time, such as layout related tags. However, using asterisks also increases the risk of accidentally locating the wrong tag.

You can provide a list of possible tags by separating them with '|', as in this tag path:

```
html.*.p|div|td.a
```

This tag path refers to an <a>-tag inside a <p>-, -, or <td>-tag located anywhere inside an <html>-tag.

In a tag path, text on a page is referred to just as any other tag, using the keyword "text". Although text is not technically a tag, it is treated and viewed as such in a tag path. For example, consider this HTML:

```
<html>
  <body>
    <a href="url...">Link 1</a>
    <a href="url...">Link 2</a>
  </body>
</html>
```

The tag path "html.body.a[1].text" would refer to the text "Link 2".

How the Tag Finder Works

A Tag Finder can be configured using the following properties:

Find Where: In this property, you can specify where to find the tag relative to a named tag. The default value is "Anywhere in Page", meaning that named tags are not used to find the tag.

Tag Path: In this property, you can specify the tag path as described in the previous section.

Attribute Name: In this property, you can specify that the tag must have a specific attribute, for example "align".

Attribute Value: In this property, you can specify that the tag must have an attribute with a specific value. If the Attribute Name property is set, the attribute value is bound to that specific attribute name.

- "Equals Text" specifies that the attribute value must match a specified text. Note that the text must match the entire attribute value.
- "Containing Text" specifies that the attribute value must contain the specified text.
- "Pattern" specifies that the attribute value must match a pattern. Note that the pattern must match the entire attribute value.

Tag Pattern: In this property, you can specify a pattern that the tag must match (including all tags inside it), for example ".*.*Stock Quotes.*.*". Some caution should be observed in using this property, since it can have considerable impact on the performance of you robot. This is because the "Tag Pattern" may be applied many times throughout a page just to find the one tag that it matches. One way to try and avoid this is to choose "Text Only" for the "Match Against" property.

Match Against: In this property, you can specify that the "Tag Pattern" should match only the text or the entire HTML of the tag. The default is to match only the text because this is normally much faster.

Tag Depth: This property determines which tag to use if matching tags are contained inside each other. The default value is "Any Depth" which accepts all matching tags. If you select "Outermost Tag", only the outermost tags are accepted, and similarly, if you select "Innermost Tag", only the innermost tags are accepted.

Tag Number: This property determines which tag to use if more than one tag match the tag path and the other criteria. You specify the number of the tag to use, either

counting forwards from the first tag or counting backwards from the last tag that matches.

For example, if you set the tag path to "table", the Tag Attribute property to "align=center", and the Tag Pattern property to ".*Business News.*", then the Tag Finder would locate the first <table>-tag that is center aligned and that contains the text "Business News".

Configuring the Tag Finders of the Current Step

In the Robot Editor, you can configure the Tag Finders of the current step in several ways. The first way is to configure it manually. Once configured (whether automatically or manually), you can click the icon in the Page View to see the tag found by the Tag Finder.

The second way to configure the Tag Finders is to select a tag in the Page View and click the icon. This will configure the Tag Finder to find the selected tag using a tag path in simple mode.

The third way is to right-click on a tag in the Page View and then select an action from the pop-up menu that appears. If you select "Use Tag" from the menu, the Tag Finder will be configured to find the right-clicked tag using a tag path in simple mode. Similarly, if you choose another action from the menu, this will select a corresponding step action and configure the Tag Finder to find the right-clicked tag.

The fourth way to configure the Tag Finders is to select a new step action. Some actions, when selected, configure the Tag Finders so that they find the tags typically used for that action. For example, the Submit Form action will use one Tag Finder and set its tag path to "form" to locate the first <form>-tag in the page.

How to Submit a Form

Submitting a form is a common task in a robot. For example, you may need to submit a search form to get the search results that you want to extract, or you may need to submit an order form to make an order transaction. In some cases, you do not need to actually submit the form, but simply want to create a URL that represents the form submission, or modify the current values in the form. In this section, you will learn how to do these things.

Simple Form Submission

The recommended and simplest way of submitting a form in Design Studio is similar to the way you submit a form in an ordinary browser: First fill in the form and then click the form submission button.

To fill in the form, you can use the following actions:

- Enter Text
- Enter Password
- Select Option
- Select Multiple Options
- Set Checkbox
- Select Radio Button

and to submit the form, you can use the Click action.

You can also loop through field values (text input) options or radio buttons by using the following actions:

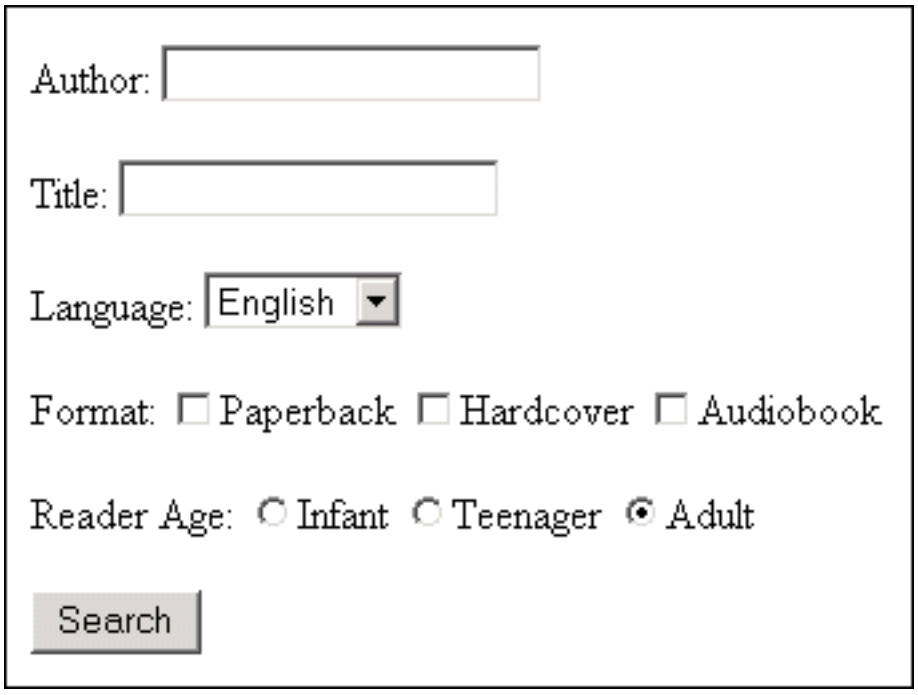
- Loop Field Values
- For Each Option
- For Each Radio Button

Form Basics

This section describes some basic properties of forms. Consider the following example of a book search form, first shown as HTML, then as it appears in a browser.

Table 8. A Book Search Form (as 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>
```



The Book Search Form in a Browser

A form contains a number of **fields**. For example, the first <input>-tag in the example form defines a field named "book_author". Note that the name of a field is usually different from what the user sees in a browser. For example, the "book_author" field will appear to be named "Author" in the browser, not "book_author".

A field can be defined by more than one tag. For example, the "book_format" field is defined by three <input>-tags in the example form. Tags that use the same field name and are of the same **field type** (text field, radio button, checkbox, etc.) define the same field.

A field can be assigned one or more **values**. For example, the "book_format" field can be assigned the value "format_pb" to select paperback format. Note that, like the field name, the value that is assigned to a field is usually different from what the user sees in a browser. For example, the user will see the text "Paperback", not the value "format_pb", when choosing the paperback format. Depending on the field type, some fields can be assigned more than one value at the same time. For example, since "book_format" is a checkbox field, we could assign both the value "format_pb" and the value "format_hc" to the "book_format" field to select both the paperback format and the hardcover format.

Most fields have a **default value**. The default value is the value that is initially assigned to the field in the form. For example, the "book_language" field has the default value "lang_0", because of the "selected" attribute.

A form is **submitted** by sending the current values of the fields to the web site. Only fields that have one or more current values are sent. For example, if none of the checkboxes of the "book_format" field in the example form are checked, no value is sent for that field.

In a browser, the submission of a form usually happens when the user clicks a **submit button**. There are two kinds of submit buttons: **normal submit buttons** and **image submit buttons**. Normal submit buttons are defined using a <button>-tag or an <input>-tag, in both cases with the "type" attribute set to "submit". If a normal submit button has a field name and value, that field will be sent with the specified value when the button is clicked.

Image submit buttons are defined using an <input>-tag with the "type" attribute set to "image". An image submit button defines two fields, named "*button name*.x" and "*button name*.y", where *button name* is the name contained in the "name" attribute of the <input>-tag. If the <input>-tag has no "name" attribute, the fields will be named "x" and "y". When an image submit button is clicked, these two fields are assigned the x- and y-coordinates of the position in the image where the mouse was clicked. Some web sites use this for creating image maps with different behavior depending on where the user clicks.

Some forms use JavaScript. For example, the <form>-tag may have an "onsubmit" attribute that contains JavaScript to be executed before the form is submitted. Similarly, an <input>-tag may have an "onclick" attribute that contains JavaScript to be executed when the user clicks on the field. The robot will automatically execute this JavaScript.

In some cases you might for performance reasons choose to ignore the JavaScript execution when submitting the form. To do this you must deselect the "Execute JavaScript" in the options [../ref/robomaker/reference/stepaction/support/BrowserEngine.html] in the form submitting step.

Which Step Action Should I Use?

As mentioned, the simplest way to submit a form is to fill in the form using the appropriate actions for this as described earlier.

Sometimes you need to loop through a form if you cannot get the desired result in a single form submission. Consider the book search example form. If you want to search for books in all available languages and for all reader ages, you cannot do this in a single form submission, because the site will not allow such a general search. Instead, you have to loop through the languages and the reader ages, and make a form submission for each combination of language and age. To do this you use the Loop Form actions: the Loop Field Values, For Each Option and For Each Radio Button. The Loop Form actions does not submit the form, so this must be done separately in a subsequent Click action that click on one of the submit buttons of the form. To loop over combination of field values you just place several steps with Loop Form actions after each other before the Click action that submits the form.

Use the Extract URL action on the submit button of the form, if you just want create a URL that represents a submission of the form.

Using the Loop Form Actions

There are three Loop Form actions: Loop Field Values, For Each Option and For Each Radio Button corresponding to the three kinds form controls text input (INPUT elements with type "text" and TEXTAREA elements), options (SELECT elements) and radio buttons (INPUT elements with type "radio"). Watch the video below or read on to learn how to use these loops.
Videos are not included in the PDF. Please visit open this url http://player.vimeo.com/video/38922810 in a browser to see the video.
Video on the use of Loops in Forms

To loop over a form you need to decide which form controls to loop over and in which order (this will decide in which order your output values will be generated). When you have done this you insert a step for each of these with the corresponding Form action. This can be done by right clicking on the control in the Page view and choosing "Forms" | <*form action*> from the popup menu, where <*form action*> is the appropriate action, e.g. if the control is a text input control then you would choose "Forms" | "Loop Field Values".

Every time a Loop Form action is executed a value is changed in a form control element in the HTLM page. This corresponds to what you would have done manually in a browser and if the form control has a JavaScript event attached to it, this event would be fired and some JavaScript executed. In some cases this

JavaScript may change the form, e.g. change the options of a SELECT element. If this is the case you must be careful and choose the right order in which to loop over the controls to ensure that the right options are available to the robot when it need them. Normally, if you follow the order that you would use when doing this manually in a browser it should work just fine.

Once all the Loop Form action steps has been inserted in the robot you should add a step with a Click action that clicks on one of the submit buttons of the form.

Uploading Files

Some forms contain file fields that allow you to upload files. A file field is defined by an <input>-tag of type file, such as the following:

```
<INPUT type="file" name="attachedFile">
```

In the Select File action, there are two ways to upload a file using a file field like this:

The first way is to upload a file from the file system. To do this, select "File in Local File System" from the drop-down box and enter the file name. When the form is submitted, the specified file will be loaded from the file system and uploaded as part of the form submission.

Note that the file name must be an absolute file name, including the drive name, if any, and the directory path to the file.

The second and most common way to upload a file is to specify the file contents to upload, instead of loading the file from the file system. To do this, select "File Contained in Variable" from the drop-down box. Then, you may select the variable that holds the file contents from the drop-down box named "File Content". Typically, you will get the contents from either a binary variable in which you have downloaded the file earlier using the "Extract Target" action, or from a variable containing text that you have extracted earlier.

Optionally, you can specify the content type and the file name of the file. The content type should be the MIME type of the contents, optionally followed by a charset. You may use one of the predefined content types, acquire it from an attribute or specify a custom content type. For example, the content type could look like this for an image:

```
image/gif
```

and like this for a plain text:

```
text/plain; charset=iso-8859-1
```

Note that when downloading files using "Extract Target", you can store the content type and file name of the downloaded data in other variables. You can then use this information when uploading the file with the "Select File" action.

Using the Pop-up Menu in the Page View

You can use the pop-up menu in the Page View as a shortcut for selecting and configuring the Submit Form and Loop Form actions. To select the Submit Form or Loop Form action in the current step, right-click inside a <form>-tag in the Page View and choose "Use Submit Form" or "Use Loop Form" in the "Forms" submenu of the pop-up menu.

If the current step contains a Submit Form or Loop Form action, you can right-click on a field in the Page View and choose "Add Assignment to Field" in the "Forms" submenu in order to assign a value to that field. A dialog will appear where you can select the value to assign.

If the current step contains a Loop Form action, you can right-click on a field and choose "Add Looping over Field" in the "Forms" submenu in order to loop through the field. A dialog will appear where you can configure a "One field with values to loop through" field group for the field.

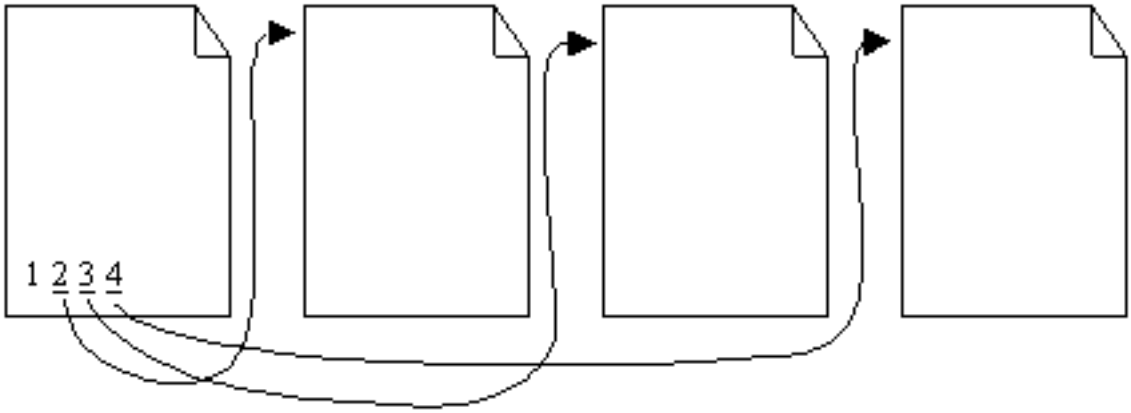
If the current step contains a Submit Form or Loop Form action, you can also right-click on a submit button in the Page View and choose "Select Submit Button" in the "Forms" submenu in order to select that submit button.

How to Loop Through HTML Pages

A robot often needs to loop through pages. For example, many web sites that present the results of a search request will do so over several pages, each containing e.g. 20 results from the search. To get the search results, you need to loop through the pages and process one page at a time. This section explains how to do this.

Pages where First Page Links to All Other Pages

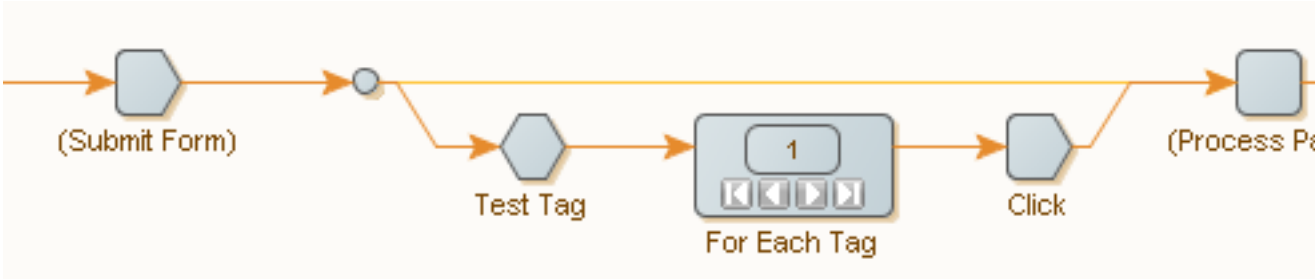
There are two common ways of linking pages together. The first one is shown below.



First Page Links to All Other Pages

Here, the first page contains direct links to all other pages. That is, you can get to any page directly from the first page by following the corresponding link. The first page sometimes also contains a link to itself.

Such pages can be looped through quite easily using a For Each Tag step, as shown in this excerpt from a robot:

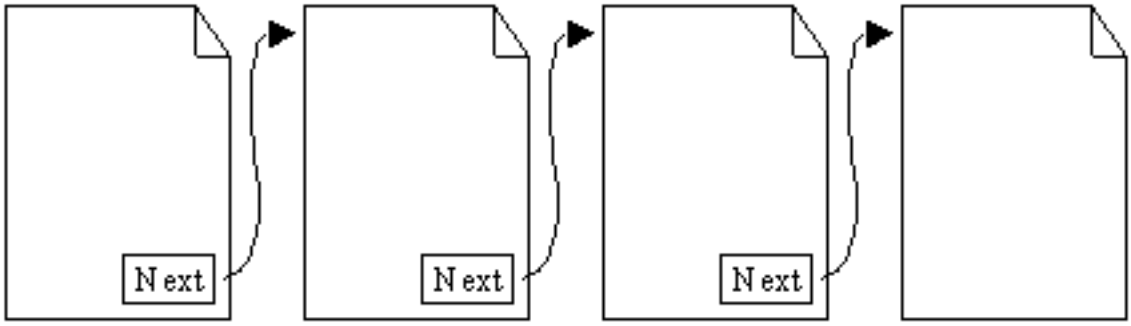


Here, we are looping through the result pages from a search request, symbolized by the step named "(Submit Form)". The first result page can be processed directly, so there is a connection from the form submission step directly to the step that processes a page, symbolized by the step named "(Process Page)". The remaining pages are looped through by the For Each Tag action in the second branch from the form submission step. First, the Test Tag step checks that there is in fact more than one page. If so, we simply

loop through the tags containing the links to the pages, load each page using a Click action, and then continue to the processing of the page. If the first page has a link to itself, the For Each Tag action should be configured to skip this first link, so that the first page isn't processed twice.

Pages where Each Page Links to Next

The other common way of linking pages together is shown below.



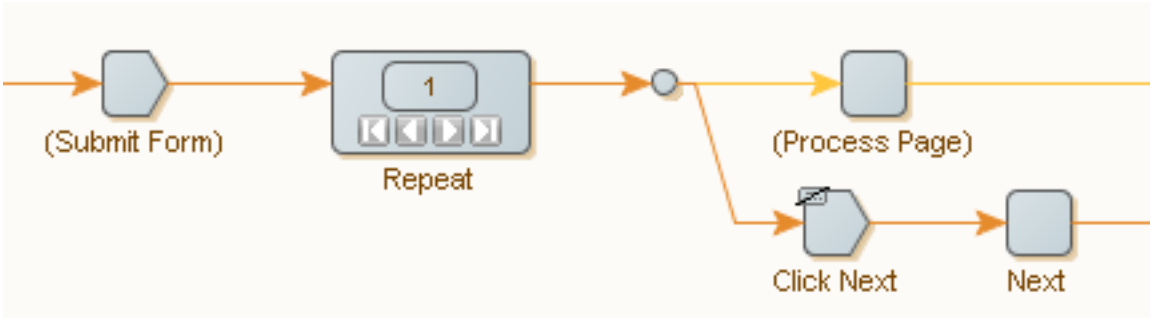
Each Page Links to Next

Here, each page simply links to the next page, typically with a link or form button named something like "Next".

To loop through such pages, use the Repeat action. The Repeat action will loop through the pages that are supplied to it by another action named Next. Watch the video below or read on to learn how.
Videos are not included in the PDF. Please visit open this url http://player.vimeo.com/video/38922812 in a browser to see the video.
Video demonstrating how to use the Repeat-Next loop.

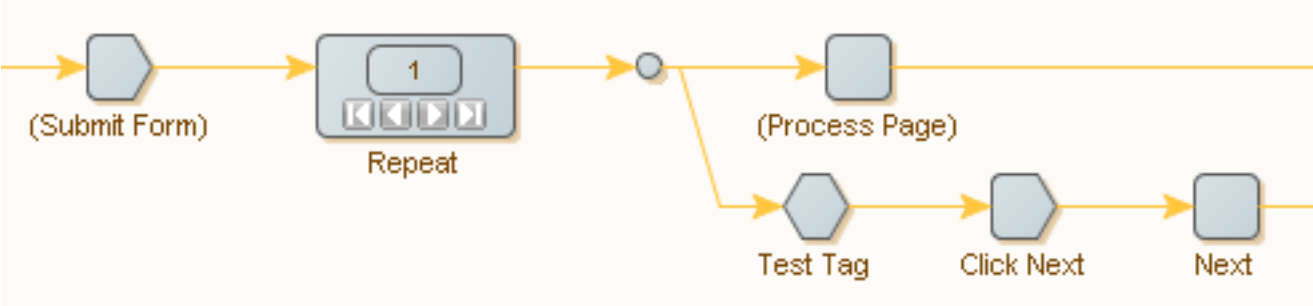
The principle is as follows: The Repeat action must be given the first page as input. It will then loop through the pages, and in each iteration it will output a page. In each iteration, we can process the current page, and we must also give the Repeat action the next page using the Next action. If we don't give the Repeat action a new page, it will not provide another iteration, i.e. the loop will end.

This excerpt from a robot shows an example:



Here, like before, we are looping through the result pages from a search request, symbolized by the step named "(Submit Form)". The form submission step will output the first result page, which we give to the Repeat action. In the first branch from the Repeat action, we process the current page. In the second branch, we load the next page by clicking its link. The Next action will send the page back to the Repeat action, which will output it in its next iteration. When the last page is reached, the Click action will generate an error. Therefore, the "Click" step is configured to terminate the loop. In the "Click" step, this is done in the "Error Handling" tab by setting the "Then" property to "Break Loop". Please see How to Handle Errors for more information on this.

An alternative way of handling the last page is shown in the robot excerpt below:



To detect when the last page has been reached, we use a Test Tag action in the second branch. The Test Tag action checks that the page contains a next-page link, for example by looking for an <a>-tag containing the text "Next". If the page contains such a link, we load this page and give this to the Next action. When the last page is reached, the Test Tag action will stop execution down the second branch, and no new page will be given to the Repeat action, causing the loop to end.

Note that finding the link to the next page can be tricky. A common mistake is to find the previous-page link on some pages instead of the next-page link, because the layout of the pages changes slightly between the first page, the subsequent pages, and the last page. Another common mistake is to not detect the last page reliably. You may have to configure the tag finders of the steps carefully to make things work (see How to Use the Tag Finders).

When you are working with a robot in Design Studio, Design Studio may not always be able to step correctly back and forth between iterations of a Repeat action. If you are not sure whether Design Studio has got it right, click Refresh to update.

How to Extract Content from HTML

Design Studio has seven step actions for extracting content from a tag in an HTML page:

- The *Extract* action is used to extract text content from the tag, optionally including the HTML tags.
- The *Extract URL* action is used to extract a URL from a tag attribute containing a URL, and making that URL absolute.
- The *Extract Clip* action is used to extract a stand-alone HTML clip from the tag, with support for preparing the HTML to appear on its own apart from its original HTML page.
- The *Extract Tag Attribute* action is used to extract the value of a tag attribute.
- The *Extract Target* action is used to extract binary data such as images and PDF files, but it handles any kind of binary data.
- The *Extract Form Parameter* action is used to extract a form parameter from a form URL in the found tag and then store its value in a variable.
- The *Extract Selected Option* action is used to extract the selected option from a <select>-tag and then store this in a variable.

Often you need to reformat (or normalize) the extracted content, and the Extract, Extract Clip, and Extract Tag Attribute actions allow you to do this by configuring a list of data converters.

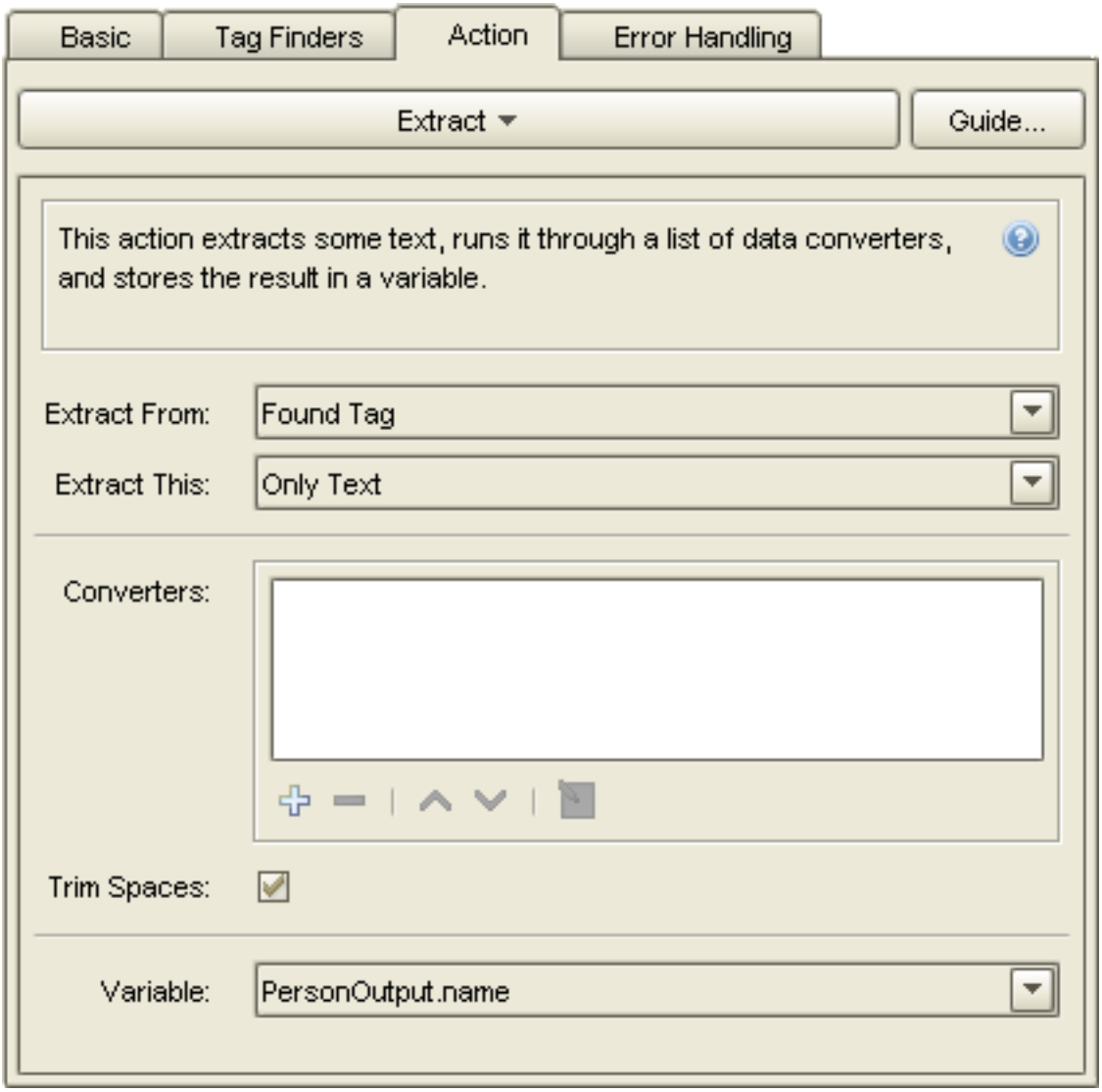
There are also two actions to extract data from various binary data formats, e.g. PDF or Flash. These are different from the ones above in that they extract the data and produce a HTML page that contains the

data in some structured form that lets your robot access the data. These actions are however used in an initial step before the actual data extraction, in which you may loop over the produced HTML and extract text from this.

- The *Extract Text from PDF* action is used to extract text from a PDF document contained as binary data in a selected attribute.
- The *Extract from Flash* action is used to extract data from a Flash object in a found tag.

Extracting Text

The Extract action is used for extracting text.



The Extract Action

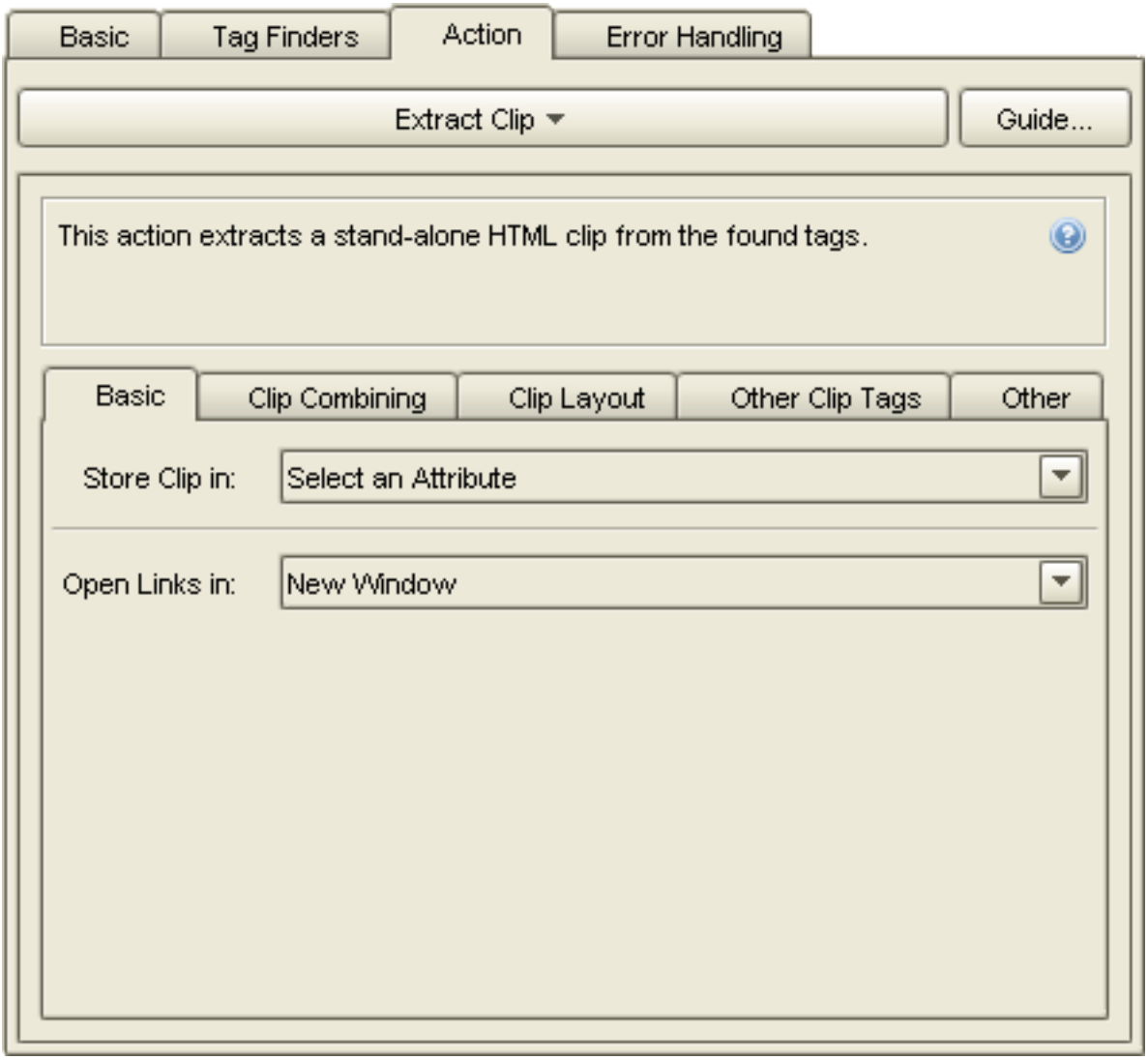
For short text, like a product name or a price, extract as "Only Text". This will simply extract the text between the tags.

If you want to extract a longer text with sections, headings etc. as plain text, but still want the text to appear close to how it appears in a browser, you should extract the text as "Structured Text". If some sort of

special markup is desired, e.g. brackets surrounding the headings, then "Structured Text" has rudimentary support for that. If the markup requirements cannot be fulfilled with "Structured Text", then use "Advanced Structured Text" which allows you to set mappings from the HTML tags into your proprietary markup.

Extracting Clips

The Extract Clip action is used for extracting a stand-alone HTML clip from a page.



The Extract Clip Action

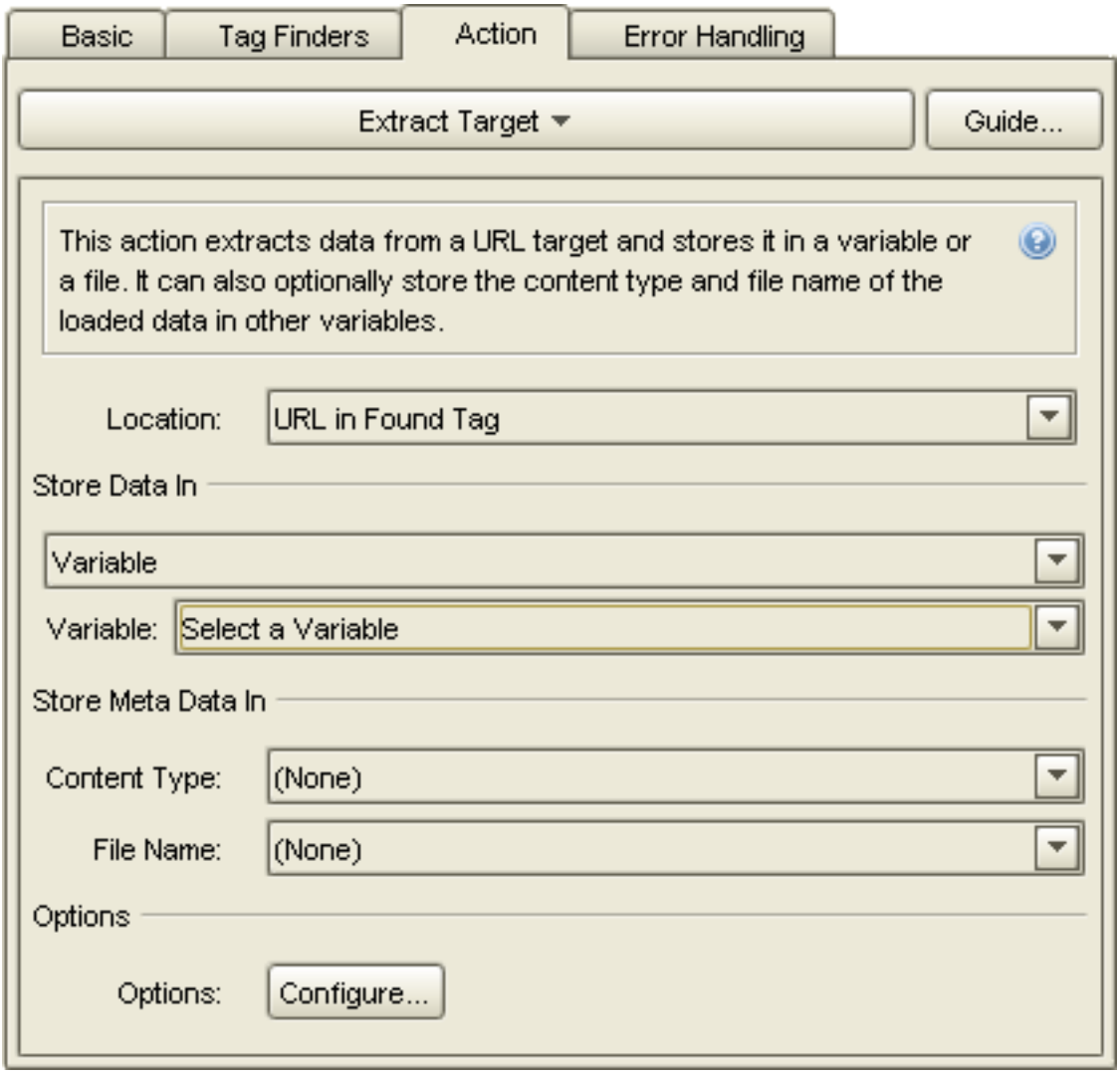
The Extract Clip action is useful when you want to extract parts of a page, or an entire page, and want to preserve the HTML formatting of what you extract. For example, you can use Extract Clip to extract web content with the original formatting preserved.

The Extract Clip action allows you to extract more than one tag from the same page and combine them into a single clip. You can choose between several ways of combining the individual clips.

The Extract Clip action can also modify and adjust the clipped HTML in various ways that make it suitable for appearing on its own, separate from its original HTML page. This includes handling of layout, URLs, and JavaScript.

Extracting Binary Data

Binary data is extracted using the Extract Target action, which will load the data from a URL and store it in a variable or directly into a file.



The Extract Target Action

Typically, binary variables are used to store the loaded data. The available types of binary variables are Binary, Image, PDF, and Session. They are all equivalent except that the Image, PDF, and Session types allow you to preview the data.

Using the Pop-up Menu in the Page View

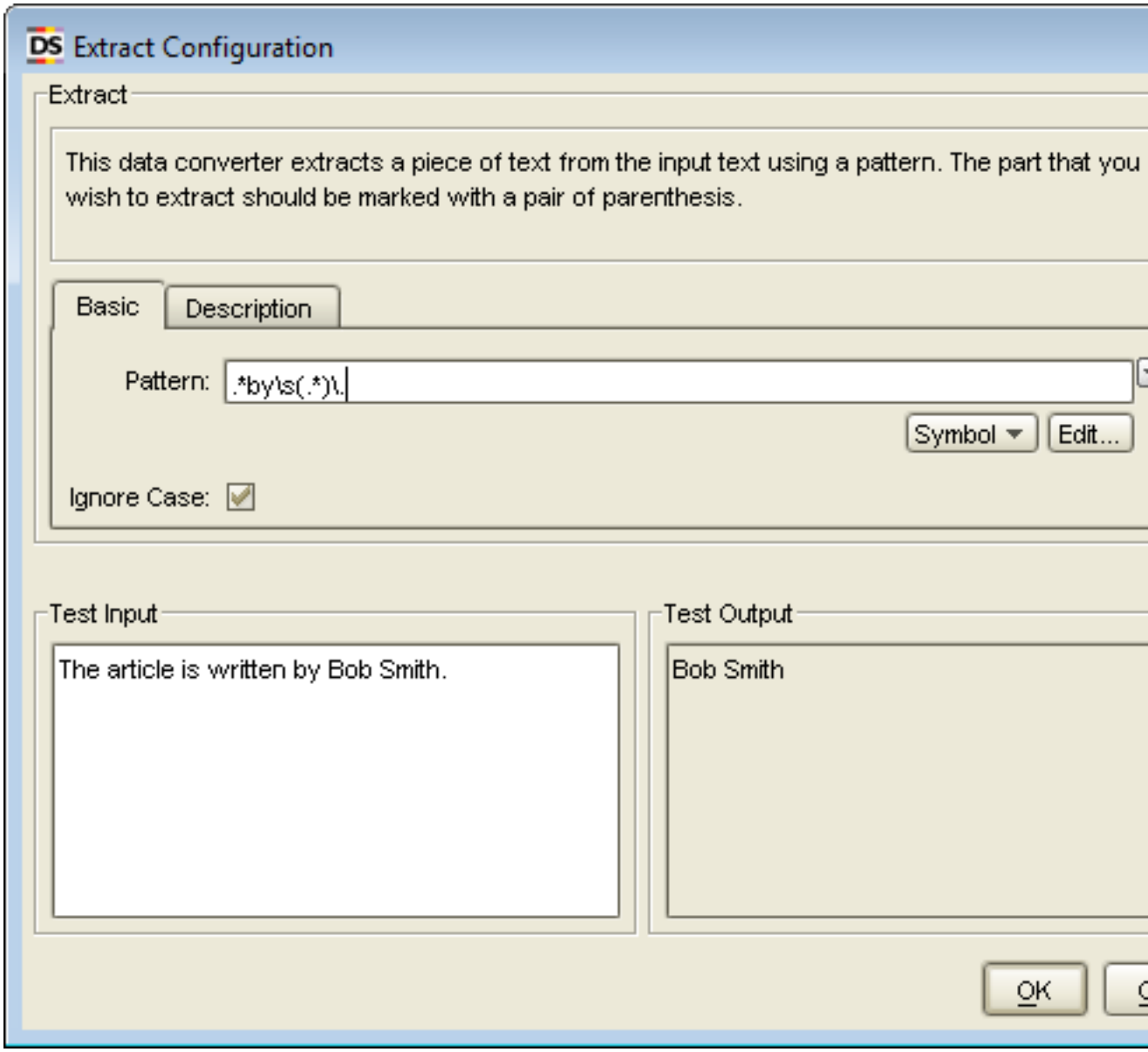
You can use the pop-up menu in the Page View as a shortcut to selecting and configuring the extraction step actions. Simply right-click on the text or tag that you want to extract from, or the link that you want to load from, and select the appropriate option from the "Extraction" menu in the pop-up menu that appears.

Performing Common Tasks

In this section, we will take a look at some common extraction tasks that you should be familiar with.

Extracting Only Part of a Text

If you want to extract only a part of the text in a tag, then you can use patterns on the text in the tag. For example, you might want to extract the name "Bob Smith" from the following text: "The article is written by Bob Smith." To do this, use the Extract data converter (do not confuse this with the Extract step action) and configure it as shown below.



Using the Extract Data Converter

The principle is to configure the Pattern property to match the entire text, with the text to extract being matched by a subpattern, enclosed by parentheses. In this case, the pattern used is ".*by\s(.*)\.", which means that the text between "by " and the period will be matched by the subpattern. For more information on patterns, see Patterns.

Converting Content

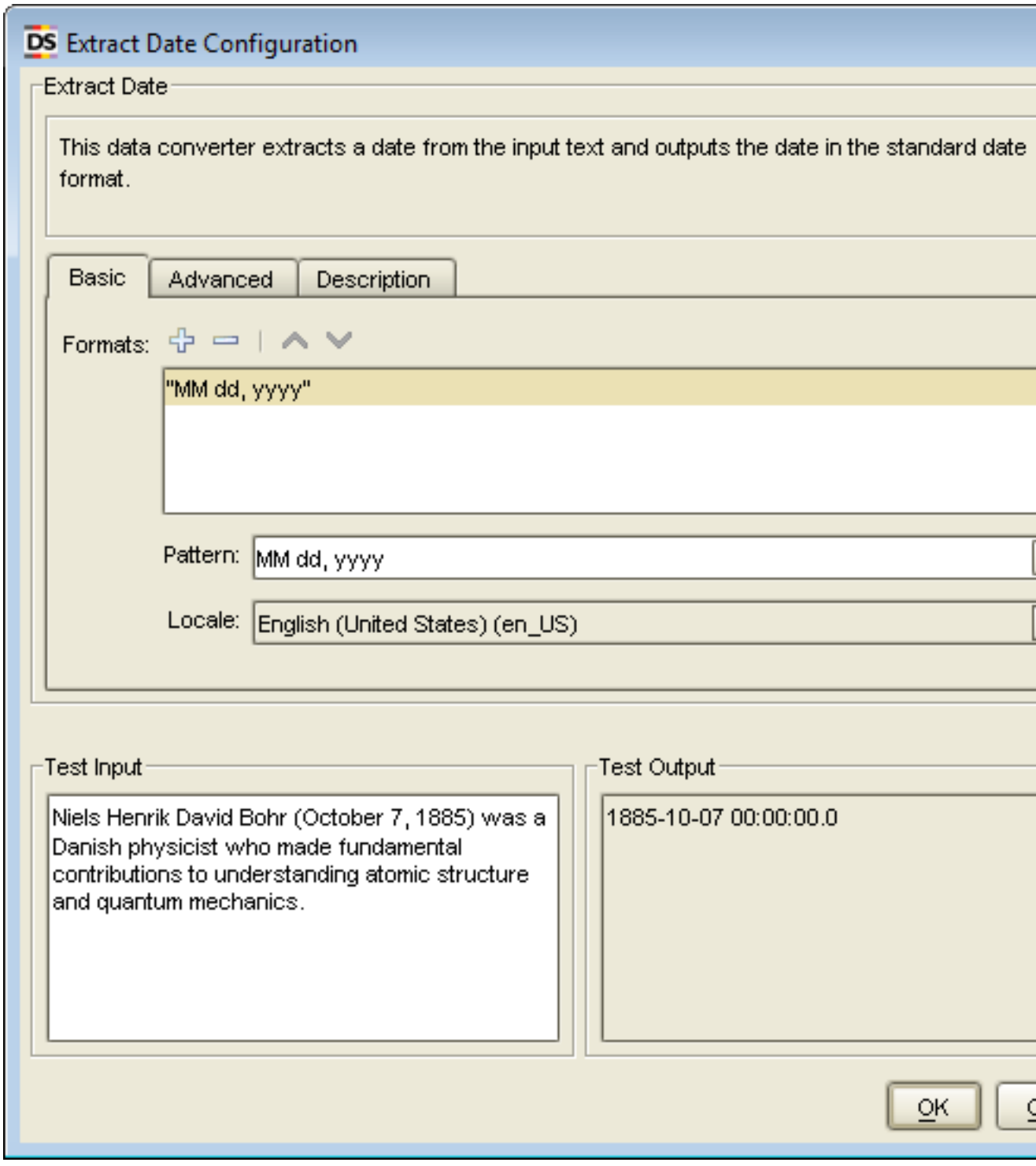
Conversion is used whenever you want to normalize content, such as when one text should be replaced by another text. For example, you might want to normalize country codes to their natural language description, e.g. "US" should be normalized to "United States". For plain text conversions, you should use the Convert Using List data converter. For conversions based on patterns or expressions, you should use the If Then data converter.

Number Extraction and Formatting

Whenever you want to extract a number from some content, you should use the Extract Number data converter. For further number formatting, you should use the Format Number data converter. Often, when you want to extract a number from some content, you add an Extract Number data converter; if you need any further formatting you add a Format Number data converter in order to reformat the text extracted by (and outputted by) the Extract Number data converter.

Extracting the Date from a Text

Extracting dates should be done in the same fashion as extracting numbers. Use the Extract Date data converter to extract the date from any text. Extract Date uses patterns to extract the date. The pattern doesn't necessarily have to match the entire text, only the date. The extracted date is then converted to standard date format, which can be formatted in any way using a Format Date data converter.



Using the Extract Date Data Converter

Below are two videos explaining how to extract dates from text.
Videos are not included in the PDF. Please visit open this url http://player.vimeo.com/video/40930103 in a browser to see the video.
Video Tutorial on Simple Date Extraction

Videos are not included in the PDF. Please visit open this url http://player.vimeo.com/video/27240628 in a browser to see the video.
Video Tutorial on Complex Date Extraction

Extracting Only a Subset of the Tags in the Found Tag

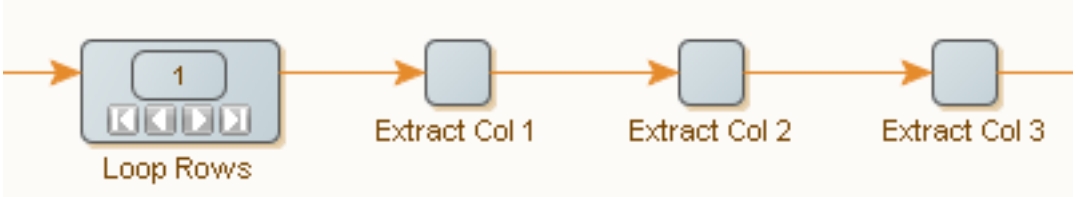
Sometimes, you want to extract from a range of tags rather than a single tag. The Extract action lets you specify a range of tags by specifying the first tag and the last tag in the range.

For example, consider the case of extracting the body text of an article, where the body text is made up of individual sections, each in their own tag, and where information about the article title and author is contained in some other tags. To extract only the body text without the article title and author, use the Extract action to extract the text, and configure the action so that only the range of tags spanning the body is extracted.

How to Extract Content From an HTML Table

Interesting content is often located in tables. Unfortunately, HTML tables are far too often irregular in both content and structure. Fortunately, Design Studio has been designed to deal with such irregularities as described below. (Note that the techniques described in this section are not really restricted to dealing with table content and structure irregularities. They can be used when dealing with all kinds of tag irregularities.)

If the table containing the interesting content is perfectly regular in both content and structure, then you can extract the content as described in How to Extract Content from HTML. The robot will typically look like this:



The first step contains a For Each Tag action that loops through the <tr>-tags in the <tbody>-tag of a <table>-tag. It is followed by several steps that each extract content from a cell (column-wise) in a table row.

Content Irregularities

Sometimes the content of cells in the same table column differs in format. For example, it might sometimes be empty, sometimes contain "Bob" (first name) and sometimes "Bob Smith" (first name and last name).

One way, and probably the simplest way, to deal with content irregularities is to use the If Then data converter in the step doing the extraction of some value. You configure its "If" and "Else If" properties so that they match each format variation. The corresponding "Then" properties then extract the matching subpattern.

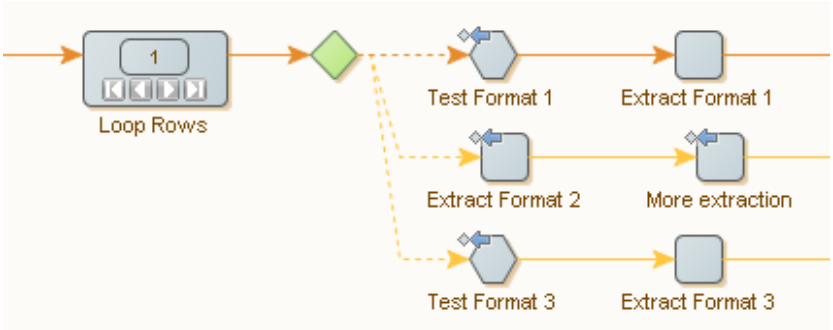
However, for the "Bob Smith" case, which contains two values (first name and last name), you need to create two steps: one that extracts the first name and one that extracts the last name. This is because the Extract action only allows you to extract one value. Each of the two steps would then contain an Extract action with an If Then data converter so that the first step extracts the first name (if any), and the second step extracts the last name (if any).

Structure Irregularities

Sometimes the rows of a table vary in the number of cells they contain. A common way of dealing with such irregularities is to test the format of each table row. For example, you might want to consider only rows containing a certain number of cells, or only rows containing a specific text.

To do this, you add a Try step after the For Each Tag step that loops through the table rows. Each of the Try step's branches handle one format. This can be done by starting each branch with a conditional step with "Try Next Alternative" error handling, for example a Test Tag action that accepts all rows matching some format (written as a pattern). The conditional step is then followed by one or more extraction steps that assume the format accepted by the conditional action. It is sometimes possible to combine the conditional step and the extraction, that is, to just try to do the extraction of a format and if it fails, try the next one.

The following robot uses both approaches. Note that the extraction of the second format is a two-step process. Because "Try Next Alternative" error handling is set up on both steps, the third branch is tried if either of the two steps fails. This represents a fairly complicated condition on the second branch:

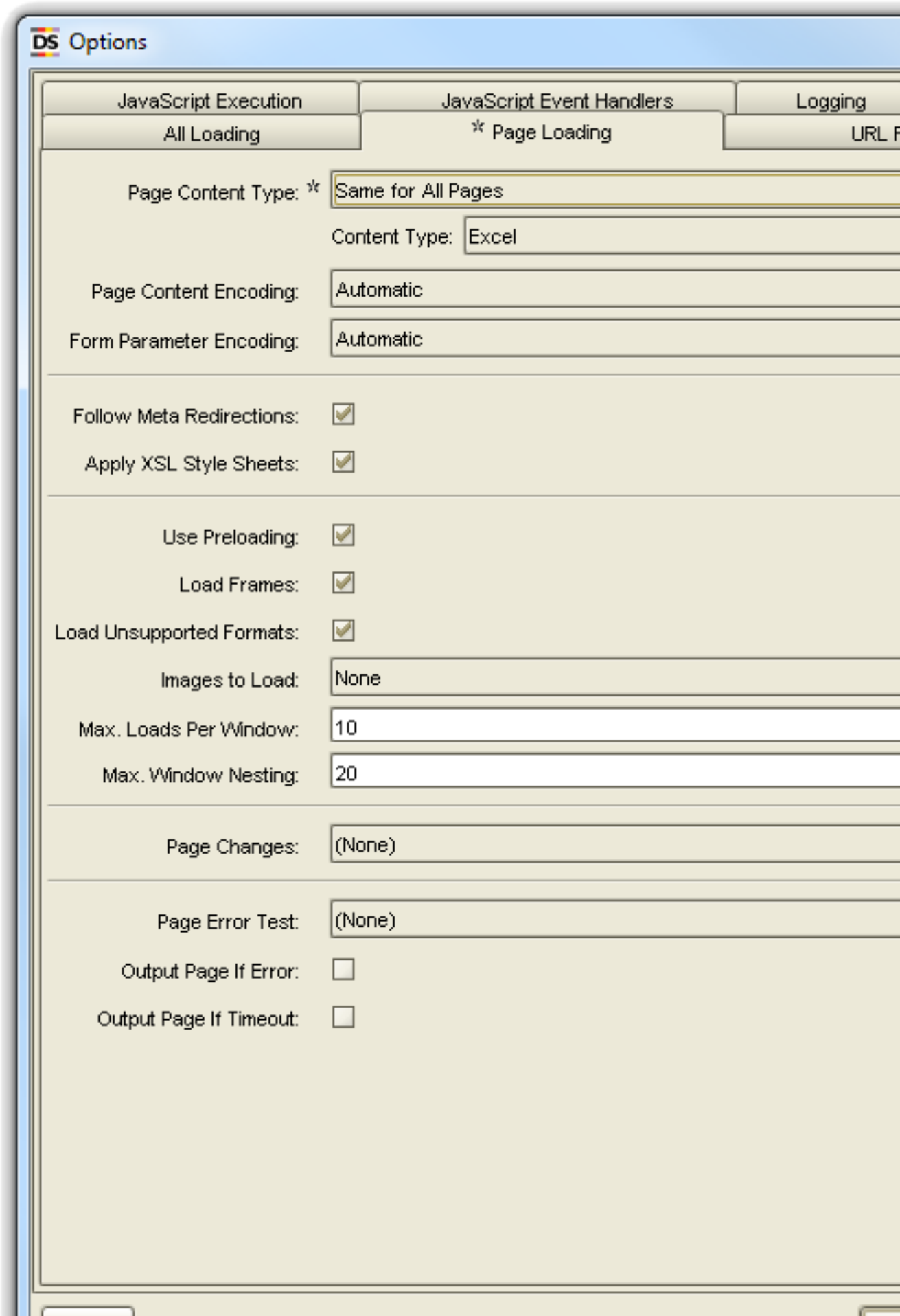


When this robot is run, each branch is executed in turn until one succeeds. This implies that the conditions in later branches do not have to repeat the conditions from earlier branches; it is known that they failed.

Note that the branches beyond the conditional steps need not be kept separate. If two or more branches share extraction steps, you may want to merge the branches after the steps that are different.

How to Load an Excel Page from a Variable

Even though the most common way to load a page in Design Studio is to use a Load Page step, a robot may also sometimes receive an Excel documents as input in a Binary attribute and you may want to have this document loaded into the robot so that your robot can loop and extract data from from this. This is possible, but it involves a bit of configuration and we will describe how to do this below. You start by inserting a Create Page step into your robot and configure this to get its Content from the Binary attribute, but to make this work you also have to tell the step that the data in the attribute is actually an Excel document and not just any kind of binary data. You do this by opening the *Option* configuration on the *Create Page* step. On the *Page Loading* tab of the *Options* dialog you set the *Page Content Type* to *Same for All Pages* and choose *Content Type*: *Excel* as show below.


Options
JavaScript Execution
JavaScript Event Handlers
Logging
All Loading
* Page Loading
URL
Page Content Type: *
Same for All Pages
Content Type:
Excel
Page Content Encoding:
Automatic
Form Parameter Encoding:
Automatic
Follow Meta Redirections:
Apply XSL Style Sheets:
Use Preloading:
Load Frames:
Load Unsupported Formats:
Images to Load:
None
Max. Loads Per Window:
10
Max. Window Nesting:
20
Page Changes:
(None)
Page Error Test:
(None)
Output Page If Error:
Output Page If Timeout:

Setting the Content Type to Excel

How to Extract Content from Excel

Design Studio has three steps for extracting content from a spreadsheet:

- The *Extract Cell* step is used to extract text content from the found range.
- The *Extract Sheet name* step is used to extract the sheet name of the sheet of the found range.
- The *Extract As HTML* step is used to extract the found range of a spreadsheet as an HTML page containing a table with the cells of the range into a variable.

For the Extract Cell and Extract As HTML steps you can specify what to extract from the cells. This is controlled by the value of the *Extract This* option. The choice here is the same as the View Modes for the Spreadsheet View. The possible options are:

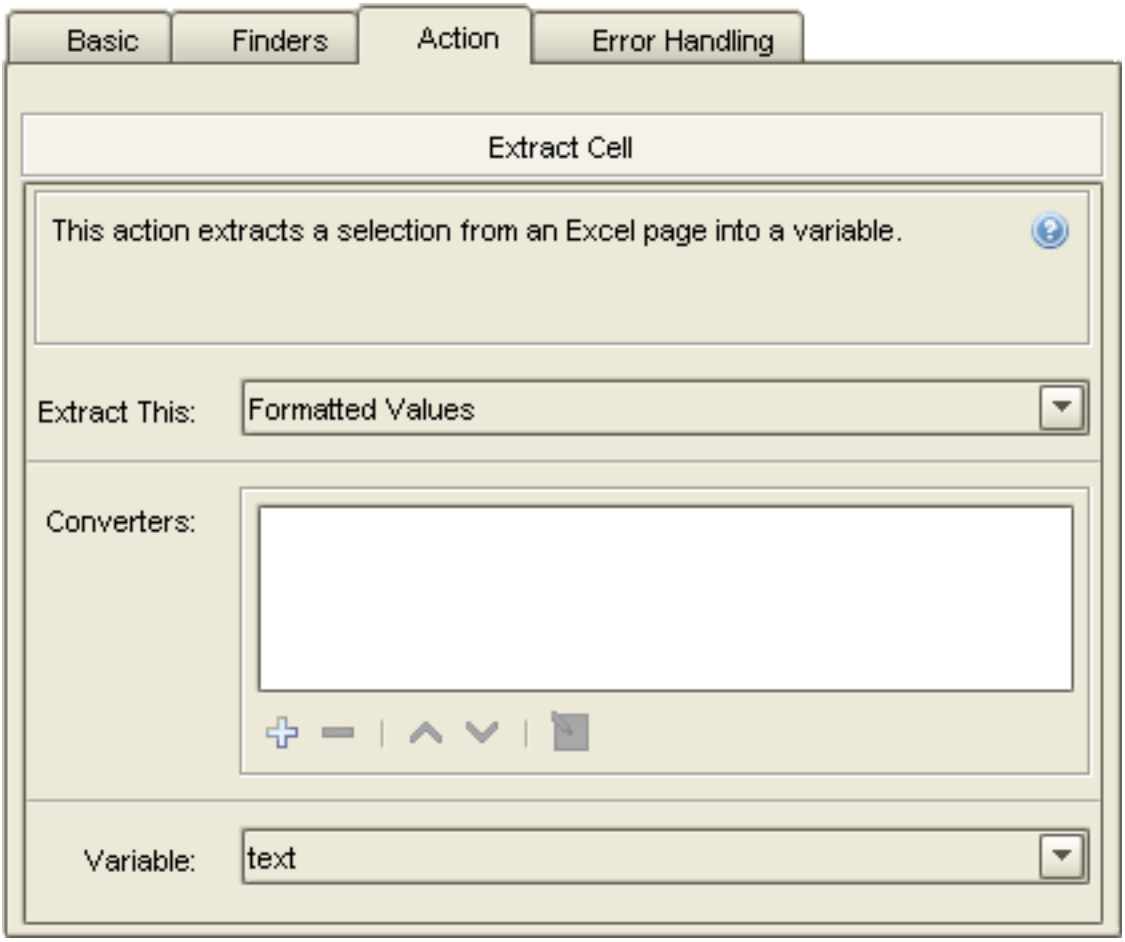
Formatted Values:	The extracted values are what you see in Excel, e.g. the values of dates and numbers are extracted formatted, which means that numbers may be with less decimals than the actual values of the cells.
Plain Values:	The extracted values are the actual values that Excel would show if the values of the cells were not formatted, e.g. numbers would not have rounding of decimals.
Formulas:	If a cell contains a formula then this is extracted or otherwise the same value as for the Plain Values option is extracted.

If you create the steps by right clicking in the Spreadsheet View then the value of the *Extract This* is set to the value of the selected *View Mode*. That is, if you have set the *View Mode* to *Formulas* and then right click in the page view and select **Extract > Extract Text** from the popup menu (into a text variable) then the *Extract This* option of the *Extract Cell* action step is set to *Formulas*.

Often you need to reformat (or normalize) the extracted content, and the Extract Cell action allow you to do this by configuring a list of data converters.

Extracting Values from Cells

The Extract Cell step is used to extract the content of a cell or a range of cells into a variable.



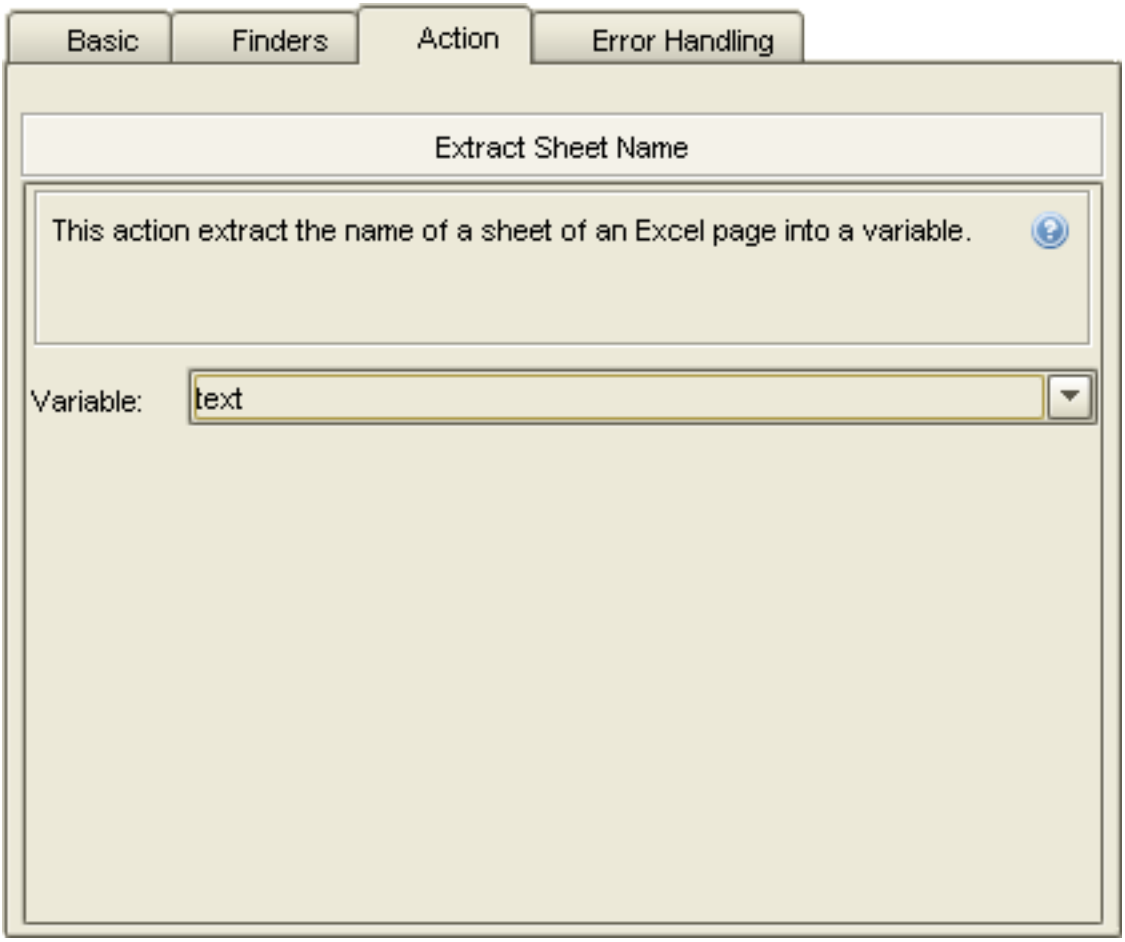
The Action Tab of the Extract Cell Step

If the found range is a single cell then the value of this cell is extracted. If the found range contains more than one cell then the values of the cells are extracted as a text in which the cells are tab separated and rows are separated by newlines. In both cases the extracted value stored in the variable is created by applying the converters to the extracted value.

The value extracted from a cell is essentially the content of the cell in Excel taking the value of the *Extract This* option into account. For a blank cell the value is the empty string and if a cell is part of a merged cell e.g. C4:D6 (created in Excel by merging cells) the extracted value is blank unless the cell is the top left cell C4 of the merged cells.

Extracting a Sheet Name

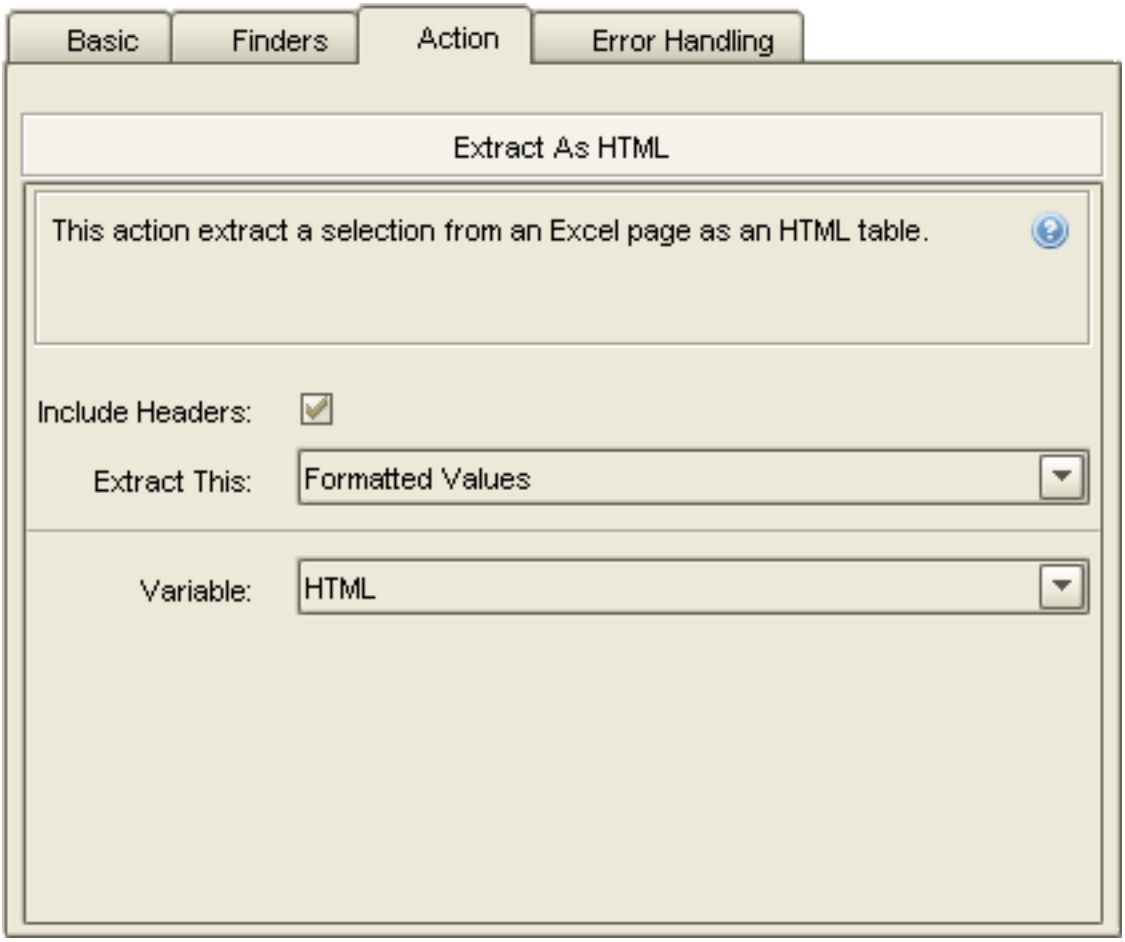
The Extract Sheet Name step is used to extract the name of a sheet. This can be useful when combined with a Test Value step to skip a sheet with a given name while looping over all sheets or if you want to extract the sheet name to an attribute of a variable of complex type so that this becomes part of a returned value.



The Action Tab of the Extract Sheet Name Step

Extracting As HTML

The Extract As HTML step is used to extract part of a spreadsheet document as HTML source code that is then stored in a structured text variable, e.g. of type HTML. The extracted code contains the extracted range (in a header tag), e.g. Sheet1:A1:H17 which means that the name of the sheet is contained in the code. The cells of the found range are placed in a table in the generated code. This step is mainly for obtaining an HTML version of a part of a spreadsheet e.g. so that this may be returned for the robot and presented in a browser, but it is also possible to use the step in a robot create an HTML page with the extracted code using a Create Page step. It is not recommended that the Extract As HTML step is used to convert a spreadsheet into an HTML page to access its content in that way, because this might result in poor performance of the robot.



The Action Tab of the Extract As HTML Step

The *Include Headers* option specifies whether the column and row headers should be included in the extracted table or not. The image below show the extracted HTML when the *Include Headers* option has been set to true. The found range was 'My Sheet'!B1:E10 which resulted in the cells from B1 to E10 being extracted to the table, but also that the table contains column headers B to E and row headers 1 to 10.

'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

Example of extracted HTML

How to Test Cell Types in Excel

If you want to test the content of a cell in an Excel page this can be done by first extracting the content of the cell and then use a Test Values step to perform the actual test. This is essentially not different from what you would do in other page types, e.g. HTML, but what if you wanted to determine the cell type of a cell that would not be straightforward or even possible by just extracting the content of a cell and subsequently performing a test on this, e.g. there is no way to determine whether a cell is blank or contains an empty text. Fortunately Design Studio contains a step to perform such a test: the **Test Cell Type** step.

There are six different cell types that you can test for and these correspond directly to what you in Excel can test for with functions *ISTEXT*, *ISNUMBER*, etc.:

Blank: This corresponds to the Excel function *ISBLANK* .

Text: This corresponds to the Excel function *ISTEXT* .

Number: This corresponds to the Excel function *ISNUMBER* . This type also includes dates since these are represented as numbers in Excel.

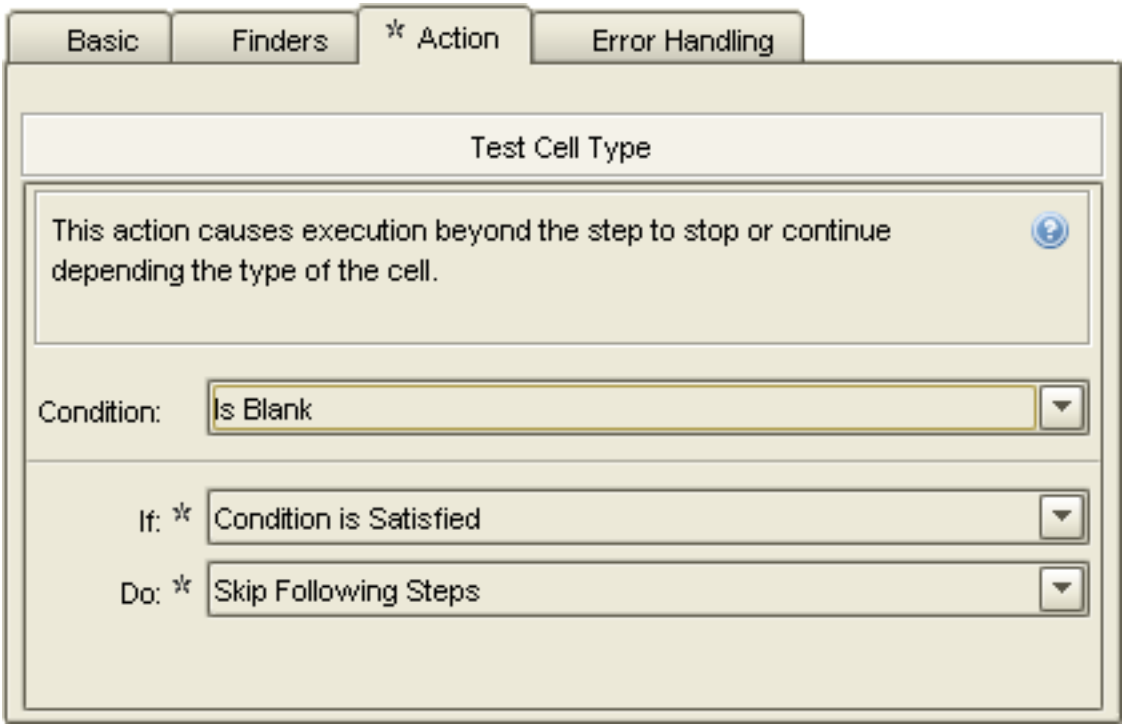
Logical: This corresponds to the Excel function *ISLOGICAL* . This corresponds to the type Boolean in Design Studio .

Error: This corresponds to the Excel function *ISERROR* .

Formula: This corresponds to the Excel function *ISFORMULA* .

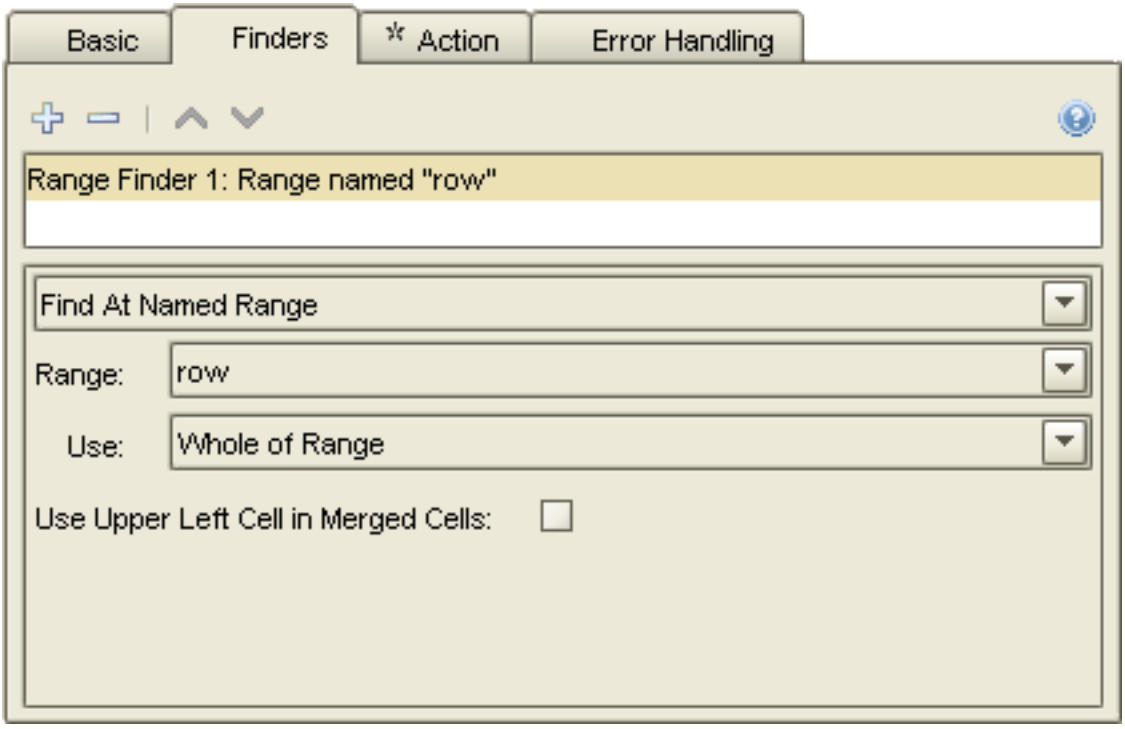
The **Test Cell Type** works like any other test step. It tests that the cell type in the found range matches a specified type and based on the result of this determines whether to continue along the branch or skip the following steps. The step is described in further details in the reference documentation on the Test Cell Type [../ref/robomaker/reference/stepaction/TestCellTypeStepAction.html] step.

An important property of the Test Cell Type step is that it can test the type of many steps simultaneously. As an example of this let us look at how one may test that an entire row is empty. This could be useful when looping over a document containing several structurally identical tables separated by blank lines. The figure below shows how to configure the step for this. In this example the branch following the step will be skipped, if the cells in the found range are all blank.



Example of Testing for Blank Cells

The figure below shows how to configure the Range Finder such that it finds an entire row. In this case we have a named range called "row" that has been set by a Loop in Excel step looping over rows and occurring before the Test Cell Type step. We have specified that the result should the entire row, by selecting "Whole of Range" for the *Use* property.



Range Finder Selecting an Entire Row

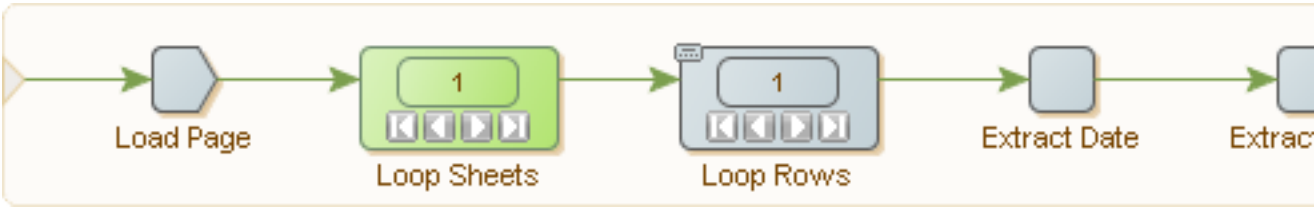
How to Loop in Excel

Looping in Excel is in many ways similar to looping in HTML, but where the possible ways of looping in HTML are very diverse, it is much simpler in Excel due to the simpler structure of Excel. Essentially you may loop over all the sheets in a document or you may loop over the cells of a sheet either by looping over the rows, columns or cells of a found range. To loop in Excel you use the **Loop in Excel** step. This step has many options in common with steps that loop in HTML, e.g. *First index*, *Increment*, etc. and these are described in detail in the reference documentation [../ref/robomaker/reference/stepaction/LoopInExcelStepAction.html].

Below is a video explaining how to loop over a table in a single sheet, extract the cells of each row into a variable of complex type and return these with a Return Value step.
Videos are not included in the PDF. Please visit open this url http://player.vimeo.com/video/61006912 in a browser to see the video.
Video Tutorial on Looping in Excel

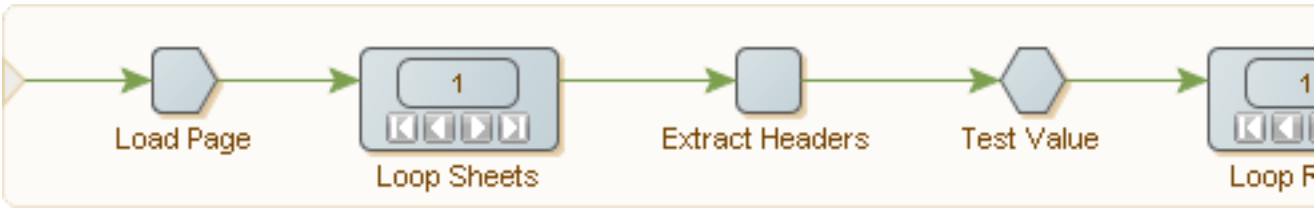
Looping over Sheets and Rows

Generally looping in Excel is a bit more complex than show in the video above. An often occurring scenario for looping in an Excel document is when the document contains a number of sheets containing tables with data of the same type, e.g. one sheet with account information for each month of the year. In this case you would like to have you robot first loops over the sheets and then over say the rows of each sheet. We would probably also like to be able the handle the situation where the document contains a sheet that does not contain data of the same type as the other sheets, e.g. maybe where a sheet is blank. The image below shows the structure of such a robot.



Looping over Sheets and Rows in Excel

The first step in this robot is a Load Page step that loads the Excel document from some URL, the robot then contains a Loop in Excel step that loops over all the sheets of the document. For each iteration of this first loop step the robot executes another Loop in Excel step that loops over each row of the sheet much in the same way as was demonstrated in the video above. The step looping over rows has its Error Handling property *Then* set to "Next Iteration", which means that if the range finder of the step fails to match a range the size of the table it will go the the next iteration. This simplified error handling will handle the simple situation where a sheet is blank, but not situations where a sheet contains a table with entirely different type of data. In the general case one would have to insert a step to extract part of the sheet followed by a step to test the structure of this. One example of this could be extracting the column headers and testing that these have some given structure. The image below shows the robot extended with this. In this the Extract Cell step named Extract Headers extracts the first row of the sheet into a variable and the Test Value step has a condition that tests the value of this. If the value matches then the robot will execute the next step (the Loop Rows step), if not the robot will skip the following steps, that is, the *Do* property of the Test Value step has been set to "Skip Following Steps".



Looping over Sheets and Rows in Excel with Test

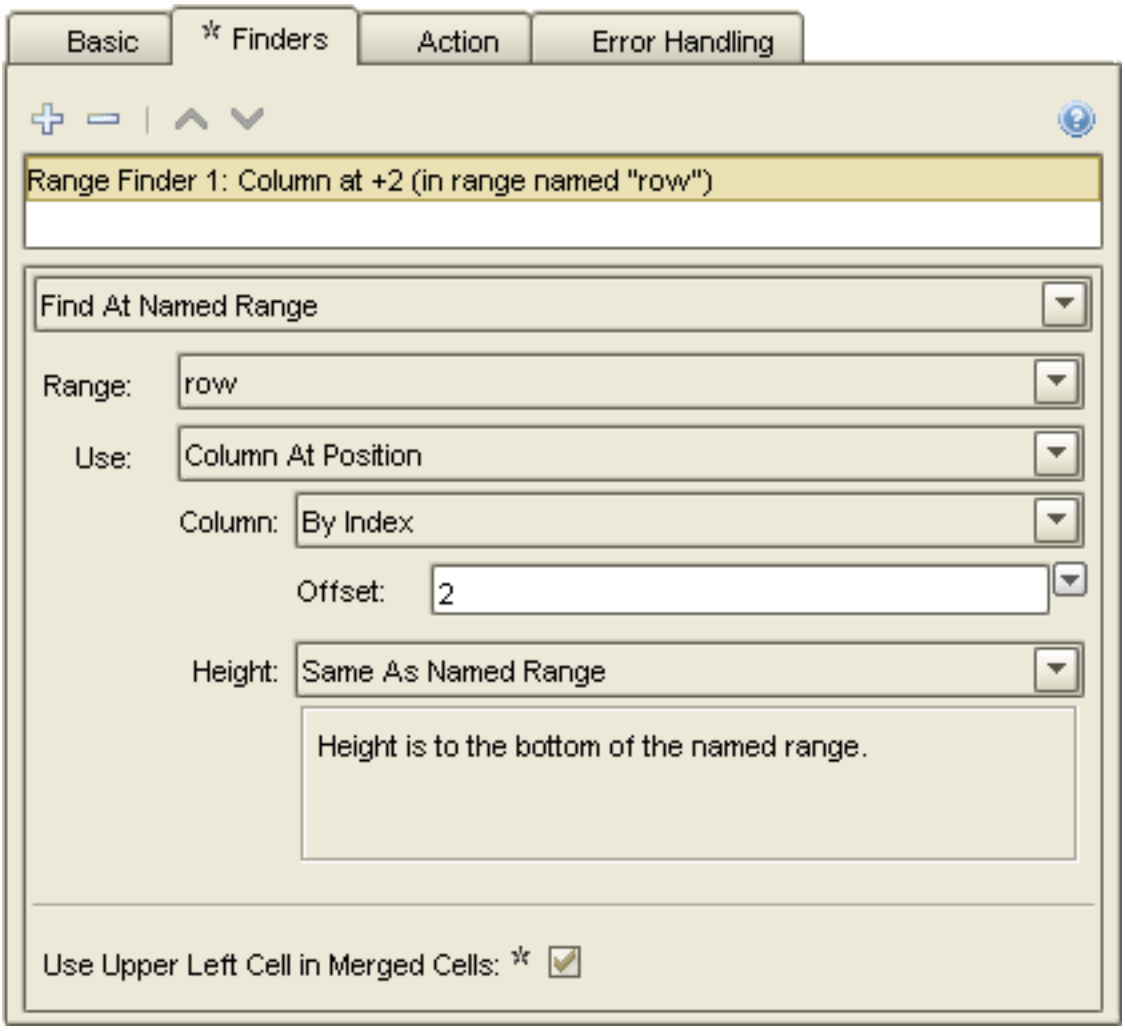
Looping and Merged Cells

The next scenario that we will look at is when an Excel Page that you want to loop over has merged cells. A *merged cell* in Excel is two or more adjacent cells that have been merged into one cell and shown as one. The content of a merged cell is stored in the upper left cell of the cells and all other cells are blank. In some case this can give problem when looping over a table that contains such merged cells. Let us look at a simple example that illustrates this. If you look at the sheet below that shows test results for students you will notice that some student have missed their test and and in some case two test shown using a merged cell.

	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					

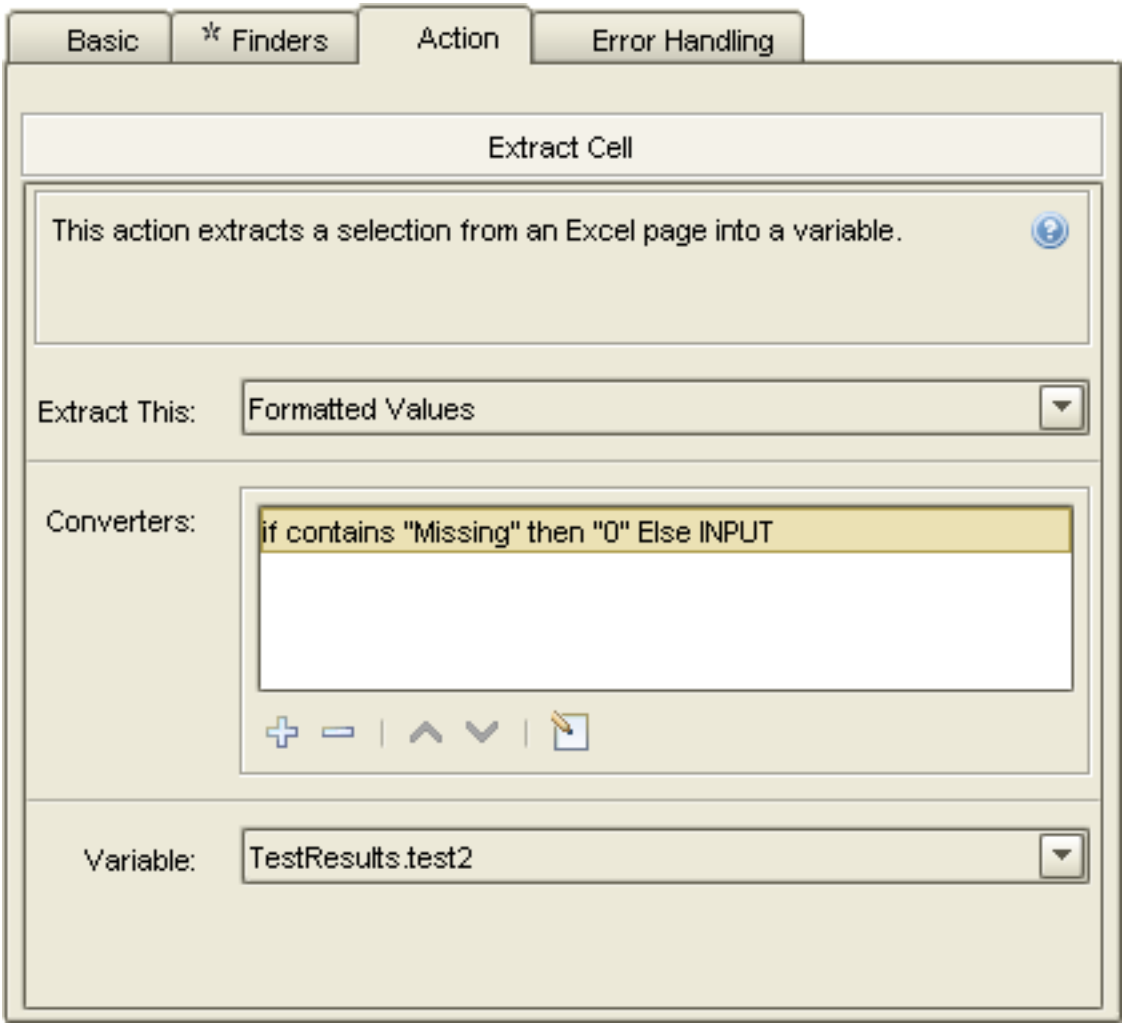
Looping over Sheet with Merged Cells

If we want to extract the students' test results by looping over the rows of the sheet, we might fail to extract the results correctly when a student has missed a test since the text "Missed" is not a number. We could solve this by inserting a test for this and then store the value 0 for a failed result. This would work fine for the cell B4 since this contains the value "Missed", but it would fail to work for C4 since the content of this would be a blank value. Instead of having yet another test for whether the content of the cell is blank, there is a simple option on all Range Finders that ensures that if it finds a single cell inside a merged cell, then it will instead return the upper left cell of the merged cell. The image below show how to configure a range finder in this way.



The Range Finder

Now all the Extract cell has to do is to test for the text "Missed" and use 0 for the result and otherwise use the extracted value. This may be done with an *If Then* data converter as shown below.



The Extract Cells Action

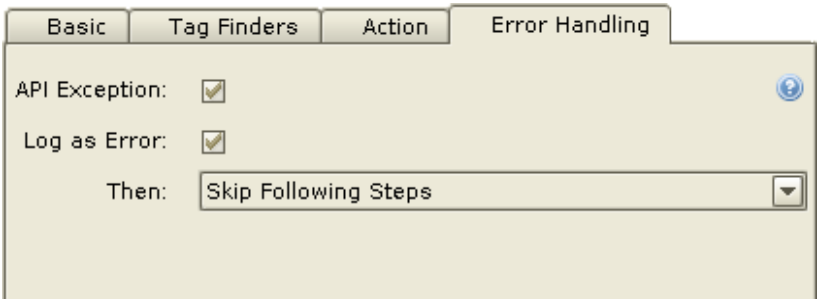
How to Handle Errors

A step in a robot may generate an error when it is executed. For example, this will happen if the tag finders cannot find the tag to work on, or if the step action generates an error. Test steps may also be configured to act as if an error occurred if the test fails. The default behavior of a robot is to report and log the error immediately, and to omit execution of the steps beyond the one that failed. However, by configuring the error handling properties of the steps in the robot, you can change this behavior. For example, you can make the robot skip a step that generates an error, or you can make it try alternative branches. In this section, you will learn how to do this.

Before we start, note that the error handling behavior that we describe here applies to runtime execution of a robot (i.e. execution in RoboServer or in debug mode), not to the execution in design mode in Design Studio. In design mode, an error is normally reported immediately, and the execution of the subsequent steps is aborted. One exception to this is when the step is configured to "Ignore and Continue" in case of errors, in which case Design Studio *does* ignore the error and executes the next step, just as it would during runtime execution.

API Exceptions and Logging

The "Error Handling" tab in the Step View has two checkboxes for selecting how (and whether) to report the error.



The "API Exception" checkbox specifies whether to report the error back to the caller of the robot. This is most useful when the robot is executed by a client via one of the APIs and runs in RoboServer. In this case the error is sent back to the caller via the API as a `RobotErrorResponse`, which causes an exception at the caller side, at least when using the default `RQLHandler`. See the reference documentation [../ref/robomaker/reference/errorhandling/ErrorHandling.html] for the details when the robot is executed in other ways.

The "Log as Error" checkbox specifies whether to log the error. Logging happens in different ways depending on whether the robot is run in the Design Studio or in RoboServer, as described in the reference documentation [../ref/robomaker/reference/errorhandling/ErrorHandling.html].

Note that the checkboxes may be unchecked or checked, but may also be marked with an asterisk * meaning that they have been explicitly set to a non-default value. This is explained in Show Changes from Default, where it is also shown how to remove the asterisk and revert to the default value. When the default value applies (that is, when no asterisk is present), be aware that the default varies according to how the error is handled.

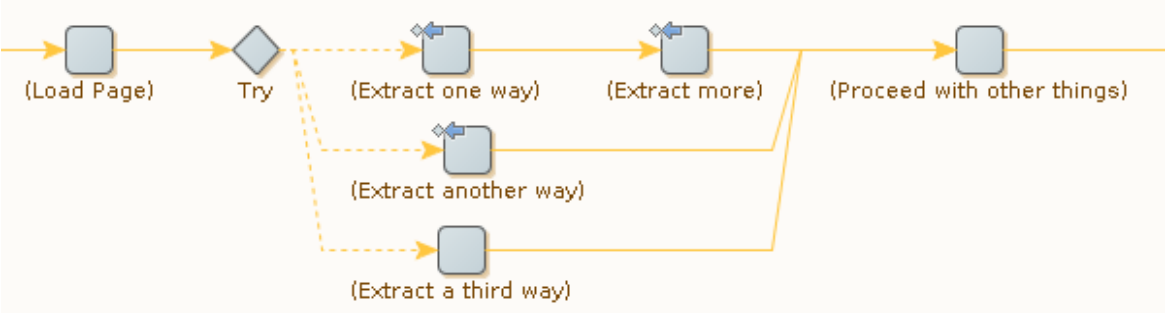
How Execution Continues After an Error

The "Then" property defines how and where robot execution continues after an error has been encountered. The possible options are described with examples in the following sections. Detailed descriptions are found in the reference documentation [../ref/robomaker/reference/errorhandling/ErrorHandling.html].

Trying Several Alternatives

In this example we will show how error handling can be used to select among several different alternative ways to achieve some goal. This relates to the discussion in Conditions and Error Handling, which also includes a tutorial video on the topic.

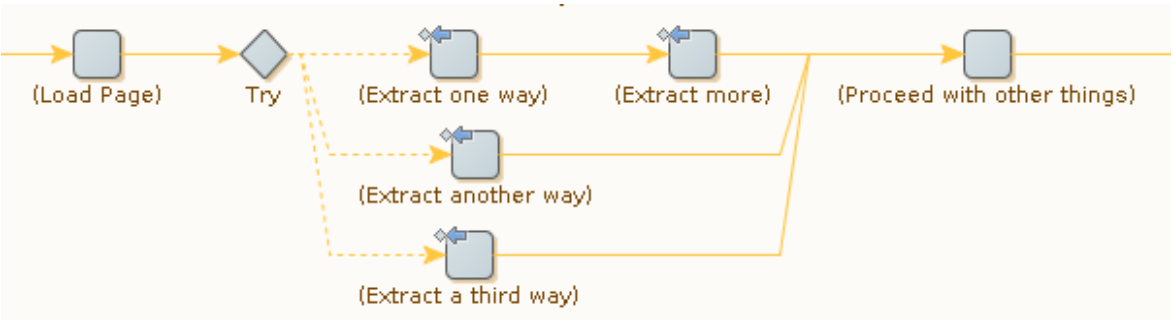
Suppose that some part of a web page has varying structure and layout, but always comes down to one of three cases. In each case there is some information we want to extract. This can be done by simply trying to do the extraction for one case at a time. If it fails, the next case is tried, until the third one, which we assume will succeed:



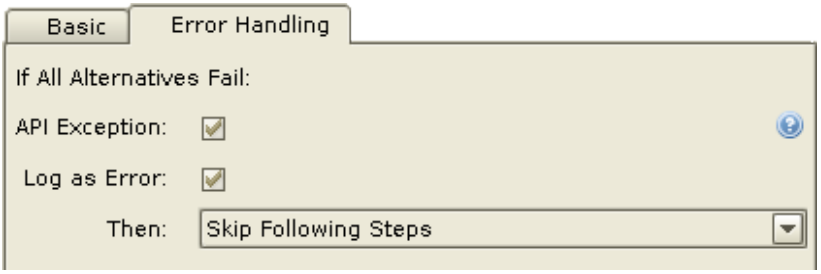
Note the ("Try Next Alternative") icons on the Extract steps, which mean that if extraction fails, the next branch from the Try step will be executed (if a branch coming out of a Try step succeeds, the next branch will not be executed). Thus the Extract steps do two things at the same time: They do some extraction from the web page, and if that cannot be done, they ensure that the next approach is tried. Please note that if any of the two steps on the first branch fails, the second branch is executed; this is an example of how the "success condition" for a branch may be expressed by a combination of steps.

This approach works best if the "third way" of extracting is bullet-proof (for example, by applying a fixed set of default values rather than actually extracting data from the web page). If the third branch accesses the web page like the first two branches do, it may not be wise to just assume that it will succeed. The next time the robot is run, the web site may have changed so much that none of the three strategies will succeed, and the robot should be able to respond to this in a reasonable manner.

This easiest way to respond is just report the problem back to the caller and log it, giving up on doing the extraction and whatever would follow it. This can be achieved simply by making the third branch inform the Try step if it fails to do its work, just like the first two branches:



Now if all three branches fail, the Try step will report and log the incident. This comes from the default configuration of error handling *on the Try step*:



(For a Try step, "Skip Following Steps" means that no additional action will be taken beyond reporting and logging.)

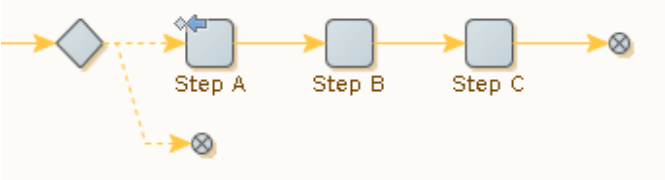
Alternatively, it is possible to make the Try step propagate the problem back to an earlier Try step for handling. We will explore this idea in the section on Try-Catch.

Shortcuts for Common Cases

Try steps and "Try Next Alternative" error handling (as described in the previous section) are very flexible tools. Used in the proper way, they support many different ways to handle errors, and in this section, we will show a couple of simple and common cases. In fact, these cases are so common that they are also supported by specialized error handling options, which we will also show.

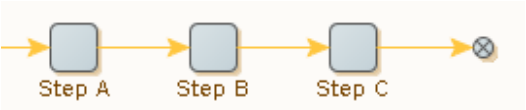
Skip Following Steps

In many cases, a robot must be able to handle optional elements on a web page. That is, if the elements are present, they must be handled (for example, data must be extracted), but if they are absent, the handling of these elements can simply be skipped. Their absence is not an error, but an expected situation. This can be expressed as follows in a robot. Step A tests if the elements are present (for example by trying to extract something), while steps B and C do further processing that depends on the success of step A:

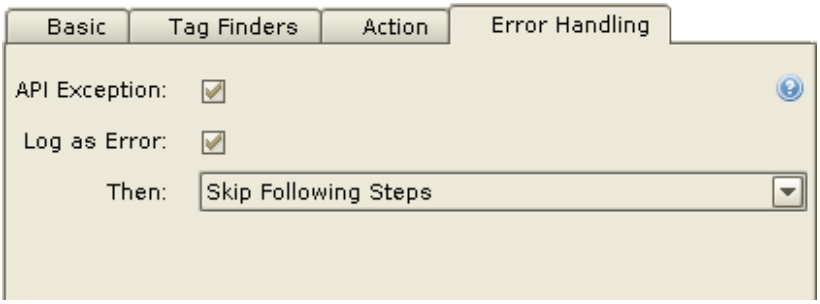


If step A is not successful (because elements are missing on the web page), its "Try Next Alternative" () error handling will send notice to the Try step (which is unnamed in this example). This will cause the second and empty branch to be executed, after which execution of that whole branch that starts in the Try step is done. Thus steps B and C are not executed if step A is not successful.

This situation happens so often that a specific error handling option "Skip Following Steps" has been introduced as a shortcut. It makes it possible to simplify the example as follows:



The error handling for step A is configured as follows. This is the default configuration for all new steps:

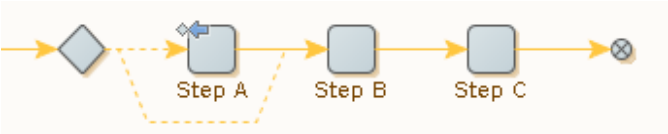


Strictly speaking, you need to uncheck the "API Exception" and "Log as Error" checkboxes to get exactly the same behavior as shown above with the Try step. This is because the default values for these checkboxes are different for these two ways to do error handling.

Note that if step B had been similar to step A (that is, if step B had also had "Try Next Alternative" error handling), this same shortcut could still be used.

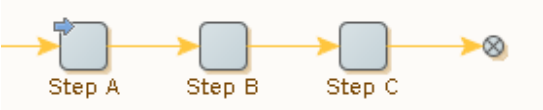
Ignore and Continue

Sometimes, some action (extraction, whatever) needs to be done if some condition is met, and otherwise it can simply be skipped. Subsequent steps do not depend on the result (or a proper default for the result has been set up in advance). This can be expressed as follows:

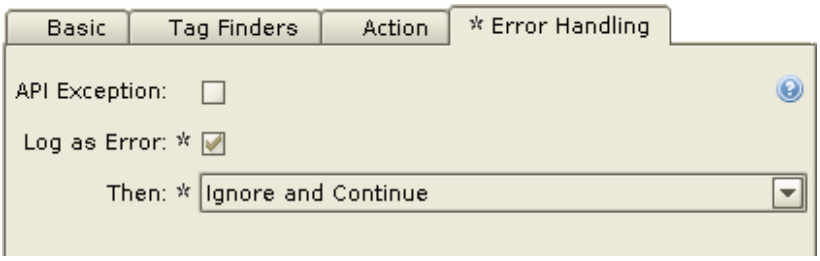


If step A is not successful, its "Try Next Alternative" () error handling will cause the second and empty branch from the (unnamed) Try step to be executed. After this, execution continues at step B with the same robot state that was input to step A. Thus step A is effectively skipped.

This can also be done without the Try step by using the error handling option "Ignore and Continue" () on step A:



One interesting possibility is to have the situation logged even though it is otherwise ignored. This can be achieved by configuring error handling on step A as follows:



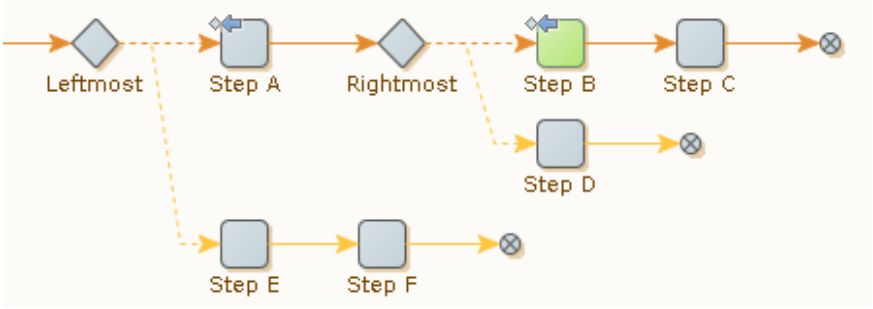
The same can be done if you prefer to use the method with a Try step.

"At" Target

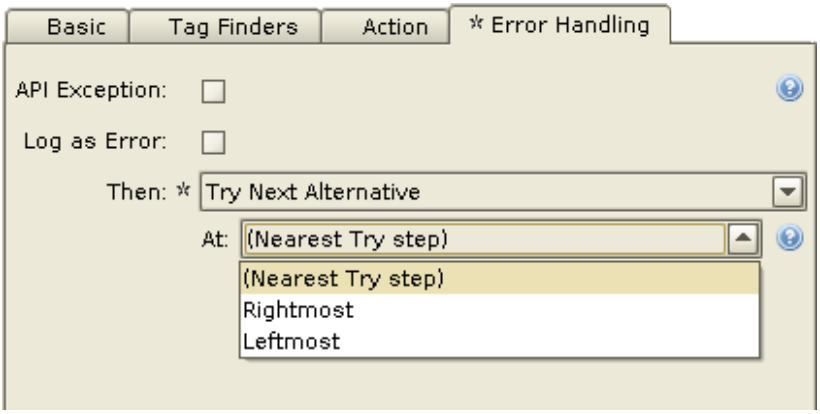
By its very nature, "Try Next Alternative" error handling refers to a Try step. This Try step must be a prior step on the current execution path, that is, it must be one of the steps whose execution led up to the current step. Otherwise the "next alternative" part of "Try Next Alternative" does not make much sense.

This opens an interesting possibility: What if there are several Try steps on the current execution path? If that is the case, the robot will be in the process of executing a "current branch" relative to each of these Try steps, and for each of them the "next branch" will be a different one. For example, if an error occurs in step B in the following robot, "Try Next Alternative" will continue execution at

- the next branch at the rightmost Try step, so that execution skips step C and continues at step D; or
- the next branch at the leftmost Try step, causing execution to skip both steps C and D and continues at step E?



The answer is that you can choose which one, as seen in the error handling configuration for step B:



The default is the "nearest Try step" but other Try steps on the execution path can be selected explicitly. The reference to the Try step is by name; if several Try steps have the same name, the nearest (rightmost) one with that name is implied. This can be used to advantage as described in Try-Catch.

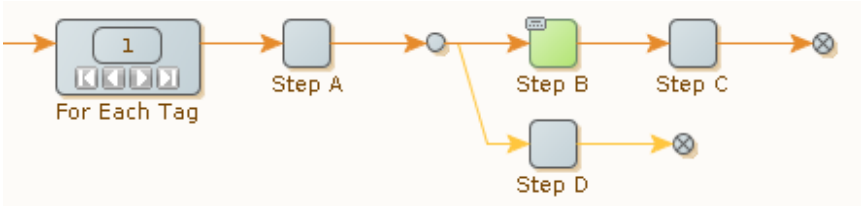
Although we here describe "At" targets only in relation to "Try Next Alternative" error handling, such references may be used also with the "Next Iteration" and "Break Loop" error handling options that are described in the following section. In those cases, the references will be to loop steps rather than Try steps.

Looping

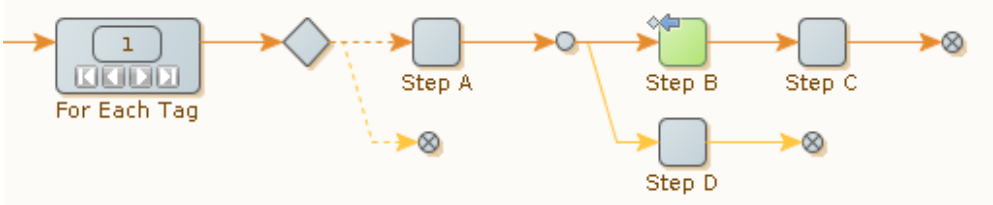
Sometimes when an error occurs or a test fails, the proper reaction is to abandon execution of the current iteration of a loop, or the whole loop. This is supported by way of two specialized error handling options.

Next Iteration

In the following robot, step B has error handling "Next Iteration". This means that if an error occurs during execution of this step, execution of the current iteration of the loop is stopped. Thus neither step C nor step D will be executed; instead execution continues at step A with a robot state that reflects the next tag among those that the loop step iterates over.



This error handling option is a kind of shortcut, as the same effect can be achieved with the aid of "Try Next Alternative" and a Try step in the following way:



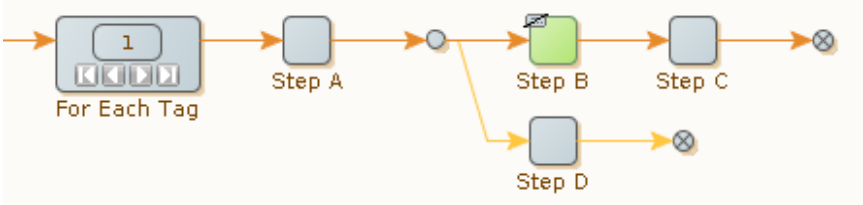
Note that doing this transformation in general requires use of "At" Targets because other Try steps may interfere.

If the robot contains several loops steps after each other, it is possible to select the one in which to go to the next iteration. This is described in "At" Target.

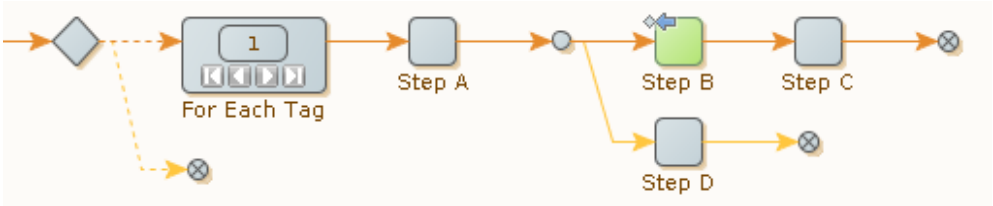
"Next Iteration" does not work with Repeat-Next loops. The word "next" has very different implications for the browser state in these two cases.

Break Loop

Instead of completing a single iteration of the loop with "Next Iteration", the whole loop may be aborted with "Break Loop" as follows:



Also this error handling option is a kind of shortcut. The following robot will have the same effect:



Note that "Break Loop" *does* work with Repeat-Next loops, as opposed to "Next Iteration".

Try-Catch

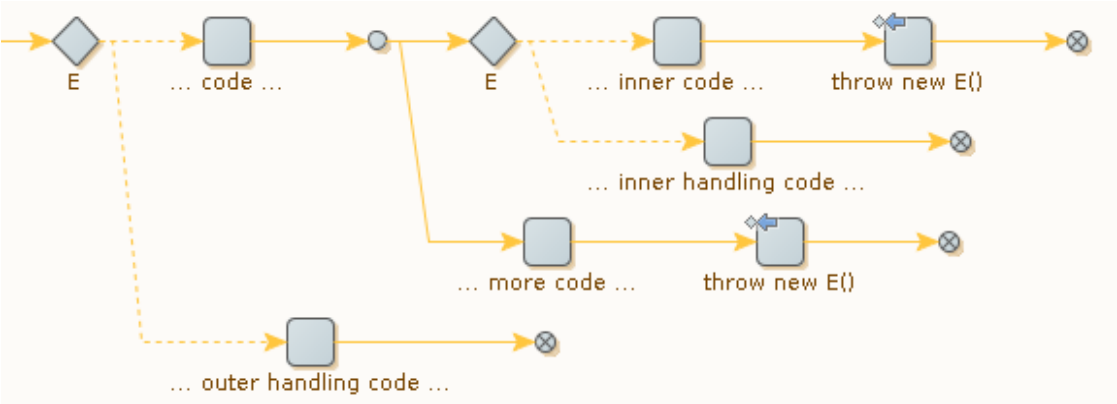
When "Try Next Alternative" error handling is used with an explicit "At" reference to a target Try step, that step is identified by its name. Most often, the fine distinction between the target step and its name is not important, but it can be exploited to provide exception handling functionality similar to the try-catch constructs in Java or C#.

In those programming languages, a section of code between "try" and "catch" has special error handling. If a specific error is signaled within this section (by "throwing" a named "exception"), the piece of code following the similarly named "catch" is executed. Try-catch constructs can be nested, and a named "exception" is always handled by the innermost enclosing "catch" with a matching name. For example:

```
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 ...
}
```

In robots, something similar can be done with Try steps. Remember that an "At" reference to a Try step with a given name always means the *nearest prior* step with that name (along the current execution path). It is permitted to use the same name for several Try steps, even on the same execution path. Thus each try-catch construct is modelled with a Try step having the same name as the "exception". The Try step has two branches, one for the code part of the "try" construct, and one for the code part of the "catch" construct:



The correspondence between the Java/C# syntax and the Design Studio terms is as follows:

Java / C# Syntax	**What to use in Design Studio**
`try { ...code... }`	The first branch of a Try step (the steps correspond to `code`)
Name of an exception	Name of a Try step
`throw new E()` within the code of a `try`	Handling an error with "Try Next Alternative at E"
`catch E { ...code... }`	The second branch of a Try step named "E" (the steps correspond to `code`)

Thus, the core idea is: When Try steps are used for error handling, name the Try steps after the error situations they handle. The advantages are:

- This helps make the purpose of each Try step clear.
- When errors are handled on a general level (with a Try step more to the left in the robot), it is still easy to do specialized handling in some cases (with the aid of a second Try step with the same name).

View Error Handling on Steps in the Robot View

Steps in a robot, for which special error handling has been set up, will as default be marked with a small symbol in the Robot View. The symbol depends on the kind of error handling that has been defined for

the step and they allow the user to visually identify the steps that have custom error handling defined. If the user do not wish to have the steps with custom error handling marked, it can be disabled in the Design Studio Settings dialog found in the Options menu (see the Robot Editor section).

How to Write a Robot with Input Variables

Robots taking input variables are probably the most advanced kinds of robots that you can write in Design Studio because the requirements of these robots are often quite strict. Generally, they should be speedy and highly reliable. (These requirements might, of course, apply to all robots; however, typically robots taking input variables are executed in real-time whereas normal extraction robots are executed in batch; also, extracting values incorrectly is typically less critical than filling out a bank transfer incorrectly.)

Let us look at the speed and reliability requirements in turn.

The **speed** requirement can be fulfilled through optimization. In Design Studio, the step actions involved with navigation, i.e. Click and similar actions, consume by far the larger part of the total execution time (80% or more). Hence, you should avoid any needless navigation whenever possible. One way to eliminate needless navigation is by linking directly to the relevant information. For example, instead of navigating to a login form from the front page you should load the login form directly. Sometimes, such direct navigation is not possible because of sessions and cookies, but often it is.

The **reliability** requirement involves writing the robot in such a way that it either carries out its task smoothly, or reports that it cannot. This is not as easy as it may sound. In fact, there are situations in which a robot cannot possibly know whether it succeeded or not. For example, if the robot submits an order and the web site does not transmit a response, then what happened? (You have probably tried something similar yourself in a normal browser.) However, a robot should be written so that it minimizes such uncertainties whenever possible. In more concrete terms, each time the robot interacts with some uncertain element (such as a web site), it should analyze the interaction thoroughly in order to detect possible errors. If an error is detected, then it should be reported, usually in the form of a returned value that somehow tells the application that invoked the robots what went wrong. If no errors are detected, then the robot can proceed to its next action. See How to Make Robots More Robust for more information.

When writing robots taking input variables, you use, more or less, the same step actions and data converters as for any other kind of robot. However, some step actions and data converters are especially useful with robots taking input variables. You should check the reference documentation on the following items:

- The Convert Variables action [../ref/robomaker/reference/stepaction/ConvertVariablesStepAction.html] for converting variable values extracted from a web site, or converting input variable values before inserting them into a form.

- The Test Variables action [../ref/robomaker/reference/stepaction/TestVariablesStepAction.html] for testing a variable value, e.g. an input variable value, according to one or more conditions, such as "price < 5000".

- The Get Variable data converter [../ref/robomaker/reference/dataconverter/GetVariableDataConverter.html] for fetching a variable value for subsequent processing or insertion into a form input field.

- The Convert Using List data converter [../ref/robomaker/reference/dataconverter/ConvertUsingListDataConverter.html] for converting content. This data converter is useful for normalizing input variable attribute values for insertion into a web site form, or normalizing content extracted from a web site.

How to Create and Reuse Snippets

A snippet can be created in three ways:

- **Creating a snippet from a selection of steps**: Select one or more steps (must be steps that can be grouped and not a single Group step) and you then select the action 'Create snippet from selection'  . This will prompt you for a name for the new snippet, create a snippet of that name containing the select steps and insert a new Snippet step instead of the selected steps.

- **Creating a snippet by turning a group step into a snippet step**: Select a Group step and then select the action 'Convert group to snippet'  . This will prompt you for a name for the new snippet, create a snippet of that name containing the steps of the group and insert a new Snippet step instead of the selected Group step.

- **Create a snippet by selecting New Snippet in the File menu**. A wizard will appear, that prompts you to give the name the snippet. After creating the snippet, an empty snippet will appear in your project, and an editor for the snippet will open. Please note, that you cannot edit the snippet contents (the steps inside the snippet) inside this editor, only edit the Description and the referenced variables list.

Variables and Snippets

Just like steps anywhere in a robot, the steps in a snippet can use variables. The steps of Snippets are always edited inside a robot. In that context the variables defined on the robot can be used in the snippet. Reusing the snippet in another robot requires you to define the variables used by the steps in the snippet on each of the robots that uses the snippet.

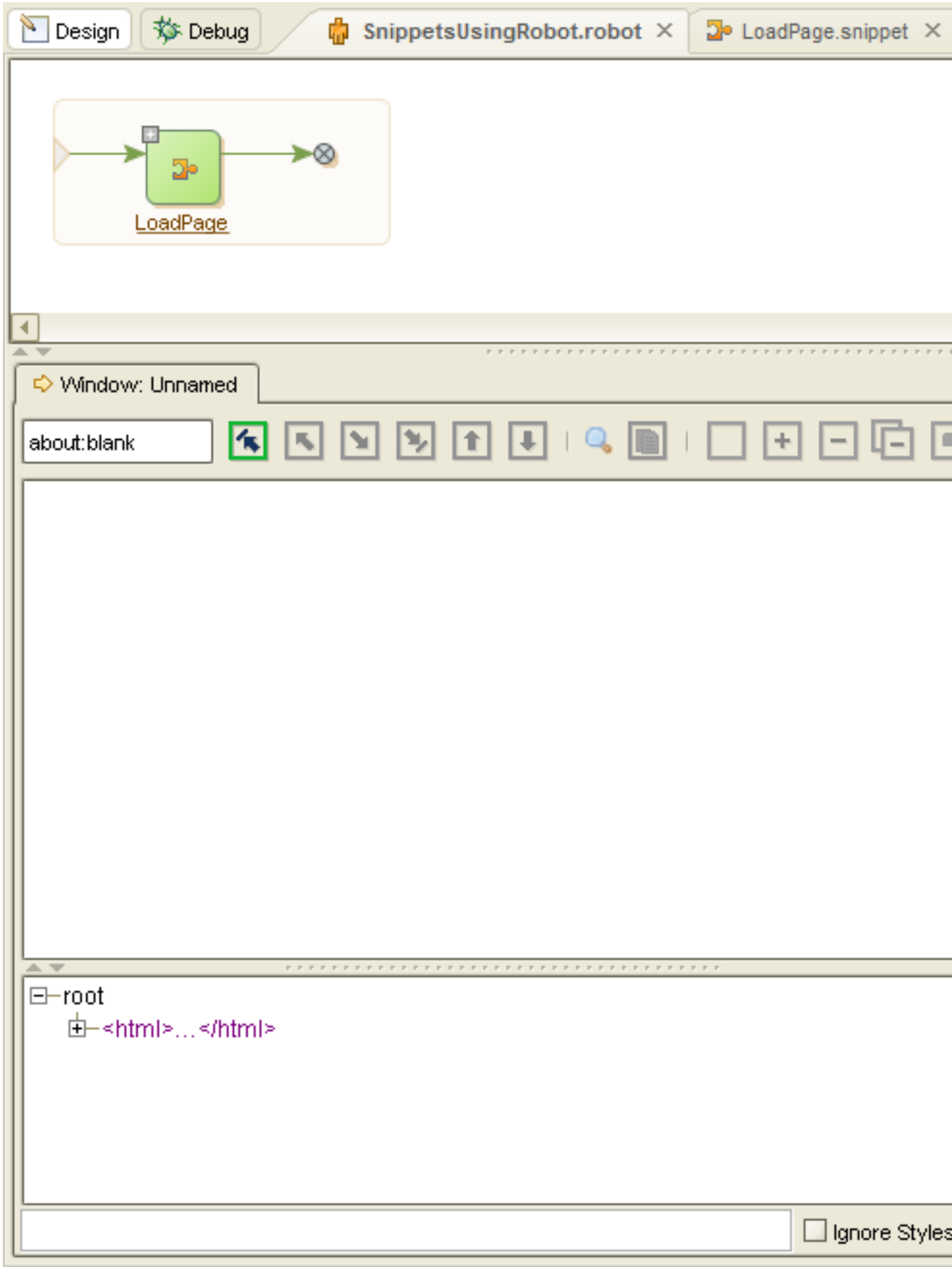
A snippet can define its own variables. Open the snippet in its own editor to define variables on the snippet. If the snippet already contains steps that use variables that existed in the robot where the snippet was edited the steps are marked with a red flag.


Design
Debug
SnippetsUsingRobot.robot
LoadPage.snippet
Load Page
A snippet step can be shared between robots.
Description:
Saved with Version:
9.2.0
Previous Versions:

A snippet in its own editor, that uses a variable not defined on the snippet itself.

Notice the active variables editor in the lower right corner, exactly as on robots.

If a snippet defines variables, using the snippet in a robot will automatically add the snippet variables to the set of variables of the robot.


Design
Debug
SnippetsUsingRobot.robot
LoadPage.snippet
LoadPage
Window: Unnamed
about:blank
root
<html>...</html>
Ignore Styles

A robot that uses a snippet that defines a variable 'url'.

As you can see the variables imported from snippets are marked in the variables list.

A robot should not contain variable definitions by the same name as variables defined in the snippets it uses. If it does so, the variable types must match. Removing a snippet from a robot also removes the variables imported by that snippet.

Good Snippet Practice

When using snippets in your projects a number of things should be considered

- It is a good idea to put **non default robot configuration** that is needed to execute the steps inside the snippet on the steps of the snippet instead of having to remember to set them on each robot that uses the Snippet.
- When inserting a snippet into a robot, take care that names of variables defined on the snippet do not conflict with variables defined on the robot. Design Studio can handle a situation where a variable defined on the snippet has the same name as a variable defined on the robot. Good practice is to **primarily define variables on the snippet** if they are needed for the snippet to work in another context. This makes reuse of the snippet easier.
- It is good practice to **document the context** in terms of named tags and/or windows needed in order for the snippet to work in the description of a snippet.
- A snippet may contain Snippet steps referencing other snippets. However, it is not legal to have cyclic references where a snippet ultimately contains itself. Design Studio will report an error in that case. So **be careful when including snippets inside snippets**.

How to Make Robots More Robust

Web sites often change without notice. Such changes may result in the robot failing to do its task, unless you are careful. **Robustness** is the term used to describe how well robots cope with web site changes. The more changes the robot can deal with (and still work correctly), the more robust it is.

Robustness, however, comes at a price. It is more challenging and time-consuming to write robust robots than writing shaky robots. (The same goes for writing a program in any programming language.) It involves analyzing the web site in question, and to understand how it responds in various situations, such as when a registration form is filled out incorrectly. In a sense, writing robust robots involves a kind of reverse engineering of the web site logic, and usually the only way to do this is through **exploration**.

There are two different approaches to robustness that each serves a different purpose:

- Succeed as much as possible.
- Fail if not perfect.

Let us look at each approach in turn.

Succeeding as much as possible might, for a robot extracting news type variables, mean that it should extract as many news items as possible. In Design Studio, you will use conditional actions Try steps, and data converters to deal with different layouts, missing information, and strangely formatted content.

Failing when things are not perfect might, for an order submission robot, mean that it should fail immediately if it cannot figure out how to enter a field correctly, or the order result page does not match

an exact layout. In this sense, failing does not mean to generate an API exception. Instead, it means that the robot should return a value dedicated to describing errors and failure causes. Robots taking input variables will often choose to fail, rather than to succeed as much as possible. In Design Studio, you will use dedicated error type variables, error handling, and conditional actions to detect and handle unexpected situations.

For more information on Design Studio techniques that can be used to make robots more robust, you should consult the following sections: How to Extract Content from HTML, How to Extract Content From an HTML Table, How to Handle Errors, and How to Use the Tag Finders.

How to Reuse Sessions

A session is the result of browsing on a website, and consists of the page, the page URL and the cookies and authentications obtained in the course. However, obtaining a session where the wanted information is easily reached can require a number of navigation steps such as logging in.

If a robot is run frequently enough, and the response time needs to be very small, getting to a suitable session in the robot can require more time than is available. However, if the session could be obtained once, and then shared between robots and robot runs then great time savings could be achieved. This is exactly what session reuse provides by the means of a session pool.

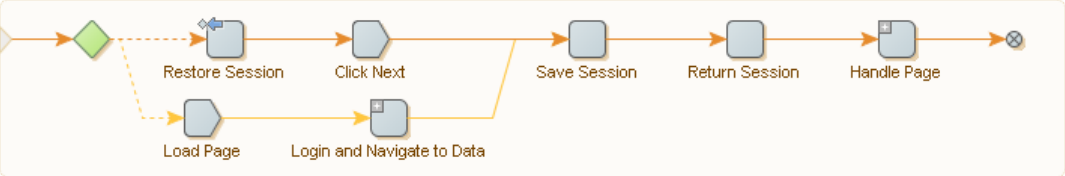
There are two step actions used for session reuse:

1. The Save Session action, which saves a session in the session pool or a variable.

2. The Restore Session action, which restores a session from the session pool or a variable.

In order to restore a session from the session pool, it is necessary to identify it. A session is identified by a site name, a username and a password.

Let's look at an example. Assume that we have a robot that logs in to a web site, collects some data and returns this. However, the data that we seek to collect is distributed over many linked pages, e.g. with a next page link. We want the first invocation of the robot to log in to the site and return the data of the first page, and each subsequent invocation should then return a new chunk of the data (the next page). So what we want is to share the session of a logged-in user between robot invocation, but we also want to remember how much of the data we have returned. The robot could look something like this:



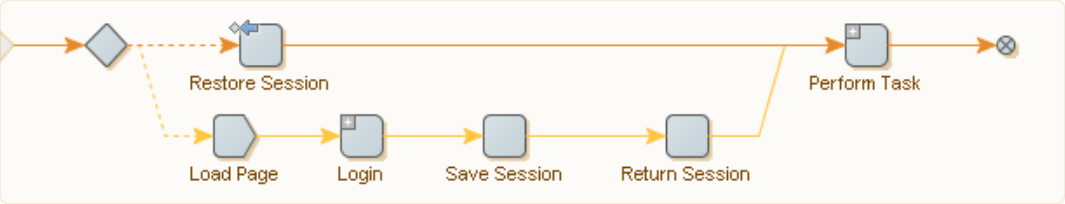
When the robot is called, it will first try to restore a session from an input variable, and if one exists that session will be used and the next step will click on a next page link to get a fresh page of data. If no session is passed to the robot, the step will fail, and the second alternative will be executed, which does the logging in and also navigates to the relevant page on the site where the data may be found.

If the robot execution gets through one of the two alternative branches it reaches the Save Session step. This will save the session so that it may be used the next time the robot is called. But for this to be possible, we need to return the session to the caller of the robot. This is handled by the Return Session step which is a normal Return Value step that returns the value of a variable containing the session, i.e. the variable is of a type which has an attribute of attribute type Session in which our Save Session step stored the session. Finally if the robot reached the end of the data, that is, there is not a next page link on the page, then the

Click Next set will produce an error. This will be ignored by the robot, because we will set Error Handling to Skip Following Steps, but if we have a check mark in API exception then the caller would get an exception, e.g. if the robot is called from Java, it can use this to know that the end of data has been reached.

After the session has been saved, the remaining steps of the robot will extract the data from the page, e.g. by looping over a table and returning a value for each row.

The session pool optimization could also be used in the case where you have several robots that log in to the same site and perform different tasks. If the log in process is time consuming, and the robots are called many times to do simple tasks, it may be a good idea to save the session after log in and then reuse it. The figure below shows a blueprint of a session reusing robot. It can be seen that any such robot that does not have a valid session will perform a log in and return the saved session for other robot to reuse. A robot should normally never rely on a session being available, but always provide a fallback for obtaining a session.



In Design Studio, it is important to understand a bit about the inner workings of the session pool if you want to utilize it. This is because the execution of a robot in Design Studio is not controlled by the natural flow of a robot run, but by the user interaction. First a session should be stored by executing the step containing the Save Session action. Selecting the step following the Save Session step does this. After this the Restore Session action will be able to pick up the stored session. Saved sessions will remain in the session pool even if you refresh the cache by clicking the icon, or after loading different robots. There is currently no way to remove a session from the session pool besides restarting Design Studio.

How to Modify an Existing Type

If you need to change a type after you have written robots using variables of that type, you need to be careful. Your robots might stop working if you do something wrong.

You should take care when performing any of the following changes to a type that is already used by variables in existing robots — if you do, you will not be able to use those variables (however, the robots may still be loaded without the variables):

- Change the name of a type.
- Delete a type.
- Remove or rename an attribute in a type if, in a robot, one or more variables of that type have assigned values different from the default to that attribute.
- Change the attribute type of an attribute if, in a robot, one or more variables of that type have assigned values to that attribute which are not compatible with the new attribute type.

If your robot is open while you make any of the above changes you will see a red status bar at the top of the Robot Editor with a text explaining the problem. The status bar also contains a button that you may click to reload the robot with the compromising variables removed. You can, of course, also solve the problem(s) by making the appropriate changes to your types. If you do this, you can return to your robot and continue working.

The following changes to a type can be automatically carried over to robots without having to remove any variables of that type (you may have to reload), but some errors might be generated when the robots are executed (you can subsequently fix these errors):

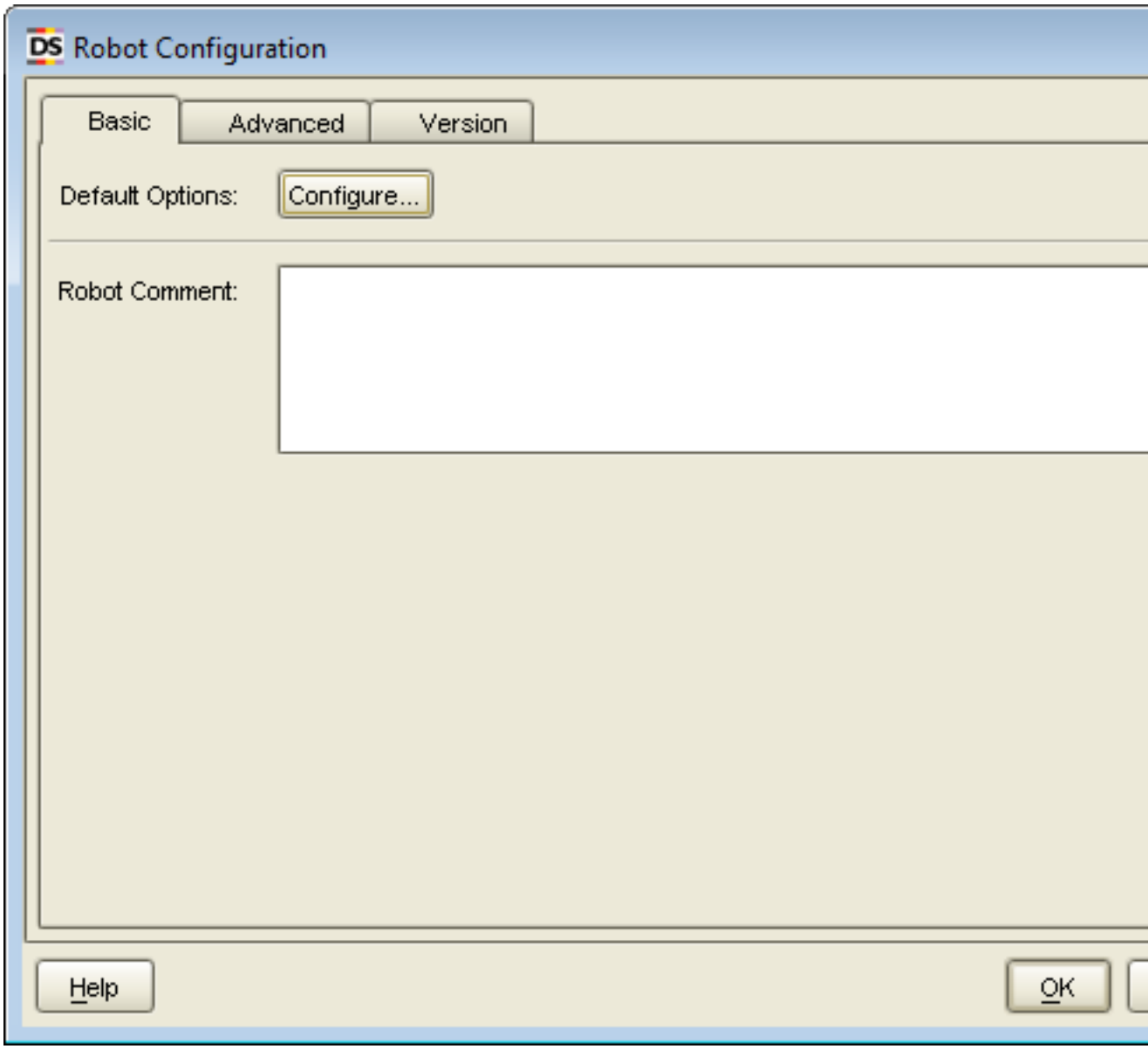
- Change the name of an attribute.
- Change the "Required" property of an attribute from false to true.
- Add a new attribute that has the "Required" property set to true.
- Delete or rename an attribute that has been assigned a value in a variable.
- Change the attribute type of an attribute that has been assigned a value in a variable.

You can always make the following changes, without affecting existing robots:

- Change the "Required" property of an attribute from true to false.
- Change a comment (no matter where).
- Add a new attribute that has the "Required" property set to false.

How to Configure Robots

A robot has a number of properties that you can configure by clicking the icon in the Design Studio toolbar, or selecting "Configure Robot" in the File menu of the Design Studio Main Window. This brings up the **Robot Configuration Window**.



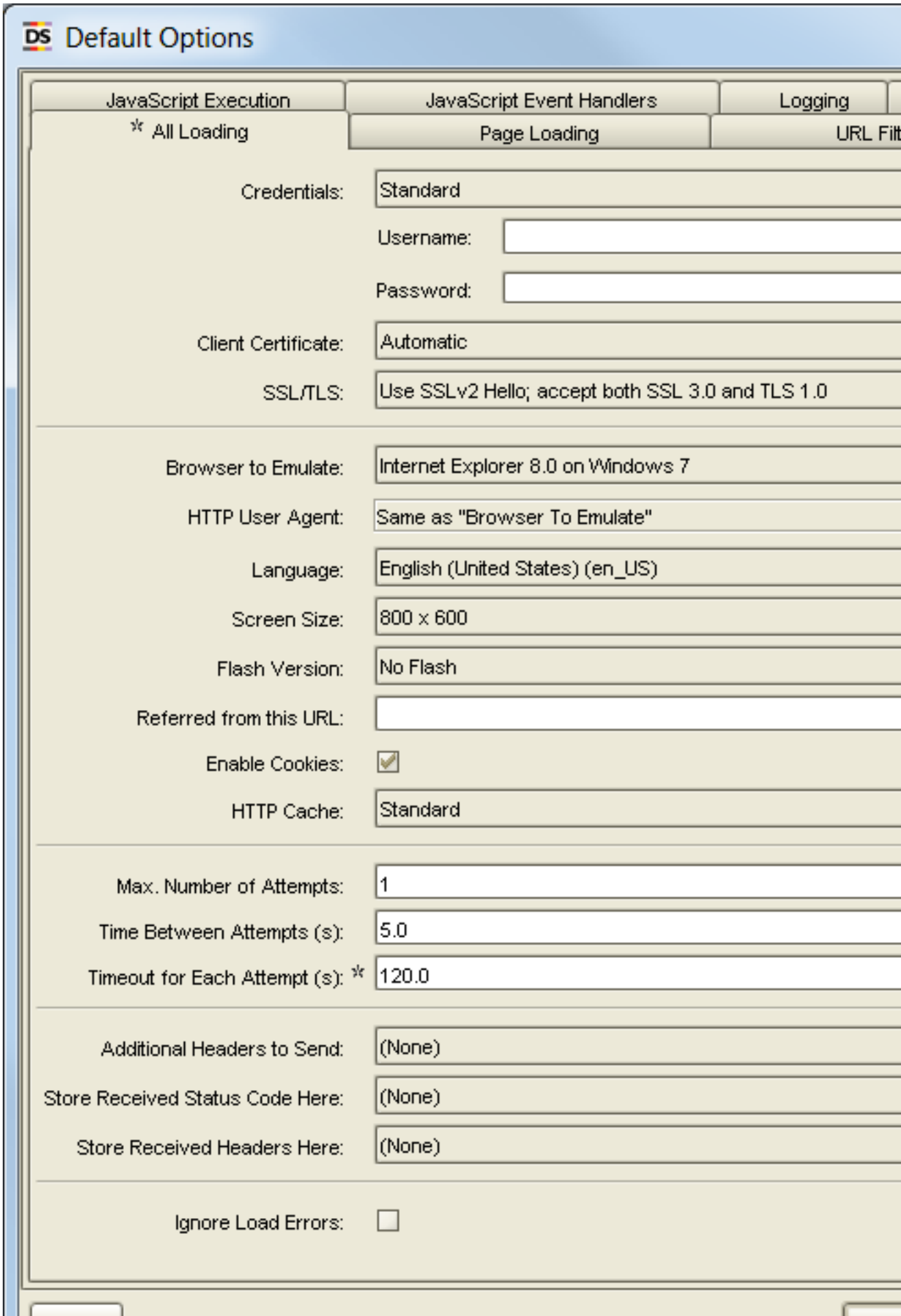
The Robot Configuration Window

In the Basic tab, you can configure default options which apply to all step actions of the robot. A step action can override these global options as needed. You can also enter a comment for the robot. This is useful if you want to document how the robot works, what should be taken into account when editing the robot, etc.

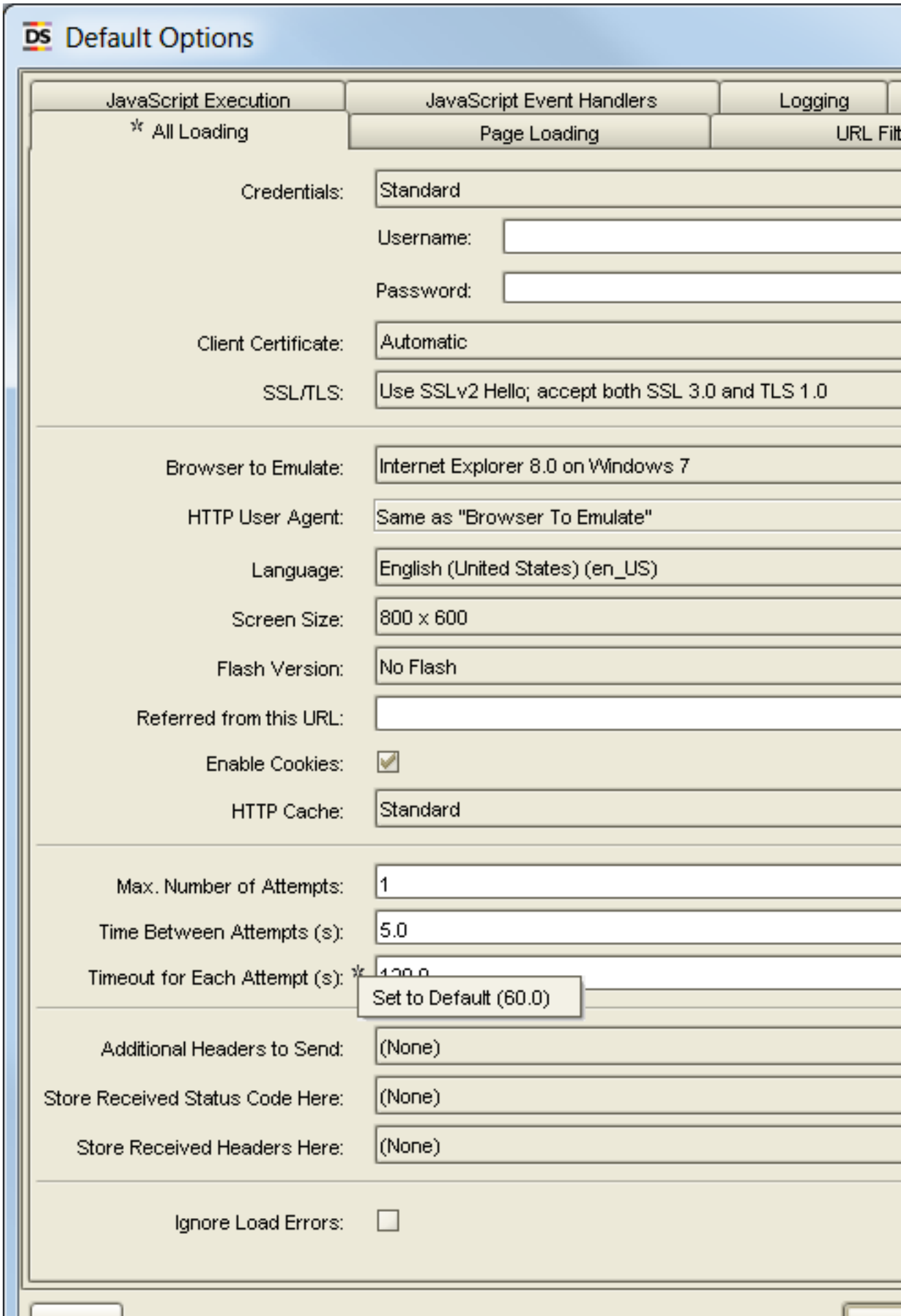
In the Advanced tab, you can specify an optional proxy server to use for all page and data loading done by this particular robot. You will probably only need to use this property rarely. Normally, it is better to specify one or more proxy servers under Design Studio settings. See Proxy Servers for further details on this. The proxy server specified for a particular robot will override proxy servers specified any other way. Furthermore the proxy server can be changed during robot execution with the Change Proxy [../ref/robomaker/reference/stepaction/ChangeProxyStepAction.html] action.

Show Changes from Default

Often, it can be difficult to see if the configurations of certain parts of a robot are done in a way that is *non-standard*, so to speak, e.g. a Page Load step that has been given a longer time out than the default value of 60.0. You would have to open dialogs and maybe click on tabs to get to the place where a property has been changed from the default. Then you would have to look at all the properties displayed to see if any of these have a value that is non-standard. Also, to be able to do this, you would have to remember what the default values for all the properties are. This is why we have introduced the **Show Changes** feature in Design Studio. Properties having a well-defined default value are marked with an asterisk * next to the property name if their value has been changed. The figure below shows this for the Options Dialog.


Default Options
JavaScript Execution
JavaScript Event Handlers
Logging
* All Loading
Page Loading
URL Filt
Credentials:
Standard
Username:
Password:
Client Certificate:
Automatic
SSL/TLS:
Use SSLv2 Hello; accept both SSL 3.0 and TLS 1.0
Browser to Emulate:
Internet Explorer 8.0 on Windows 7
HTTP User Agent:
Same as "Browser To Emulate"
Language:
English (United States) (en_US)
Screen Size:
800 x 600
Flash Version:
No Flash
Referred from this URL:
Enable Cookies:
HTTP Cache:
Standard
Max. Number of Attempts:
1
Time Between Attempts (s):
5.0
Timeout for Each Attempt (s):
*
120.0
Additional Headers to Send:
(None)
Store Received Status Code Here:
(None)
Store Received Headers Here:
(None)
Ignore Load Errors:

Notice that there are also asterisks on tabs. This means that some property on that tab has been changed. If you right click on the property name or the asterisk a popup menu will let you reset the value to its default. As a short cut for this you can double-click on the property name or the asterisk and the value of the property will be set to its default. The popup menu will also, if possible, show you what the default value is. Also, the asterisk on a tab has an attached popup menu that can be used to reset all properties on the tab to their default values. The figure below shows this for the *Timeout for Each Attempt* property on the Options Dialog.


Default Options
JavaScript Execution
JavaScript Event Handlers
Logging
All Loading
Page Loading
URL Filt
Credentials:
Standard
Username:
Password:
Client Certificate:
Automatic
SSL/TLS:
Use SSLv2 Hello; accept both SSL 3.0 and TLS 1.0
Browser to Emulate:
Internet Explorer 8.0 on Windows 7
HTTP User Agent:
Same as "Browser To Emulate"
Language:
English (United States) (en_US)
Screen Size:
800 x 600
Flash Version:
No Flash
Referred from this URL:
Enable Cookies:
HTTP Cache:
Standard
Max. Number of Attempts:
1
Time Between Attempts (s):
5.0
Timeout for Each Attempt (s):
Set to Default (60.0)
Additional Headers to Send:
(None)
Store Received Status Code Here:
(None)
Store Received Headers Here:
(None)
Ignore Load Errors:

There is one situation in which **Show Changes** works a little different than elsewhere, and that is in the Options dialog when this is used to configure options for a step. You can generally configure options in two places:

- from the **Robot Configuration**
- on **steps** that may depend on these options

In both places, you click on a button to open the **Options dialog** to configure the options, but the default value is different for the two situations. If you open the dialog from Robot Configuration then the default is a fixed *application default*, that is the default values that the Design Studio application provides. If you open the dialog from a step configuration, the default is the one defined under Robot Configuration (*robot default*). That is, a step inherits the option values from the robot configuration unless these have explicitly been changed for the step. For example, if the option "Enable Cookies" has been deselected in the robot configuration, all steps that depend on this option will also use this setting unless you have explicitly changed this on the step. An asterisk on an option for a step indicates that the step uses a value different from the one defined in the robot configuration and not necessarily different from the application default.

There is another way in which an asterisk on the step's Options dialog has a different meaning than on the Robot Configuration's Option dialog. For options on the step's Option dialog, an asterisk means that the given option has deliberately been set to a fixed value not necessarily different from the corresponding Robot Configuration value and that it will not be influenced by any changes in the corresponding Robot Configuration value. For example, if initially the *Timeout for Each Attempt* property for a step is set to 120 and the corresponding value under the Robot Configuration is 60 then there will be an asterisk next to this option. If the Robot Configuration value is then changed to 120 so that the two values are actually the same, then the step's value is still marked with an asterisk and if the value for the robot is again changed away from 120 then the value for the step will stay unchanged at 120. If, however, you change the step's value back to its default value (the value from the Robot Configuration) using the popup menu or double-click, the step's value will use the Robot Configuration value and subsequently follow any changes made to this.

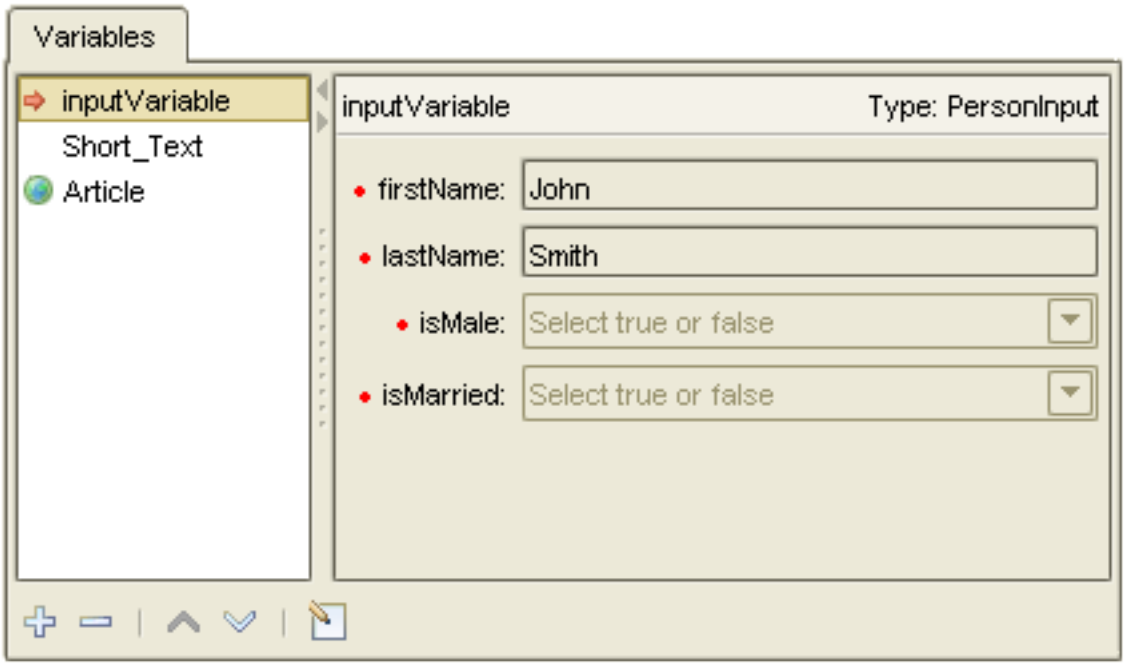
Another situation where the default value may depend on other configuration choices is found in the configuration of a step's error handling.

How to Configure Variables

When you create a new robot, you usually start by configuring its variables. Of course, you can reconfigure the variables at any time during the robot's lifetime, e.g. if, at some point, you want to change the initial value of a variable.

You configure the variables of the robot in the Variables View, located below the Step View in the Robot Editor. The variables that you specify become part of the robot state given as input to the first step of the robot.

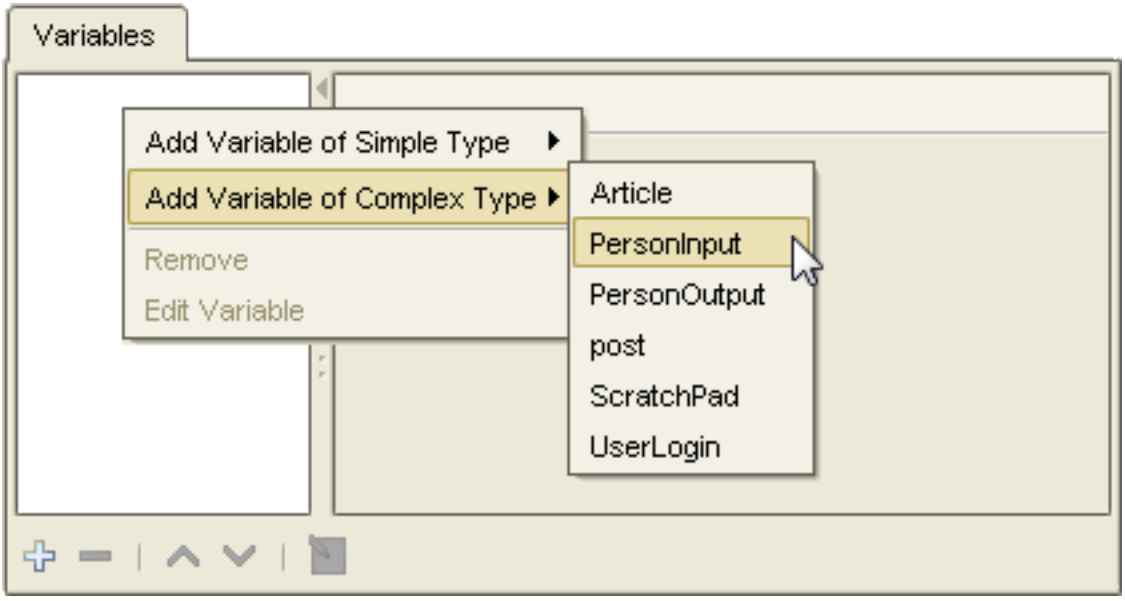
The Variables View shows a list of variables, together with details on the selected one. A variable may be set as an input variable in which case the icon is shown next to it. Also, a variable can be set as being global in which case the icon will be shown next to it (how to configure a variable is described below). The Variables View shown below contains three variables: one input variable, one normal variable, and one global variable.



The Variables View

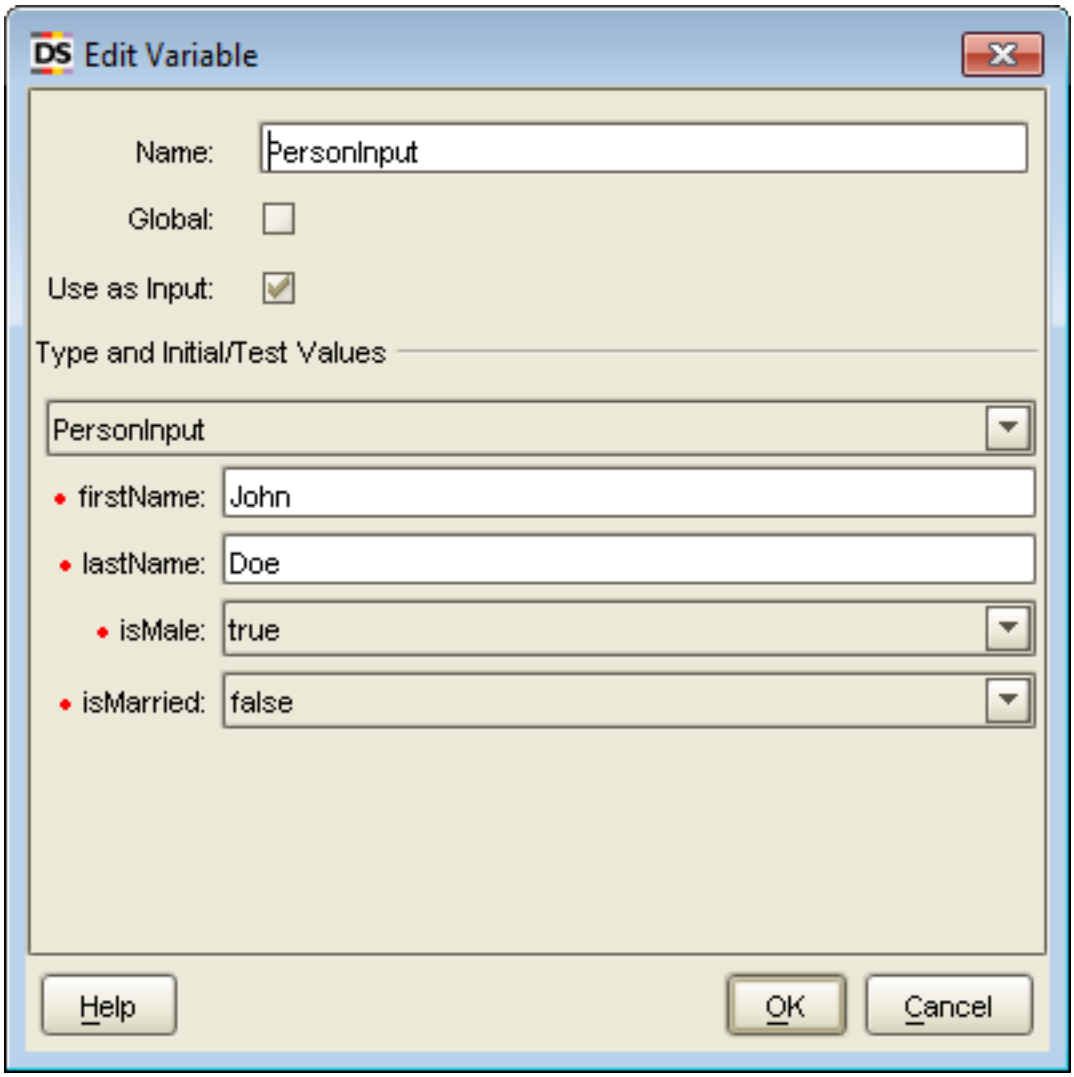
The Variables View shows the values of the variables at the current step. Because these values result from the execution of the robot, you cannot change them directly. However, you can add or remove variables.

There are two ways to add a new variable: you can click the button at the bottom of the Variable View, or you can right-click the Variable View and select a type from which the variable should be created as shown in the figure below.



Adding a new variable

Adding a variable is done using the variable configuration dialog. If the variable is added using the right-click method, where a type is already chosen, the dialog opens with the type pre-selected. The variable configuration dialog is shown in the figure below. This is also the dialog you get if you want to configure an existing variable, either by double-clicking it or by clicking the button.



The Variable Configuration Dialog

In this dialog there are a number of options. Firstly, you can choose a name for the variable. The name must adhere to some standards, for example it is not allowed to contain whitespace. When you click the 'OK' button, the dialog will tell you if the name is invalid, and you must change it to continue (or click 'Cancel'). Secondly, you must select a type for the variable. Types and their connection to variables are described here. When you have selected a type, a number of input fields appear, depending on which attributes the type has. These fields can be used to provide the variable with initial values. You do not have to manually give the variable a name. If you have not entered a name, when clicking 'OK' the dialog will ask you if you want to generate a name from the type name.

Note: the variable configuration dialog allows you to edit initial values. In other words, this dialog does not, as the Variables View, show current values. The values you provide are those that the variable has at the start of the execution.

The two remaining options in the dialog are the two checkboxes 'Global' and 'Use as Input'. These are used to configure the important properties of whether a variable should be used as input to the robot, and/or whether it should be global.

If a variable is used as input, it makes it possible to supply the robot with values for that variable when running it on a RoboServer. This means that if the variable has been selected as being input, the values

entered for its attributes in the dialog should be regarded as test input and are only used when you are working with the robot in Design Studio. When the robot is run on RoboServer, the input values will be overridden (i.e. replaced) by values supplied by the client that runs the robot. Please note that variables of simple types cannot be used as input, as their usage is as temporary variables, internal to the robot. More on using input variables can be seen here.

The 'Global' checkbox marks the variable as global, meaning that the variable keeps its value during the entire execution of the robot. This is different from normal variables, whose values are not kept across loop iterations and branches.

Global variables provide a way to create counters and do other kinds of computation across iterations and branches. Global variables can also be used for accumulation of data across iterations or branches, such as accumulating a text consisting of comma-separated values.

Note: In Design Studio, the values of the global variables depend on which specific steps you have executed to get to the current step. Unless care is taken to execute the right sequence of steps, the values will be different from when the robot is actually executed.

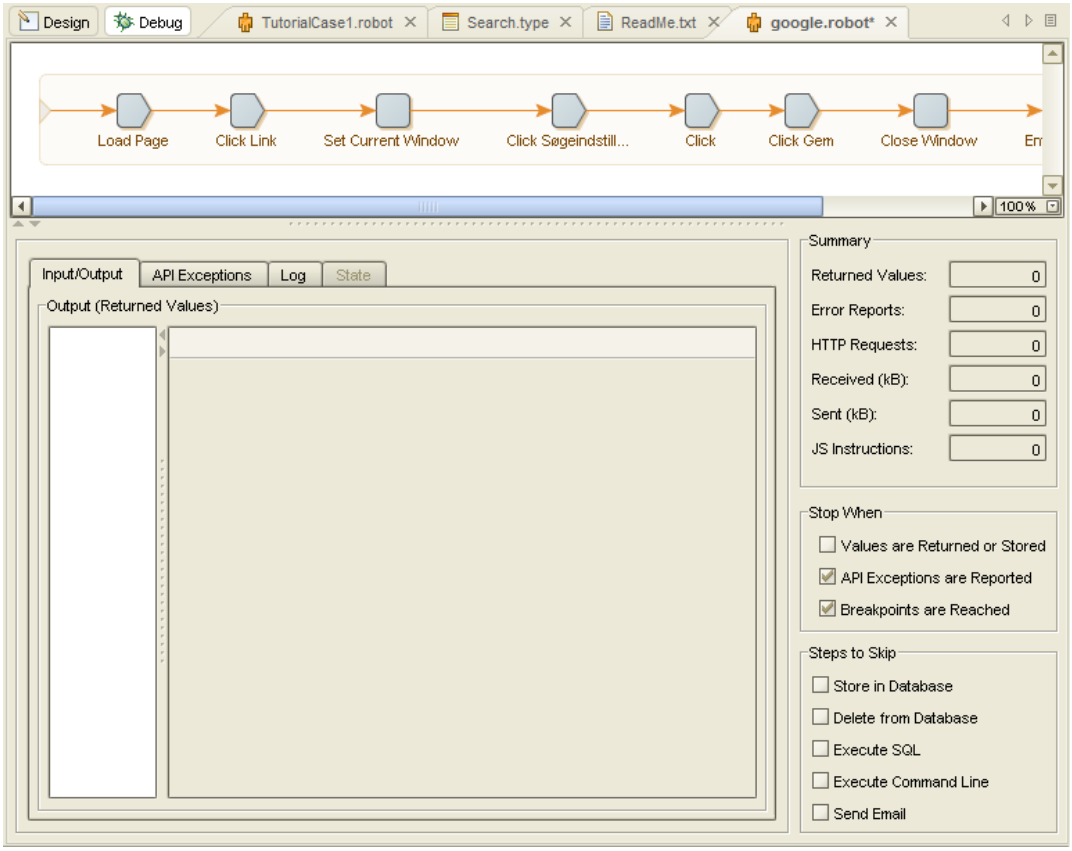
To remove a variable, simply right-click it and chose 'Remove'. Alternatively, select a variable and click the button below the list.

How to Debug a Robot

In this section, we will take a closer look at how to debug a robot using the debug mode that is built into Design Studio. Debug mode allows you to execute a robot in the same way as it will be executed by RoboServer. This allows you to check that the robot does what you expect.

Basic Debugging

To switch to debug mode, click the icon or the Debug button in Design Studio. To start debugging the robot, click the icon.



The Debug Mode of the Robot Editor

As the robot is being executed in debug mode, you can watch the current location in the Robot View. You can also watch the results of the execution in the main panel. In the "Input/Output" tab, the "Input" panel shows the input variables, if any, and the "Output" panel shows all values that have been returned so far during the execution. If the robot has no input variables, the "Input" panel is not shown. In the "API Exceptions" tab, you can see all API exceptions that have been generated so far during the execution. In the "Log" tab, you can see what has been written to the log so far during execution. In the "State" tab, you can see the robot state, if any. Also, in the "Summary" panel to the right of the main panel, you can see a summary of the execution, containing the number of returned values, the number of API exceptions generated, statistics on the number of HTTP requests, the amount of sent and received data, and the number of JavaScript instructions that have been executed.

It is important to understand that execution in debug mode is independent of the execution done in design mode in Design Studio. Therefore, debug mode has its own current step and its own current robot state, independent of the current step and current robot state in design mode. In debug mode, the current step is the step that is about to be executed, or is being executed, in the debugging process, and the current robot state is the input to that step.

You can stop the debugging at any time by clicking the icon. You can also make the debugging stop when certain events occur. This is done in the "Stop When" panel. Here, you can choose whether the debugging should stop when values are returned, when API exceptions are reported, and when breakpoints are reached. Of course, debugging will always stop when the execution of the robot has completed.

When debugging has stopped, you can see the reason for the stop in the status bar at the bottom of the Robot Editor. If the debugging has stopped before the execution of the robot is complete, you can watch the current robot state in the "State" tab. The "Variables", "Windows", "Cookies", and "Authentications"

sub-tabs show the robot state in the same way as in the State View in Design Studio. The "API Exception" sub-tab shows the API exception, if the execution stopped because an API exception was reported.

If debugging has stopped before the execution of the robot is complete, you can resume the debugging by clicking the icon. You can also restart the debugging by clicking the icon. This will abort the current debugging process and make the debugger ready to start a new debug from the start of the robot. The debugging is also restarted automatically whenever the current robot is modified or replaced by another robot in Design Studio.

If the robot has input variables, the input values of these can be edited in the "Input" panel, and when you press Enter, the debugging will be restarted with the new input values. The input values cannot be edited while a debug is running, so if you want to change the input values, you must first restart the debugging.

Debugging from the Current Location in Design Mode

You can start debugging from the current location in design mode by clicking the icon in Design Studio. This will switch the Robot Editor to debug mode and execute as directly as possible to the location that is current in design mode. When that location is reached, debugging will stop. You can then continue debugging from this location by clicking the icon.

This feature is useful if you just want to debug a part of the robot, like a specific branch or a specific iteration of a loop action.

If Design Studio is already debugging the robot when you press the icon, it will have to restart the debugging before it executes to the location, and it will ask you for permission to do this.

Going Back to Design Mode from a Debugging Location

When debugging has stopped at some location in the robot, you can switch back to design mode at the same location by clicking the icon in debug mode. This allows you to closer examine that location in design mode, and perhaps modify the steps around that location, or modify some other part of the robot.

You can also switch to design mode and go to the location where a value was returned. To do this, select the value in the "Output" panel in the "Input/Output" tab and click the "Goto" button in the lower right corner of the tab. This is useful if a value has not been extracted correctly, and you want to find out why.

You can also switch to design mode and go to the location where an API exception was reported, or where an error occurred. When you view an API exception in the "API Exceptions" tab or the "API Exception" sub-tab in the "State" tab, you can click the "Goto" button to the right of the location code to go to the location where the API exception was generated. You can also click on a "Goto" button to the right of a specific error to go to the location where that error occurred. This is, of course, very useful when you want to find the reason for the error and fix the problem.

When you have done what you want in design mode, you can resume the debugging. If you haven't modified the robot, you can simply click the icon. If you have modified the robot, the debugging will have been automatically restarted, so you cannot resume it directly. Instead, you can start a new debugging session from the current location in design mode by clicking the icon.

Using Breakpoints

While debugging, you can make Design Studio stop at a specific step in the robot by setting a **breakpoint** on that step. The easiest way to do this is to right-click on the step in the Robot View and select "Toggle Breakpoint" in the pop-up menu. The breakpoint will be indicated by a small icon in the step.

When Design Studio reaches a breakpoint during debugging, it will stop, unless you have chosen not to stop at breakpoints in the "Stop When" panel. You can resume the debugging by clicking the icon.

You can remove the breakpoint from a step by right-clicking on it and selecting "Toggle Breakpoints". If you select one or more steps, you can remove all breakpoints of these steps by selecting "Remove Breakpoints". You can also remove all breakpoints in the robot by clicking the icon.

Single-Stepping

You can execute one step at a time in debug mode. This is called **single-stepping**. It is useful if you want to examine the execution very closely.

You can single-step when Design Studio is ready to start a new debug, or when it has stopped during a debug. To execute the next step, click the icon. When that step has been executed, execution will stop. You can then click the icon again to execute the next step, and so on. At any step, you can also resume normal execution by clicking the icon.

Stepping-Into

If you have used the Group step [../ref/robomaker/reference/stepaction/GroupStep.html] to create a group containing several steps, when debugging, a breakpoint can be created at a Group step in the same way as when creating a breakpoint at any other step. If the group is collapsed, however, using single-stepping will treat the group as a single step and not enter any of the grouped steps. If the group is expanded, the steps in the group will be visited when using single-stepping. If a group is collapsed, using the icon, instead of the icon, it will result in the debugger **stepping-into** the group and visiting each step. If you wish to leave the group and advance the debugger to the step following the Group step, you can use the icon to step out.

How to Use the Browser Tracer

Sometimes things do not behave as expected and it can be difficult to figure out just what is going on in a complex website. The Browser Tracer can assist you in this. It can trace JavaScript execution and HTTP traffic in Design Studio and external browsers and compare these traces side-by-side so that differences can easily be identified.

The Browser Tracer is available from the Tools menu in Design Studio.

Setting Up a Browser

A browser, such as Internet Explorer, can be traced by setting it up to use a special proxy server which is built into Design Studio and started when Design Studio starts. This proxy server typically runs on port 9999, but if you start multiple instances of Design Studio, additional instances will use different ports. You can see the exact port number in the Browser Tracer window.

In Internet Explorer, setup the proxy server by opening Internet Options and choosing LAN Settings from the Content tab. Enable "Use a proxy server for your LAN" and type "localhost" in the Address field, and 9999 in the Port field. You should also clear the browser's cache because cached JavaScript files cannot be traced.

Tracing

To record a trace for either Design Studio or a browser connected through the Browser Tracer's proxy, click the icon for the source you want to trace. While recording, things may run much slower than normal

since vast amounts of data is collected. Thus, you should make sure to disable recording by clicking the icon again once you have traced what you wanted.

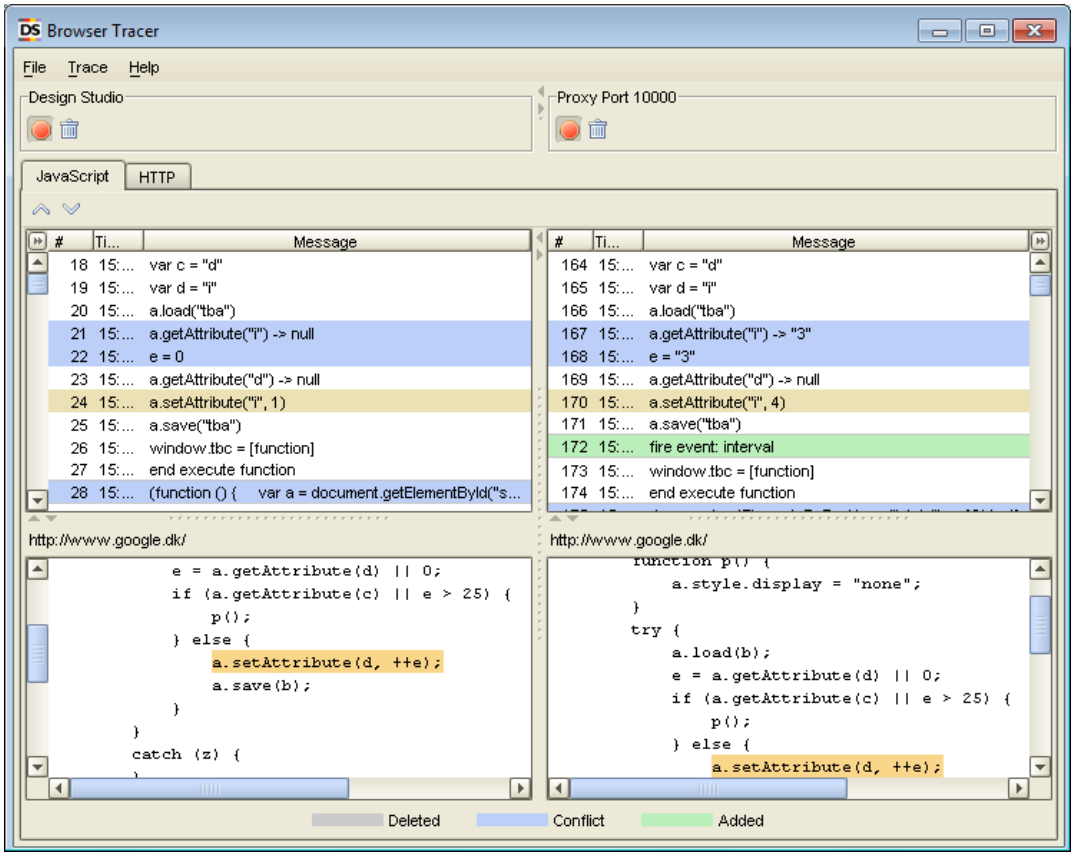
In a typical tracing scenario you would do the following:

1. Enable trace recording for Design Studio.
2. Execute the step action in Design Studio that you are interested in, say, a Load Page.
3. Disable trace recording for Design Studio.
4. Enable trace recording for the proxy.
5. Perform the exact same actions in your browser, say, load a page.
6. Disable trace recording for the proxy.

Now, you have produced two traces which you can compare side-by-side in the difference view.

The Difference View

In the difference view two traces are shown side-by-side. Conflicts are highlighted with a blue color, trace entries that only occur in the left view are gray, and entries that only occur in the right view are green.



The Browser Tracer Window

JavaScript Trace

Below each JavaScript trace, the JavaScript source code for the currently selected trace entry is shown. When a trace entry is selected, the corresponding source code line is highlighted in the source view. The

trace entry is the runtime result of the execution of the highlighted source code line. Each source code line may, of course, be executed multiple times, in which case multiple trace entries are produced - all corresponding to the same source code line.

Stepping through trace entries can help you understand how a piece of JavaScript code works.

HTTP Trace

The HTTP trace shows HTTP traffic. Selecting a trace entry shows the details about that HTTP event in the detail view below the trace. The detail view shows the request and response headers, as well as the request and response data sent. Normally, only POST requests will contain request data.

Saving and Loading Trace Sessions

Trace sessions can be saved and loaded at a later time. A trace session contains both the Design Studio trace and the proxy trace, and both JavaScript and HTTP traces. Saving a trace session can be useful if it is large and you want to look at it in detail at a later time, or if you want to mail it to someone else.

Note, that bug reports submitted from Design Studio will automatically contain the current trace session, if any.

Design Studio Settings

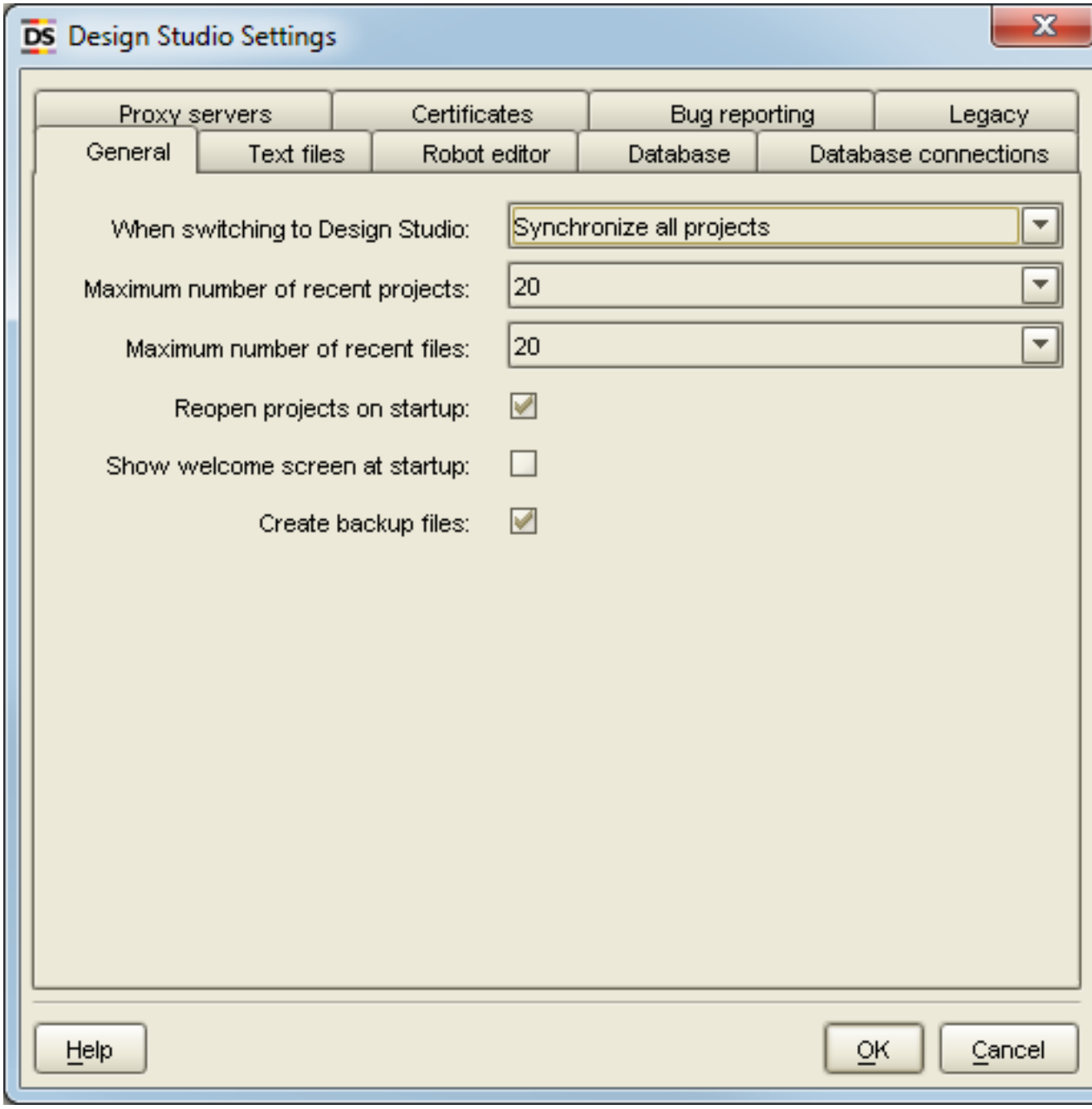
Settings in Design Studio can be altered in the Design Studio Settings dialog found under Options in the menu bar. Setting are divided into the sections listed below.

- General Settings
- Text files
- Robot Editor
- Database
- Database Connections
- Proxy Servers
- Certificates
- Bug Reporting
- Legacy

The sections are shown on different tabs in the dialog and described in detail in following subsection of this user guide.

General

The General tab of the settings dialog is shown as default upon opening the dialog. On this tab the user may alter some of the more general settings of Design Studio. This section describes each of the fields shown on this tab.



General settings in Design Studio

Each of the fields have been predefined with default values. The user may choose to simply leave the values at their default, but they may also be altered to meet individual needs. The table below describes each field in detail.

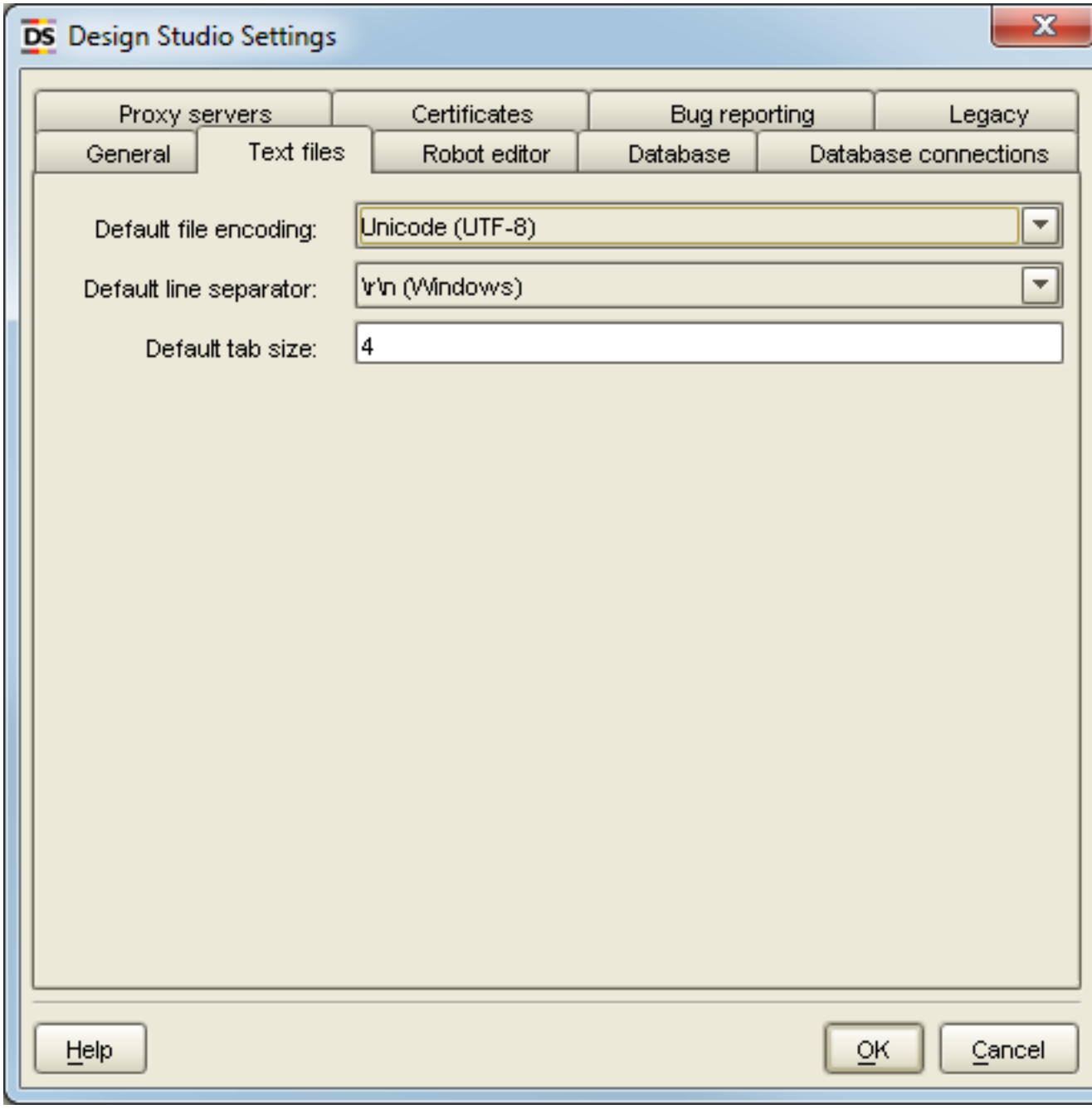
Table 9. Options descriptions

Option	Description
When switching to Design Studio	Lets the user specify what should happen when switching to Design Studio.

Option	Description
Maximum number of recent projects	Design Studio keeps a local history of recent projects. This option allows the user to specify the maximum number of recent projects to keep in history. Recent projects can be accessed under File -> Recent projects.
Maximum number of recent files	Design Studio keeps a local history of recent files. This option allows the user to specify the maximum number of recent files to keep in history. Recent files can be accessed under File -> Recent files.
Reopen projects on startup	When this checkbox is checked, projects that are open when Design Studio closes will be opened again when Design Studio is started.
Show welcome screen at startup	With this checkbox checked, the welcome screen will be showed every time Design Studio is started.
Create backup files	Design Studio will create backup files whenever a saved file is changed when this checkbox is checked. Backup file are characterized by ending with ~ .

Text Files

The Text files tab of the settings dialog lets the user alter settings for the text files in Design Studio.



Text files in Design Studio

Default settings for the text files have been predefined, but they can be changes if needed. The table below describes each of the fields in detail.

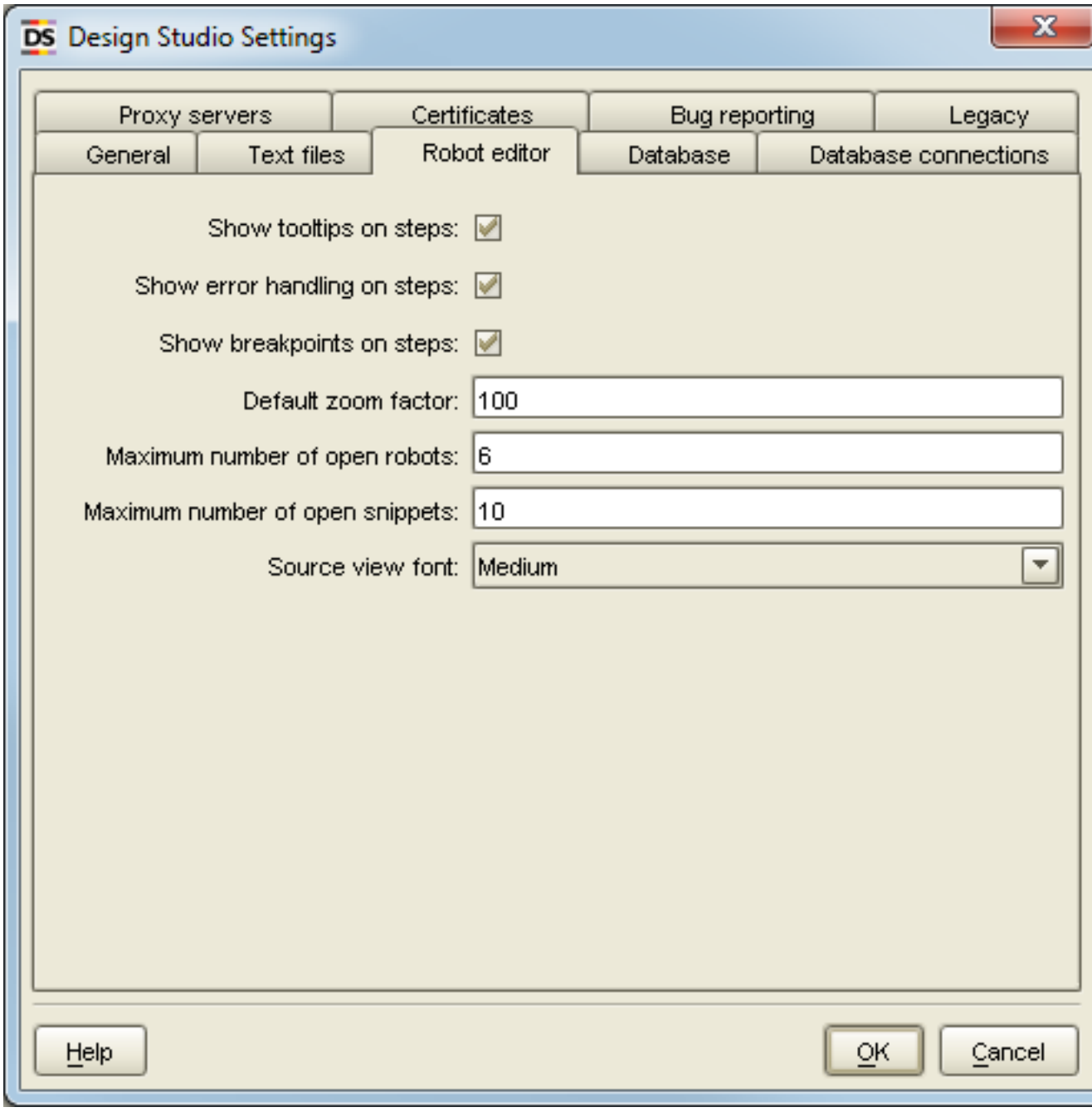
Table 10. Options descriptions

Option	Description
Default file encoding	Specifies the default encoding of text files.
Default line separator	Specifies the default line separator in text files.

Option	Description
Default tab size	Specify the default tab size in text files.

Robot Editor

A number of robot editor settings can be changed in Design Studio.



Robot editor settings in Design Studio

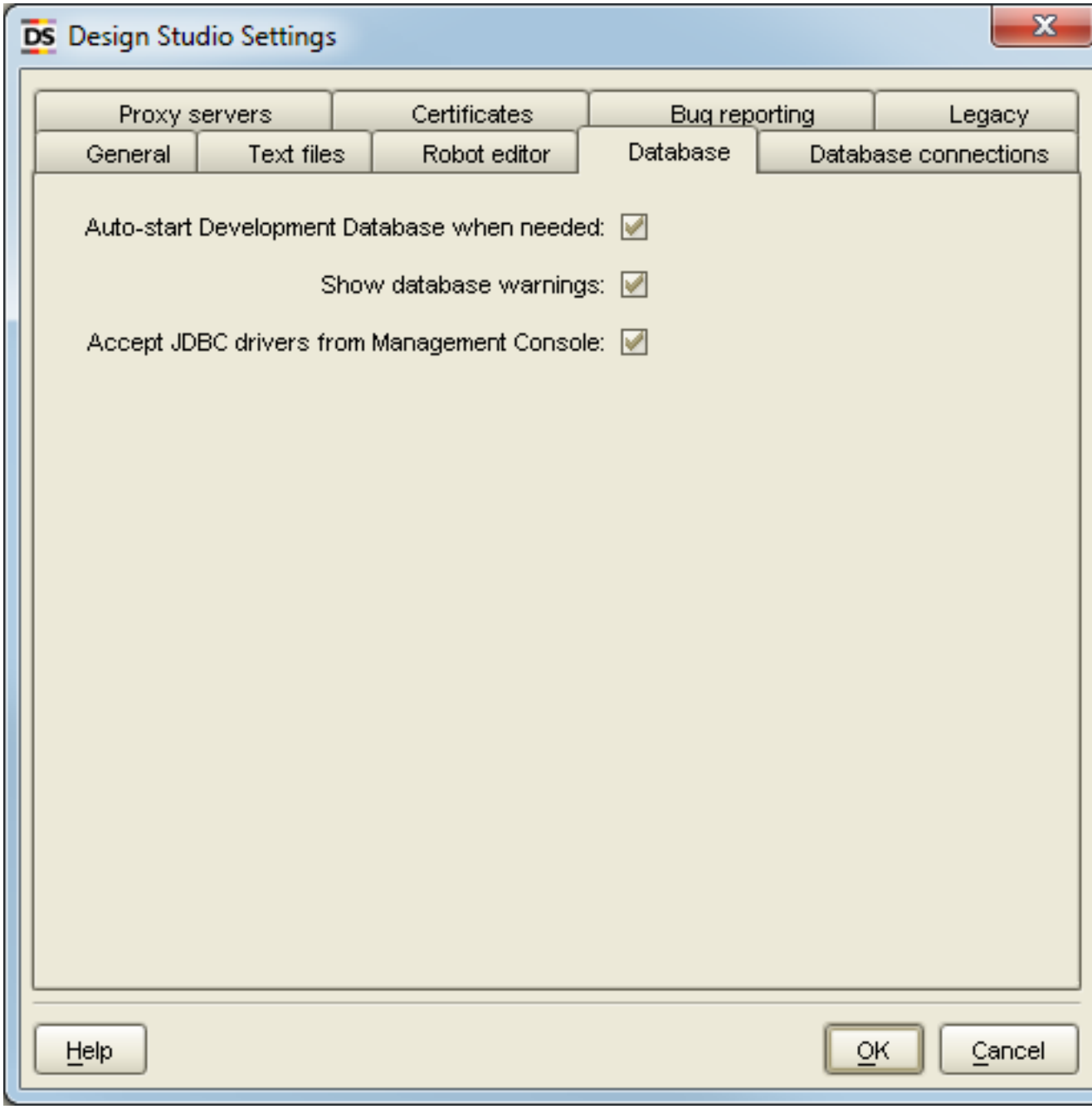
Robot editor settings are predefined with default values. The table below describes each of the fields in detail.

Table 11. Options descriptions

Option	Description
Show tooltips on steps	When this checkbox is checked, tooltips will be shown for robot steps. To disable tooltips on steps uncheck this box.
Show error handling on steps	Robot steps with custom error handling are marked with a symbol to make it visually obvious that custom error handling apply to the step.
Show breakpoints on steps	If a breakpoint is set on a robot step, the step is marked with a symbol to make it visually obvious that execution will stop at this step if run in the debugger.
Default zoom factor	The zoom factor to apply when opening robots in the robot editor. Changes to the zoom factor for an open robot can be changed manually in the bottom right corner of robot editor.
Maximum number of open robots	The maximum number of open robots in the editor. If the maximum number of robots have been opened, the user is asked to select which robot(s) to close when attempting to open more robots.
Maximum number of open snippets	The maximum number of open snippets in the editor. If the maximum number of snippets have been opened, the user is asked to select which snippet(s) to close when attempting to open more snippets.
Source view font	

Option	Description
	Specifies the font size of the text in the source view (the bottom section of Design Studio)

Database



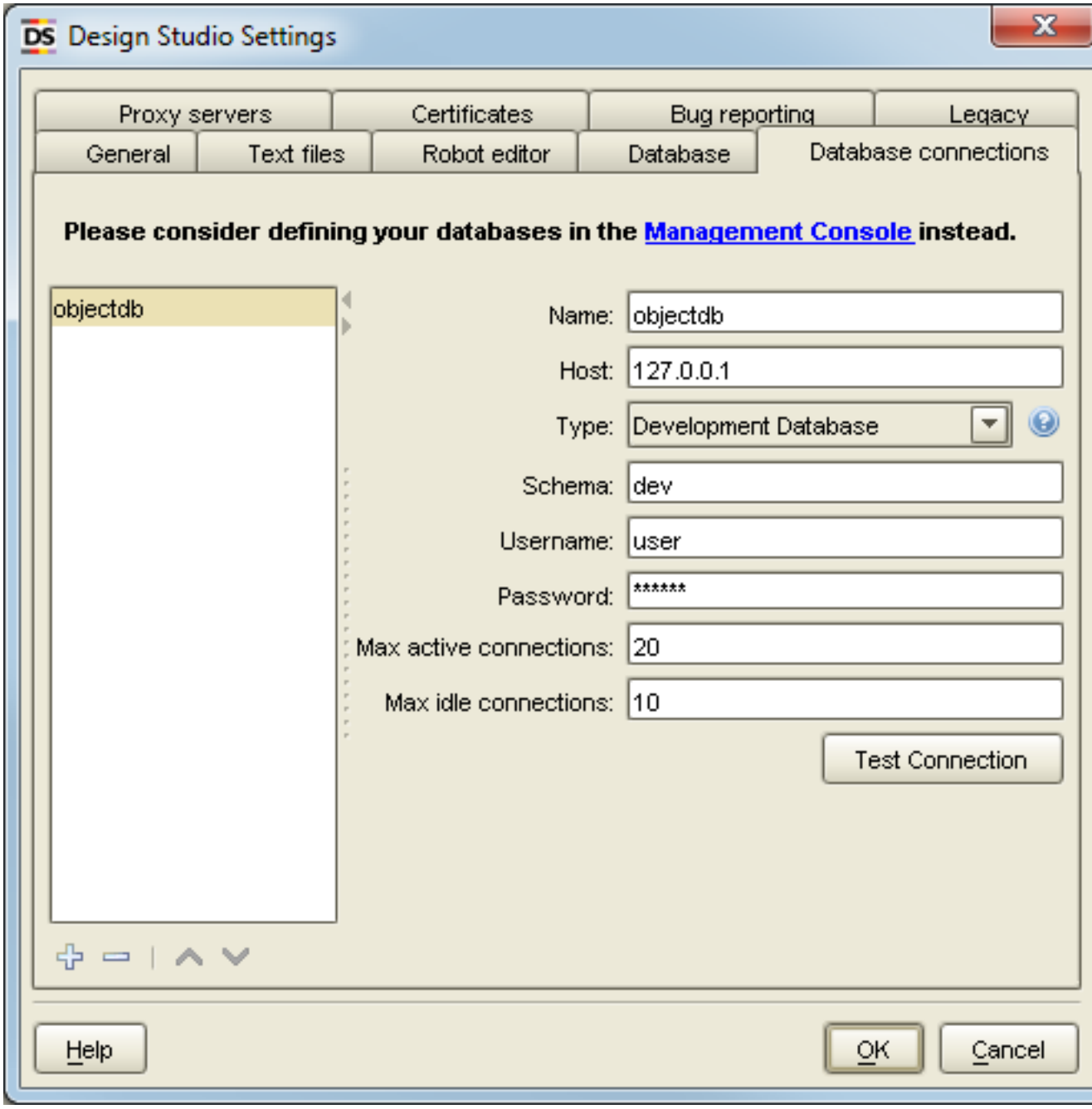
Database settings Design Studio

Table 12. Options descriptions

Option	Description
Auto-start Development Database when needed	If this option is checked the Development Database will be automatically started if a mapping to the Development database exists
Show database warnings	With this option checked, database warnings, such as missing tables, will be shown at the top of the robot editor
Accept JDBC drivers from Management Console	JDBC drivers are distributed from Management Console to Design Studio if this option is checked. Users should only rarely need to disable this option.

Database Connections

This section describes how to create a database in Design Studio. It should be noted that databases created in Design Studio are only available in Design Studio. In order to make the database available both in Design Studio and on the server the databases must be configured in the Management Console.



Configure database connections in Design Studio

In the left side of the window is a list of the created connections. You can create new connections, remove connections, or change their order, using the buttons below the list. The currently selected connection is configured in the right side of the window.The fields name, host, type and schema are mandatory and must be specified.

The different database types are defined in the Management Console and are automatically distributed to Design Studio at startup. New database types must thus be created in the Management Console

Below is an in-depth description of the database fields:

Table 13. Options descriptions

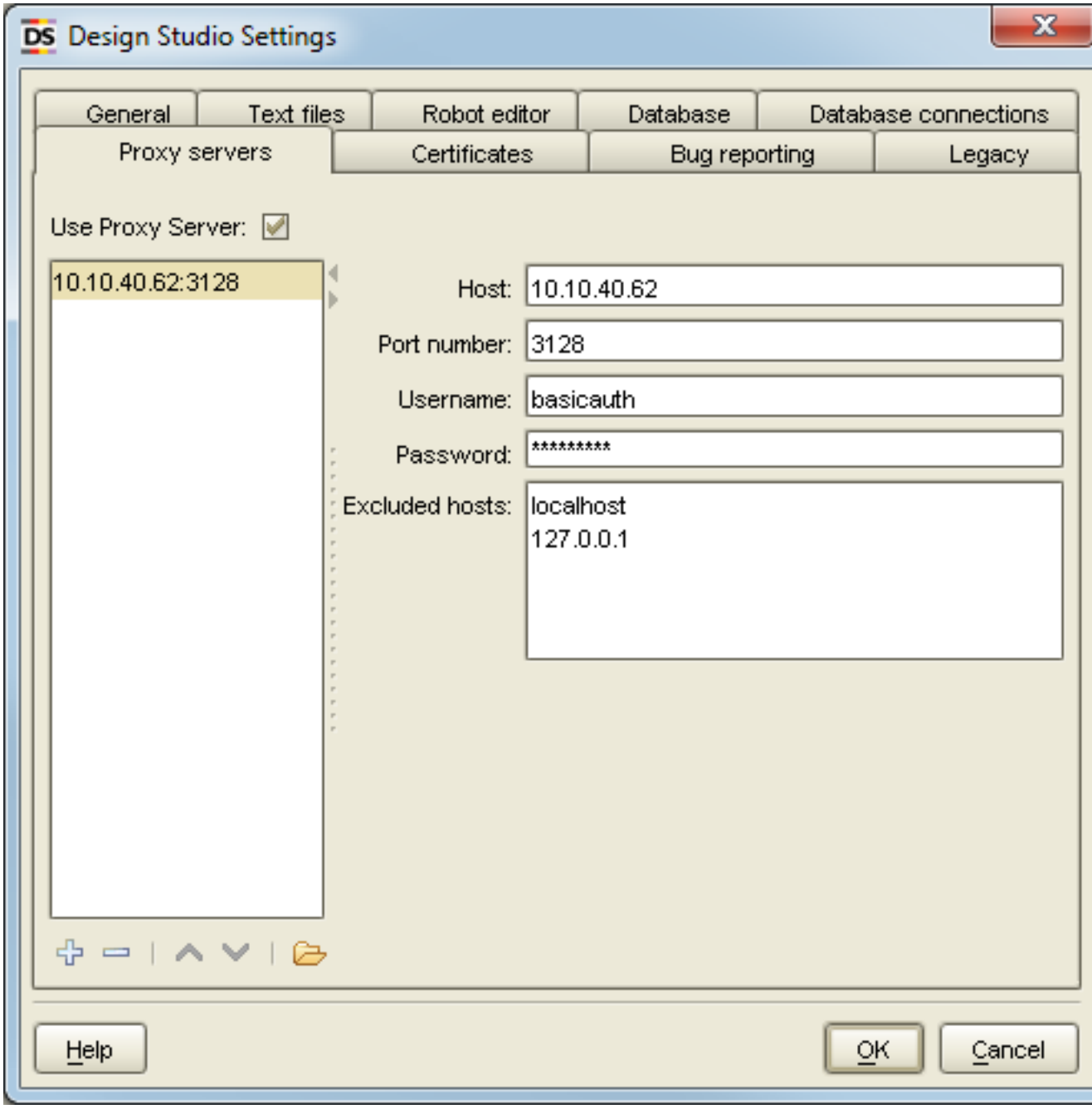
Option	Required	Description
Name	Yes	A name for uniquely identifying the database in Kapow Katalyst. The name is used for internally referencing the database and it may only contain alphanumeric characters and underscores.
Host	Yes	The host name of the database server. This can be an IP-address, or the fully qualified domain name, e.g. myhost.kapowtech.com.
Type	Yes	The type of database, e.g. Oracle. The different types of databases are configured in Management Console and provided automatically at Design Studio startup.
Schema	Yes	The name of the database schema (or catalog).
User Name	No	The user name for the database.
Password	No	The password for the database.
Max Active Connection	Yes	The maximum number of concurrent connection that Kapow Katalyst (RoboServer or Design Studio) will create to the database. The connections are managed by a connection pool, which means that existing connection will be reused, before creating new connections.
Max Idle Connection	Yes	The maximum number of idle connections allowed. If more than this amount of connection have been created, due to heavy load they will automatically be closed when they are no longer needed.

You can test the current connection by pressing the “Test Connection” button. Note: This will only test the connection to the database, it will not test that you have the proper permissions in the database.

Connecting to Oracle: If you are using an Oracle database, you need to enter a username *and* a role in the “Username” field. For example, if the username is “sys”, and the role is “sysdba”, you should enter “sys as sysdba” in the “Username” field.

Proxy Servers

Under Design Studio Settings you can specify a number of proxy servers that can be used by Design Studio.



Configure proxy servers in Design Studio

In the “Proxy Server” tab of the Design Studio Settings proxy servers are specified using the following properties:

Table 14. Options descriptions

Option	Description
Use Proxy Server	Check this checkbox to enable the use of a proxy server.
Host	The host name of the proxy server. This can be an IP address, or the fully qualified domain name, e.g. myproxy.kapowtech.com.

Option	Description
Port Number	The port number on the proxy server. Leave this blank to use the default proxy server port 8080.
User Name	The user name to use if the proxy server requires login.
Password	The password to use if the proxy server requires login.
Excluded Hosts	Here, you can specify a list of host names that the proxy server should not be used for. You should specify one host name per line. Each host name can be an IP address, or a fully qualified domain name, e.g. www.kapowtech.com.

Note that a list of proxy servers can also be imported from a file using the open file button underneath the proxy server list. The file may hold an arbitrary number of proxy server definitions each of which must comply with the following format:

```
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
```

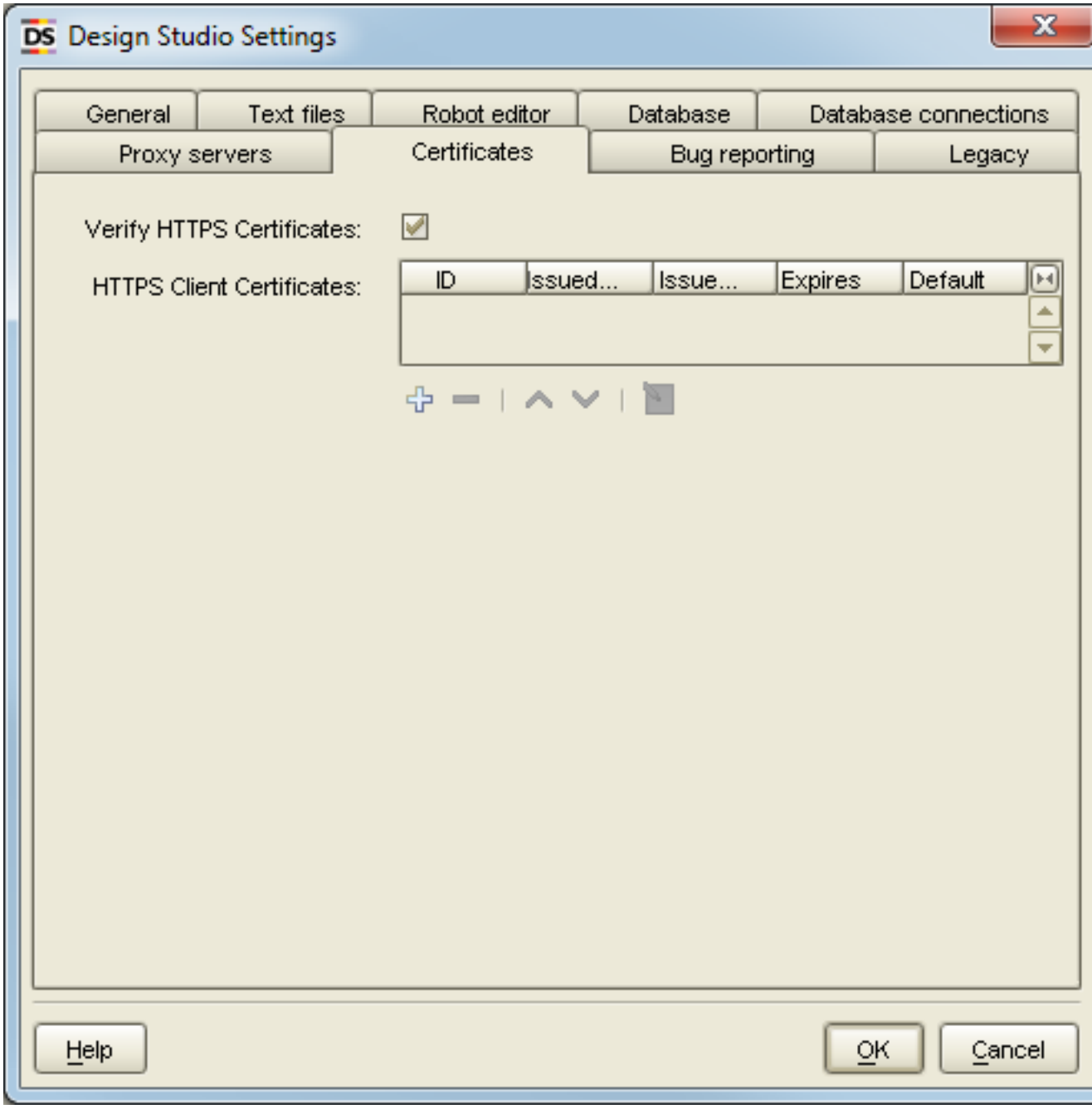
Where *proxyName* is a name chosen to identify that particular proxy server. Each proxy server must have its own unique *proxyName*.

When multiple proxy servers are specified, a new proxy server will be chosen every time a robot is run.

Note that you can also specify a proxy server for an individual robot. This is done when you configure the robot in the Robot Configuration Window in Design Studio. Such a proxy server will override the proxy servers specified here. See the Design Studio User's Guide for more information. Furthermore, the proxy server can be changed during robot execution with the Change Proxy [../ref/robomaker/reference/stepaction/ChangeProxyStepAction.html] action.

Certificates

A robot may need to verify the identity of a web server that it accesses (via HTTPS). Such a verification is routinely (and invisibly) done by ordinary browsers in order to detect phishing attacks. However, the verification is often not necessary when robots collect information, because the robots only access the web sites that they have specifically been written for. Thus the verification it is not enabled by default.



Certificates in Design Studio

Verification is done in the same way a browser does it: The web server's certificate is checked based on an installed set of trusted HTTPS certificates similar to those you can configure in a browser.

Table 15. Options descriptions

Option	Description
Verify HTTPS Certificates	When a robot accesses a web site over HTTPS, it will verify the site's certificate provided that this option is checked. Verification is done based two sets of trusted certificates: The set of root certificates and an additional set of server certificates.

Option	Description
HTTPS Client Certificates	A list of client certificates that the robots can use. Certificates can be added/removed by using the buttons underneath the list.

Note that root certificates are installed with Design Studio just as root certificates are installed with your browser. They are found in the `Certificates/Root` folder in the application data folder.

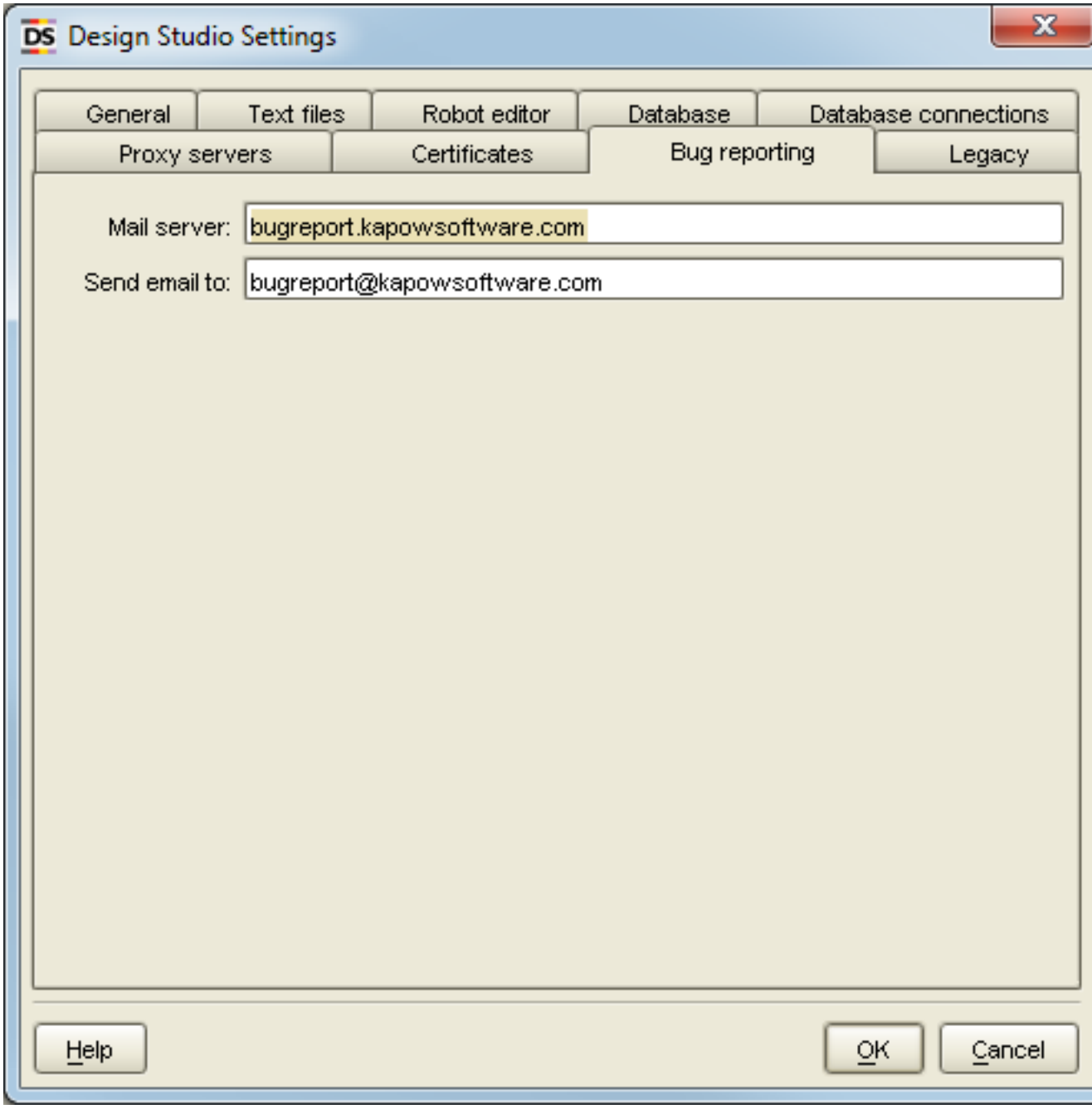
Some HTTPS sites may use certificate authorities that are not included by default. In this case, you need to install the appropriate certificates for Design Studio to load from these sites. Most often, these would be installed in the `Certificates/Server` folder in the application data folder.

To be precise, it does not matter - for the purpose of handling HTTPS sites - whether you add certificates to the set of root certificates or to the set of server certificates.

To install a certificate, you need to obtain the certificate as a PKCS#7 certificate chain, as a Netscape certificate chain, or as a DER-encoded certificate. You install the certificate by copying it to either of the two folders mentioned above. The name of the file containing the certificate does not matter.

Bug Reporting

Design Studio has a built-in mechanism for handling internal errors that occur during its operation. When an internal error occurs in Design Studio, the user will be able to send a bug report with details about the error. The user can also send a bug report on his own initiative, by selecting the Report Bug action from the Help menu.



Bug reporting in Design Studio

Users are generally encouraged not to change the default settings in this tab unless absolutely necessary.

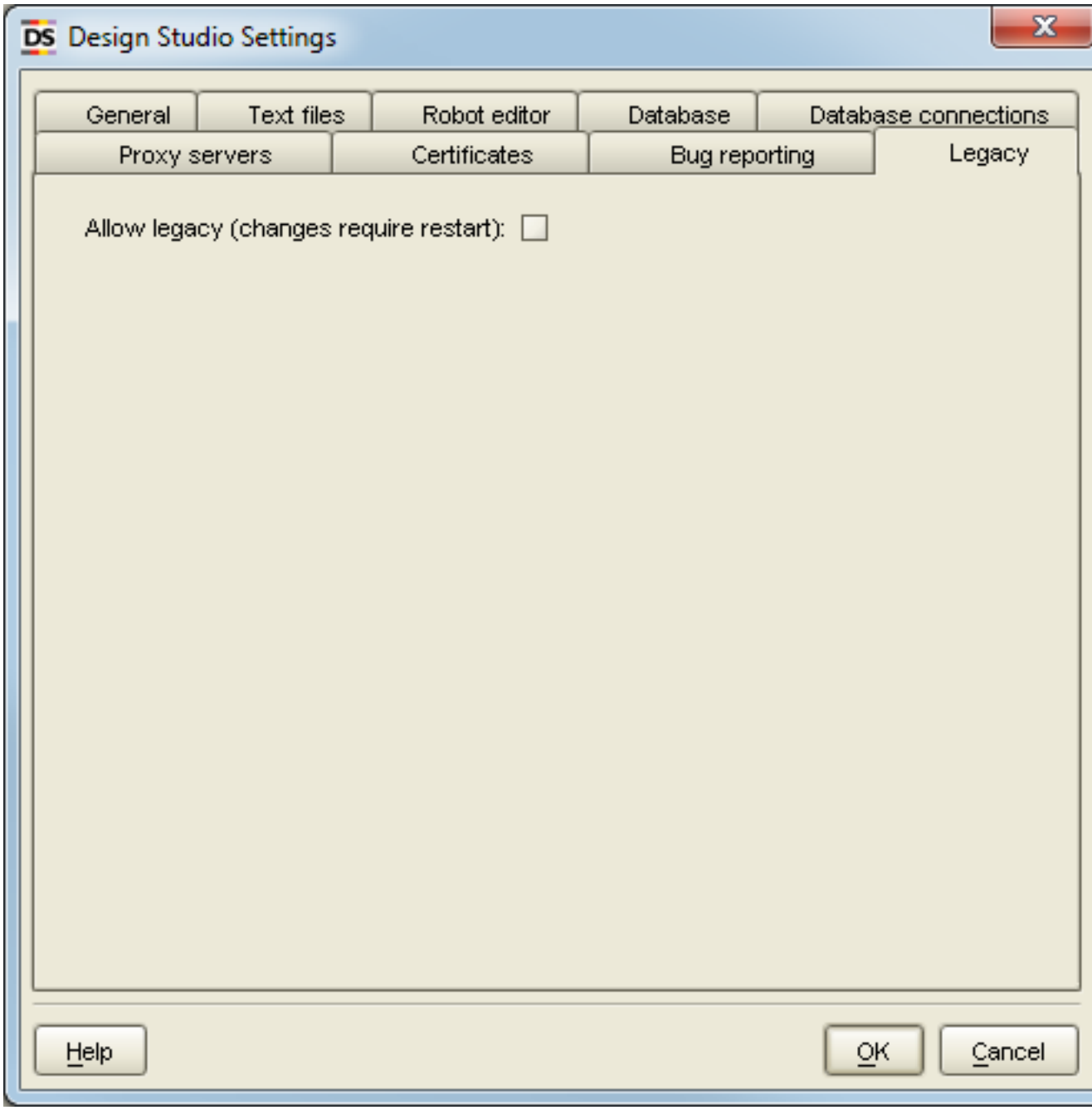
Table 16. Options descriptions

Option	Description
Mail server	Specifies the mail server to use when sending bug reports.

Option	Description
Send email to	The email address to which bug reports should be send.

Legacy

The Legacy tab lets the user alter settings regarding legacy features of Design Studio.



Legacy settings in Design Studio

Legacy features should only be enabled by users who have uses them in earlier version of Design Studio and who rely on them for successful continued use of Design Studio.

Table 17. Options descriptions

Option	Description
Show Legacy Properties	Specifies whether to display legacy features like environments (should only be enabled for users who used these features in version of Design Studio prior to version 7.2). Note that changes to this option requires restart of Design Studio.

Management Console User's Guide

The Management Console is a web-based interface providing point and click monitoring and management of the Kapow Katalyst platform servers.

In the Management Console you can

- Enable collaboration and sharing using the repository.
- Manage user roles and permissions, and centrally administer the solution (Enterprise Edition).
- Schedule robots from the repository.
- Browse extracted data and export to MS Excel.
- Access detailed logs of production results and errors.
- Monitor RoboServer health and resource usage in a graphical dashboard.
- Configure clusters of multiple RoboServers.
- Deploy robots from Design Studio to the repository.
- Run Kapplets.

This User's Guide describes the concepts used in the Management Console as well as the user interface. Before going into this amount of detail, you may want to take the Management Console Beginner's Tutorial.

Management Console Structure Overview

The Management Console is divided into 5 functional areas.

Dashboard: The Dashboard gives you a quick overview of the Management Console. The information is presented through portlets, which shows the health of your RoboServers, Schedules, and Robots.

Kapplets: A Kapplet is a web-based user interface that allows you to run a robot without any programming.

Schedules: A schedule is a plan for running one or more robots, typically at pre-planned points in time and in a repeating fashion. A schedule does not run robots itself; it merely provides the plan for when the robots should be run (which is done by passing them to the configured servers).

Repository: Robots, type definitions and resources can be uploaded from Design Studio to the Management Console repository or can be uploaded manually through the web interface of the Management Console. Uploaded robots can be executed as part of a Schedule or through client code that executes robots using the Kapow Java or C# APIs. You can also use the APIs to programmatically query or update the repository.

Data: The Data View allows you to see the data your robots have stored in databases. The data view also allows you to export this data to Excel or XML.

Logs: Here you can view the execution history of your schedules. If database logging is enabled, you can also view the RoboServer logs which contains details of every robot execution.

Admin: In the Admin section you can configure settings for the Management Console. It also enables you to manage the clusters of RoboServers and their settings, as well as manage projects and permissions. This is also where you configure the license and create/restore backups.

Note: The Management Console is available in two editions: The SMB and the Enterprise edition. Some features, such as High Availability and user management are only available in the Enterprise edition. The number of schedules and robots is limited in the SMB edition.

Note: In the Enterprise edition, the Admin tab will only be accessible to administrators.

Starting the Management Console

The Management Console component is an optional part of a RoboServer. You start the Management Console by starting a RoboServer and instructing it to function as a Management Console, it may still also run the RoboServer functionality.

A RoboServer with Management Console functionality may be started with the "Start Management Console" item in the Start menu (on Windows). On Linux and Unix variants, you can use the command line:

```
RoboServer -MC
```

This will start a RoboServer as Management Console only, and thus not capable of running any robots. The Management Console's web interface is accessible by connecting to the port configured in Settings, see Configuring the Embedded Management Console). If you start the Management Console this way, it cannot be restarted or shut down via Control Center or the ShutDownRoboServer program.

The Management Console can also be started using the following command:

```
RoboServer -p 50000 -MC
```

This will a start RoboServer listening on a socket at port 50000, and providing Management Console functionality via a web interface on a configured port (the port is configured in Settings, see Configuring the Embedded Management Console).

Regardless of how the Management Console is started, a license key must be entered into the web interface before it can function as a license server for RoboServer and Design Studio instances.

For production scenarios, we recommend that you start one RoboServer with the Management Console but not add that RoboServer to a cluster. That way you prevent that Management Console is starved because the RoboServer is using all the resources. After setting up the RoboServer running the Management Console, you can start as many RoboServers as you like using simply the parameter `-p 50000`. These are the only RoboServers that should be added to the RoboServer Clusters on the Clusters tab.

Starting RoboServer gives more information on how to start RoboServer.

In the Enterprise edition, the Management Console may also be installed as a standalone web application. Deployment in standalone mode enables additional features, such as project-based permissions.

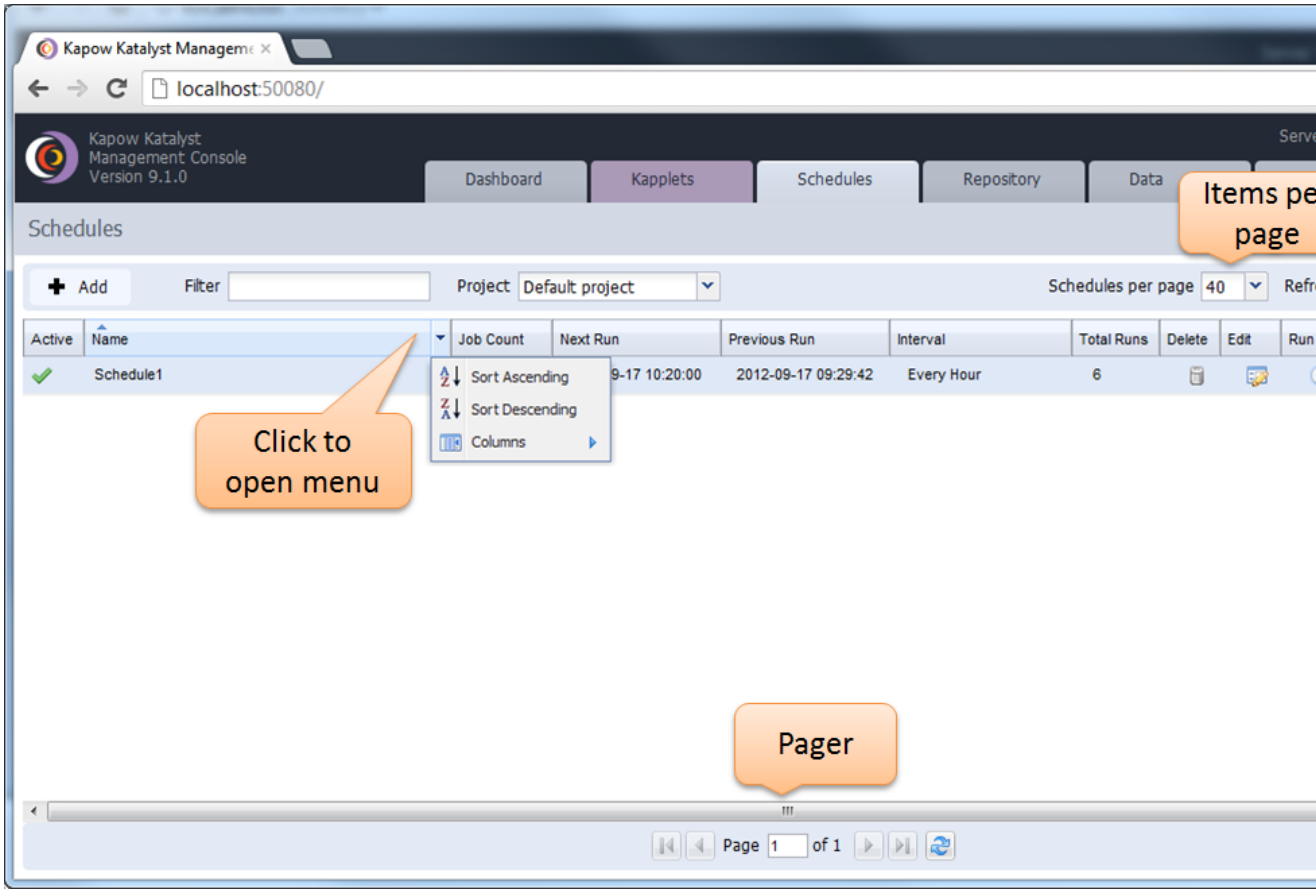
Management Console User Interface

The Management Console has a web-based user interface, which makes it easy to access it from any machine on the same network. If it is installed on the machine named `hostname` you simply need to point your browser to

http://*hostname*:50080 [http://hostname:50080]

The port number `50080` is configurable as described in Configuring the Embedded Management Console.

Navigating the Management Console's user interface is mainly done through the tabs at the top of the window as seen below.



Management Console Main Window

Most subsections of the Management Console displays items of a certain type in a table. When there are many items to display, they are divided into "pages". In the above image, you will notice the setting "Schedules per page", which you can use to select how many items (schedules in this case) to display at the same time. Please note that this number does not adapt automatically to the height of the browser window. Thus empty space below the table does not always mean that you are seeing the last items — it may also mean that the "per page" setting is too low compared to the window height.

At the bottom of the table you can navigate between the individual "pages" of display, thus moving upwards and downwards within the same table. The button will refresh the display.

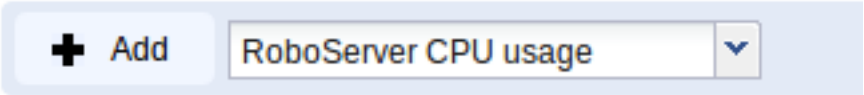
The tables support sorting. Click any column heading to sort by that column. Click the same column heading again to reverse the sorting order. Sorting is performed on the whole table, not on each "page"

individually. Thus if you have more than one "page", you may see a completely different set of rows if you change the sorting order.

The contents of each of the tabs will be explained in the following sections.

Dashboard

The Dashboard contains a number of portlets which display the status of various areas of your system. When you first access the dashboard it will contain 4 portlets; you may add additional portlets to the dashboard by selecting them from the drop down, and pressing the add button



Adding Portlets to Dashboard

If you have a large screen, you may benefit from selecting to display the portlets in 3 columns rather than

2. Each portlet contains a toolbar with 4 iconic buttons  . The first button minimizes the portlet, the second refreshes the portlets data, the third displays info about the portlet, the fourth closes the portlet (it may be added again). Whenever you add, remove, or rearrange any of the portlets on the dashboard, the layout is stored in a cookie inside your browser so that the next time you visit the dashboard, your previous layout will be restored.

The dashboard contains the following portlets:

Most Errors, by Robot - last 100 runs	The graph shows the 100 latest runs for the 10 robots with the most errors (relative to extracted values). You can click a point on the graph, and the Management Console will switch to the log view, and display the run information for this particular run. Notice that the Y-axis is logarithmic. For this graph to work you must enable database logging on at least one cluster.
Summary, last 24 hours	Displays summary for each project in the Management Console. The view allows you to see the 20 most recent execution of each schedule. The health bar gives a quick overview of the status of each Project and schedule. The health bar is colored red, orange, yellow or green. Each color denotes the status of the schedule execution: **Red** — Indicates that at least one schedule error occurred. A schedule error is any *non-robot* error that occurred during the execution of the robot. **Orange** — Indicates that at least one of the robots that executed in this schedule gave an error (And that there were no schedule errors). **Yellow** — Indicates that the schedule was ignored because it was already executing. **Green** — Schedule executed successfully, no schedule errors or robot errors. The size of each color on the health bar is relative to the number or schedule runs with the given status.
RoboServer memory usage	Displays the memory usage over time (over the last 55 hours) for each RoboServer listed on the Clusters Tab

RoboServer CPU usage	Displays the CPU usage of the RoboServer process. If you have multiple RoboServers there will be a graph for each one.
RoboServer KCU usage	Displays the KCU usage of each of the RoboServers.
RoboServer Wait Time	The time a RoboServer has waited for KCU to become available. If there is any wait time, the performance of you robots can be increased by acquiring a license with more KCUs.
Total bytes loaded	Displays the total bytes loaded (over the last 55 hours) for each RoboServer listed on the Clusters Tab. The graph resets at midnight.
Total executed robots	Displays the number of executed robots (over the last 55 hours) on each RoboServer listed on the Clusters Tab. The graph resets at midnight.
Concurrent robots	Displays the number of concurrent robots for each RoboServer listed on the Clusters Tab. The graph resets at midnight.
Total HTTP requests	Displays the total number of HTTP requests loaded by each RoboServer listed on the Clusters Tab. The graph resets at midnight.
Rejected clip sessions	Displays the number of times each RoboServer refused to create a clip session because it already has `-maxClippingSessions` active sessions. Graph resets at midnight. This graph is only important when running clipping robots. If all servers reject creation of clip sessions, no new clips can start, and the `-maxClippingSessions` command line parameter should probably be increased (if there is enough memory available on the RoboServer). See Starting RoboServer for RoboServer parameters
Restarts of pruned clip sessions	Displays the number of times a clip session was destroyed before the user was done with the clip. Clip session may be destroyed due to timeouts (configured in the robot) or if a RoboServer is restarted. When a clip session is restarted, the user will see the first page of the clip regardless of his prior location. If you see a lot of restarted sessions, you should consider increasing the timeout in the robot, and maybe also the -maxClippingSessions, see Starting RoboServer. Graph resets at midnight.

Kapplets

The Kapplet user's guide is found here.

Schedules

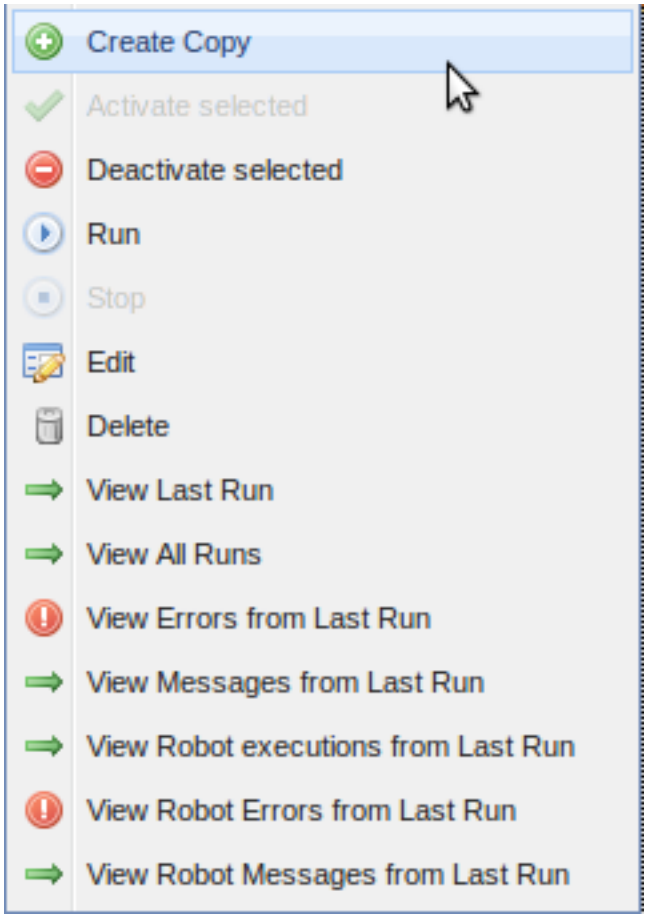
The Schedules subsection enables you to manage the schedules on this Management Console. A schedule denotes a selection of robots and plans for running them. Running the schedule means running the selected robots (in parallel or sequence) optionally executing pre and post run script. The following information is displayed for each schedule in the list of schedules. The information is presented in columns, some of the columns are hidden by default, and can be shown by clicking the down arrow on the column header, and selecting the columns.

Table 18. Schedule Information

Column	Description
Active	Whether the schedule will actually be run when planned. You may want to make a schedule inactive for several reasons. Some examples are: • Because the function performed by the schedule currently is not needed. • Because errors have been found in the robots and you don't want the schedule to run before you have fixed these errors. • Because you want to trigger the schedule manually each time it should run. This may be appropriate for some robots and schedules, e.g. for preparation or clean-up tasks.
Name	The name of the schedule.
Project Name	The name of the project that the schedule belongs to (useful when the 'All' project is selected)
Robot count	A combination of total and active robots. If all robots are active it will simply list the number of active robots, if 2 of 3 robots are active it will list `2 (3)`
Total robots (hidden by default)	The total number of robots in the schedule. (To see the robots, edit the schedule as described below.)
Active robots (hidden by default)	The number of active robots in the schedule. (To see the robots, edit the schedule as described below.)
Next Run	The time when the schedule is planned to be run next.
Previous Run	The time when the schedule was last run.
Interval	The planned interval between two consecutive runs of the schedule.
Total Runs	How many times the schedule has been run.
Created By (hidden by default)	The name of the user that created this schedule.
Modified By (hidden by default)	The name of the user that last modified this schedule.
Object DB (hidden by default)	(The Object DB is a legacy feature, and used with collection robots prior to version 7.2.) The name of the database where to store values extracted by this schedule's robots. When not empty, a Database Storage Environment [../ref/roborunner/environment/DatabaseStorageExecutionEnvironment.html] will be passed along with the robots to RoboServer.
RoboManager DB (hidden by default)	(The RoboManager DB is a legacy feature, and used with collection robots prior to version 7.2.) The name of the database where to store log information data about robots run via this schedule. When not empty, a Database Robot Info Environment [../ref/roborunner/environment/DatabaseRobotInfoExecutionEnvironment.html] and a Database Message Environment [../ref/roborunner/environment/DatabaseMessageExecutionEnvironment.html] will be passed along with the robots to RoboServer.
	The cluster that the schedule executes on.

Column	Description
Cluster (hidden by default)	
Delete	Click this button to delete the schedule.
Edit	Click this button to edit the schedule. This is also useful for viewing more details on a specific schedule. You can also edit a schedule by double clicking the row.
Run/Stop	Click to manually run the schedule. This is especially useful for inactive schedules. If the schedule is already running, it will be stopped. That is, all its running robots will be stopped as quickly as possible. The schedule will be seen as "running" until all its robots have been stopped.
Errors	The number of schedule errors during the last run of the schedule. Schedule errors do **not** include robot errors. Schedule errors are errors that prevent the schedule from running, such as a deleted cluster or errors in pre-, post-processing.
Robot Errors	The number of robot errors that occurred in robots run by this schedule.

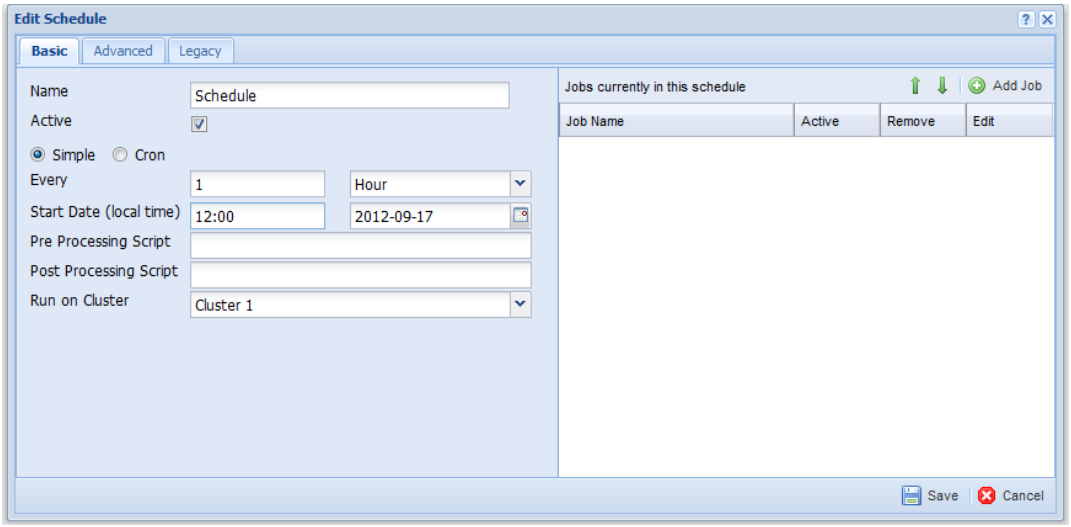
You can right click a schedule to bring up the following popup menu



Schedule context menu (right click)

Clicking the **Add** button in the upper left corner opens a dialog for creating a new schedule. If the current project selection is 'All', when clicking the add button, you have to select an actual project before you can add a new schedule.

Clicking the button in the "Edit" column or double-clicking anywhere in the row for an existing schedule brings up the edit dialog, which is useful for changing the schedule or for viewing all details about it.



Create New Schedule Dialog

The dialog contains three tabs: Basic, Advanced, and Legacy. The basic tab contains everything necessary for setting up a normal schedule. On the advanced tab you can configure runtime constraints. The legacy tab contains a number of obsolete options, which are provided for backwards compatibility when running older collection robots.

The following information can be configured for schedules:

Table 19. "New/Edit Schedule" Dialog Fields

Field	Description
Name	The name of the schedule.
Active	An active schedule is marked with a check mark.
Simple / Cron	Used to select between two different ways of defining the time plan for a schedule.
Every	(Available only for Simple schedules.) The desired time interval between two consecutive runs of the schedule. This is entered as an integral number with a unit, e.g. "1 minute" or "3 hours".
Pattern	(Available only for Cron schedules.) A pattern defining when the schedule should be run. See Cron Schedule [../ref/scheduler/reference/CronSchedule.html] for details of the format.
Pre-processing script	The name of a script that will be run as part of the schedule, before any robots are run. This can be a windows `.cmd` or a Linux `.sh` file. The field must contain the absolute location of the file, like `c:\scripts\truncatedb.cmd`. If you use this you have to remember to copy the scripts when importing projects or restoring backups.

Field	Description
Run on cluster	The name of the cluster to run this schedule on.
Post-processing script	The name of a script that will be run as part of the schedule, after all robots have been run. This can be a windows `.cmd/.bat` or a Linux `.sh` file. The field must contain the absolute location of the file, like `c:\scripts\truncatedb.cmd`. If you use this you have to remember to copy the scripts when importing projects or restoring backups.
Robots (to the right)	See the table below.
Max runtime (Advanced tab)	Select the maximum running time for each robot in the schedule. When a robot has executed for this period of time, the server will stop it, and an error will be logged.
Max extracted values (Advanced tab)	Select the maximum number of values each robot may output. If the robot outputs more than this number of values, the server will stop it, and an error will be logged.
Run robots sequentially (Advanced tab)	If checked, the robots will execute in the order listed on the basic tab.
Use RoboManager Database (Legacy tab)	Check (and specify a database) to collect historical data about robots run via this schedule. See the above table for details.
Use Object Database (Legacy tab)	Check (and specify a database) to store objects extracted by this schedule's robots. See the above table for details.
Use email notification (Legacy tab)	Check to receive an email whenever a robot fails. If several robots in a schedule fail, you will get one email for each robot each time the schedule runs. This will pass an Email Message Environment [../ref/roborunner/environment/EmailMessageExecutionEnvironment.html] along with the robots to the RoboServer. Email notification works only if you configure an SMTP server in the Options Tab and enter the desired email addresses in the following field.
Email addresses (Legacy tab)	A comma-separated list of email addresses to which notifications will be sent.

On the right-hand side you see the list of jobs that will run when the schedule triggers.

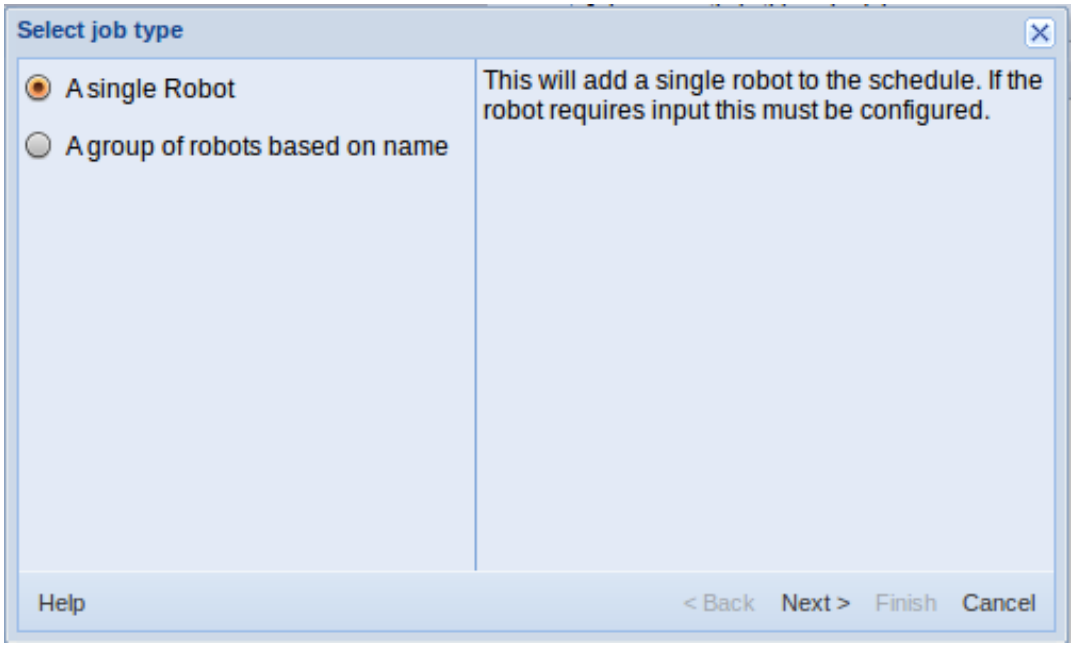
Table 20. Robot Information In a Schedule

Column	Description
Job Name	The display name of the job. This is selected when the job is created.
Active	Whether the job will actually be run when the schedule is run. You may want to make a single job within a schedule inactive for much the same reason you might make an entire schedule inactive.
Remove	Click this button to remove the job from this schedule. This will *not* delete any robots referred to by the jobs.
Edit	Click here to edit the job.

You can add jobs to the schedule by clicking the **Add Job** " button in the upper right-hand corner. This will open a wizard that will take you through the steps of creating a job.

Adding Jobs

When you click **Add Job** , a wizard will open to guide you through the job creation. The first step is shown below



Select a Job Type

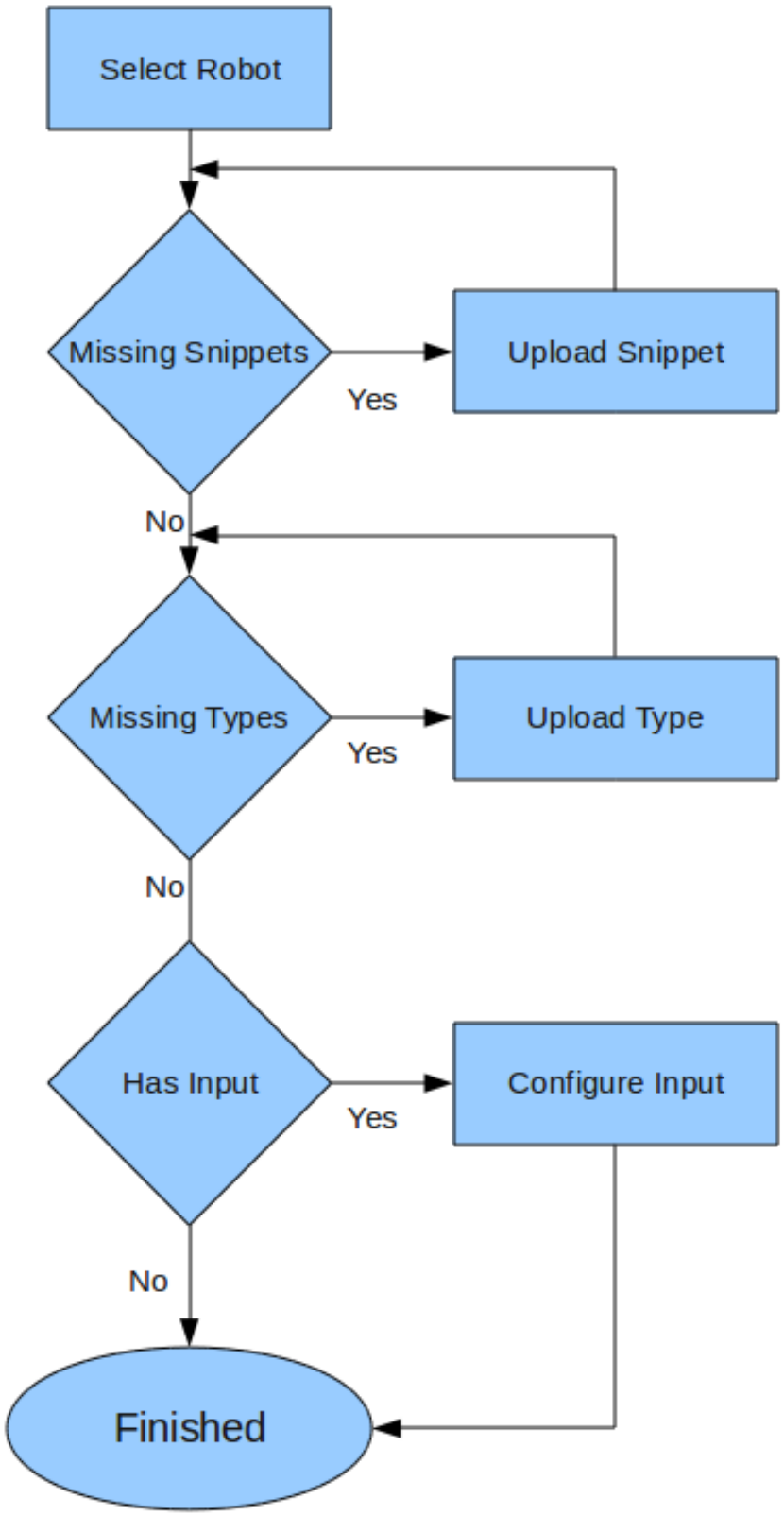
Table 21. Job Types

Job Type	Description
Single Robot	This will add a job that runs a single robot. If you need to pass input to a robot, you must choose this option.
Multiple Robots	This will add a job that runs any number of robots who's name matches a given criteria. The criteria are *Name starts with*, *Name contains*, and *Name matches pattern*.

Once you have selected the job type, the wizard takes you down one of two different tracks.

Adding a single Robot

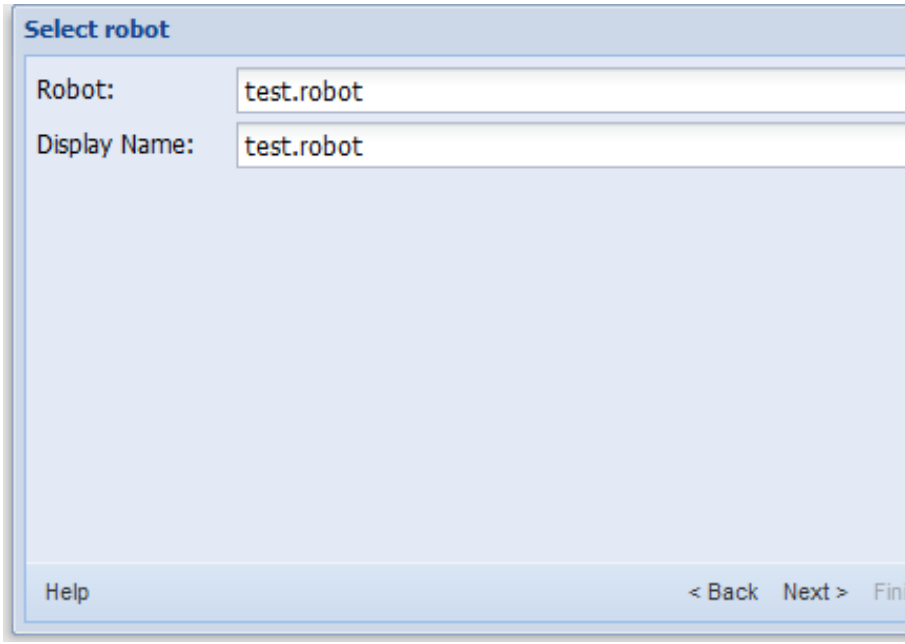
The wizard for adding a single robot contains one to four steps depending on the robot you select. The wizard follows this flow listed below. The four rectangles are the 4 possible steps.



Select single robot flow.

Select Robot: Use the dropdown to select a robot.

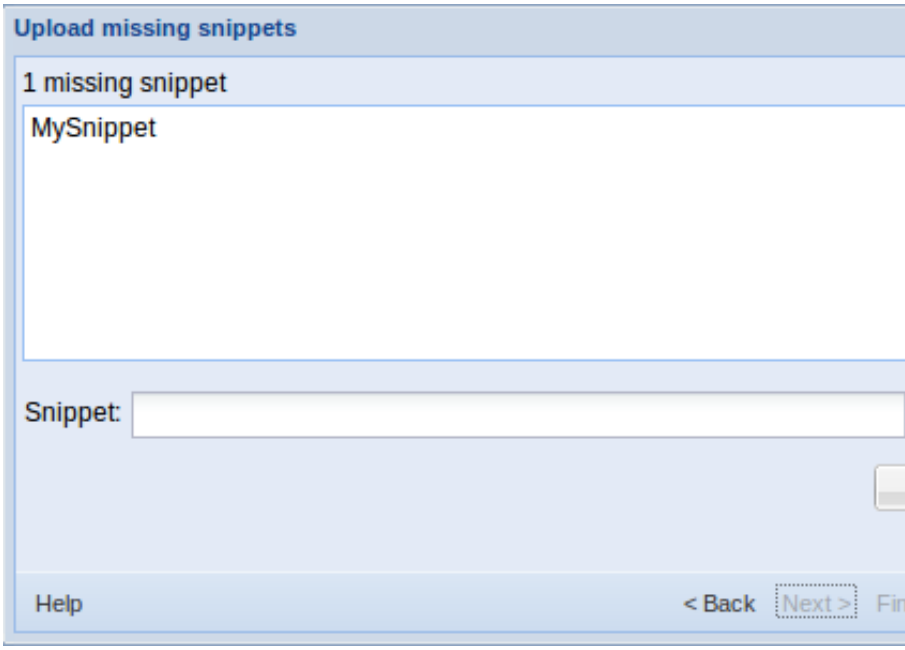
If all snippets and types used by the robot are already uploaded, and the robot doesn't have any input variables, you can click Finish, otherwise you must click Next.



Select single robot.

Upload missing snippets:

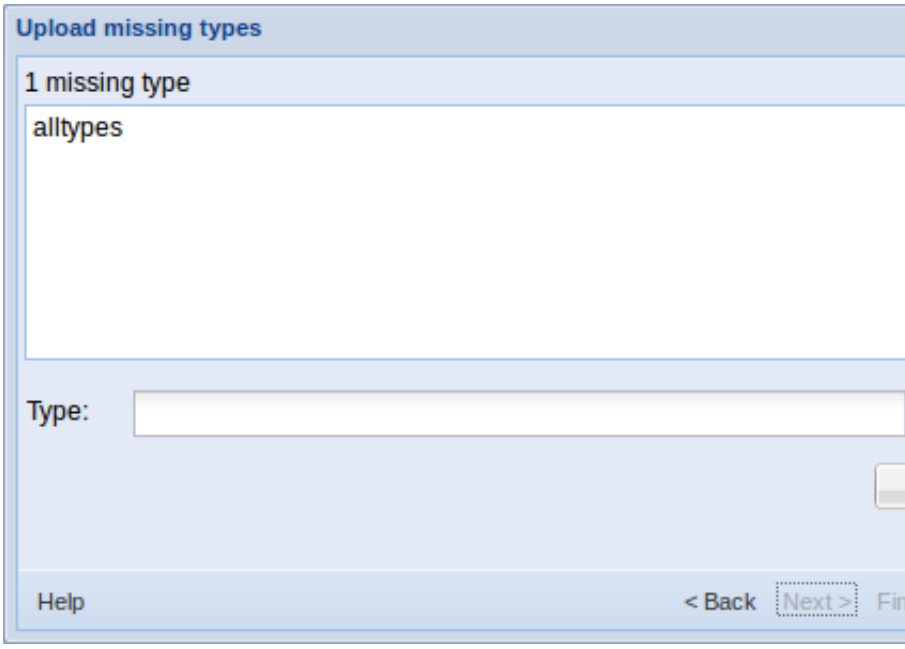
If the robot you have selected uses any snippets which have not been uploaded to the Management Console's repository, you will have to upload them now.



Upload missing snippets

Upload missing Types:

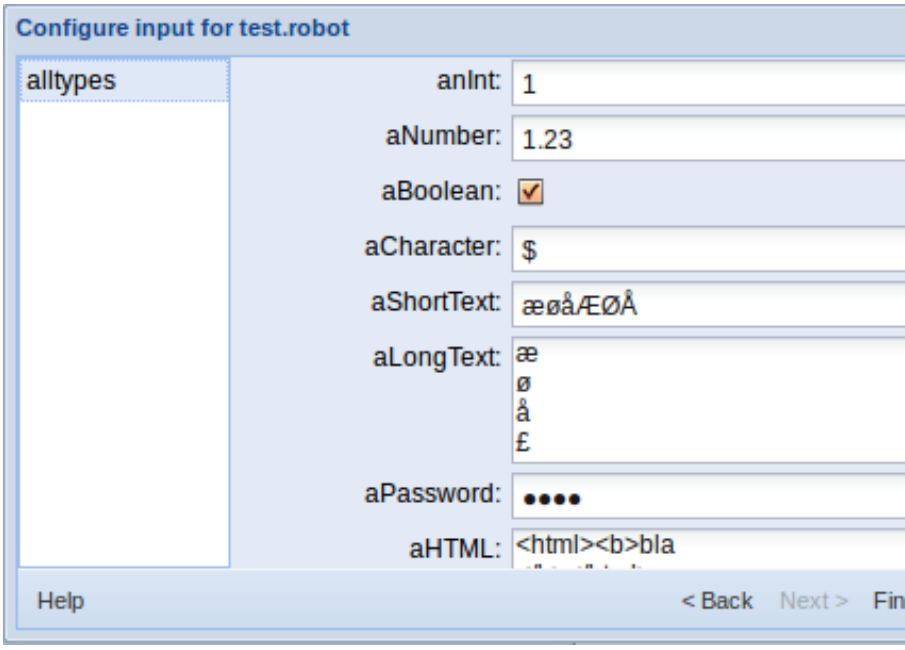
If the robot you have selected uses any types which have not been uploaded to the Management Console's repository, you will have to upload them now.



Upload missing types

Configure Input:

Here you configure the input that should be given to the robot, when it runs as part of this schedule. If an attribute is of a binary type, you can use the drop down to select a resource which is already uploaded, or click the Upload button and upload one. If an attribute is required, it will be underlined with red. You can't finish the wizard if any of the required fields are missing.



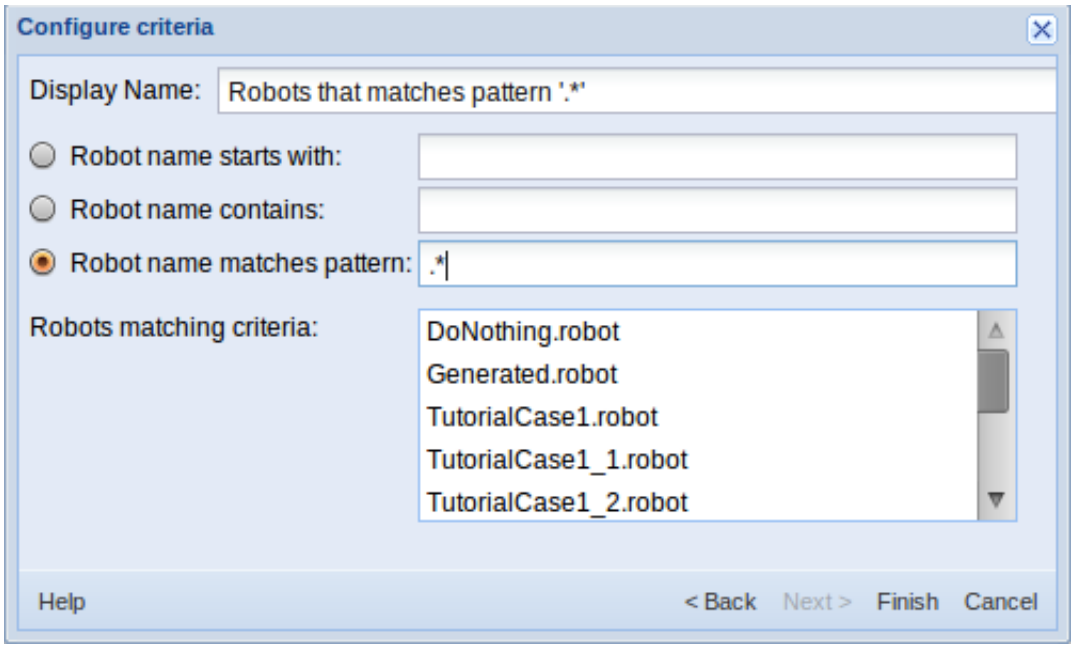
Configure input

Adding a group of robots based on name

You can select to add a single job that will run all robots who's name matches a given criteria. The criteria is evaluated when the schedule is run, so any robots uploaded after the job is created will be included if they match the configured criteria.

Because the criteria is evaluated at runtime, any errors such as missing snippets/types will be logged as errors in the Schedule Run log. The same is true if a robot that uses input variables is run because it matches the criteria.

The wizard for creating a group based on a name criteria, contains a single step.



Create a job that selects robot based on their name.

Use the radio buttons to select the criteria type, or start typing in one of the text fields. The list at the bottom will display the robot(s) matching the selected criteria. As long as you don't edit the Display Name field, it will change depending on your criteria selection, but once you start editing it, the automatic naming is disabled. You may click finish even if no robots match the configured criteria, since you can later upload a robot that will match.

If you have many robots it may take some time to refresh the list of robots matching the selected criteria.

Note: When this job type is added to a schedule configured to run its jobs sequentially, you can't control the order within the group of robots matching the criteria (but they will be execute sequentially).

The **Help** button in the upper right-hand corner of the Schedules tab will open the Management Console User's Guide in your browser.

Alternate schedule creation

It is also possible to create a schedule when located on the Robots Tab. This is done by selecting any number of robots, right-clicking and choosing *Create Schedule* from the context menu. This will open the New Schedule Dialog with the robot(s) already added.

Repository

The Management Console keeps a repository of robots, types, snippets, resources and OAuth credentials. This section helps you managing this repository.

Robots

This subsection help you manage the robots in the repository on a per project basis. In order for robots to be able to run in a schedule, they have to be uploaded to the repository. When the robot is uploaded, it is *copied* into the Repository. Thus, if changes are later made to the robot in Design Studio, it needs to be uploaded again (this can easily be done from within Design Studio). Doing so will not remove the robot from schedules with which it is already associated. Rather, these schedules will use the new version of the robot the next time they run.

Each robot belongs to a project. At the top of the Robots tab, you can select the project for which robots are shown.

Within a given project, you cannot have two different robots with the same name in the Repository. They will be considered the same robot, and the one you upload last will overwrite the previous one. Two different projects may contain robots by the same name, though.

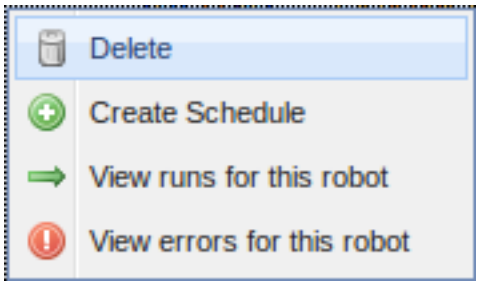
The robots are displayed in a table, with a default of 40 robots per page. The information is structured in columns

Table 22. Robot Information

Column	Description
Name	The name of the robot. If the robot uses a type or a snippet which is not present in the repository, the name will be marked in red
Project Name	The name of the project that the robot belongs to (useful when viewing All projects)
Version	The Kapow Katalyst version last used when editing the robot.
Size	The size of the robot in bytes.
Schedules	The names of the schedules that will run the robot.
Delete	Click this button to delete the robot from the Repository. The robot is automatically removed from any schedules that used to run it. If you don't have a copy of the robot in the file system, it is irrevocably lost.
Input Types	Types used in input variables in the robot. In order to execute the robot, .type files corresponding to each of these types must be present
Returned Types	Types of values returned by the robot. When executing the robot via the API, it may return values of these types. In order to execute the robot, .type files corresponding to each of these types must be present.
Stored Types	Types of values stored in a database by the robot. In order to execute the robot, .type files corresponding to each of these types must be present.
Snippets used	Names of the snippets used by this robot. If a Robot uses snippet A and snippet A uses snippet B, only snippet A will be listed here.
Created By (hidden by default)	The user name of the user who first uploaded the robot. This feature is only available when running a stand-alone Management Console.
Modified By (hidden by default)	The user name of the user who last modified the robot. This feature is only available when running a stand-alone Kapow Katalyst.

Column	Description
Last Modified	The date of the most recent modification of the robot.
Run Now	Click this button to start immediate execution of the robot on RoboServer. This feature is not available for robots that take input.
API	Click this button to see example Java or C# code for executing the robot on RoboServer.
REST	This will open a window that allows you to invoke the robot as a REST service.
Download	Click this button to download a copy of the robot from the Repository and save it to the file system.

Right clicking a robot brings up the following popup menu



The Delete and Create Schedule options are available when multiple robots are selected. If you have multiple robots selected and create a schedule, the New Schedule Dialog will open with all the selected robots added. If any of the robots added this way requires input, you will have to add it later.

Clicking the "**Add Robot**" button in the upper left corner opens a dialog for uploading a new robot. If you upload a robot with the same name as an existing one, the existing one will be replaced, with no changes to schedules that run it. If you upload a new robot, it is *not* added to a schedule automatically; you will need to do this yourself.

An alternative way of uploading a robot is to use one of the Upload functions in Design Studio. This works in exactly the same way, except that Design Studio also uploads the necessary types and snippets. If your Management Console Repository contains multiple projects, you will be prompted to choose which project to upload the robot into.

Executing Robots

Once a robot has been uploaded to the Management Console, it can be executed in 4 different ways. Most often you will execute robots as part of a schedule or as a Kapplet, but they may also be executed programmatically either through the Java/.Net APIs or as RESTful services.

API

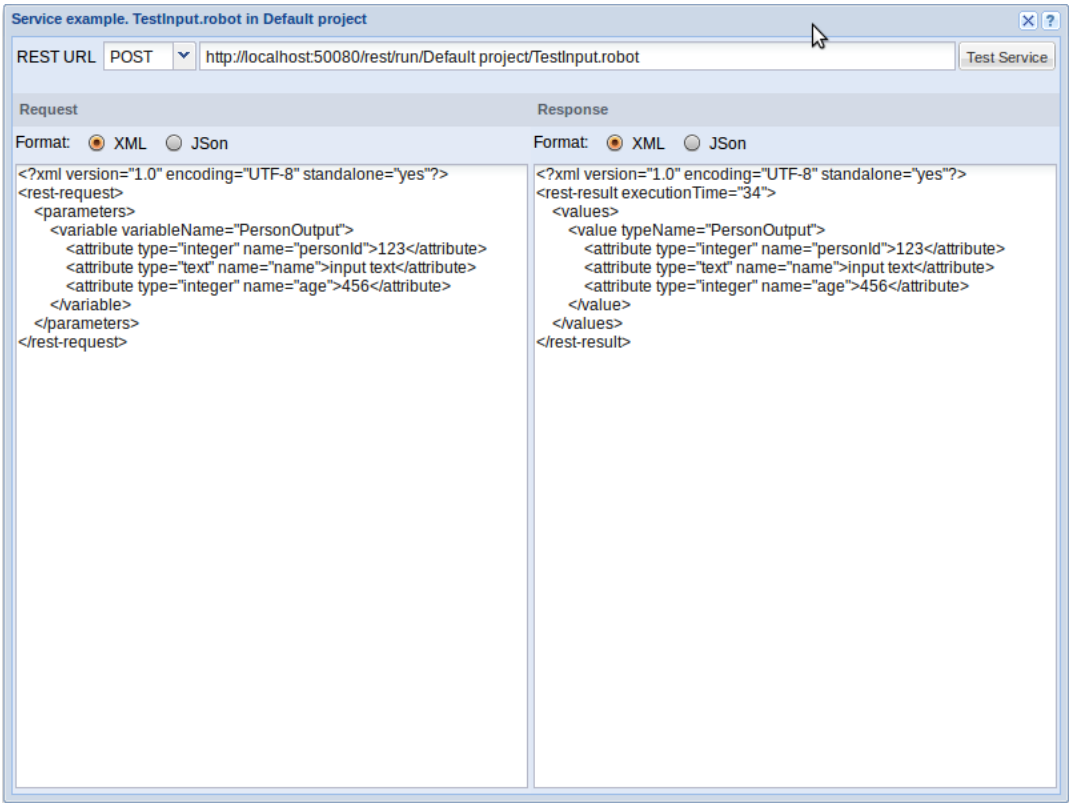
On the Robots Tab you will find a column named API. Clicking this column will bring up the code generation window. This will generate template code for either Java or .Net.

Before you start using the API to execute robots, it is recommended that you read the relevant Programmer's Guide to understand how the API works: Java Programmer's Guide, .NET Programmer's Guide.

REST

As a new feature in 8.3, robots may be executed as RESTful services. This allow you to invoke a robot from any programming language, or directly from a browser using JavaScript.

On the Robots Tab you will find a column named REST. Clicking in this column will bring up a window that allows you to test your robot as a service.



Service Window

The left-hand side of the service window allows you to construct a request. You then click the `Test Service` button in the upper right corner to execute the robot. The result is then displayed in the right-hand side of the window.

The format buttons allow you to configure the formats of the request and responses while testing, but when you call the service from code, the format is controlled by the **Accept** and **Content-Type** HTTP headers. The Content-Type header specifies the request format, and the Accept header specifies the desired response format.

Robots that require input must be invoked using POST. Robots without input may be invoked using either GET or POST.

REST services are easily invoked from a robot by using the Call REST Web Service action.

If the project or robot name contains any non-ASCII characters, you have to make sure that the URL is encoded properly (UTF-8 URL encoding). This is automatically done in robots, but if the service is called from code, the developer is responsible for encoding the URL.

Note: Robots run as services will stop the first time the robot generates an API exception. This is different from scheduled robots which will continue to run regardless of any API exception generated by the robot. You should think of REST services as something short-lived, like a Google search or translating a sentence.

Each robot that is run as a service uses a request thread. When the Management Console is running embedded in a RoboServer, there is a maximum of 40 request threads. These 40 threads are used for all types of HTTP requests, such as users accessing the Management Console, uploads from Design Studio, and the Repository API. If you need to run a higher number of concurrent REST services, you will need to install a standalone version of the Management Console on Tomcat so that you can control the number of request threads.

Types

This subsection lists the types that have been uploaded to the Management Console Repository. At the top of the types tab, you select the project whose types should be displayed.

When a schedule runs, the robots linked to it are executed on RoboServer. Many robots require types as definitions of either input, output or both. These types must be available in the Repository (in the same project as the robot) or running the robot will fail.

When you upload a type to the Repository, it is *copied* into the Repository. Thus, if changes are made later to the type in Design Studio, it needs to be uploaded again. Since each type name must be unique (within each project), uploading the type again overwrites the previous version.

You may upload multiple types by the same name only if they are placed in separate projects.

The following information is displayed for each type.

Table 23. Type Information

Column	Description
Name	The name of the type.
Size	The size of the type in bytes.
Project Name	Displays the project the type belongs to (useful when viewing All project).
Delete	Click this button to delete the type from the Repository. If you don't have a copy of the type in the file system, it is irrevocably lost.
Last modified	The timestamp for the last change of this type.
Download	Click this button to download a copy of the type from the Repository and save it in the file system.
Created By (not shown by default)	The user name of the user who first uploaded the type. This feature is only available when running a stand-alone Kapow Katalyst using LDAP.
Modified By (not shown by default)	The user name of the user who last modified the type. This feature is only available when running a stand-alone Kapow Katalyst using LDAP.

Clicking the "Add Type" button in the upper left corner opens a dialog for uploading a new type. Note that type may implicitly be uploaded with the aid of Design Studio. This happens when you use Design Studio to upload a robot that uses the type. Because Design Studio knows about the dependencies between robots and type, it always uploads the necessary types along with the robot.

If you click the in Delete column, you will be prompted to confirm the delete. If the type is used by a robot or snippet, the confirmation message will include the usage count. If you delete a type that is used by a robot or snippet, the robot (or robots using the snippet) will no longer be able to execute.

Snippets

This subsection lists the snippets that have been uploaded to the Management Console Repository. At the top of the snippets tab, you select the project whose snippets should be displayed.

When a schedule runs, the robots linked to it are executed on RoboServer. Some robots use snippets , these snippets must be available in the Repository (in the same project as the robot) or running the robot will fail.

When you upload a snippet to the Repository, it is *copied* into the Repository. Thus, if changes are later made to the snippet in Design Studio, it needs to be uploaded again. Since each snippet name must be unique (within each project), uploading the snippet again overwrites the previous version.

You may upload multiple snippets by the same name only if they are placed in separate projects.

The following information is displayed for each snippet.

Table 24. Snippet Information

Column	Description
Name	The name of the snippet.
Project Name	Displays the project the snippet belongs to (useful when viewing All project).
Input Types	Types used in input variables in the snippet. In order to use this snippet in a robot, the .type files corresponding to each of these types must be present
Returned Types	Types of values returned by the snippet. In order to use this snippet in a robot the .type files corresponding to each of these types must be present.
Stored Types	Types of values stored in a database by the snippet. In order to use this snippet in a robot the .type files corresponding to each of these types must be present.
Snippets used	Name of the snippets used by this snippet. If a Snippet uses snippet A and snippet A uses snippet B, only snippet A will be listed here.
Size	The size of the snippet in bytes.
Delete	Click this button to delete the snippet from the Repository. If you don't have a copy of the snippet in the file system, it is irrevocably lost.
Created By (hidden by default)	The user name of the user who first uploaded the snippet. This feature is only available when running a stand-alone Kapow Katalyst using LDAP.
Modified By (hidden by default)	The user name of the user who last modified the snippet. This feature is only available when running a stand-alone Kapow Katalyst using LDAP.
Last modified	The timestamp for the last change of this snippet.
Download	Click this button to get a copy of the snippet out of the Repository and save it in the file system.

Clicking the "Add Snippet" button in the upper left corner opens a dialog for uploading a new snippet. Note that snippets may implicitly be uploaded with the aid of Design Studio. This happens when you use Design Studio to upload a robot that uses the snippet. Because Design Studio knows about the dependencies between robots and snippets, it always uploads the necessary snippets along with the robot.

If you click the Delete column, you will be prompted to confirm the delete. If the snippet is used by other snippets or robots, the confirmation message will include the usage count. If you delete a snippet that is used by a robot (or a snippet used by this robot), the robot will no longer be able to execute. Any robot that is missing a required snippet, will be listed with it's name in a red font on the Robots tab.

Resources

This tab shows the resources uploaded to the Management Console. These resources can be used for input to robots which have an input variable with binary attributes.

The following information is displayed for each resource.

Table 25. Resource Information

Column	Description
Name	The name of the resource.
Size	The size of the resource in bytes.

Column	Description
Project Name	The name of the project the resource belongs to (useful when viewing all projects).
Delete	Click this button to delete the resource from the Repository. If you don't have a copy of the resource in the file system, it is irrevocably lost.
Download	Click this button to download a copy of the resource from the Repository and save it to the file system.
Created By (not shown by default)	The user name of the user who first uploaded the resource. This column will only contain relevant information when running a stand-alone enterprise version of Kapow Katalyst that has individual user login.
Modified By (not shown by default)	The user name of the user who last modified the resource. This column will only contain relevant information when running a stand-alone enterprise version of Kapow Katalyst that has individual user login.

Resources can be uploaded by clicking the **Add Resource** button in the upper left corner of the tab, or uploaded in the dialog for configuring input for a robot.

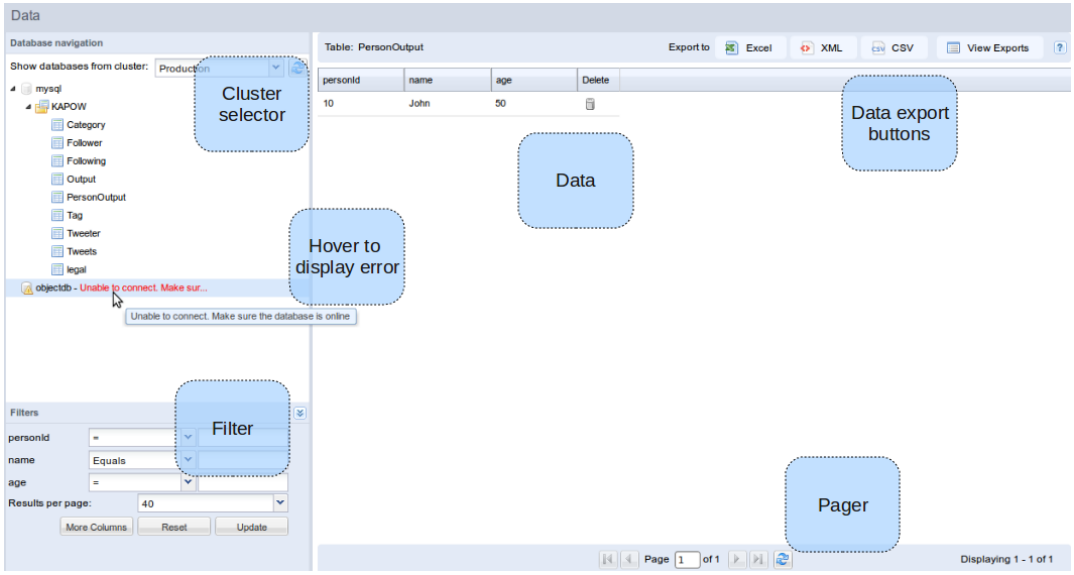
OAuth

This tab contains the OAuth applications and users that have been authenticated using the Management Console. The credentials of the users can be used as input to robots in a schedule, allowing robots to access APIs on behalf of the authenticated user without having access to the username and password.

Please consult the OAuth section of the documentation for more information on how to create and manage robots that access APIs that are protected by OAuth.

Data

The Data view allows you to view and export data extracted by Robots.



Data View

In the upper left-hand corner of the Data View, you can see the database navigation tree along with a cluster selector. You can view data from the databases for each cluster. So to view data from a given cluster,

simply select it, and the databases defined in that cluster's settings will be shown in the database navigation tree. Next to the cluster selector is a refresh button for use when the cluster databases have changed in which case it will re-populate the database navigation tree to show the new information.

When you click a database name, the tree will open, and you will see the various schemas in the database. When you click a schema you will see the Kapow tables in that schema. When you click a table, the contents of that table will be loaded into the data grid on the right. Once the data is loaded, a number of filter will be visible underneath the database navigation tree. The filters work exactly like the filters on the Logging Tab. Binary and longtext attributes are not filterable.

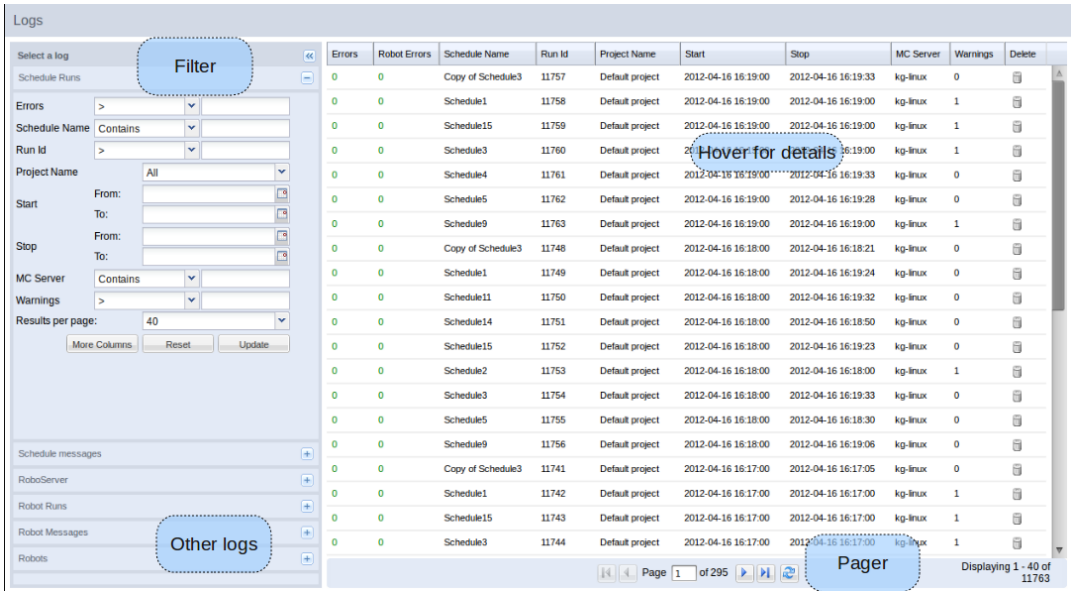
If you click the Delete column, a window will open allowing you to delete one or more rows for the table. Double clicking a row will bring up a window with the content of that row, allowing you to copy the data.

Above the data grid, you see an Export bar, containing 4 buttons. The first 3 buttons allow you to export the table data to either Excel, XML, or CSV format. The fourth button allows you to see previous export, and download them again. The exports are automatically deleted the next time the Management Console is restarted, also the oldest exports are deleted when the number of exports exceeds 100. There is no limit to the number of rows you can export to CSV or XML, but Excel files are limited to 10000 records, to prevent the system from running out of memory.

The list of schemas/catalogs available under each database in the navigation tree, is controlled by the permissions of the user who's credentials are used (in the cluster settings database configuration).

Logs

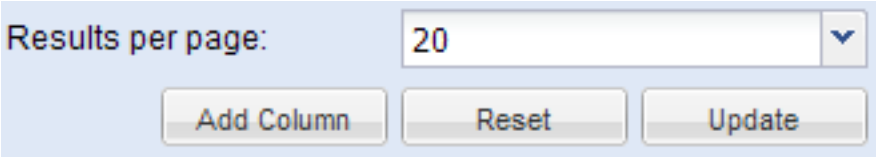
This tab is used to view the logs created the Management Console and by the RoboServers' database logging. The 'Schedule Runs' and 'Schedule Messages' are Management Console logs that report information on schedules. The remaining logs are RoboServer logs containing information on the status of the RoboServers and on robots and robot runs. The Management Console logs are written to the Management Console database and are therefore always available. The RoboServers logs, however, require that a logging database is set up in the Management Console settings (see Settings->Log Database) and that database logging is enabled on the RoboServers via the logging cluster settings.



Log View

You select one of the 6 log views, by clicking the  icon next to the log you desire to view.

The page layout for each of the six logs are identical. When you click the  a number of filters are displayed to the left, and the data is loaded into the grid to the right. Below the filters there is a select box, and 3 buttons



Adding columns to data grid

The Add Column button is used to add and remove columns to the grid to the right. Reset clears any filter configuration entered. Update loads the grid with data based on the configuration of the filters. The select box controls the number of results per page (for the next update). If there are more results than the number selected per page, you can navigate to the next page using the controls under the data grid. You may also hit the Return button in any filter text field to trigger an update.

If you don't delete the log messages, automatic cleanup systems are in place that will make sure the logs do not grow beyond certain sizes. For the RoboServer logs, the RoboServer's automated cleanup system will do so by deleting the oldest messages first. In Settings->RoboServer Log Database you can configure the number of messages required before a cleanup is triggered. The schedule logs cleanup thresholds are defined in Settings->Schedule Logs.

Schedule Runs	Displays execution information for each schedule that executes, such as when the schedule started and when it finished. The context menu allows you to navigate to the individual schedule message, or to the robots that were executed as part of this schedule run. If you click in the Delete column, a window will open allowing you to Delete the Run and messages. When deleting, you always have to delete the messages, but you can keep the run information if you like. You can delete this run and messages, or all runs or messages matching the current filter. Deleting many runs or messages, may take some time.
Schedule Messages	Displays individual message entries for a given schedule run. This allows you to see exactly why a schedule failed to run. **Note:** Robot errors for a given schedule run can be found by selecting "View robots view errors" from the context menu. You can delete one or more records by clicking the Delete column. This will open a window in which you can select to delete only this message, all message matching the current filter, or all RoboServer log messages (the option to delete message matching a filter is disabled if there are more than 500 matching results. This is due to performance).
RoboServer	Displays general messages from RoboServer or the Management Console. You can double click or use the context menu to open the message in a separate window, making it easier to copy the error message. You can delete one or more records by clicking the Delete column. This will open a window in which you can select to delete only this message, all message matching the current filter, or all RoboServer log messages (the option to delete message matching a filter is disabled if there are more than 500 matching results. This is due to performance).

Robots	This is a simple summary view of all robots ever run (for as long as data is available). If you double click a row or use the context menu, you will navigate to all runs for the given robot.
Robot Runs	Displays information for each robot run. This log contains a number of extra fields which are not shown by default, but can be added by clicking the Add Column below the filters. Double clicking a row takes you to all messages logged during this run, this is also available through the context menu. The context menu also allows you to view all runs for the same robot, or view the input this robot was given when this run was executed. If you click the Delete column, a window will open allowing you to Delete the Run and messages. When deleting, you always have to delete the messages, but you can keep the run information if you like. You can delete this run and messages, or all runs or messages matching the current filter. Deleting many runs or messages may take some time.
Robot Messages	Displays individual error messages belonging to a robot run. Double clicking a row brings up a window which allows you to easily copy the error message. Using the context menu, you can navigate to the run this message belongs to. For robot errors, the Location Code column in the data grid contains a link. When you click the link a small .robotDebug file is downloaded. If you open the file with Design Studio, the robot will be loaded, the input from the run will be set, and the robot will navigate to the location of the error, allowing you to quickly debug errors.

Admin

This tab contains a number of sub-tabs that is used to administer the Management Console.

Task View	Displays the robots and other tasks that are being executed by the Management Console.
Clusters	Add or delete RoboServers and clusters, and configure cluster settings.
Projects	Create and delete projects.
Users (Enterprise Only)	View information on the Management Console users.
Settings	Configure various Management Console settings.
Backup	Create and restore backups and import/export projects.
License	Enter license information and view information on the current license as well as Design Studio seats in use.

Task View

This task view shows the scheduled robots, as well as pre- and post-processing tasks that have been started by the Management Console's Scheduler. Its purpose is to give an overview of the current activity across all servers and robots; if you want to know about the outcome of previous robot runs, you should inspect the Logs.

Because of update delays, short-running robots may never (or almost never) show up in this tab.

The following information is displayed for all currently running robots:

Table 26. Run Information

Column	Description
Name	The name of the task that is running.
Schedule	The name of the schedule that the task runs within.
Project Name	The name of the project that the task belongs to.
Status	The task may be in one of the following states: Queued: The task is queued and waiting for execution. A task may be queued for the following reasons No Servers: There are currently no servers in the cluster, or all servers are offline. The task will start to execute when a server is added, or an offline server comes online. No Capacity: All servers are executing at maximum capacity. The task will start to execute when other tasks finish and capacity becomes available, or if additional server are added to the cluster. Waiting for other task: The task may be waiting for another task to finish. Post processing may not be run until all robots have finished, and robots may not be run before pre processing has finished. Also if robots on a schedule are configured for sequential execution, robots will remain queued, until the previous robot completes. Running: The task is currently executing.
Last status change	The time of the last status change.
Stop	Click this button to stop the task even though it has not finished running.

Clusters

This section enables you to manage clusters and RoboServers known to the Management Console. All servers in the list can be monitored using the portlets on the Dashboard. By default, the list contains one

cluster containing one RoboServer, namely the one that also runs the Management Console functionality. In larger setups with multiple RoboServers and clusters, it is recommended that the Management Console is deployed on a standalone web container (if license permits), or on a RoboServer which is not used for running robots.

The following information is displayed for each server.

Table 27. Server Information

Column	Description	
Cluster/Server	For Clusters	The name of the cluster. Will be suffixed by - *SSL* if the cluster uses SSL. If the cluster has unapplied settings this will also be shown here, and the name will be displayed in blue. If the cluster has invalid settings, the name will be displayed in red.
	For RoboServers	The name and port of the RoboServer. If the RoboServer is incorrectly configured the name will be displayed in red. If you hover over the red name, a tooltip will inform you of the error.
Version	Shows the version of the software on the running RoboServer.	
KCU	The number of KCU assigned to this cluster. KCU in a cluster are distributed evenly between online RoboServers in the cluster. To adjust the KCU on a cluster click the Assign KCU button.	
Running Robots	Shows the number of robots currently running on the RoboServer.	
Queued Robots	The number of queued robots on the RoboServer	
Max Robots	The maximum number of concurrent robots on the RoboServer. Can be configured in the Clusters settings	
Max Queue	The maximum number of robots that can be queued on the RoboServer. Can be set in the Clusters settings	
License Type	The license type of the RoboServer, this is either Production or Non-Production.	
Cluster Mode/Server Status	For clusters, shows the Cluster Mode. For RoboServers, shows if the server is Online or Offline.	
Delete	Click this button to delete the cluster/server from the Management Console. This will remove any associated information from the Dashboard portlets.	

Create a Cluster

When you create a cluster you specify the name of the cluster, and the cluster type. If you create a non-production cluster you can assign KCU from the non-production license, and similarly if you create a Production cluster you can assign KCU from the production license. If you select the SSL option, all RoboServers in the cluster must use the SSL RQL service.

After you have created a cluster, you can add RoboServers to the cluster.

Load Distribution and fail over

When a cluster needs to execute a robot, it finds the RoboServer with the highest number of available slots. Available slots are calculated based on how many robots are already running on the RoboServer and how many robots it can run concurrently (the max concurrent robots in the Clusters settings.)

If a RoboServer in a cluster goes offline, the KCU is automatically distributed evenly among the remaining RoboServers.

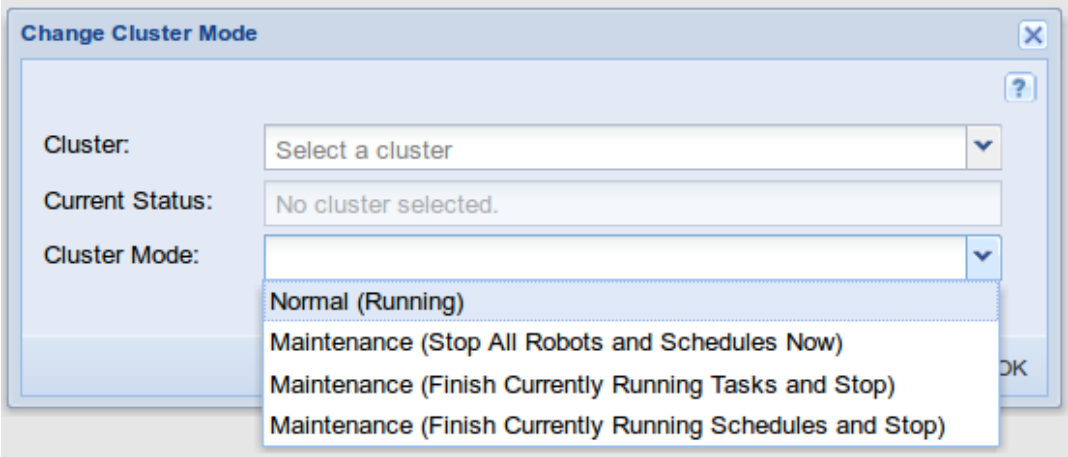
Cluster Mode

A cluster can be in three different modes: Normal, Maintenance or Pending Maintenance. The maintenance mode is used to set clusters into a mode where nothing is run on them. The point of this is to be able to update clusters settings in a way that ensures that all robots in a given schedule use the same settings, and that settings are not changed in the middle of schedule runs. The table below explains the different modes.

Table 28. Cluster Modes

Cluster Mode	Description
Normal	In the Normal cluster mode, a cluster is operating normally, and schedules and individual robots are executed as expected. In this mode, Cluster settings will not be "applied" to the RoboServers in order not to change settings in the middle of schedule runs.
Maintenance	In Maintenance mode, no robots are allowed to run on the cluster (unless initiated from the API, see below). For cluster settings to be applied to the RoboServers, a cluster MUST be in Maintenance mode.
Pending Maintenance	There are 3 ways of setting a cluster into Maintenance mode. These specify what to do with currently running or queued robots and schedules. When setting a cluster into Maintenance mode, one of the 3 ways are chosen and until nothing is running on the cluster (except possibly API robots, see below), it will be in Pending Maintenance mode. The 3 ways of settings a cluster into Maintenance mode are explained below. Stop All Robots and Schedules Now — This will attempt to stop all currently running robots. Any queued robots are de-queued and not run. This is the fastest way of settings a cluster into Maintenance mode. Finish Currently Running Tasks and Stop — This will ensure that all robots that have been started and are currently running will finish before the cluster goes into Maintenance mode. A schedule running several robots where some are queued and some running, will only have it's currently running robots run; any queued robots are de-queued. Finish Currently Running Schedules and Stop — This will make sure that all running and queued robots are finished. This means that any started schedules will finish before the cluster is set into Maintenance mode.

To change cluster mode, click the 'Change Cluster Mode' button above the cluster list. This will open the following dialog:



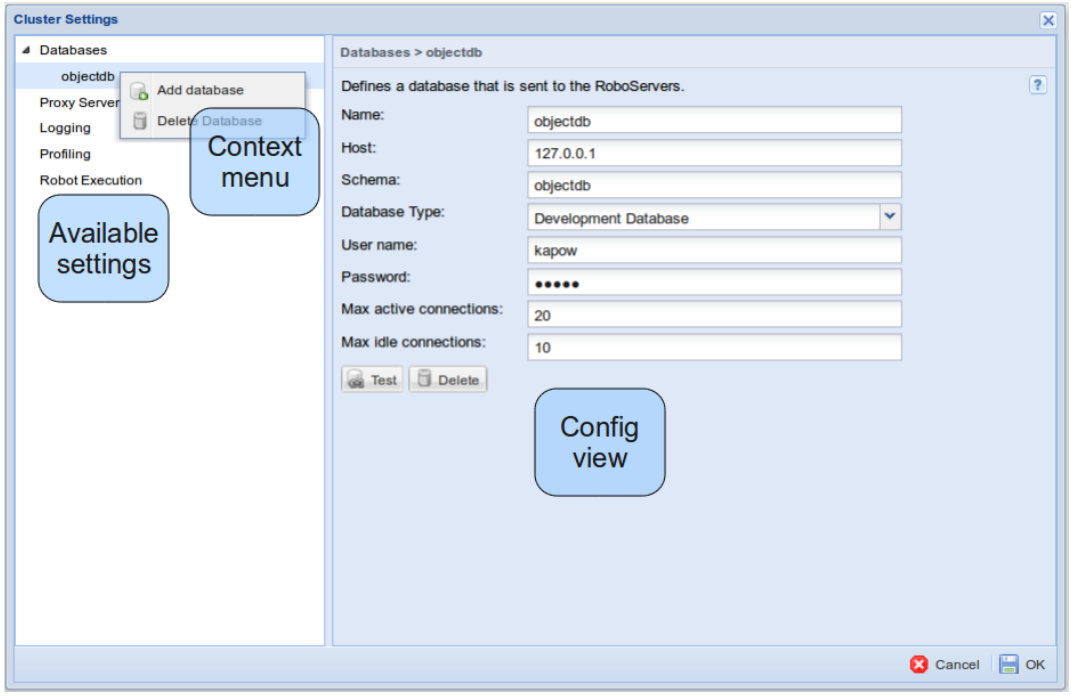
Changing cluster mode

In the dialog, the cluster that should have it's mode changed must be selected, and one of the above described ways of transitioning into maintenance mode must be selected. **Hint:** the cluster mode can also be changed by right-clicking the cluster in the list and selecting the appropriate sub menu-item under 'Change Cluster Mode'.

The cluster modes are, as the name implies, only relevant on a cluster level. As such, they are a way for the Management Console to control when tasks can be executed on clusters. They do not, however, control the individual RoboServers. This means that robots started from the **API** are not attempted stopped when going into Maintenance mode. Therefore, the settings of a RoboServer can be updated while API robots are running. It is guaranteed, though, that a robot will not have its settings updated during its execution. So, for instance, if database are updated while one or more API robots are running, the robots will use the databases that were configured when they started. Next time they are run, they will use the new settings.

Cluster Settings

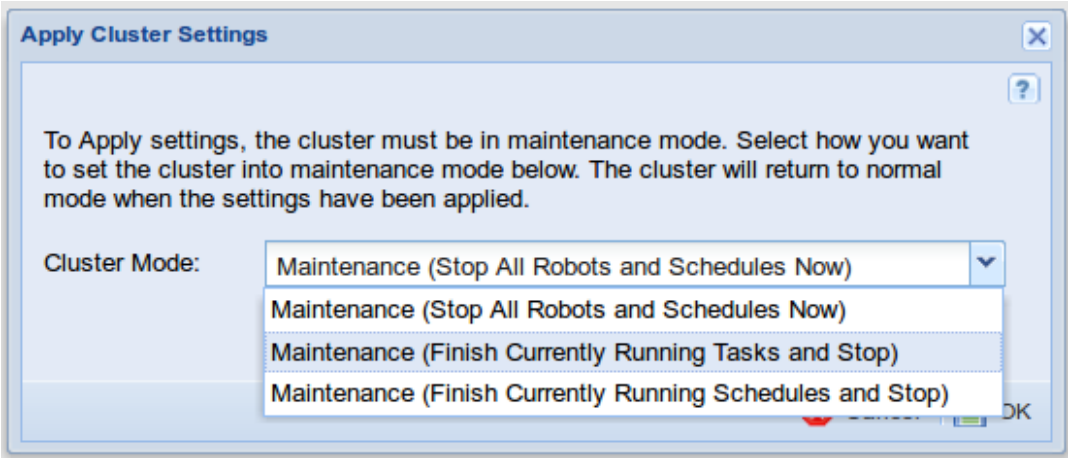
The cluster settings are settings that are sent to each RoboServer in the cluster. Through the cluster settings one can configure, for instance, which databases are available for robots executing on the RoboServers. The cluster settings dialog is opened by right clicking a cluster and selecting the "Cluster Settings..." menu item or by clicking the button in the "Settings" column.



The cluster settings

When modifying the cluster settings, any changes settings are indicated with a bold font so that one has an overview of the changes made before confirming by clicking the 'OK' button.

In order for the settings to be sent to the RoboServers, they must first be applied. **Applying settings** is done by settings the cluster into maintenance mode (see the section on cluster modes for more information). If, when confirming a number of settings changes by closing the cluster settings dialog with a click on the 'OK' button, a cluster is already in maintenance mode, the settings are applied right away. If the cluster is not in maintenance mode, the following dialog appears:



Applying cluster settings
In the apply settings dialog, one selects how the cluster should transition into maintenance mode in order to apply settings. When the settings have been applied, the cluster will return to normal mode.

The following sections describe each part of the cluster settings in more detail.

Databases

The databases that are defined on a cluster are those that are available to robots running on the cluster RoboServers. This means that for a robot to access a cluster database, it must use the name that the databases is given in the cluster settings. A database uses a database type. Database types are defined in the Management Console settings.

If a cluster has been selected as providing Design Studio with Shared Databases, the databases that are defined on the cluster are also sent to Design Studio.

For users upgrading from pre 9.2.0 versions, it is possible to import databases from the legacy db.properties files in which databases were previously defined. This is done by selecting the 'Databases' node in the tree, and clicking the 'Import Databases From File' button. This will open a window in which the contents of the file can be pasted.

The properties of the databases are the same as those described here.

To add a new database, you either select the database category, and click the 'Add Database' button, or you right-click the database category and click the add button. You can import databases from legacy properties files. This is done by selecting the database category and clicking the 'Import Databases From File' button. This will open a window in which you can paste the contents of the file you wish to import.

Proxy Servers

This section allows you to configure a list of proxy servers that will be used on the RoboServers. If only a single proxy server is defined, that one will be used. If a list of proxy servers is defined, for the first connection a random proxy server is selected. The following connections then use the proxy servers in round-robin order.

The properties of the proxy servers are the same as those described here.

To add a new proxy server, you either select the proxy server category, and click the 'Add Proxy Server' button, or you right-click the proxy server category and click the add button. You can import proxy servers from legacy properties files. This is done by selecting the proxy servers category and clicking the 'Import Proxy Servers From File' button. This will open a window in which you can paste the contents of the file you wish to import. The file must have the format described at the bottom of this section.

Logging

In this sections logging is enabled or disabled for the cluster RoboServers, and how to log is defined. There are three types of logging possible, these are described below.

Database Logging
: If database logging is enabled, the RoboServers will log to the RoboServer Log Database which is defined in the Design Studio settings. If 'Log Robot Input To Database' is selected, the input that robots are given will be logged to the database. **NOTE:** a RoboServer log database MUST be configured and enabled for database logging to work. If no database has been configured, the cluster settings will be invalid (you will be informed of this, so you can correct the problem).

Log4J
: Using log4j to log you can control where the log goes using an ordinary `log4j.properties` file. The `log4j.properties` file is found in the `Configuration` folder in the application data folder. The default `log4j.properties` file logs all robot run information, robot messages and server messages into a file. The logs are placed in the `Logs` folder in the application data folder. More advanced setups include storing in the Windows event log, rotating files, and the syslog. See the log4j manual for details.

When enabling this log, RoboServer will log using three different loggers.

Table 29. Log4J Loggers

Logger	Description
log4j.logger.kapow.server.messagelog	This logger is used for server logging that is not tied to a particular robot run.
log4j.logger.kapow.robot.log	To this logger information about starting and stopping robots. It includes execution time of the robot.
log4j.logger.kapow.robot.messagelog	Messages logged by robots e.g. as part of handling errors in steps. This also includes profiling if profiling is setup to output into the log.

E-mail Logging

This will send an e-mail whenever a robot logs an error message or the server has a error message.

Table 30. E-mail configuration

Property	Description
To Address	Who should receive the e-mail. You can add multiple addresses separated by ",".
From Address	The from address to be used on the e-mails.

Profiling

If you enable profiling of robots you can see the execution time for the individual steps and the entire robot. This is very useful if you have a slow running robot and want to improve performance.

Profiling is configured using the following properties:

Table 31. Profiling Properties

Property	Description
Output Type	If you choose Summary, only one statement summarizing the execution is written to the profiling log for each robot execution. If you choose Detailed, a detailed statement is written to the profiling log for every step executed in a robot provided the execution time of the step is above the threshold defined by the Threshold property.
Output Target	This controls where the profiling information is send. The possibilities are to send it to the console, to a file or in the log.
Threshold	This threshold defines the smallest execution time for a step (in milliseconds) for which to include profiling information for that step.
Log page URL in wait messages	If checked, the page URL is printed before page load wait time.

Robot Execution

On this section, properties of the RoboServer robot execution can be defined. It is possible to specify how many robots that can run concurrently on the (individual) RoboServers, and how many can be queued. Please refer to this section for help on tweaking the numbers.

Table 32. Robot Execution Properties

Property	Description
Max concurrent robots	How many robots are allowed to execute concurrently on each RoboServer.
Max queued robots	How many robots are allowed to reside in the queue on each RoboServer.

Projects

In this section of the Admin tab, you may configure which projects are available in the Management Console.

Projects are a way to segment robots, types, snippets, resources and schedules. Robots only have access to the type, snippets and resources contained in the project they belong to. Names of robots, types etc. also need only be distinct within a project.

Note that if you delete a project, all of the robots, types, snippets, resources and schedules associated with it will be deleted as well.

By default, the Management Console contains a single project.

When deploying the Enterprise edition on a standalone web container, you can also configure project permissions for users based on their group membership (for instance LDAP groups), see Project Permissions for details.

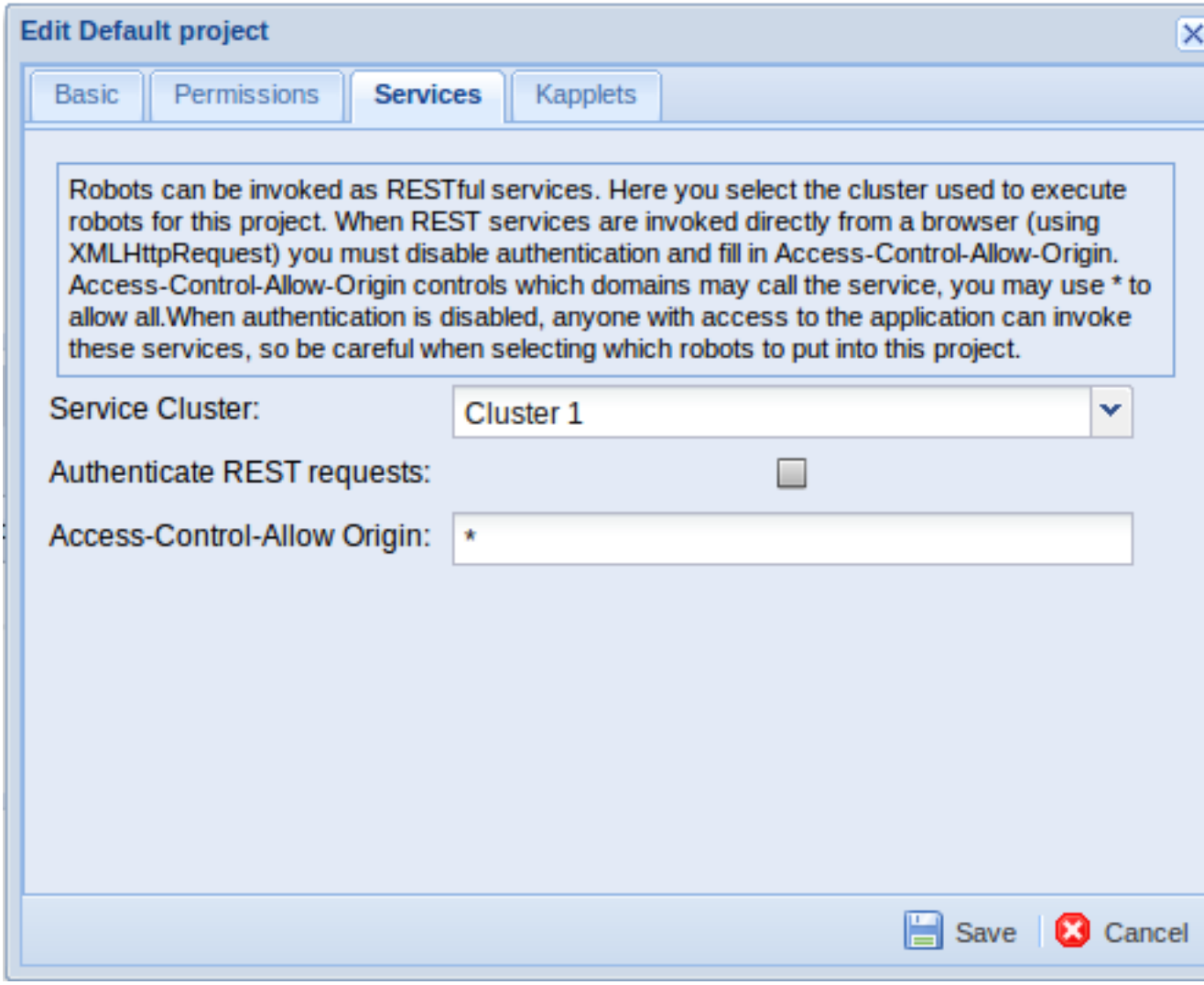
The project name is used as a foreign key in the log tables. This is needed to determine who has permission to view the log files in the Enterprise version. This means that when you rename a project, all existing Robot Runs and Robot Messages belonging to this project must be updated to reflect the new name. Otherwise they will not show up when the logs are filtered by project. If the Management Console is connected to the logging database when you rename a project, the Management Console will automatically rename the Robot Run/Message entries in the log database. However if it is not connected, or the connection is lost during this update, the administrator must manually run the following SQL to update the log tables.

```
UPDATE ROBOT_RUN SET projectName = '<newName>' where projectName = '<oldName>'
UPDATE ROBOT_MESSAGE SET projectName = '<newName>' where projectName = '<oldNa
```

Where `<oldName>` is the old project name, and `<newName>` is the new project name.

Rest Services and Kapplets

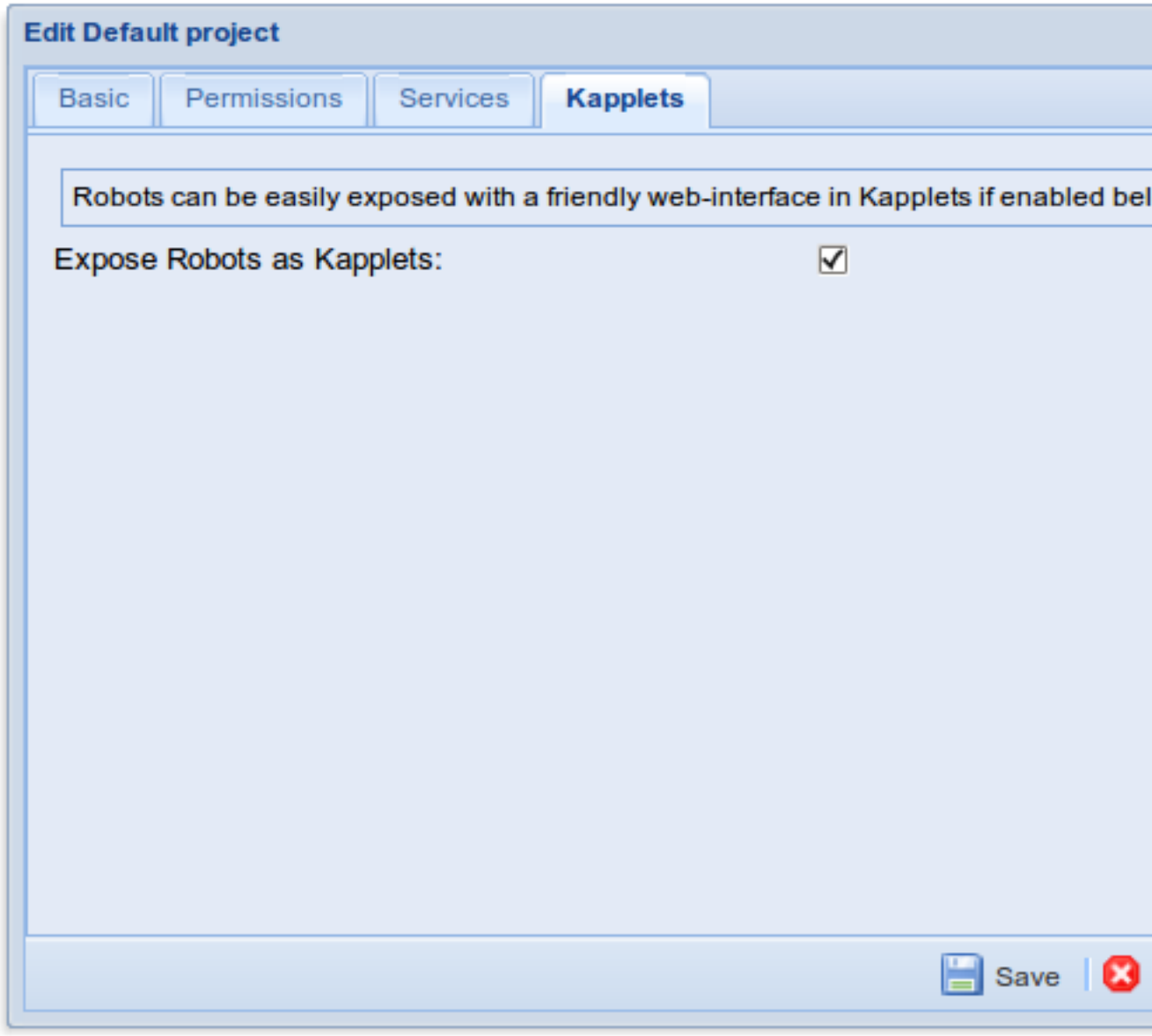
Robots can be exposed as Rest services and/or Kapplets. If you open the Edit Project dialog then on the Services tab you can configure the runtime for these services.



Projects - service tab

By default Rest services are protected by basic authentication. When invoking Rest services directly from a browser (using XMLHttpRequest), you have to disable the authentication, as you would otherwise be exposing login credentials in the JavaScript source files. When calling Rest services from a programming language like Java, Ruby, C# etc, it is a good idea to have the services protected by authentication (assuming that you can keep the credentials stored in a secure manner).

There are certain restrictions when calling a Rest service from a browser, unless the service is located on the same web server as the web page from which it is invoked. When calling a Rest service from another domain (referred to as CORS Cross-Origin Resource Sharing), you have to include certain headers for the client to be allowed to process a resource from another domain. The Access-Control-Allow Origin is one such header. If you call a Rest service in a cross domain fashion, you must specify the domain, from which the page that generated the request was loaded. If a page on http://example.com contains a page with JavaScript that generates a request to a service located on http://kapowsoftware.com, then the service response from http://kapowsoftware.com must contain the header *Access-Control-Allow-Origin: http://example.com* or built in security mechanisms in the browser will prevent it from processing the cross-origin response. As you can see in the above screen shot, you may use * as a wildcard, which means that any domain can invoke your Rest service in a cross domain fashion.



Projects - Kapplets tab

In the Kapplets tab, you can specify if the robots of the project should be exposed as Kapplets.

Users

On the users tab it is possible to view data on users that have logged in. It is also possible to delete any configuration stored by the user, by pressing the Delete button in the user row in the table.

Settings

This tab enables you to configure the Management Console. You have to click the save button for the changes to take effect. Additional options, which affect how you connect to the Management Console (such as port numbers and security settings) are configured in the Settings application as described in Configuring the Embedded Management Console.

Schedule Logs Cleanup

Here you configure the thresholds for the schedule logs cleanup.

Every 10 minutes the Management Console looks at the number of schedule runs and messages. If there are more records than the thresholds defined here, cleanup will occur and records will be deleted until the numbers of records are below the listed threshold.

If you have many schedules, or your schedules execute often, you may want to increase the thresholds, otherwise the information available on the Logs Tab may not be sufficient.

Table 33. RoboServer Log Database Cleanup

Property	Description
Max schedule runs	The maximum number of schedule runs recorded in the database before cleanup triggers.
Max schedule messages	The maximum number of individual schedule messages recorded in the database before cleanup triggers.

RoboServer Authentication

In this section, you configure the credentials (user name and password) that the Management Console will use when running robots on the RoboServers belonging to the configured clusters. The Management Console will use the same set of credentials for all RoboServers. These credentials must match the configuration done as described in Requiring Authentication for all the configured RoboServers.

RoboServer Log Database

This is the logging database used by all RoboServers belonging to clusters where database logging has been enabled in the cluster settings. The database is configured in the same way as the cluster databases.

Cleanup thresholds can be configured for the RoboServer log database in the same way as for the Schedule Log.

Table 34. RoboServer Log Database Cleanup

Property	Description
Robot message count threshold	The log will automatically clean up the database when the number of robot messages in the database exceeds this threshold. The cleanup will delete the oldest robot runs and the messages for the deleted runs. If you experience performance problems with the log database you can lower this threshold. If you wish to store more historic messages you can increase this threshold.
Robot run count threshold	The log will automatically clean up the database when the number of robot runs in the database exceeds this threshold. The cleanup will delete the oldest robot runs and the messages for the deleted runs. If you experience performance problems with the log database you can lower this threshold. If you wish to store more historic messages you can increase this threshold.
Server message count threshold	The log will automatically clean up the database when the number of server messages in the database exceeds this threshold. The cleanup will delete the oldest server messages. If you experience performance

Property	Description
	problems with the log database you can lower this threshold. If you wish to store more historic messages you can increase this threshold.

In order to use a database for logging, you must prepare your database server by either creating a new database (schema), or simply making sure an existing database is available. You must obtain a username and password with rights to create tables, drop tables, create indexes, drop indexes, select, insert, update, and delete in the database.

Both the Management Console and RoboServer will create the log tables automatically when they are started (if the tables do not already exist). However you may also create them using the following scripts

Table 35. SQL scripts for log tables (right click and chose *save as*)

Database	Create Tables	Drop Tables
IBM DB2	create [sql/logging/db2_create.sql]	drop [sql/logging/db2_drop.sql]
Derby	create [sql/logging/derby_create.sql]	drop [sql/logging/derby_drop.sql]
MySQL	create [sql/logging/mysql_create.sql]	drop [sql/logging/mysql_drop.sql]
Oracle	create [sql/logging/oracle_create.sql]	drop [sql/logging/oracle_drop.sql]
Microsoft SQL Server	create [sql/logging/sqlserver_create.sql]	drop [sql/logging/sqlserver_drop.sql]
Sybase	create [sql/logging/sybase_create.sql]	drop [sql/logging/sybase_drop.sql]

If you are upgrading from version 7.2, and want to use the logging tables already created, you will have to modify the `ROBOT_RUN` table, since a new column has been added. You must add a column named `PROJECTNAME` of type `VARCHAR(255)` to the `ROBOT_RUN` table (`NVARCHAR2` on Oracle, `NVARCHAR` on Sybase and Microsoft SQL server).

If you are upgrading from version 8.1 or earlier, and want to use the logging tables already created, you will have to modify the ROBOT_MESSAGE table, since a new column has been added. You must a a column named `PROJECTNAME` of type `VARCHAR(255)` to the `ROBOT_MESSAGE` table (`NVARCHAR2` on Oracle, `NVARCHAR` on Sybase and Microsoft SQL server).

SMTP Server

This section allows you to configure a mail server that will be used for sending notifications to Kapplet users (if notification is selected when starting the Kapplet), and for sending e-mail log messages if e-mail logging has been enabled in the cluster settings. **NOTE:** for Kapplet notification e-mails to work, a from address must also be specified under Kapplet results.

The SMTP server is configured using the following properties.

Table 36. SMTP server

Property	Description
Host	The SMTP server host name

Property	Description
Port	The SMTP server port.
User	If the SMTP server requires authentication then enter the user name here.
Password	If the SMTP server requires authentication then enter the password here.
Encryption	• **NONE:** Credentials and email are sent in clear text. • **TLS:** TLS encryption is used. **This only works if the SMTP server has a trusted certificate** (if the server has a self-signed certificate it must be exported and imported into the JVM's truststore using the keytool utility). *Uses the STARTTLS SMTP extension.* • **SMTPS:** SMTP over SSL. A secure channel of communication is established, over which the Credentials a email is sent. **This is rarely supported by SMTP servers** , but will work even if the server uses a self-signed certificate.

Base URL

This is the application base URL. It is used when generating results links in Kapplets, so that the e-mails can contain links to the results residing on the Management Console server. Normally, the base URL is configured automatically, and it is not necessary to change it.

Shared Databases

Shared databases are databases which are sent to all Design Studio clients being activated by (getting license from) the Management Console. To simplify the user interface, these databases are configured on a cluster. So in order for the databases available in a certain cluster to also be available to Design Studio clients, simply select the cluster here. If one desires to have a special set of databases available for Design Studio (for instance, maybe one is not interested in having production databases sent to Design Studio) a special Design Studio cluster can be created on which the shared databases are defined. The default cluster for shared databases is the default, and automatically created, Production cluster.

Kapplet Results

In this section Kapplet results settings are configured.

Table 37. Kapplet Results

Property	Description
Delete results after (hours)	How many hours that Kapplet results are residing in storage before they are automatically cleaned up (deleted).
From address (on result e-mails)	A from address used when sending results e-mails. Note that results e-mails also require a properly configured SMTP server.

Database Types

A database type defines a given type of database, e.g. MySQL. A database type consists of the following properties.

Table 38. Database Type Properties

Property	Description
Name	The name identifying the database type.
JDBC driver	The JDBC driver class of the driver (the driver must be uploaded under Database Drivers).
Connection URL template	A template string defining how connection URLs for databases of the given type look. Possible variables to use in the template string are the following: `${ServerName}` Defines the server name (host) of the database server. ${Schema} Defines the schema (or database/catalog depending on database vendor) of the database. An example of a connection string template is the following '`jdbc:mysql://${ServerName}/${Schema}`'. This string defines the connection string for a MySQL database running on the default port (no port is specified), on the server given by `${ServerName}` and using the schema given by ${Schema}. The variables are given values when databases of the given type are created (see the cluster databases section.
Validation query	The validation query for a database is the query to use when validating connections to the given database. Such queries varies greatly between different types of databases.

The database types are also sent to Design Studio clients.

To add a new database type, you either select the database type category, and click the 'Add Database Type' button, or you right-click the database type category and click the add button. **NOTE:** adding a new database type should be considered an advanced and in some cases experimental feature, meaning that the default database types are those that are supported, and there is no guarantee that a whole new type of database will work as expected throughout the product. However, modifying the database types in order to, for instance, add a type that defines a MySQL server running on a non-standard port would be a normal use-case.

Database Drivers

The Database Drivers section of the settings contains uploaded database JDBC drivers. These drivers are required by the various databases types. As with database types, uploaded database drivers are also sent to Design Studio clients. Note that if you run your Management Console on, for instance, a MySQL database, you need to provide Tomcat with the MySQL driver. This means that MySQL databases will work when used in the Management Console (on the log tab or when testing the connection). However, for the MySQL databases to also work on all RoboServers, it is necessary to upload the MySQL driver here so that it can be sent to the RoboServers (and Design Studio).

HINT: on certain database types, you may need to tweak parameters or settings to be able to store files larger than a certain size. For instance, on MySQL databases, you may have to increase the value of the '`max_allowed_packet`' variable which in many installations are set to 1 MB meaning that database drivers larger than 1 MB cannot be stored. Please consult the documentation for your database if you encounter problems when uploading drivers. To help identify any problems, you will receive an error message, and the Management Console log will contain further details.

SECURITY NOTE: by default, only the admin user is allowed to upload JDBC drivers IF he is accessing the Management Console from the localhost. If you are running the Management Console in embedded mode, you can change this behavior in the RoboServer Settings application, please refer here for further details. If you are running the Management Console in Tomcat, refer to this section instead.

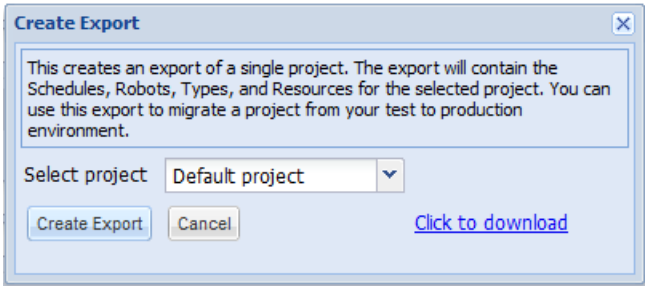
Backup

This tab enables you to make a copy ("Backup" or "Export") of all or parts of the Management Console configuration into a file, and to read back ("Restore" or "Import") information from a file created this way.

It is important to understand the difference between creating/restoring a "backup" and exporting/importing a "project". A backup contains all of the Management Console configuration including all schedules and all projects in the repository, and can be restored only in its entirety. It may be used to restore the system after data loss, or when upgrading to a later version of Kapow Katalyst. In the latter case, you will create a backup of the old instance of the Management Console and restore it into the new instance (see below for more information on restoring from an instance of the Management Console prior to release 8.1. Also note that prior to release 8.1, "Create Backup" was named "Export"; the term "export" is now used only for "Export Project").

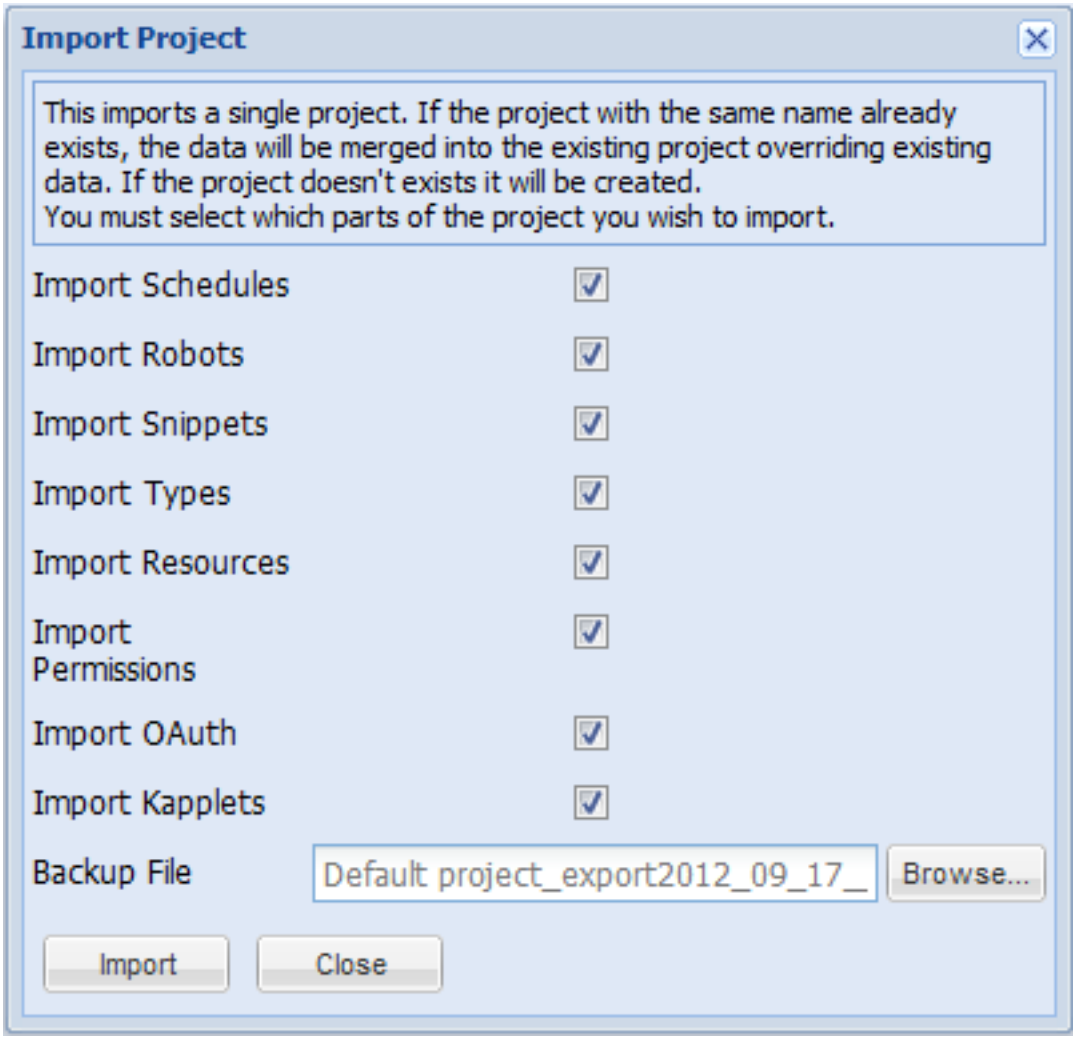
"Export Project" creates a file with information pertaining to a single project. This comprises the schedules, robots, types, and resources within the project. Such a file may be used to copy schedules, robots etc. from one Kapow Katalyst system to another, for example when moving from a test environment into production. It is possible to import into an existing project on the target system; in this case items from the file will be merged into the project, overriding present items when names match. It is also possible to do a selective import that includes only some kinds of items.

Creating a backup or project export file is a multi-step process as follows: When you click the **Create Backup** or **Export Project** button, a dialog opens. When exporting a project, you must select the desired project in this dialog. Then click the **Create ...** button in order to create the desired backup/export file; this may take a while. When the file is ready, a download link appears. You should note that the link is active only while the dialog is open; you cannot copy the link and use it after closing the dialog. The following shows the dialog for "Export Project" with an active link, that is, after the export file has been created.



Export Project Dialog

Restoring from a backup or importing a project is similar: First, click the **Restore Backup** or **Import Project** button. In the dialog that appears, locate the backup/project file that you want to restore/import. When importing a project, also select the kinds of items that you want to restore (as shown in the example below). Finally, click the **Restore** or **Import** button to actually perform the restore or import.



Import Project Dialog

Note about restoring an export from a Management Console prior to release 8.1: As described above, it is possible to restore from a backup made with an earlier release of the Management Console. Three caveats should be kept in mind: First, the terminology was changed in release 8.1. Previously, the term "export" was used for what is now named "backup". That is, if you want to migrate contents from a Management Console prior to release 8.1, you must use its "Export" function to create an export file. Secondly, this file must be converted to the new backup file format with the aid of the "Upgrade Pre-8.1 Management Console Export" feature in the Tools menu of Design Studio. Thirdly, the new project based permissions introduced in 8.2 means that permissions from previously version can't be upgraded automatically. After you restore an older backup, you will have to assign project permission manually.

License

On this tab, information on the current license is shown, and a new license can be entered. The Management Console has room for two license keys: A Production and a Non-Production key. If your production environment is completely separated from your development/QA environment you will need to install two Management Consoles: one should contain the Production license key, while the other should contain the Non-Production license key. In mixed environments, a single Management Console should contain both keys.

This tab also shows information on the Design Studio seats currently in use (Design Studio clients that have gotten a license from this Management Console and are currently active).

JMX

The Management Console offers a small amount of information via the JMX protocol. The information exposed through JMX is experimental, and should not be considered a public API.

The Management Console exposes the following 4 MBeans:

Table 39. Top level MBeans

Name	Description
`DistributionController`	Information on executing schedules, and cluster configuration.
`FileBackedDataExporter`	Information on exports created from the Data View.
`RobotLibraryGenerator`	Allows you to control the internal caching of Robot Libraries within the Management Console.
`TaskQueuePerformanceTracker`	Allows you to profile the distributed Task queue between two clustered Management Console instances.

The MBeans are accessible via JConsole and Java VisualVM (+MBean plugin). Both are included in JDK 1.6+ or any other JMX client.

OAuth

OAuth is an open standard for authorization that is used with many of the popular APIs such as Twitter and Facebook. It provides a means for applications (and in the case of Kapow Katalyst: Robots) to access data or perform actions on a user's behalf without having direct access to the user's login credentials. For instance, a robot can use Twitter's API to extract the most recent mentions of a user, e.g. @KapowSoftware, using access tokens provided by that user but without having explicit access to @KapowSoftware's Twitter password.

The Management Console is used to generate access tokens, which are stored securely in a key store. The access tokens can be passed on as input to robots in a schedule. As always, the robots that perform the actual API calls and handle the returned data are created in Design Studio. Both steps will be demonstrated in an example in the following.

Supported Service Providers

In OAuth terminology, a service provider is the provider of the web service to access. Kapow Katalyst supports the following service providers:

- Dropbox
- Facebook
- Google
- LinkedIn
- Salesforce

- Twitter
- Yelp

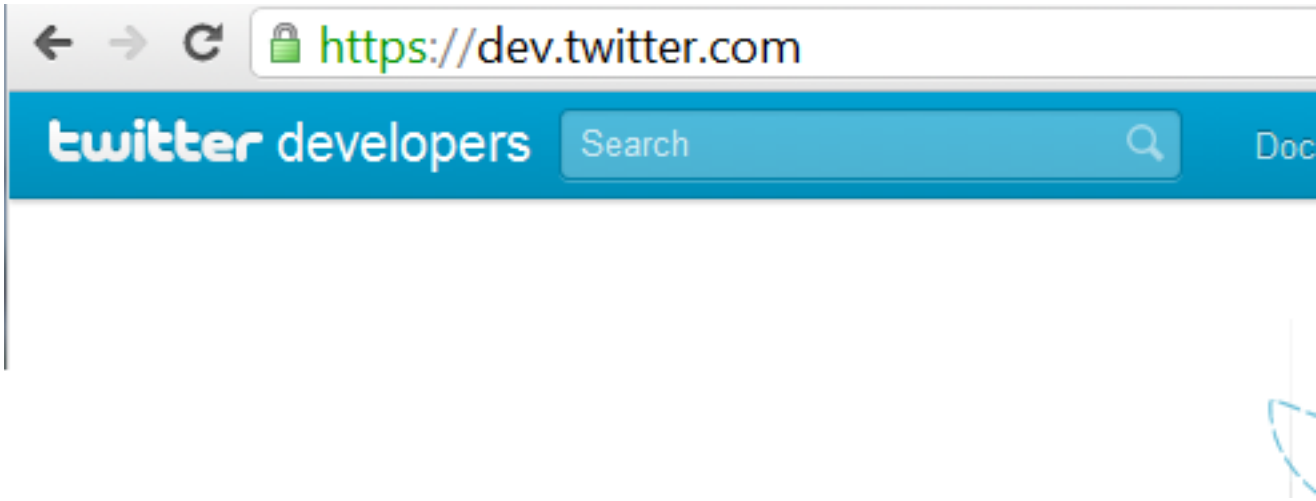
Documentation on the APIs of each service provider can be found in their respective websites.

Adding Applications

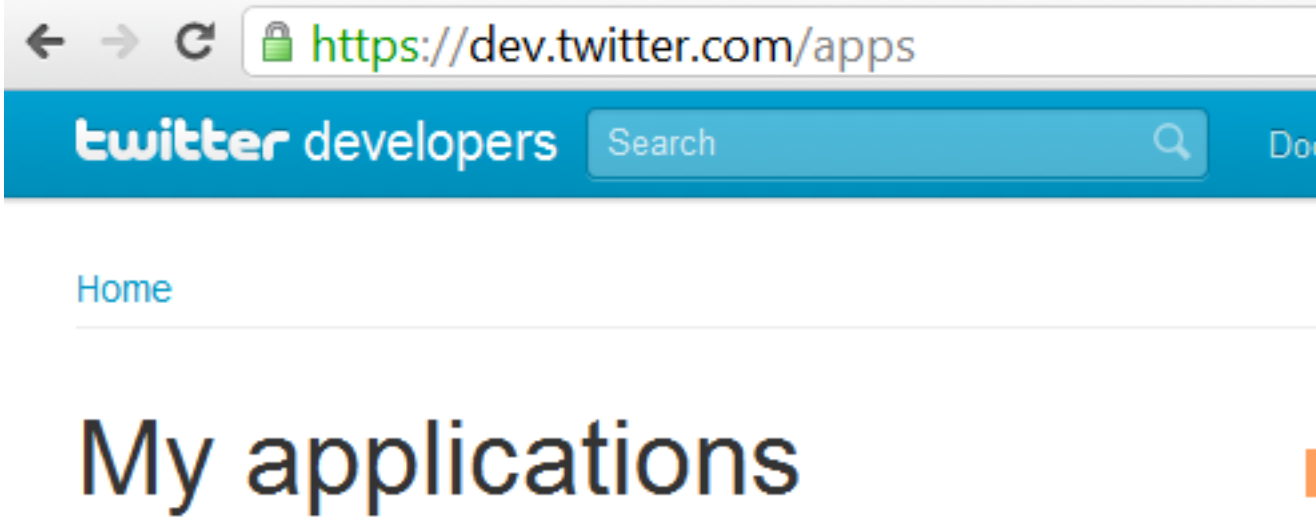
In order to access the API of a service provider, you need to create an *application* with that service provider. Creating an application will provide you with a *consumer key* (also known as *API key* or *application key*) and a *consumer secret* (also known as *API secret* or *application secret*).

Creating an application is normally done by logging in to the "developer" community of the service provider, selecting "Create New Application", or the like, and filling out the required information. Let's take a look at how this is done at Twitter:

First, log into http://dev.twitter.com (creating a new account if necessary), click the user name in the top right corner and select "My Applications".

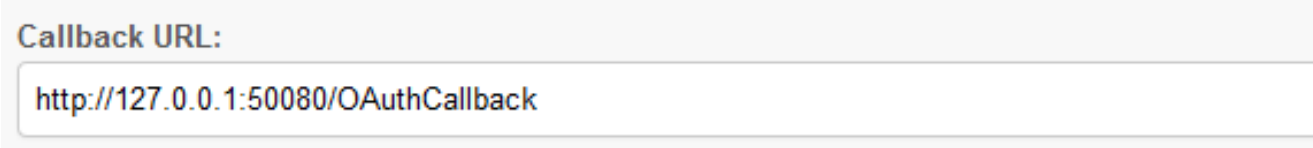


Then, click the "Create a new Application" button.



Fill out the required information, such as the name and description of the application and read through Twitter's terms of service before accepting.

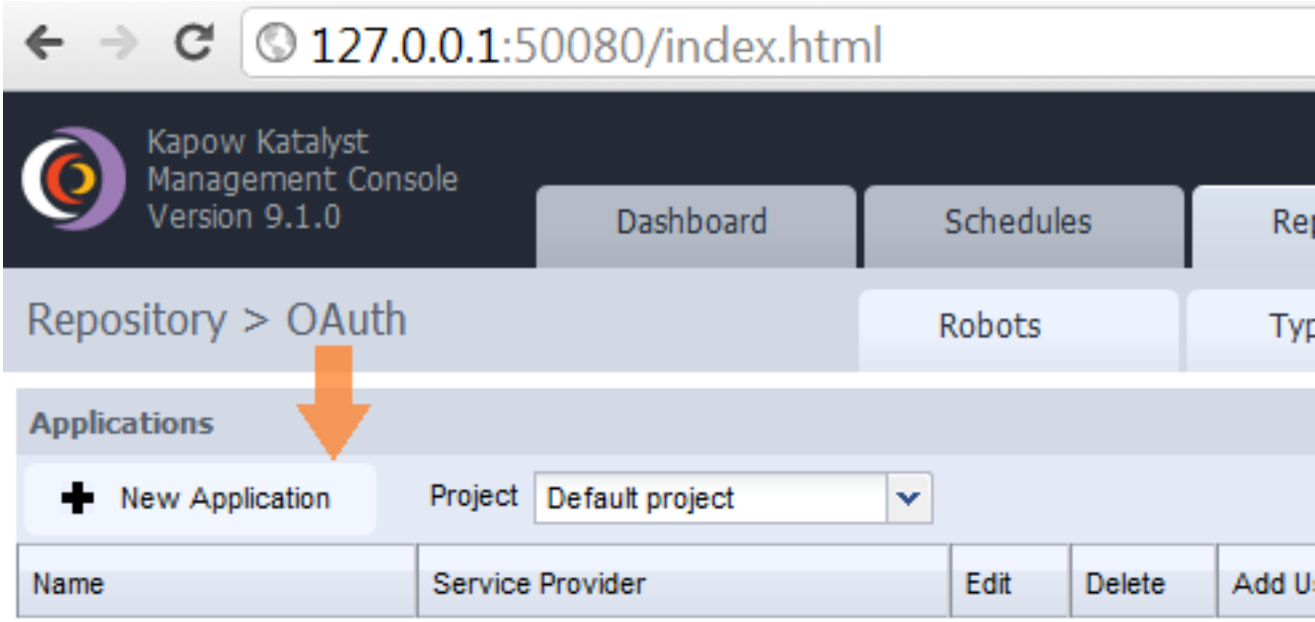
One of the fields in the form is a "Callback URL". This is the URL that Twitter will redirect a user's browser to after she has accepted to let your application interact with her Twitter account on her behalf. This field must be set to the path `OAuthCallback` under the folder in which the Management Console is deployed. For instance, if running with an embedded Management Console, it runs at `http://localhost:50080/`. In this case, the callback URL would be specified to `http://localhost:50080/OAuthCallback` - however, beware that some service providers do not allow a callback URL containing `localhost`. Twitter is one of those providers, so we will use `http://127.0.0.1:50080/OAuthCallback` instead.



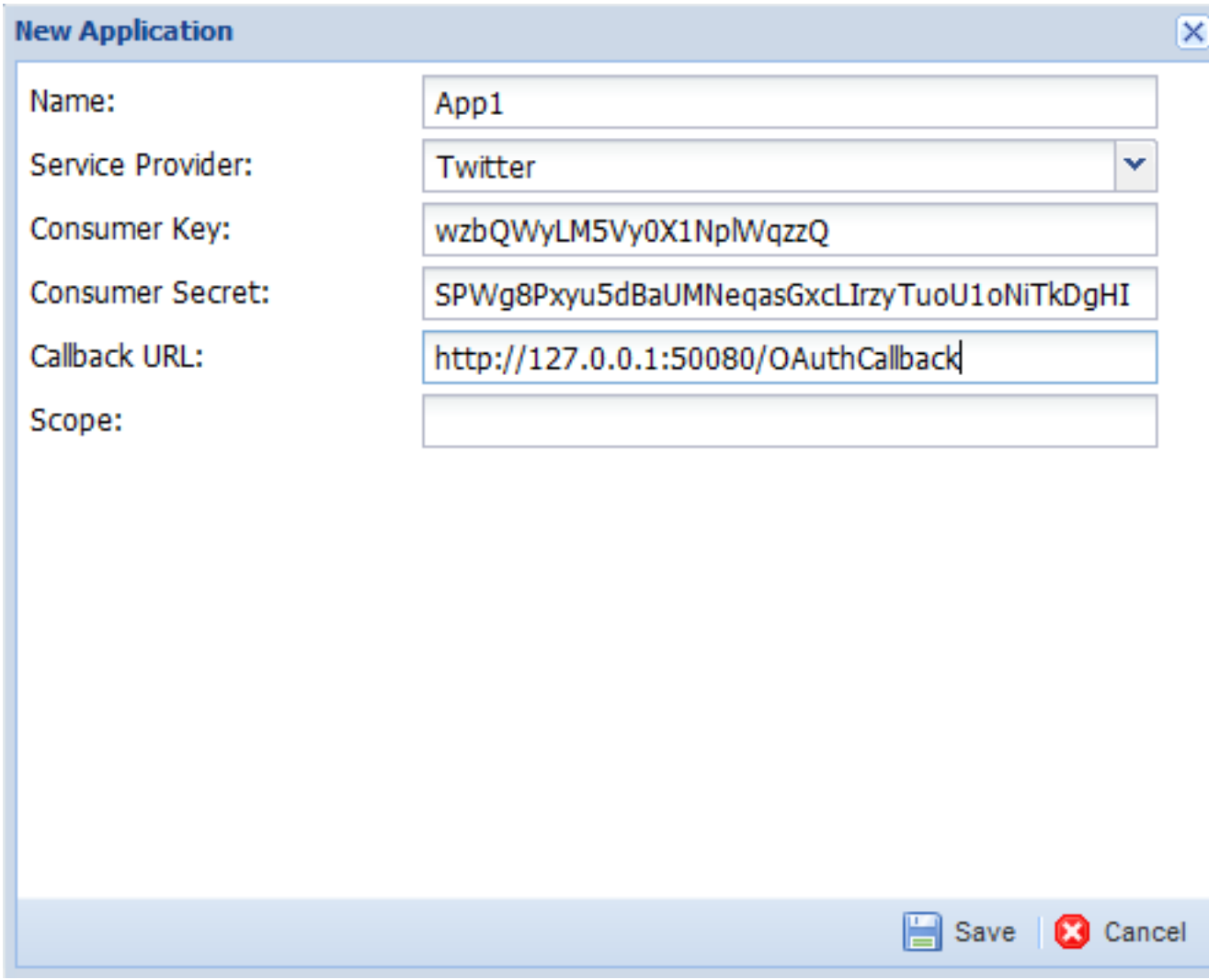
Alternatively (and this is required by some service providers), you need to specify the hostname or non-loopback IP address of the machine on which you are running the Management Console. Since this page will be loaded by the browser of the authenticating user, this need not be a public hostname or IP address.

After creating the application, we are presented with a summary of the application. We will need to copy some of these values into Management Console, so go ahead an open Management Console in a browser. Note that you should use the same IP address or hostname that was entered as callback URL; in this example we will therefore point our browser to `http://127.0.0.1:50080/`.

Now, navigate to the OAuth tab, which is a sub-tab of the Repository tab, and click the "New Application" button.



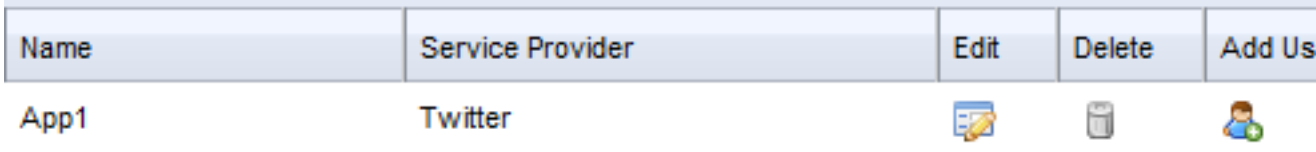
Select a name for the application (which doesn't need to be the same name as what is used when you created the application at the service provider) and select the service provider (in this case Twitter).



The consumer key and consumer secret must be copied from the summary page of the application presented by the service provider.

Enter the same callback URL as you did before and click Save. Some service providers additionally require that you specify a scope; i.e. what parts of the API that a user will authorize the application to access. For instance, when accessing Google, the scope `https://www.google.com/analytics/feeds/` must be specified if the application should be allowed to access the Google Analytics Data API. Twitter does not use the scope field, so this will be left blank in our example.

We have now set up an OAuth application in the Management Console.



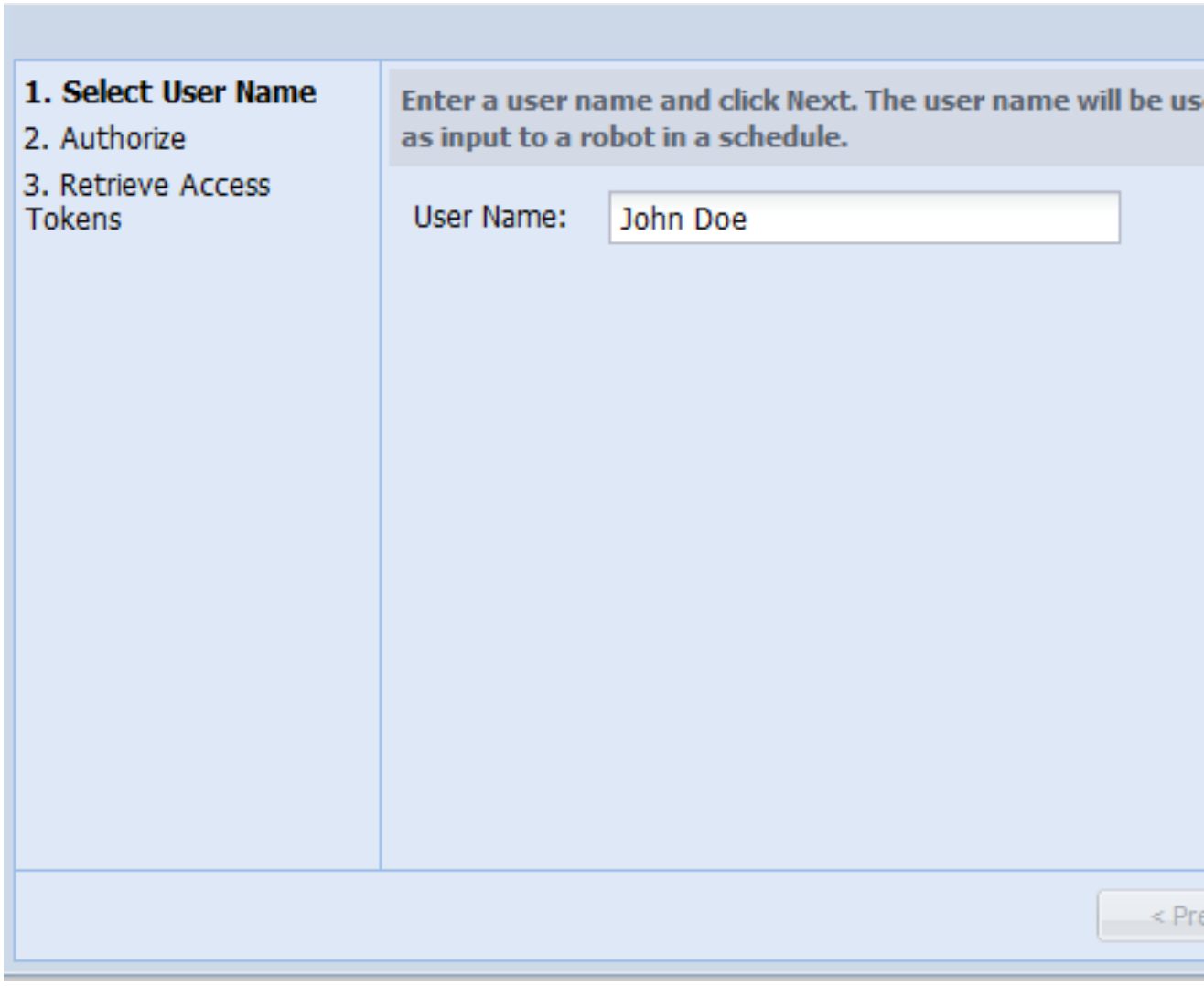
Note that if you later edit the application, the consumer secret will be displayed as "(encrypted)" for security reasons. To change the consumer secret, simply replace this value in the input field with the new consumer secret; otherwise, leave as-is when editing an application.

Next, we will be adding a user to the application.

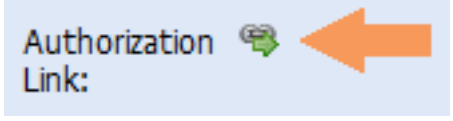
Adding Users

Click the icon in the Add User column next to the application that was created in the previous steps.

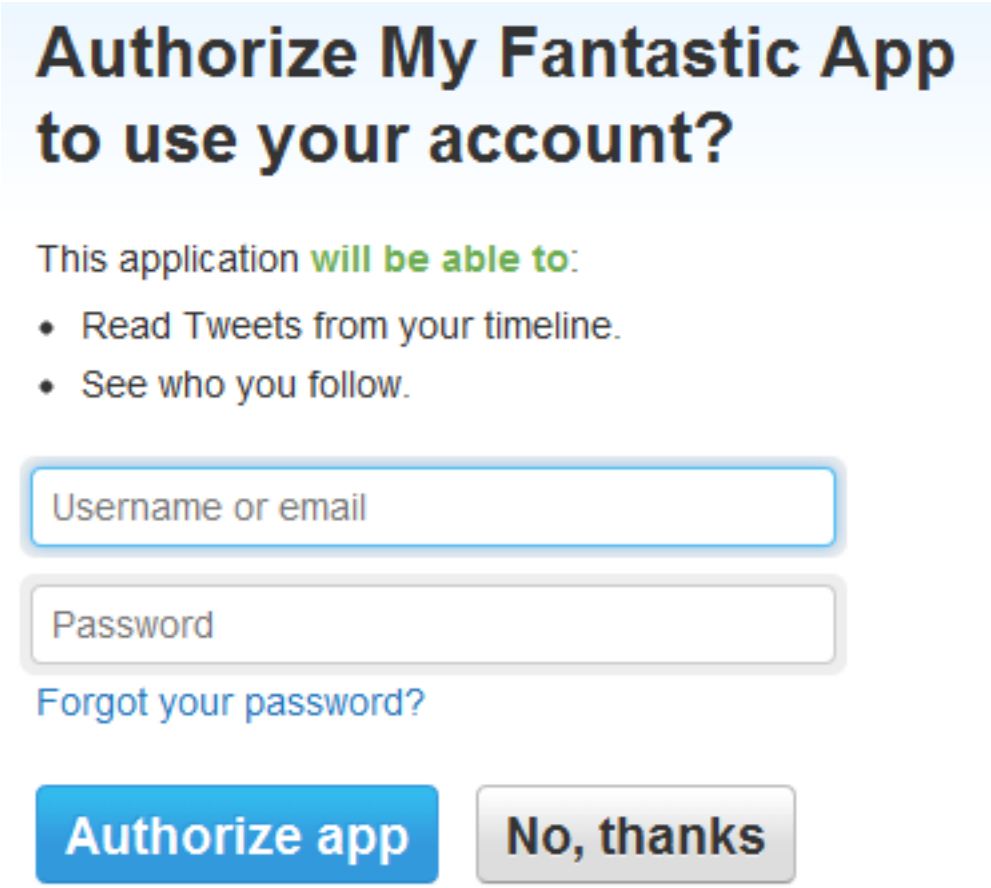
This will initiate a wizard. As a first step, select a name for the user. This doesn't need to map to the username used by the service provider but will only be used inside the Management Console.



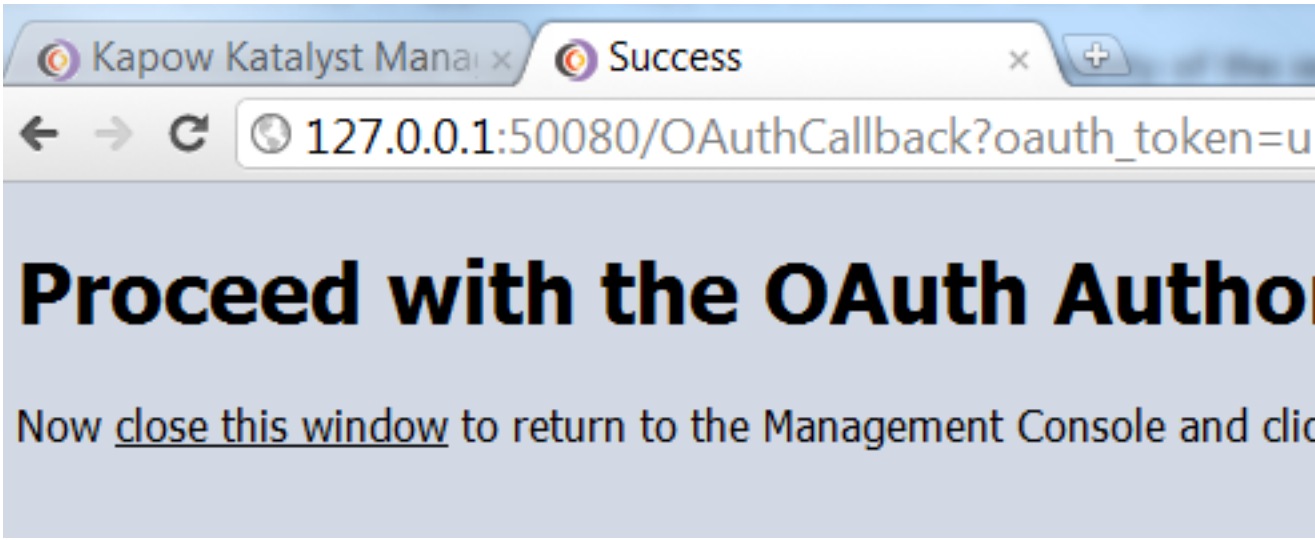
In the next screen there is an authorization link that must be clicked.



Clicking the link will take you to the service provider's website. At Twitter, this looks as follows:

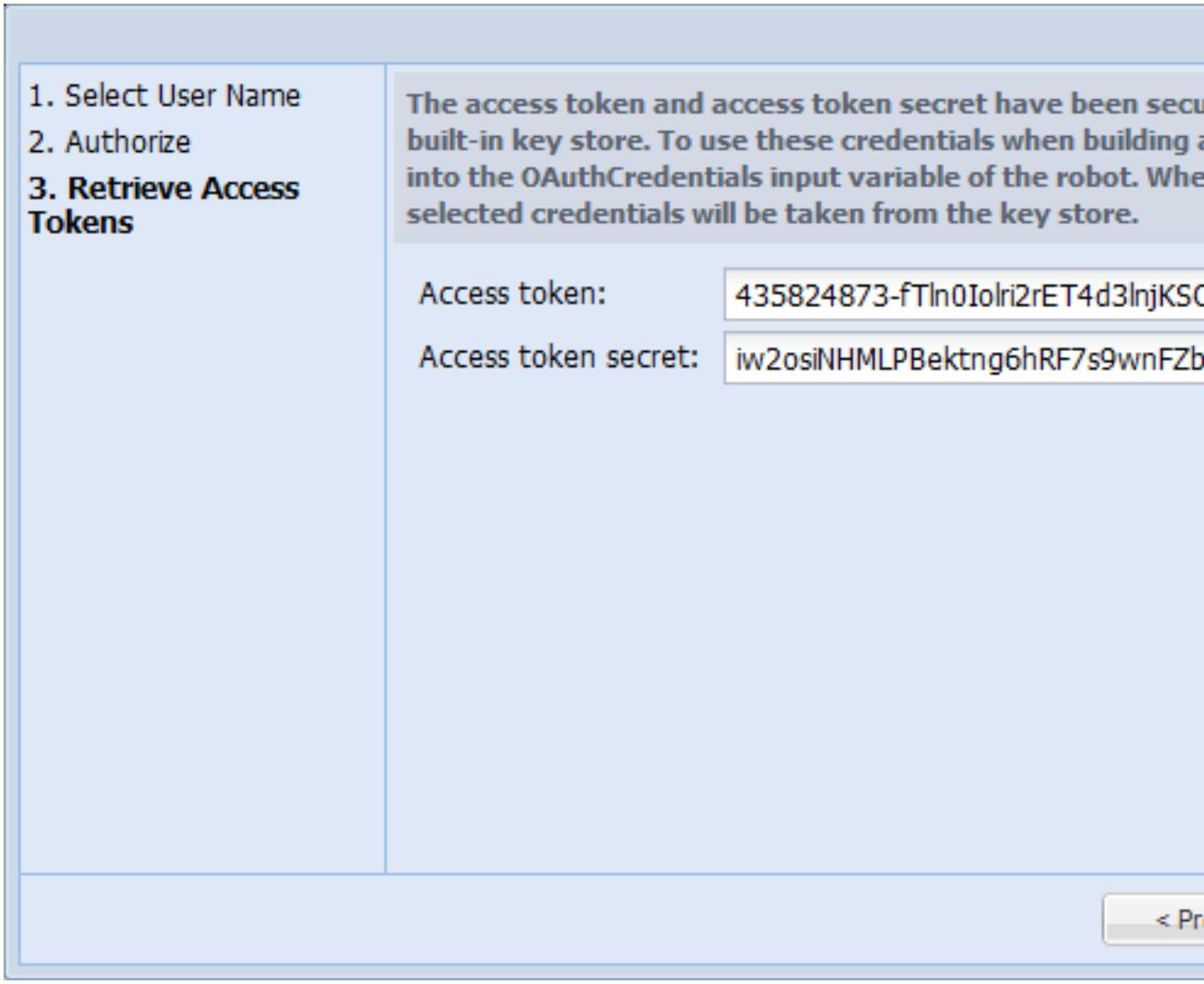


Enter the username and password and click Authorize app. The service provider now forwards us to the callback URL and if the authorization was successful, you should see this page:



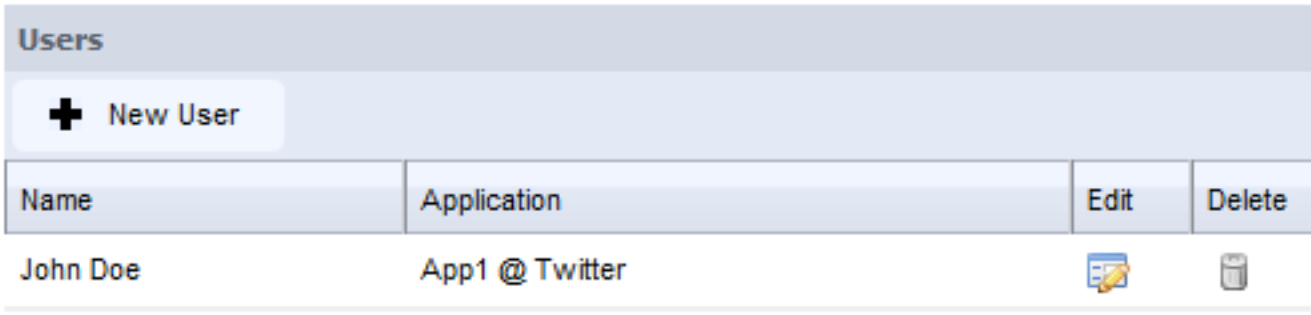
Close the browser tab and return to the Management Console. Click Next in the wizard and you will see the access tokens that can be used for accessing the service provider on the user's behalf. They have been securely stored in the Management Console's key store and can now be used as input to schedules. However, because we will need some sample access tokens as test input for the robot that we will build in

a later step, copy the values into a text editor such as Notepad. For security reasons, you will not be able to retrieve them from the key store in unencrypted form after having clicked Finish.



At Twitter, we get both an access token and an access token secret. Service providers that use OAuth 2.0 do not use an access token secret, so they will only return an access token. Some service providers will additionally return a refresh token. This is used when the access tokens returned by the service provider are only short-lived. Robots can then use the refresh token to obtain new access tokens without a user having to re-authorize through the Management Console. To create robots against the API of a service provider, one must copy all of the tokens displayed at the final step of the wizard.

After clicking Finish, we should now see a user in the Users section of the OAuth tab.



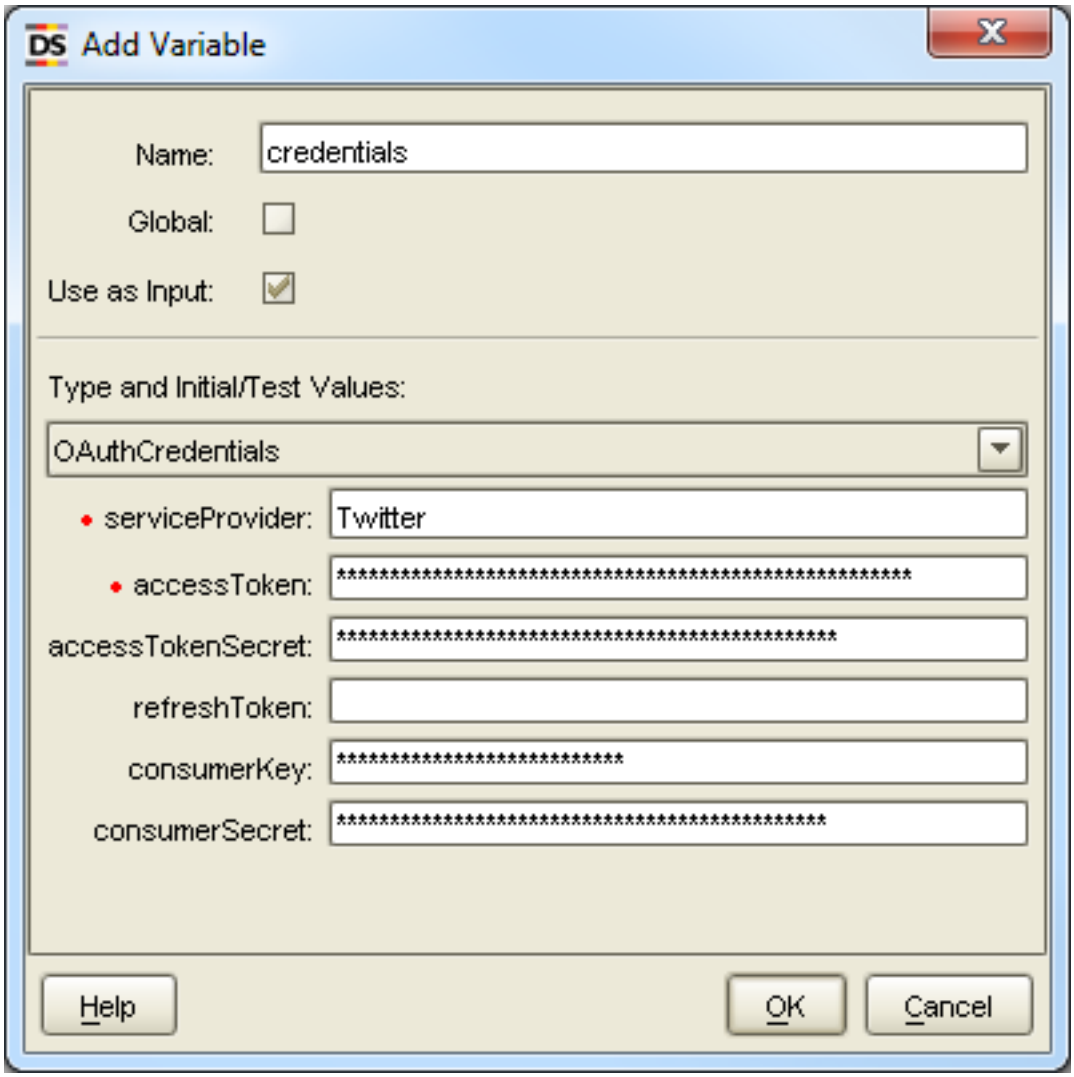
Note that if you later edit the user, the access token, access token secret and refresh token will be displayed as "(encrypted)" for security reasons. To change any of these, simply replace this value in the input field with the value; otherwise, leave as-is when editing a user.

Next, we will show how to write a robot that accesses an API that uses OAuth.

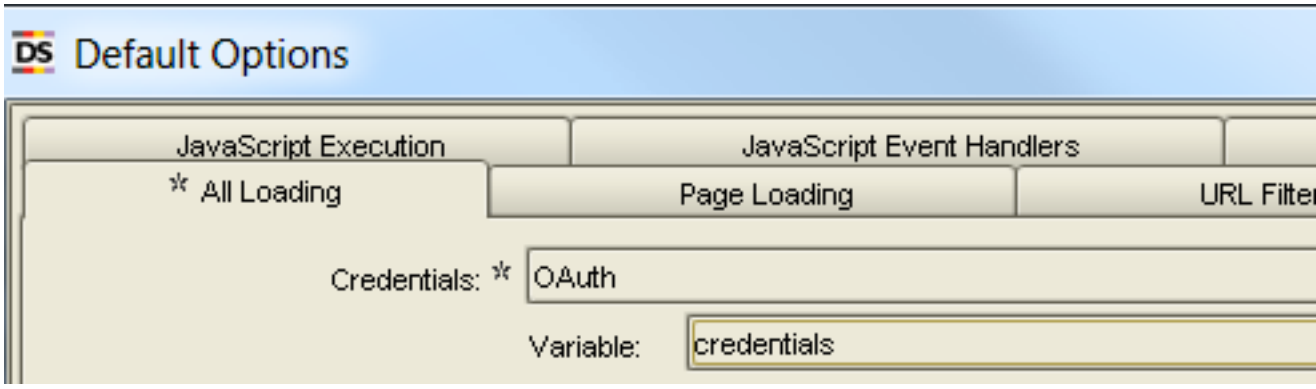
Writing Robots

In the following, we will show how to write a robot that accesses a REST API that uses OAuth as its authentication mechanism. As an example, we will use the Twitter REST API to obtain the most recent statuses by the authenticating user and the users he or she follows.

Start Design Studio and create a new robot. Do not enter a URL in the wizard, as we will not be able to access the REST API before having authenticated. Add a new input variable of type `OAuthCredentials`. Type "Twitter" as service provider, enter the access token and access token secret that was obtained when we went through the user authorization process in the Management Console wizard, and enter the consumer key and consumer secret of the Twitter application.

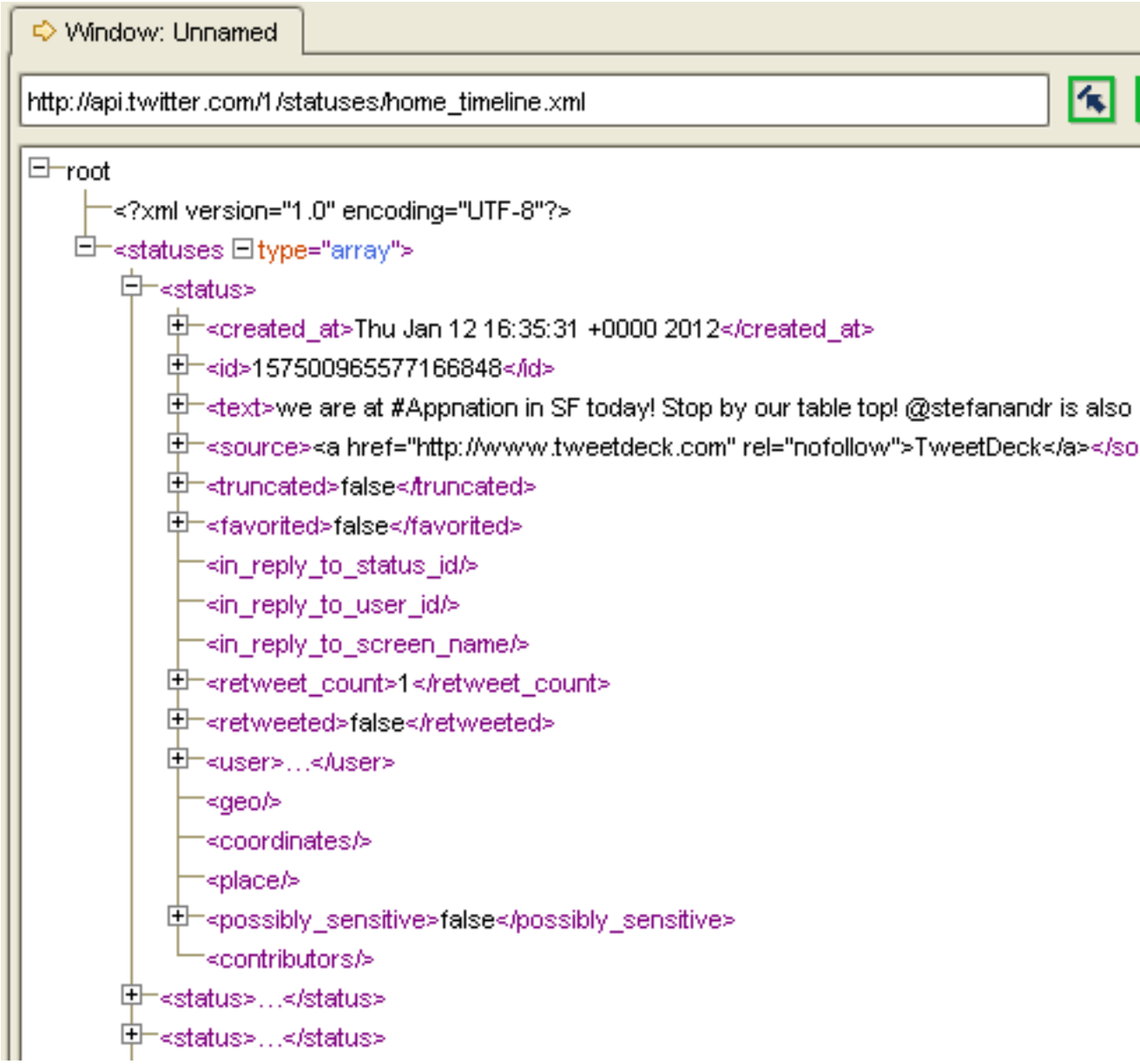


After adding the variable, open the robot configuration by clicking the button. Click the "Configure..." button on the Basic tab. On the "All Loading" tab there is a "Credentials" option. Switch it from standard username/password authentication to OAuth and select the input variable that was just added.



Press OK in both dialogs. The robot has now been configured to use OAuth and will use the specified credentials when running in Design Studio. We can now start accessing Twitter's API. For instance, to see the most recent status updates by the authenticating user and the users he or she follows, we can access

the URL `http://api.twitter.com/1/statuses/home_timeline.xml`. Enter that URL in the address bar of Design Studio and press Enter.

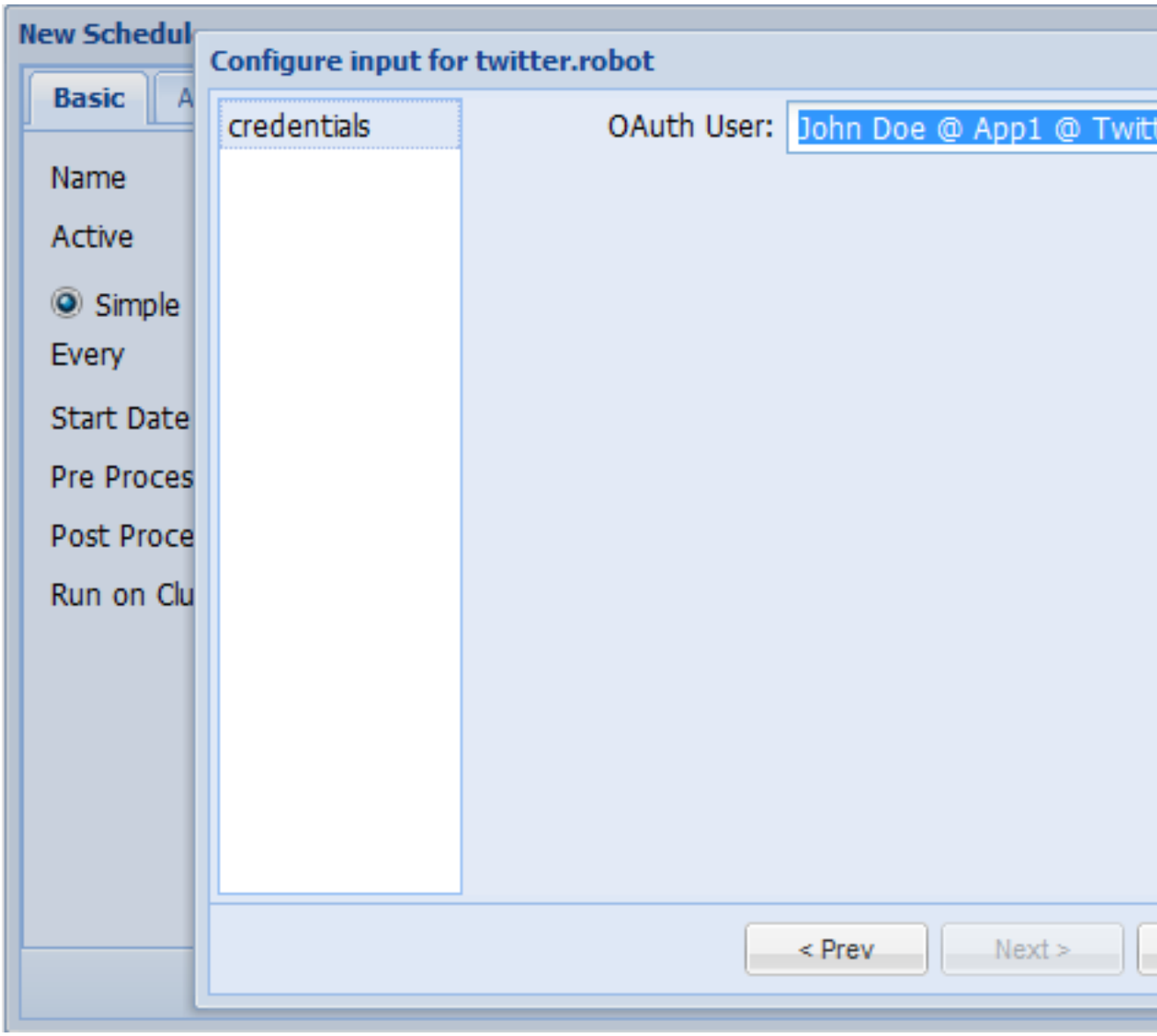


You should now see the XML that has been returned, containing the most recent statuses in the user's timeline, as above.

Next, we will look at how to use the credentials stored in the Management Console as input to the robot.

Scheduling Robots with Credentials

First, save and upload the robot that was created in the previous step to the Management Console instance where we created a user for the OAuth Twitter application. Open the Management Console in a browser and create a new schedule. Add the robot to the schedule and click next in the "add robot" wizard until you have reached step 4: Configure Input.



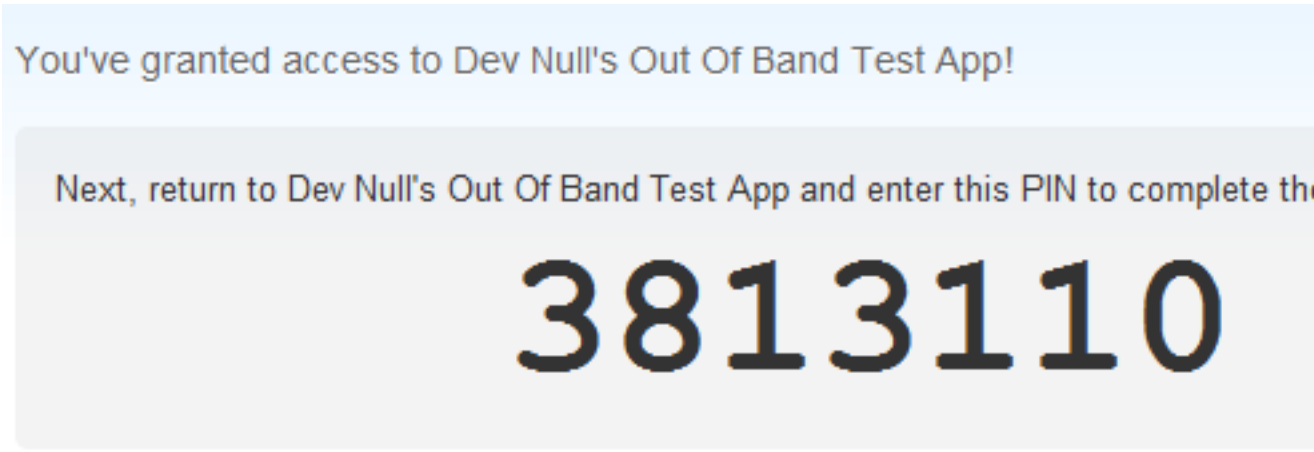
Select the OAuth user whose credentials to use when running the robot in this schedule. This will auto-populate the service provider, access tokens, consumer key and consumer secret fields when the robot is run on RoboServer.

Click Finish and save the schedule, which is now ready to run.

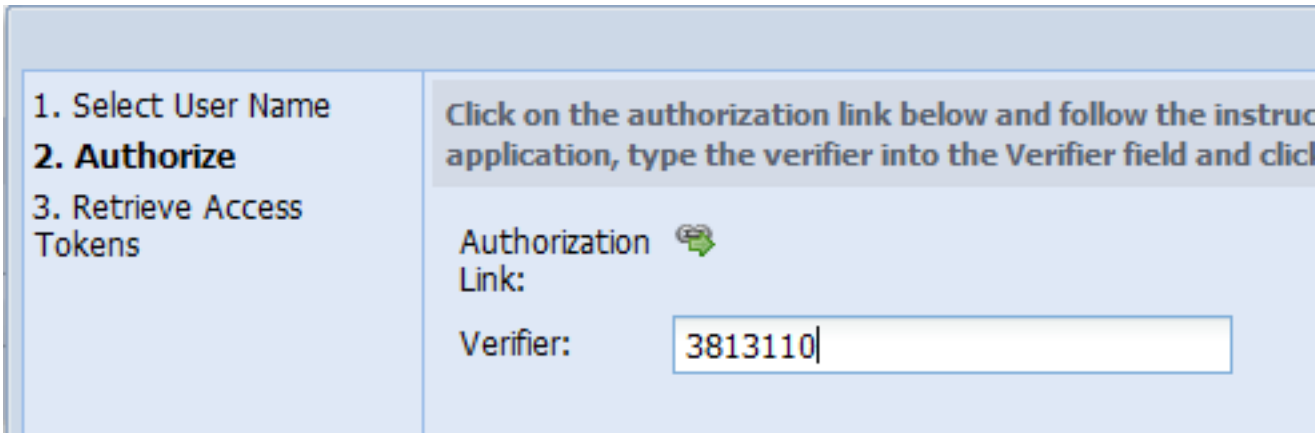
Out of Band Applications

Some service providers also offer a mechanism to authorize OAuth applications without using a callback. This is known as "out-of-band". Management Console also supports out-of-band authorization. The application created at your service provider must be registered as an out-of-band application; at Twitter this is done simply by leaving the callback URL field blank. Other service providers use the special value "oob" to signify an out-of-band application.

When you register the application in Management Console, leave the callback URL field blank here, too. When adding users to the application, clicking the authorization link will not redirect back to the Management Console. Instead, the service provider will present a verifier or PIN.



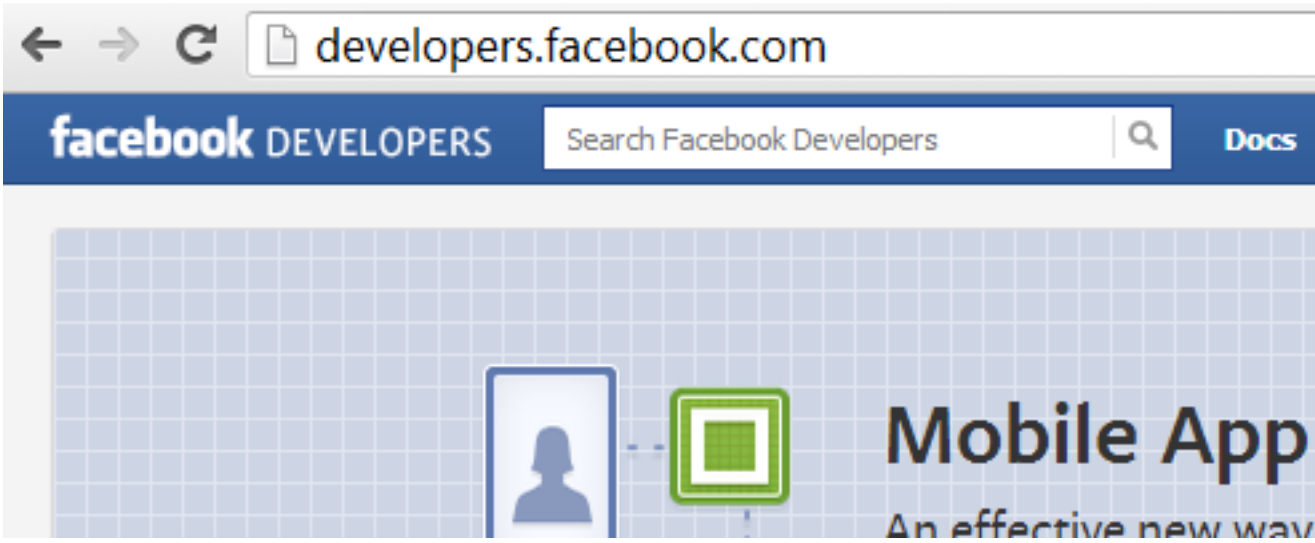
This pin must be copied into the Verifier field in the wizard before pressing Next.



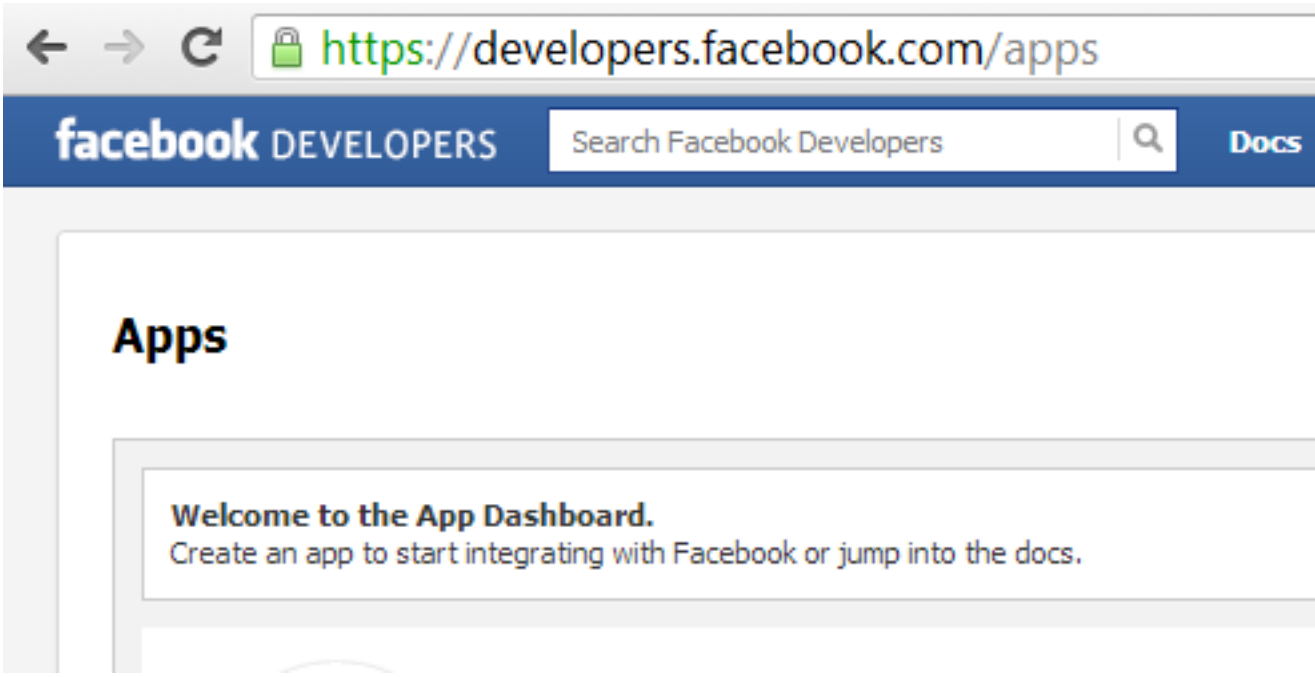
Facebook

This is a short guide to creating a robot which accesses Facebook through the Graph API. The Graph API is the API used to read and write to Facebook accounts. See `https://developers.facebook.com/docs/reference/api/` for more information.

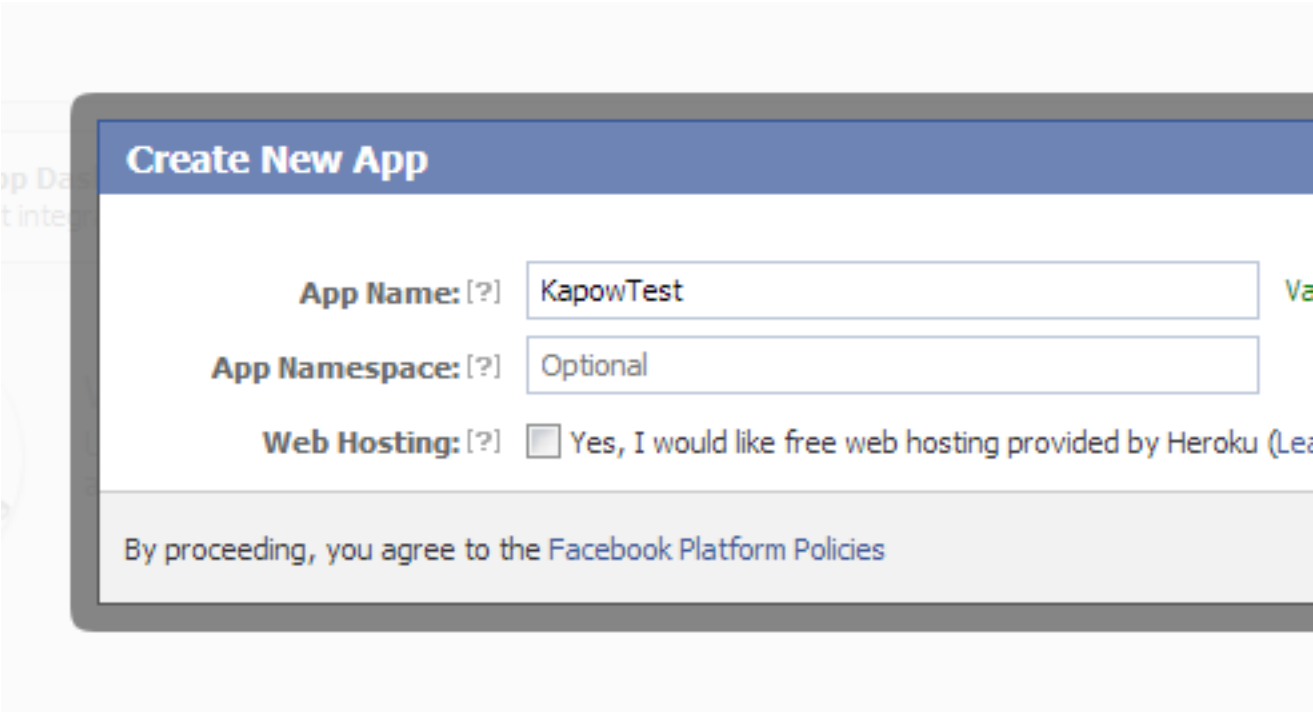
First, log into http://developers.facebook.com with your Facebook account, or create a new account if necessary. Click "Apps" in the bar at the top of the page.



Then, click "Create New App".

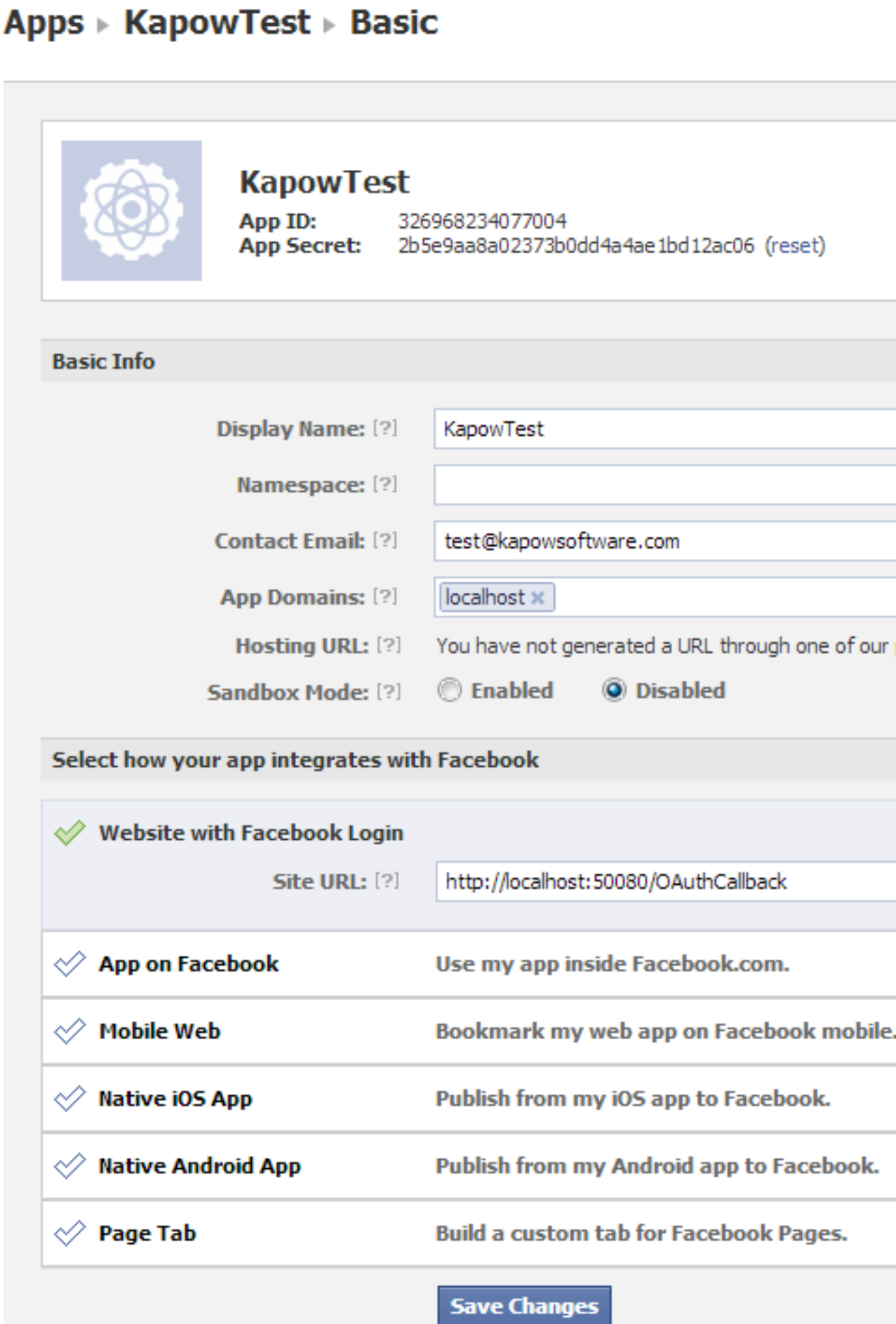


Type in a name for the new app and click Continue.

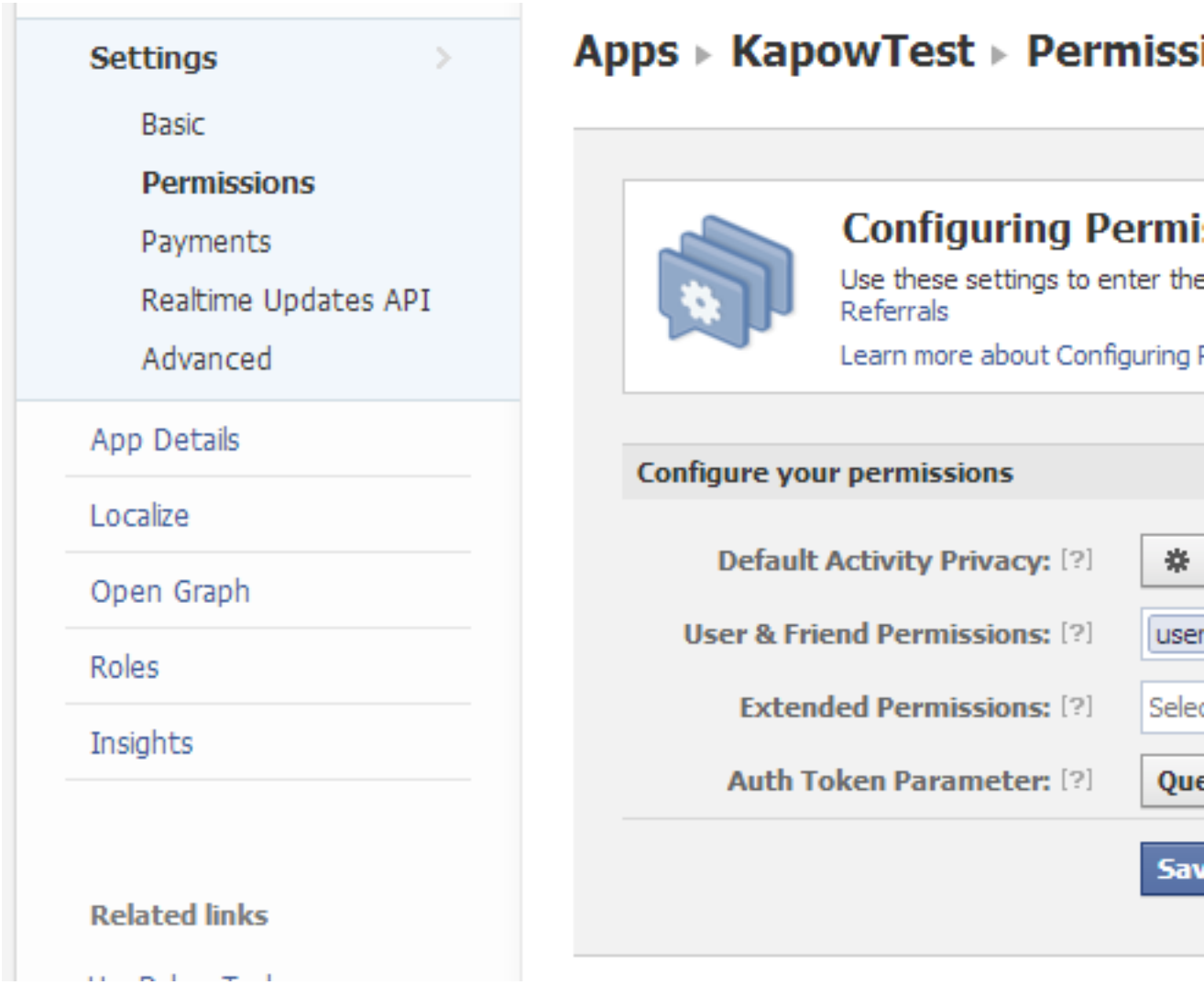


A configuration screen now appears for the new app. One of the fields is called App Domains and is used to state any domains used by the app. This must be set to the domain which hosts the Management Console. If running with an embedded Management Console write "localhost" in this field. At the bottom of the page it is possible to select how the app integrates with Facebook. Choose "Website with Facebook Login" and write the OAuth callback URL as Site URL. The callback URL is the path `OAuthCallback` under the folder in which the Management Console is deployed. For instance, if running with an embedded Management Console, it runs at `http://localhost:50080/`. In this case, the callback URL would be specified to `http://localhost:50080/OAuthCallback`.

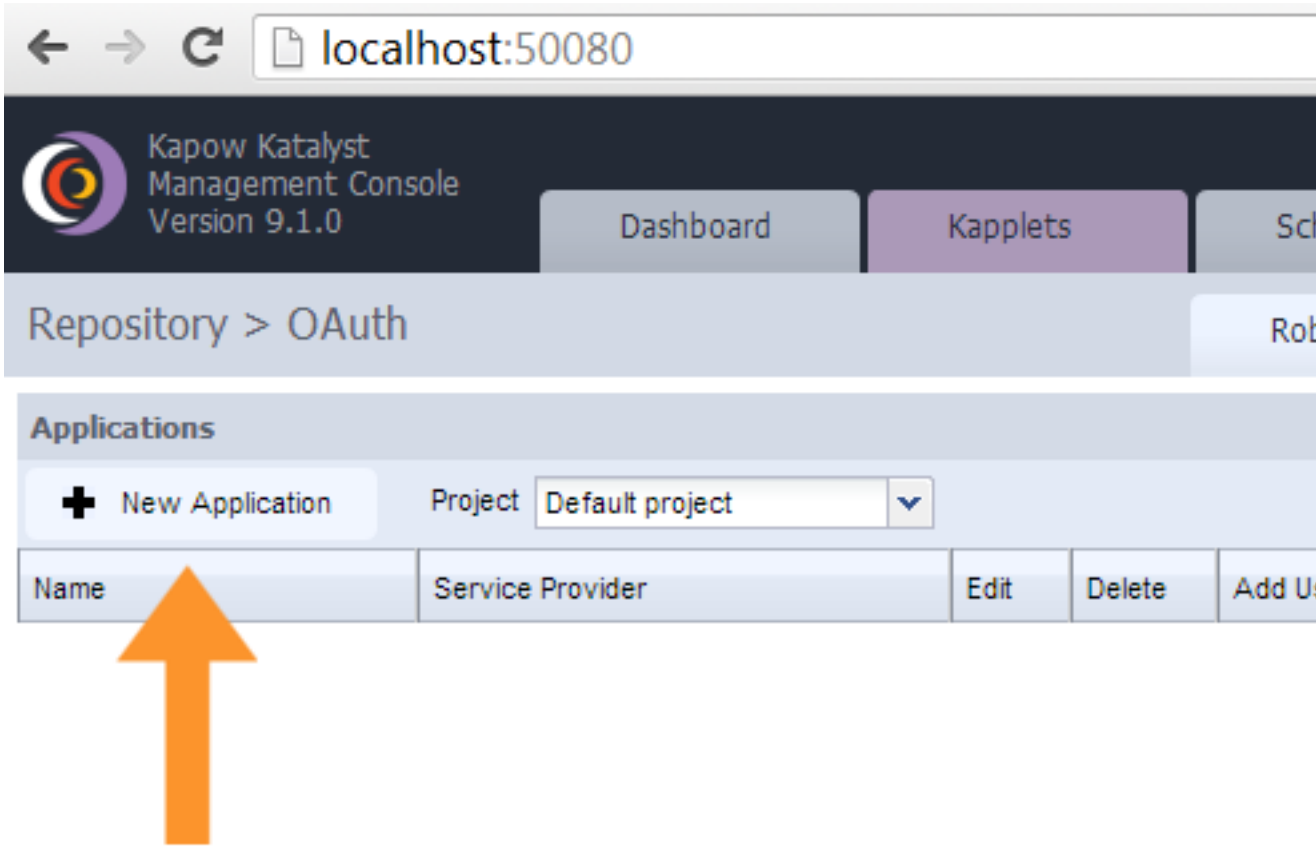
On this page it is also important to take note of the App ID and App Secret which will be used as Consumer Key and Consumer Secret in the Management Console later on.


Apps ▸ KapowTest ▸ Basic
KapowTest
App ID: 326968234077004
App Secret: 2b5e9aa8a02373b0dd4a4ae1bd12ac06 (reset)
Basic Info
Display Name: [?] KapowTest
Namespace: [?]
Contact Email: [?] test@kapowsoftware.com
App Domains: [?] localhost
Hosting URL: [?] You have not generated a URL through one of our
Sandbox Mode: [?] Enabled Disabled
Select how your app integrates with Facebook
Website with Facebook Login
Site URL: [?] http://localhost:50080/OAuthCallback
App on Facebook Use my app inside Facebook.com.
Mobile Web Bookmark my web app on Facebook mobile.
Native iOS App Publish from my iOS app to Facebook.
Native Android App Publish from my Android app to Facebook.
Page Tab Build a custom tab for Facebook Pages.
Save Changes

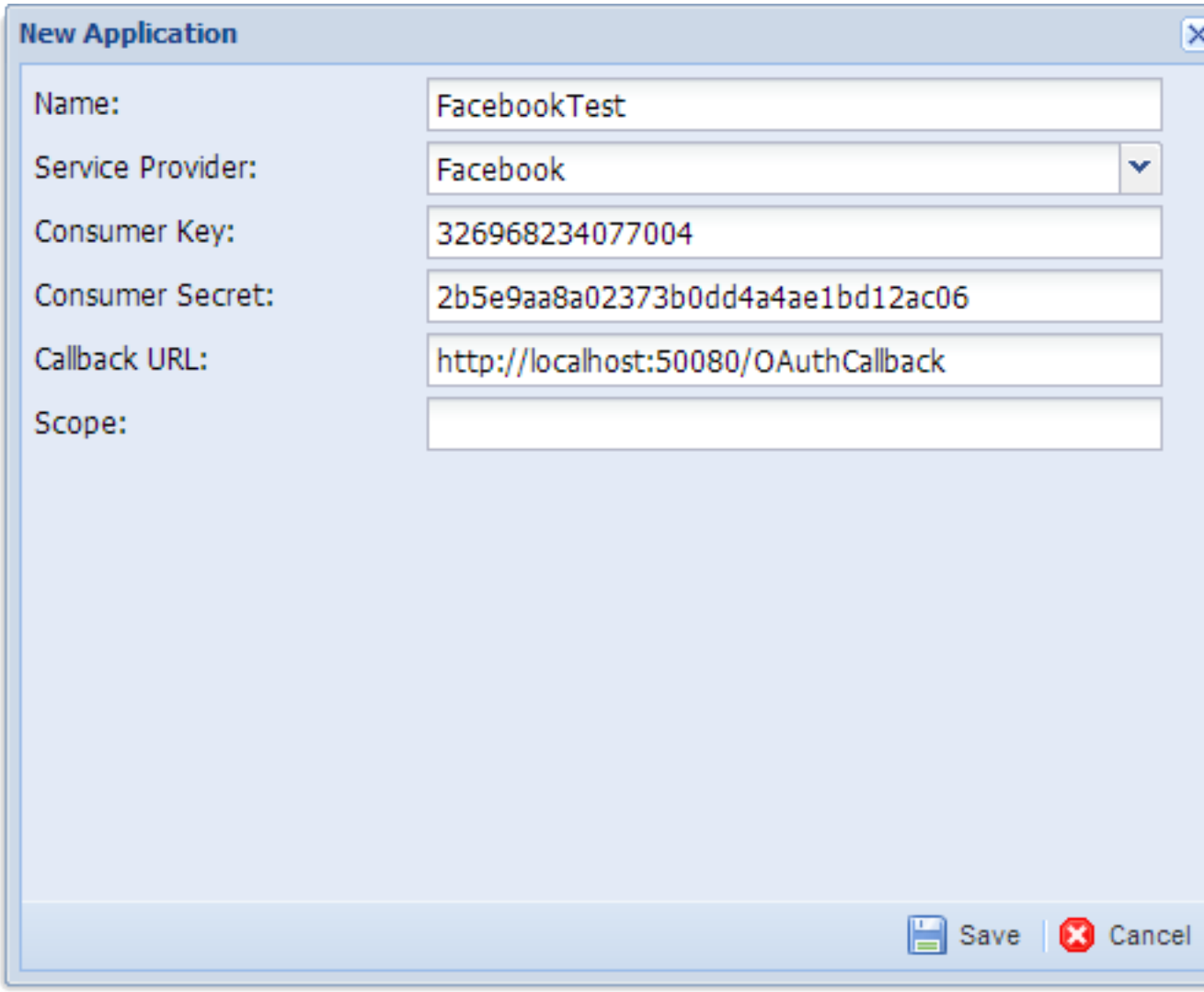
After saving the basic configuration of the app, go to the Permission Configuration by clicking "Permissions" in the menu on the left side of the page. Here it is possible to configure which permissions the app requires when being authenticated. For basic user information type "user_about_me" in the "User & Friend Permission" field. For more information on the possible permissions, please visit https://developers.facebook.com/docs/reference/login/.



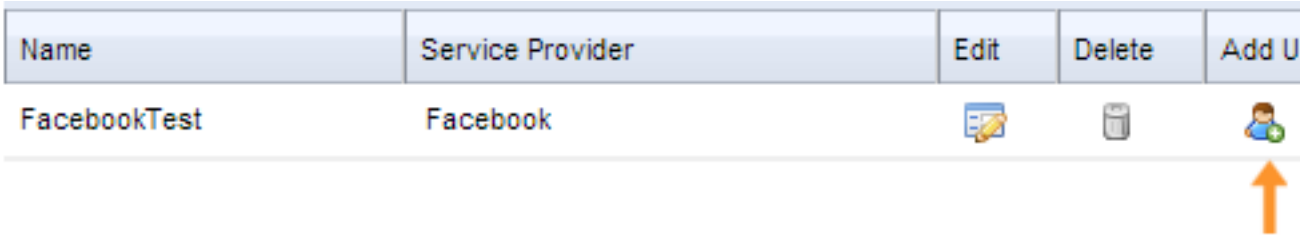
Once the app has been set up and any permissions have been saved, open the Management Console. Make sure to open the Management Console at the URL specified for the app. That would be `http://localhost:50080` for the information used above. Then navigate to the OAuth Repository and click "New Application".



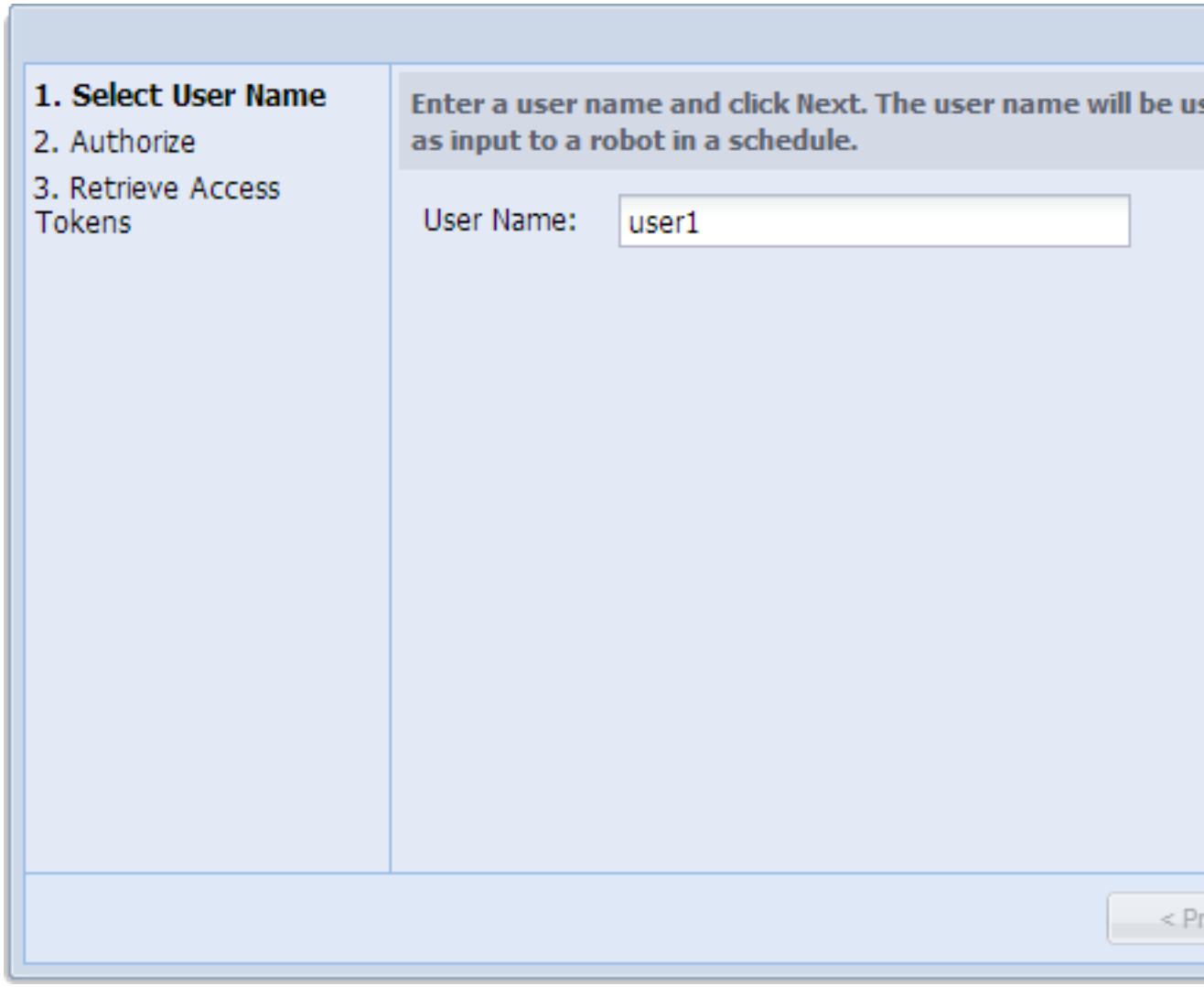
Select a name for the new application and choose Facebook as the service provider. Consumer Key and Consumer Secret should respectively be set to the App ID and App Secret noted earlier. The Callback URL should already be correctly stated.



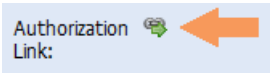
After being saved, the application appears in the repository. Click on the icon under "Add User" to add a user to the application.



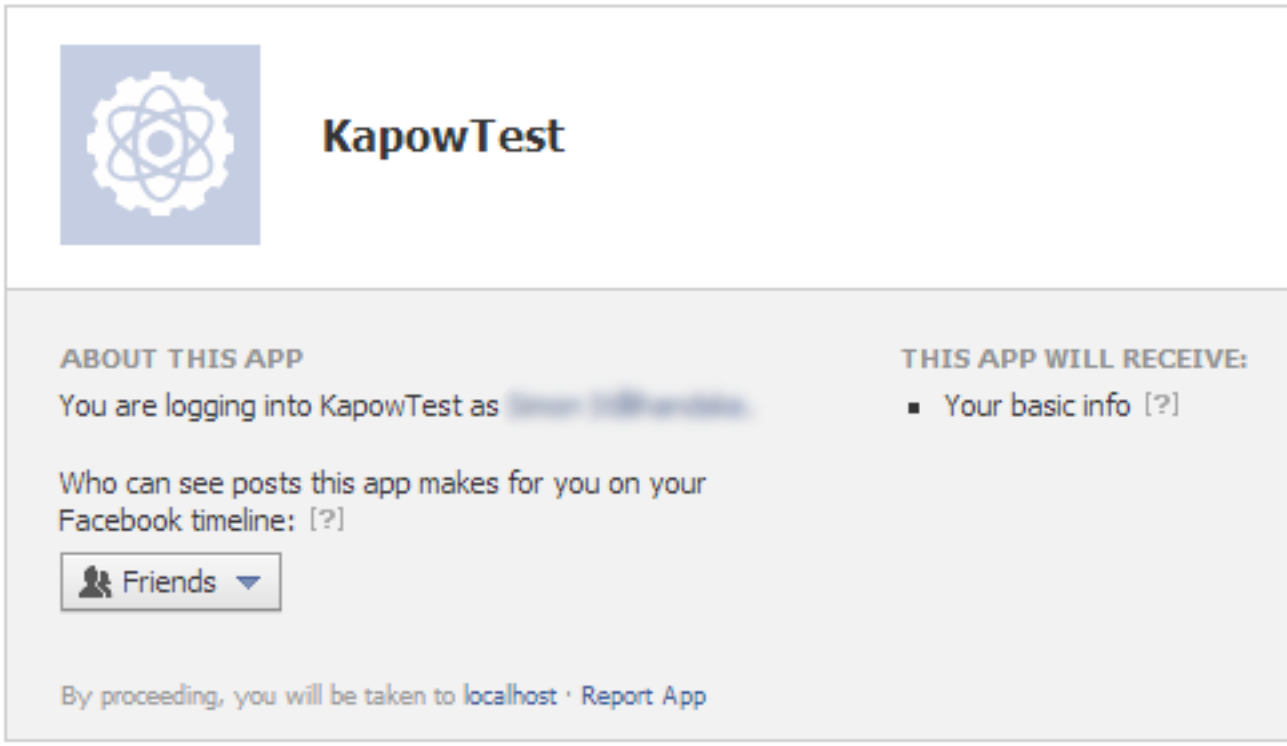
Choose a name for the user and click next.



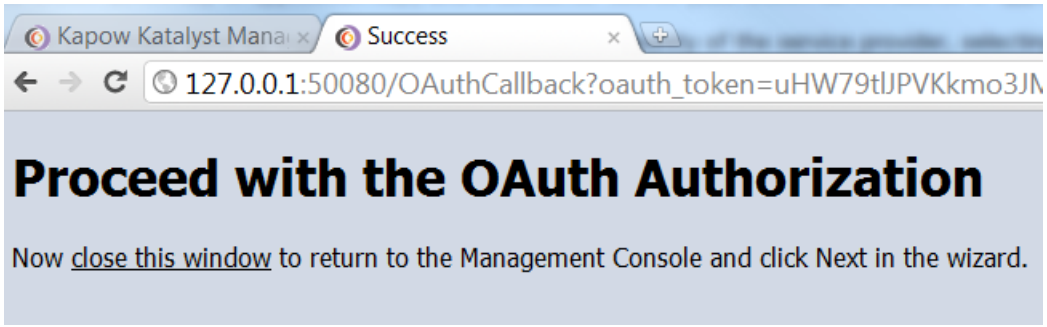
Now, click the authorization link.



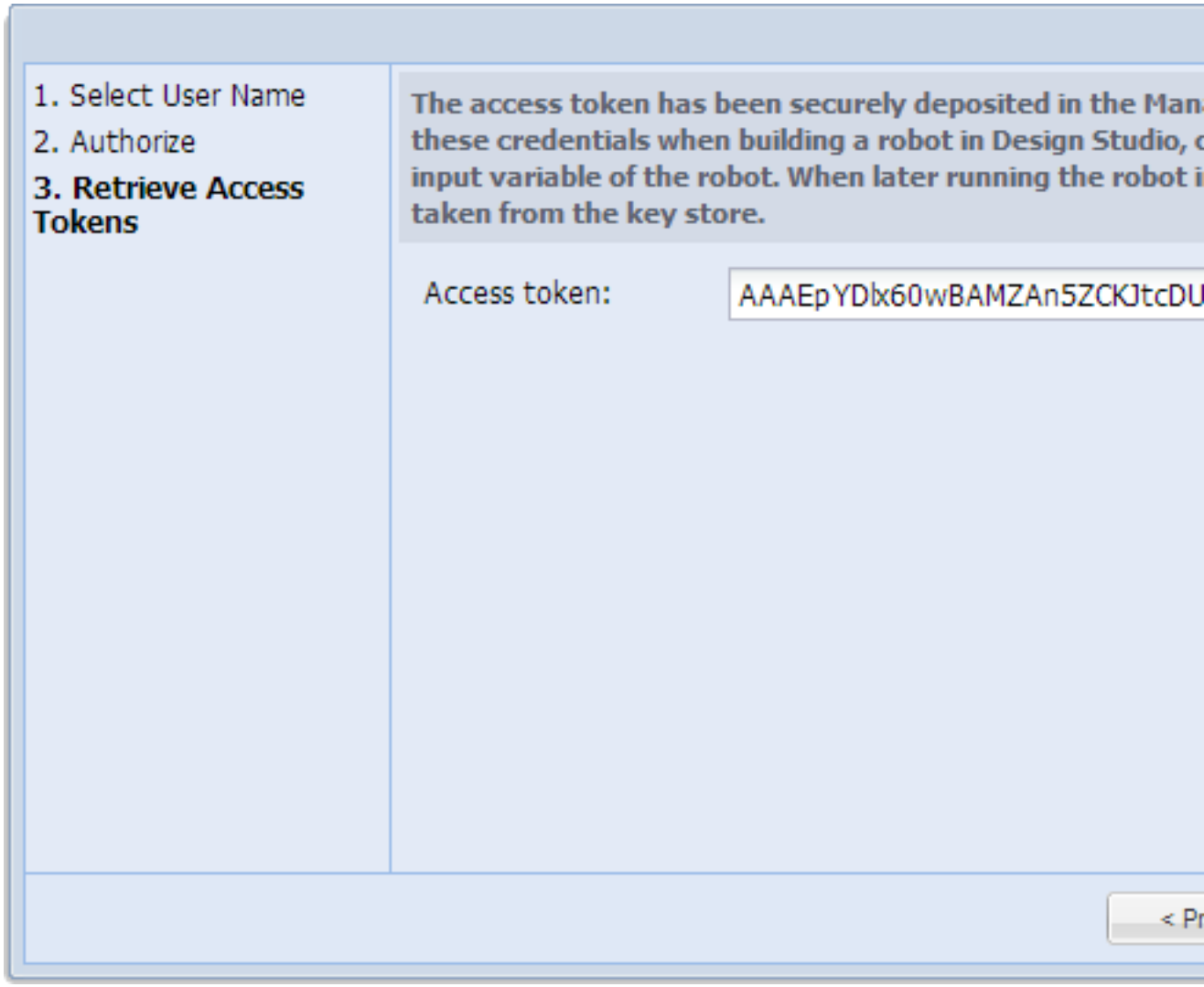
This will open a page where the app will request permission to access your Facebook account.



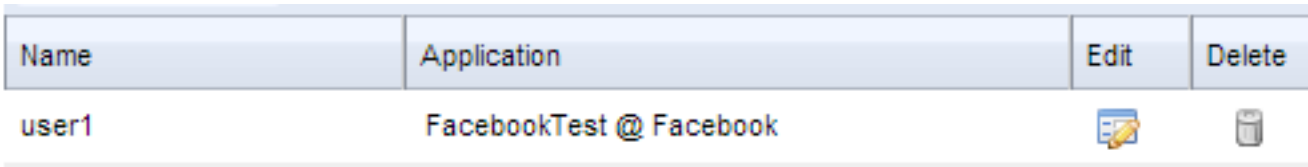
Authorizing the app will forward you to the OAuth callback page of the Management Console.



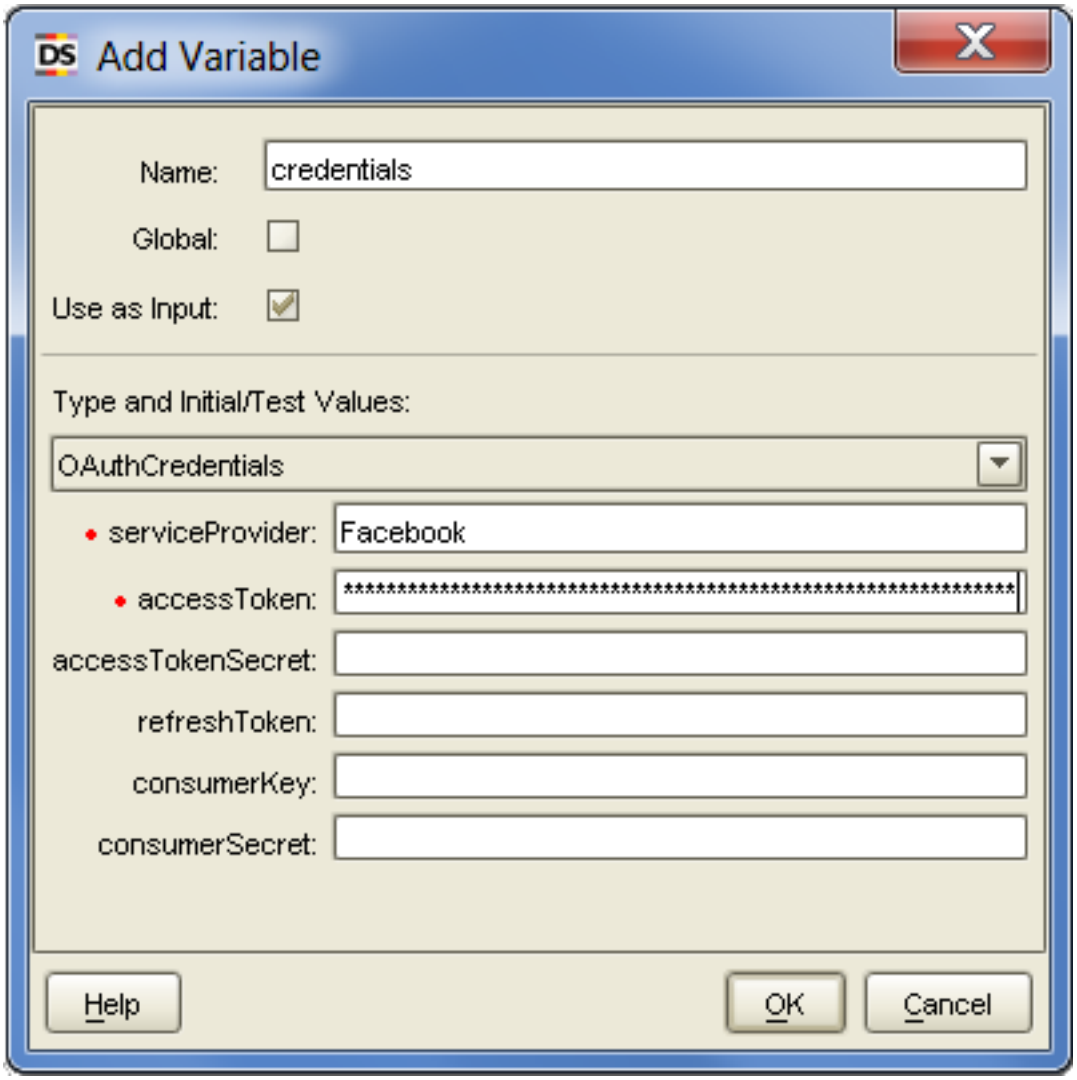
Close the OAuth callback page, returning to the Management Console. Click Next in the wizard and you will see the access token that can be used for accessing Facebook on the user's behalf. Copy this token into a notepad since it for security reasons not will be accessible later.



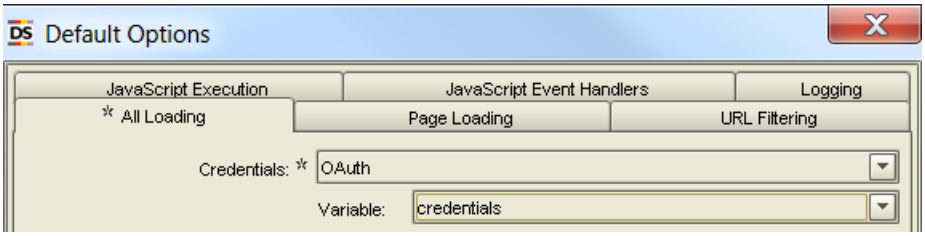
After clicking Finish, you should now see a user in the Users section of the OAuth tab.



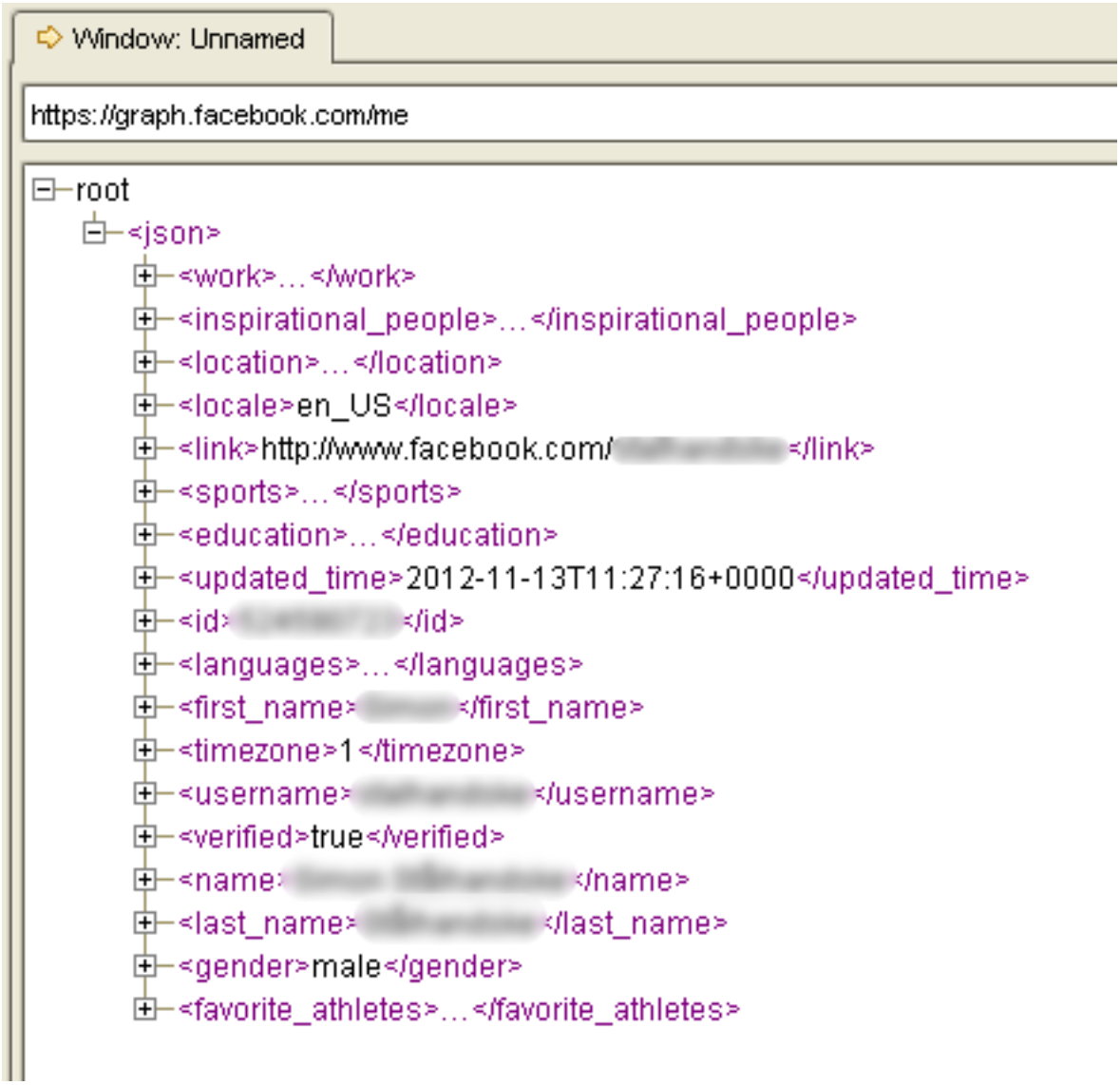
Now, open Design Studio and create a new robot or open the robot you would like to access the Facebook API. For the robot to be able to use the Facebook Graph API, make sure that it has an input variable of the complex type "OAuthCredentials" which contains "Facebook" as the service provider name and the Access Token copied from the Management Console earlier.



Then, make sure to configure the robot to use OAuth credentials.



The robot should now be able to access the Facebook Graph API. As an example try typing `https://graph.facebook.com/me` in the address bar of the browser view and press Enter.

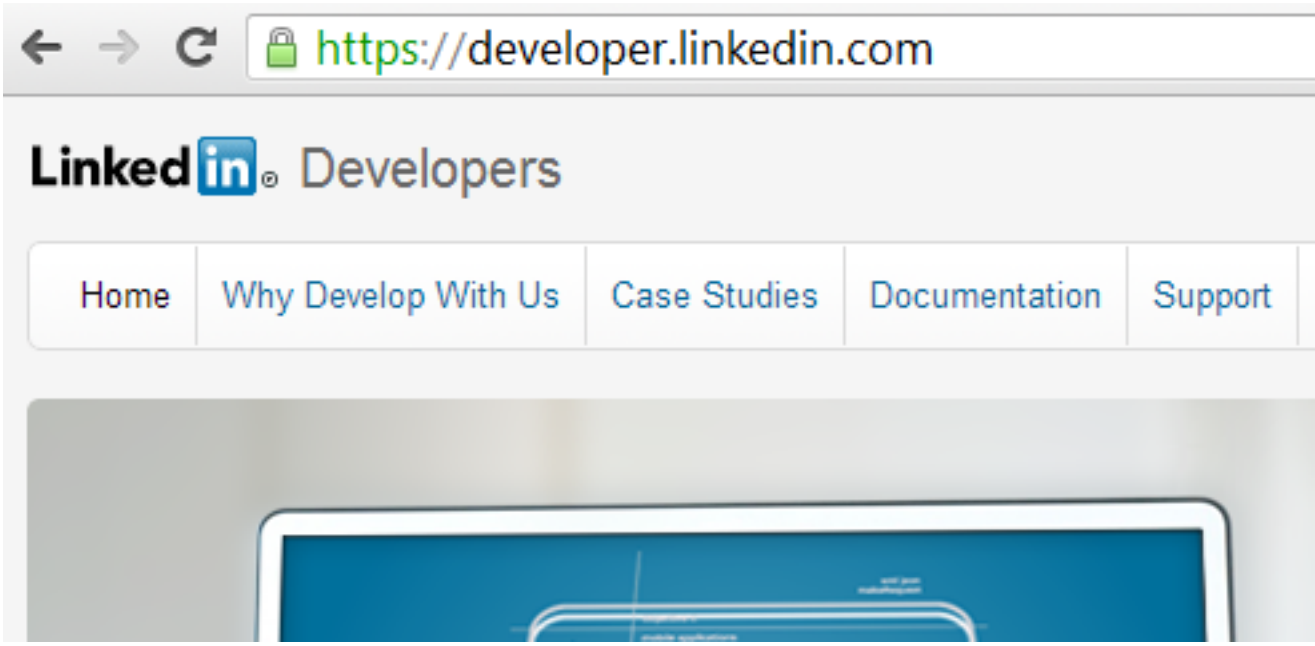


To learn more about the Facebook Graph API visit https://developers.facebook.com/docs/getting-started/graphapi/. Once a robot has been built, it is possible to schedule execution of the robot in the Management Console.

LinkedIn

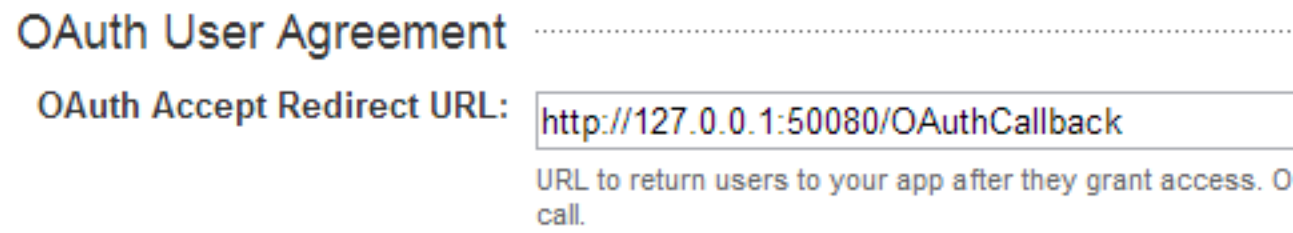
This is a short guide on how to create a robot which accesses LinkedIn through the REST API, which can be used to read and write to LinkedIn accounts.

First, log into http://developer.linkedin.com with your LinkedIn account, or create a new account if necessary. Hover over your name in the upper right corner and click "API Keys" in the menu that appears.

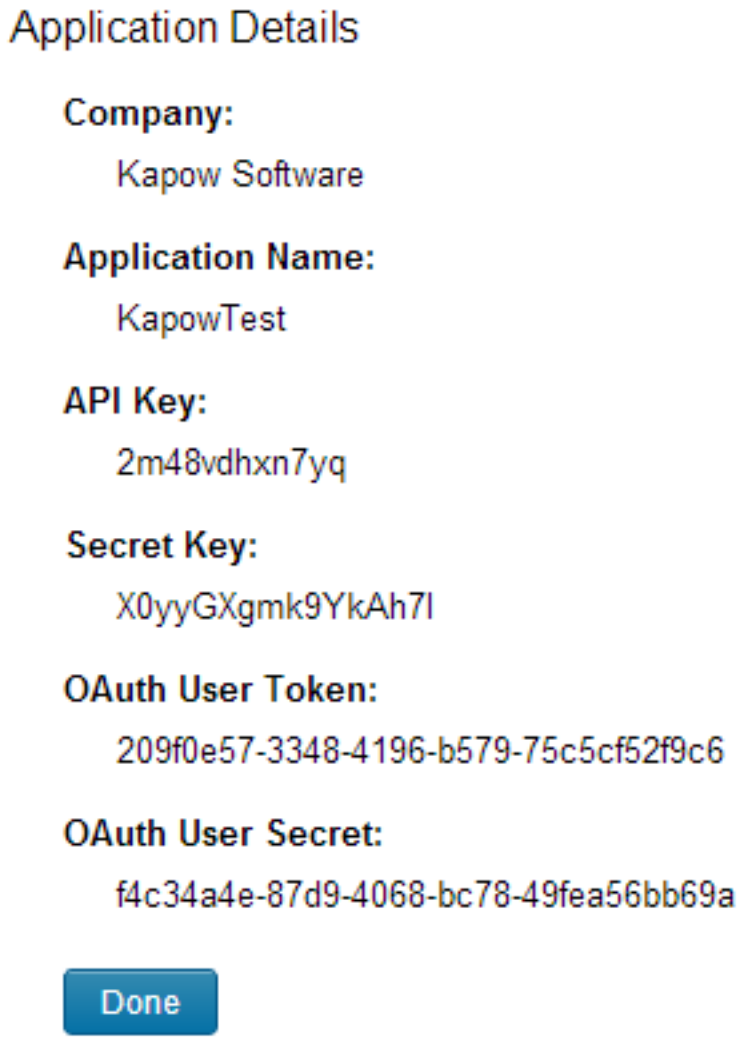


Then, click "Add New Application".

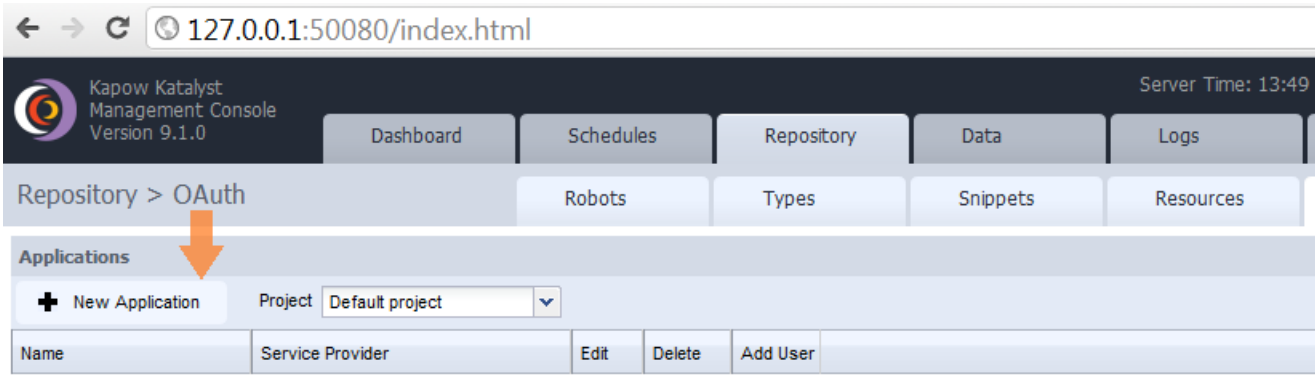
A configuration screen now appears for the new app. Fill in the form with relevant information. One of the fields is called "OAuth Accept Redirect URL". Write the OAuth callback URL in this field. The callback URL is the path `OAuthCallback` under the folder in which the Management Console is deployed. For instance, if running with an embedded Management Console, it runs at `http://localhost:50080/` or similarly `http://127.0.0.1:50080/`. In the latter case, the callback URL is specified to `http://127.0.0.1:50080/OAuthCallback`.



After reading and accepting the LinkedIn API Terms of Use and adding the application, a page with Application details is opened. On this page it is important to take note of the API Key and Secret Key which will be used as Consumer Key and Consumer Secret in the Management Console later on. The page also gives you a User Token and User Secret, but we can ignore these as we will be creating our own.

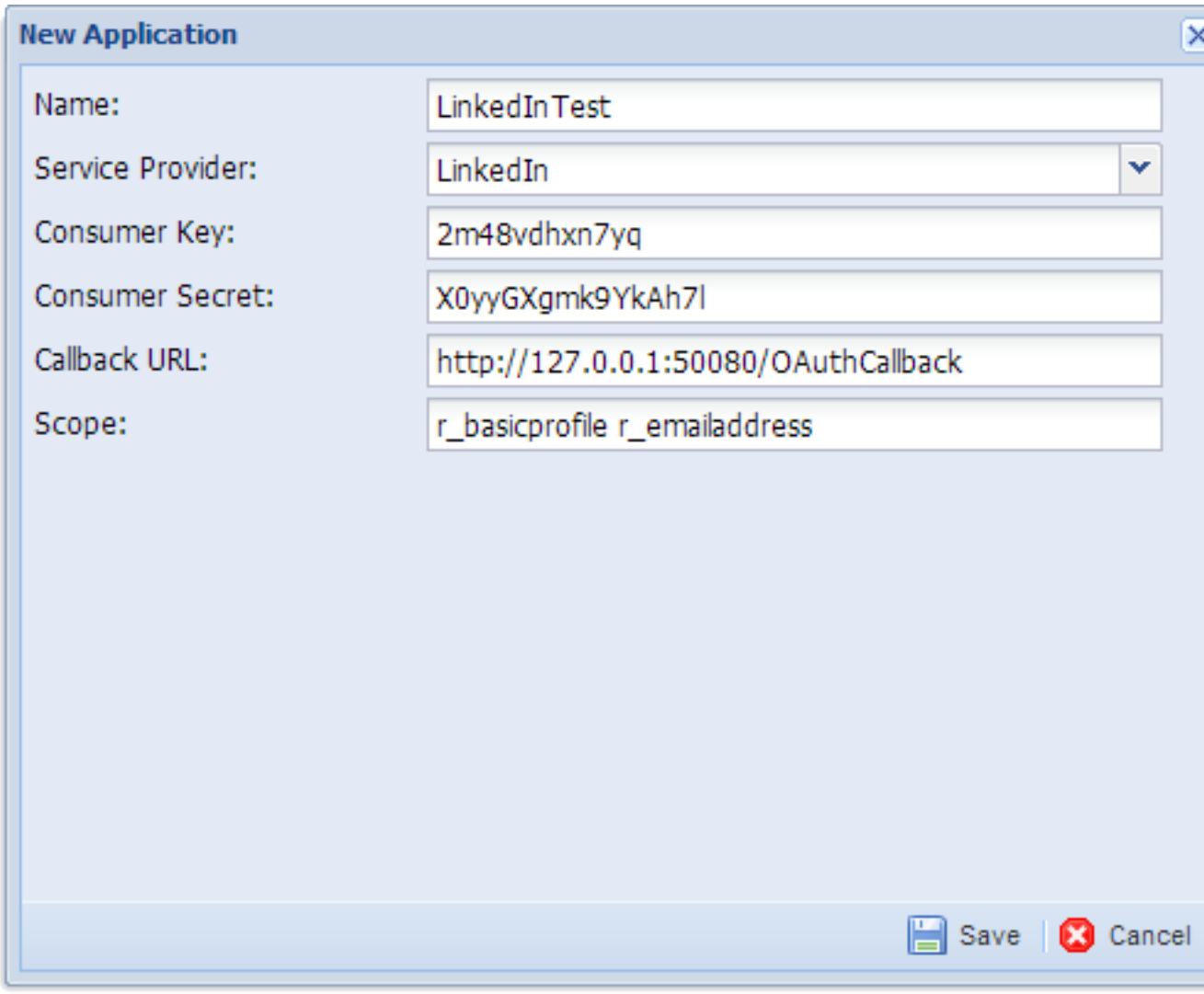


Now, open the Management Console. Make sure to open the Management Console at the URL specified for the app. That would be `http://127.0.0.1:50080` for the information used above. Then navigate to the OAuth Repository and click "New Application".

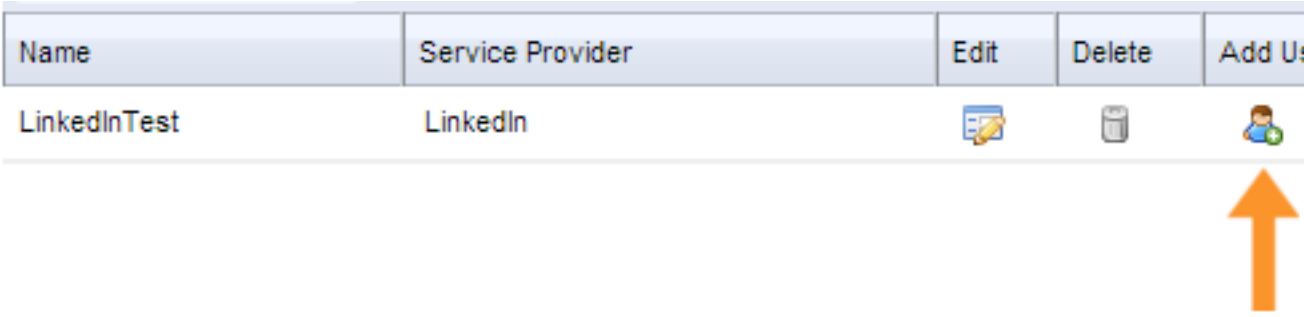


Select a name for the new application and choose LinkedIn as the service provider. Consumer Key and Consumer Secret should respectively be set to the API Key and Secret Key noted earlier. The Callback URL should already be correctly stated.

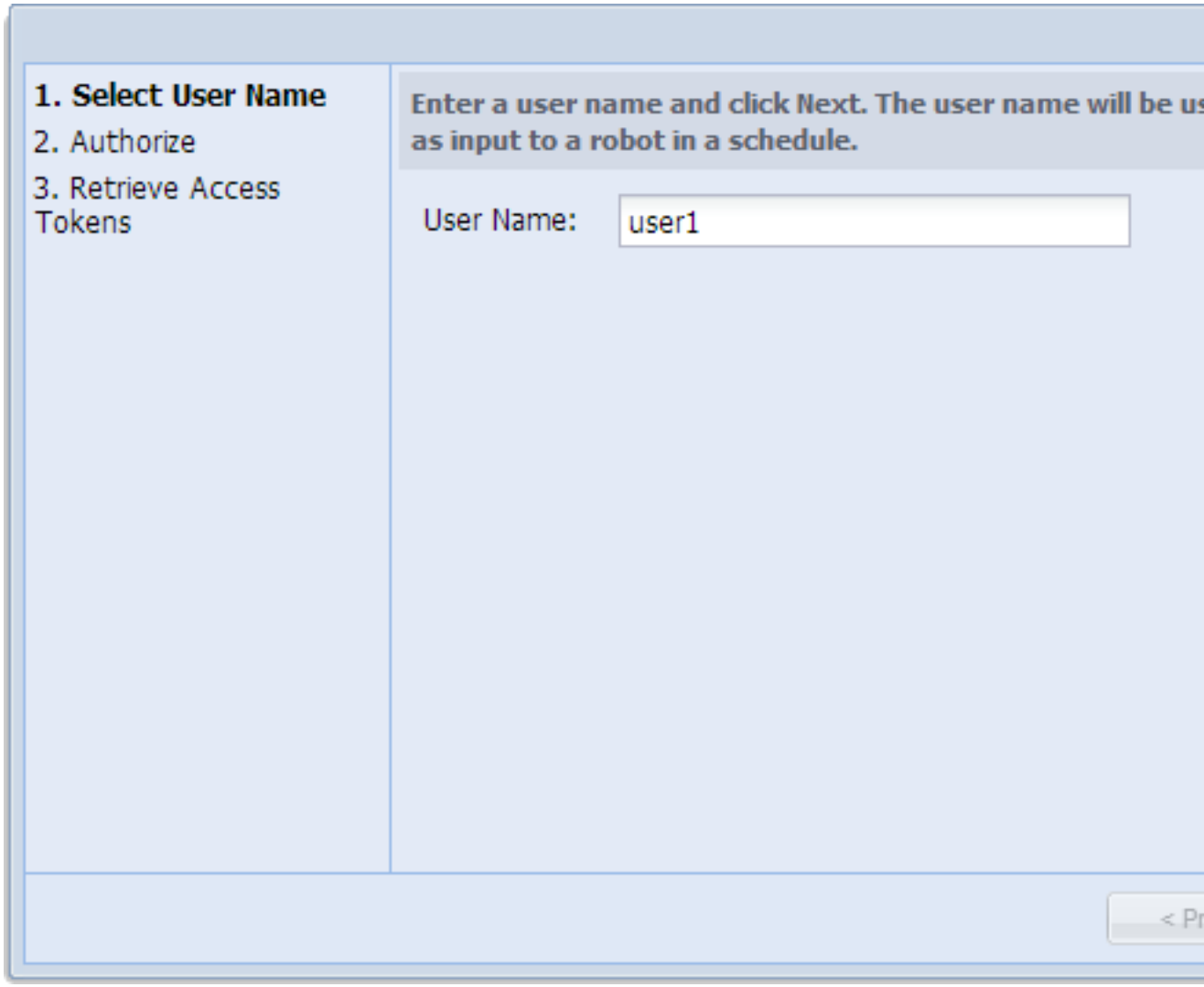
In the Scope field, we state the kinds of permissions required from the app user. In the picture below two different permissions have been requested: "r_basicprofile", which gives permission to read basic profile information and "r_emailadress", giving permission to access the user's email address. For more scope options please consult https://developer.linkedin.com/documents/authentication#granting.



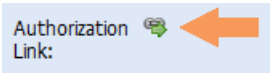
After being saved, the application appears in the repository. Click on the icon under "Add User" to add a user to the application.



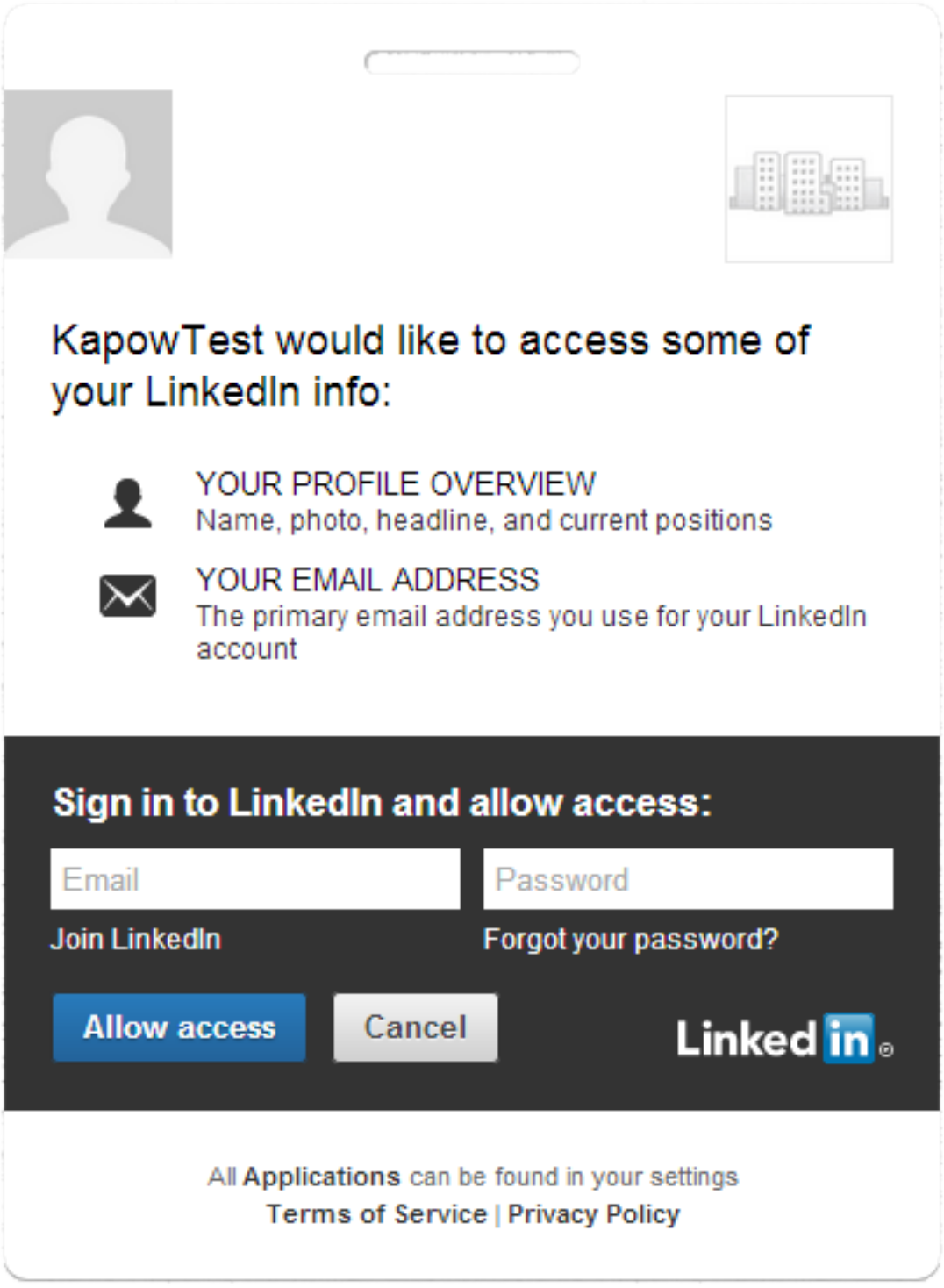
Choose a name for the user and click next.



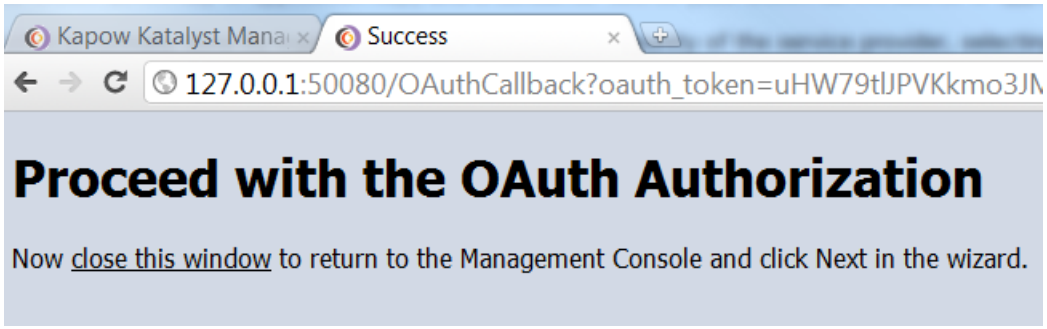
Now, click the authorization link.



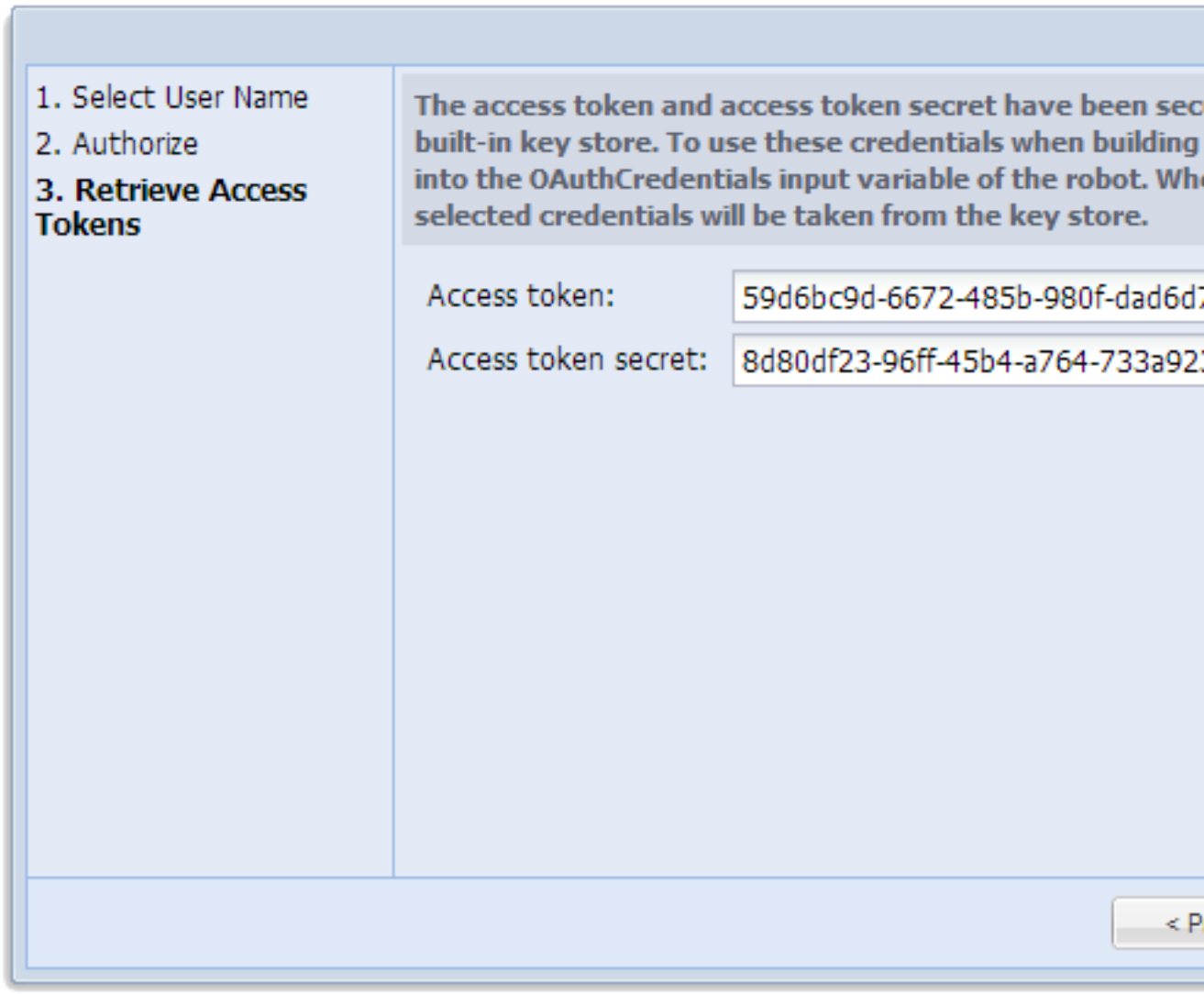
This will open a page where the app will request permission to access your LinkedIn account, specifying the two permissions requested.



Authorizing the app will forward you to the OAuth callback page of the Management Console.



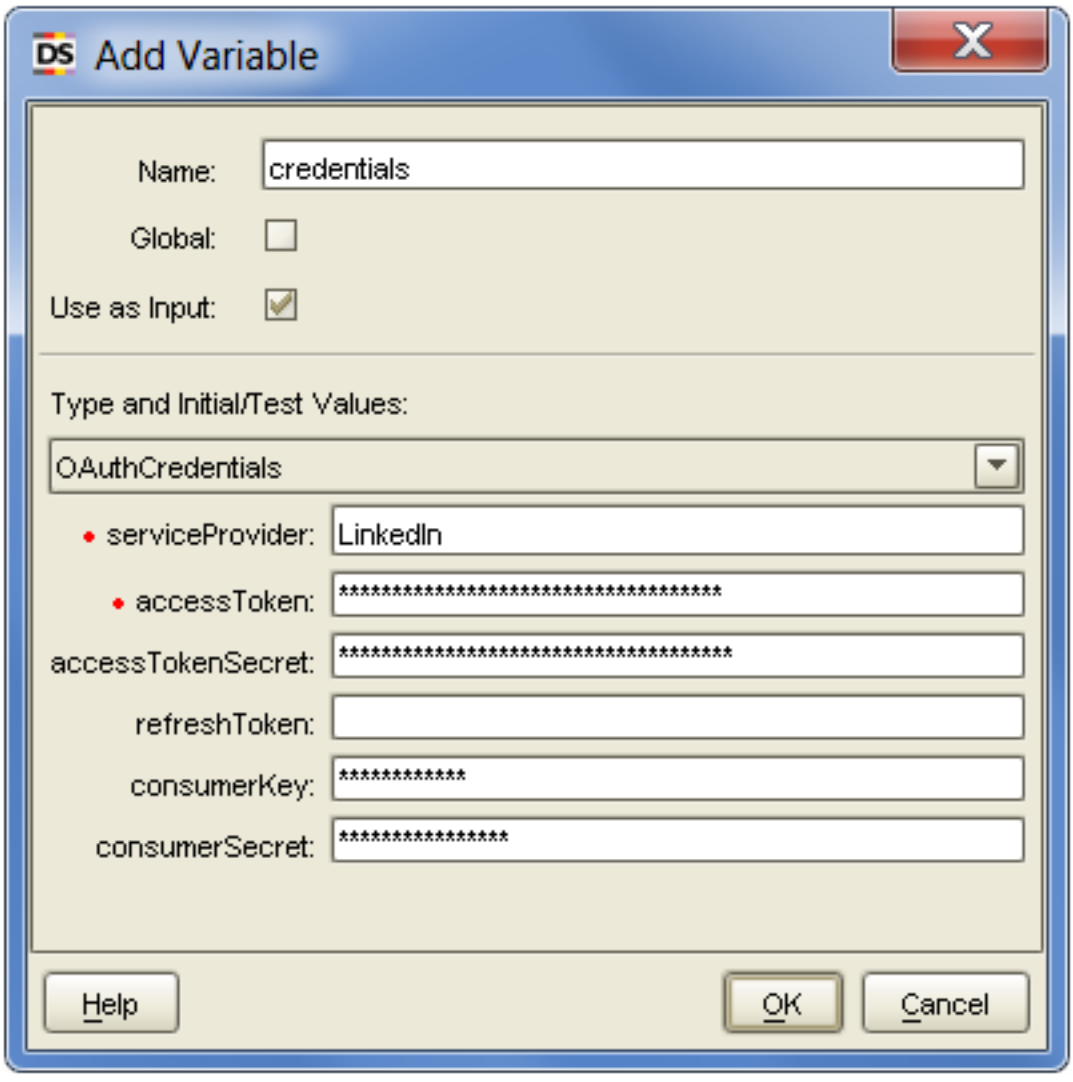
Close the OAuth callback page, returning to the Management Console. Click Next in the wizard and you will see the access token and access token secret granted for accessing LinkedIn on the user's behalf. Copy these tokens into a notepad, since they will not be accessible later for security reasons.



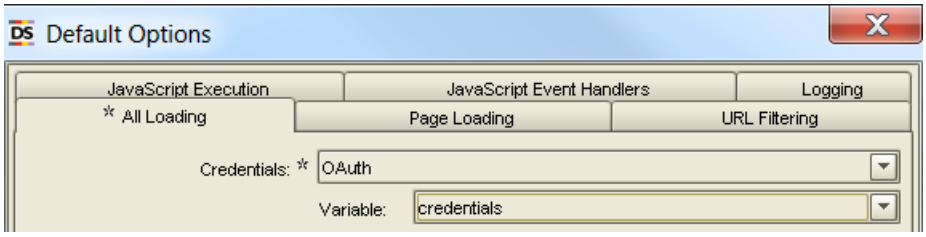
After clicking Finish, you should now see a user in the Users section of the OAuth tab.

Name	Application	Edit	Delete
user1	LinkedInTest @ LinkedIn		

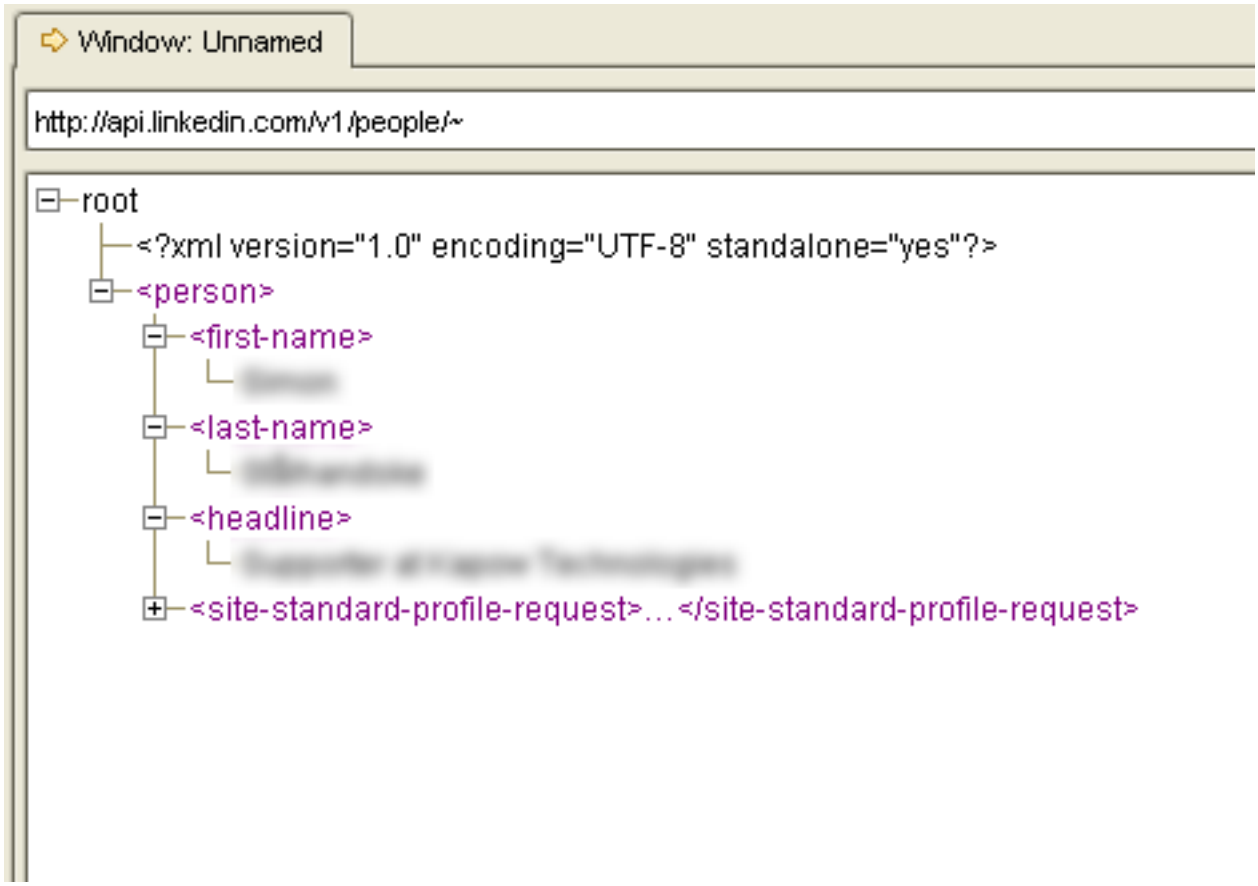
Now, open Design Studio and create a new robot or open the robot you would like to access the LinkedIn API. For the robot to be able to use the LinkedIn API, make sure that it has an input variable of the complex type "OAuthCredentials". The variable should have "LinkedIn" as the service provider name. Also, input the access token and access token secret along with the API Key and Secret Key noted earlier. The latter two should be used as consumer key and consumer secret respectively.



Then, make sure to configure the robot to use OAuth credentials.



The robot should now be able to access the LinkedIn API. As an example try typing `http://api.linkedin.com/v1/people/~` in the address bar of the browser view and pressing Enter. This should return an xml page with basic user information. Also try `http://api.linkedin.com/v1/people/~/email-address`, which returns the user email address.

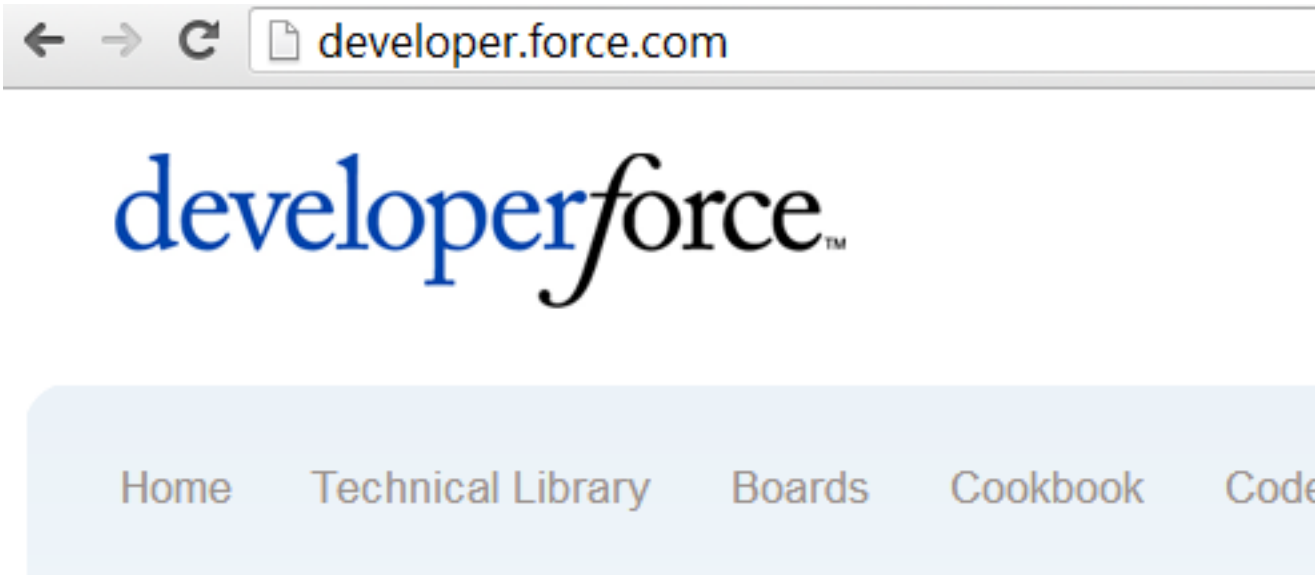


To learn more about the LinkedIn REST API visit https://developer.linkedin.com/rest. Once a robot has been built, it is possible to schedule execution of the robot in the Management Console.

Salesforce

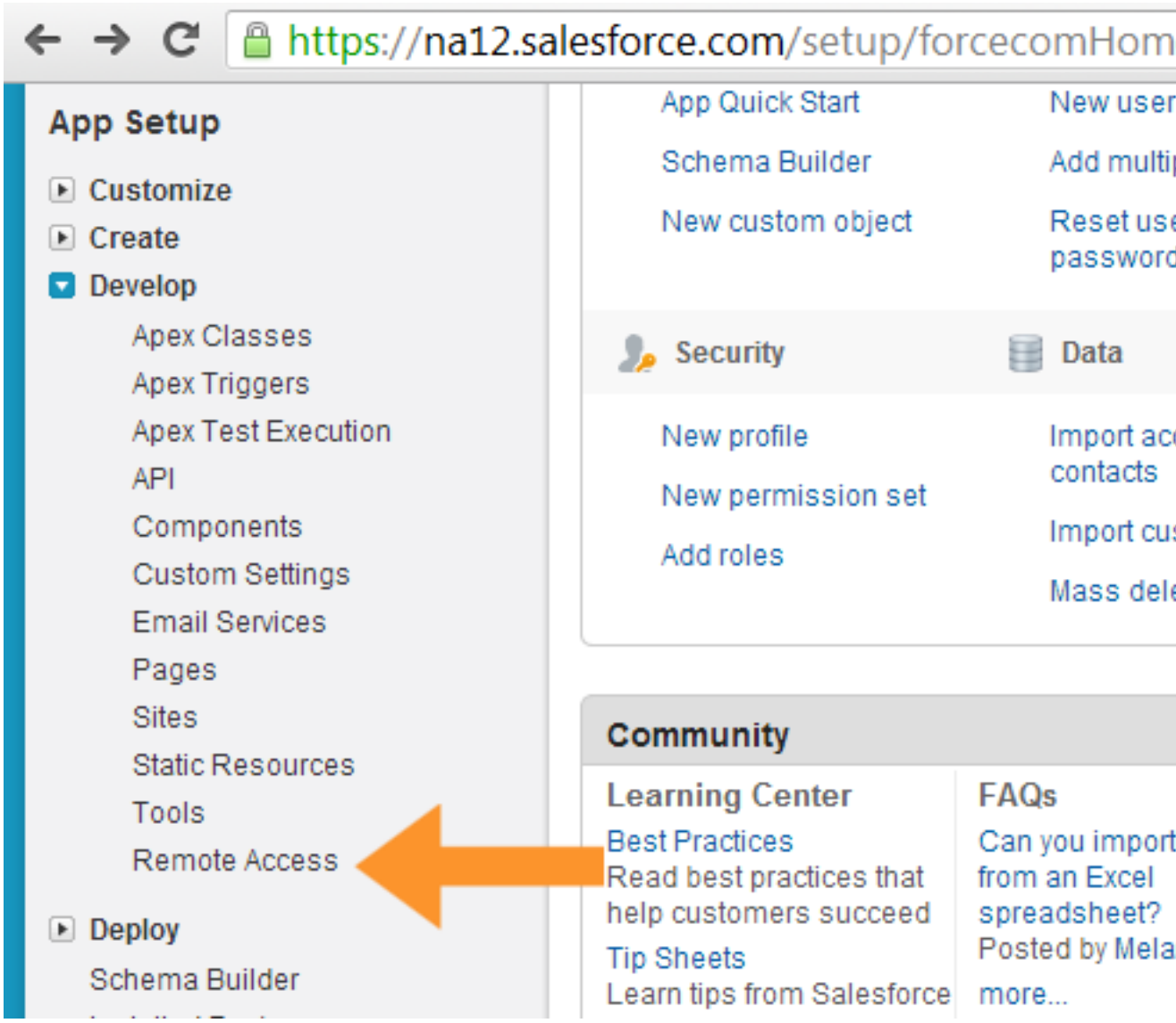
This is a short guide on how to create a robot which accesses Salesforce using the REST API and OAuth.

First, log into http://developer.force.com with your Salesforce account by clicking "DE LOGIN" at the top of the page. Create a new account if necessary.

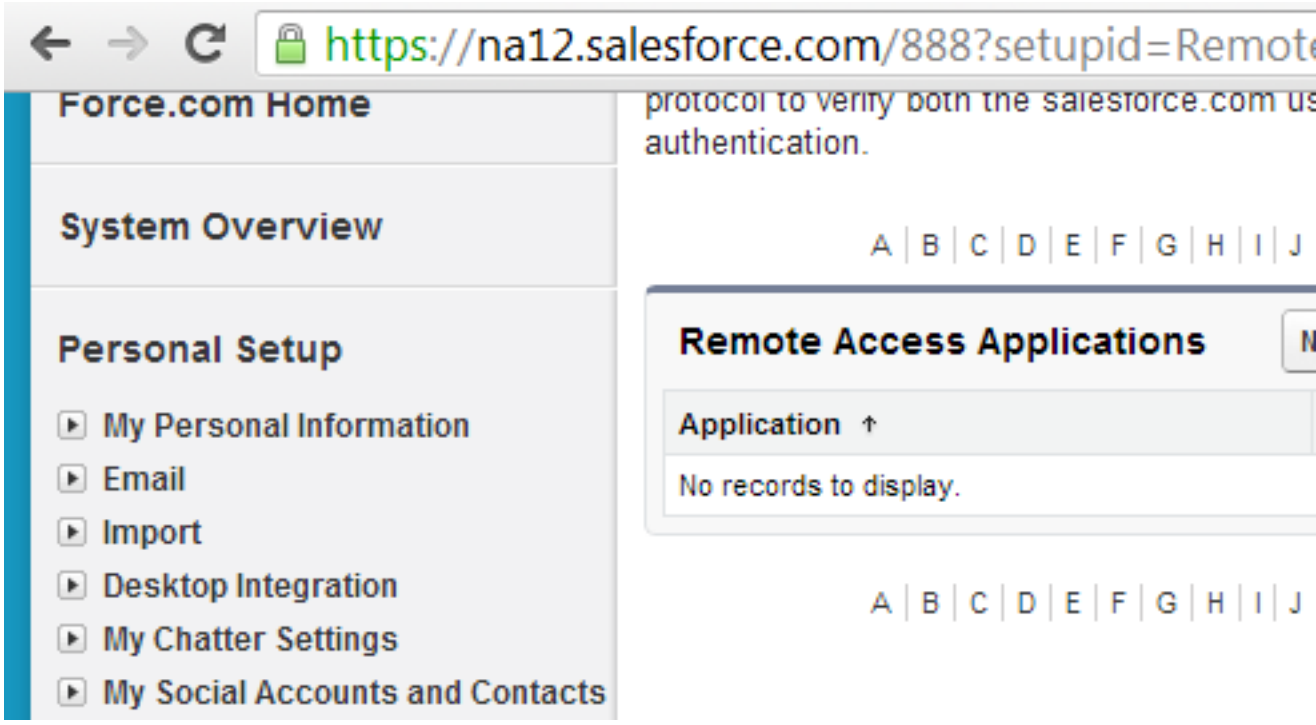


On the home page of your account, take note of the "instance" on which your account runs. The instance is given in the server name part of the URL. In the case shown below, the instance is "na12".

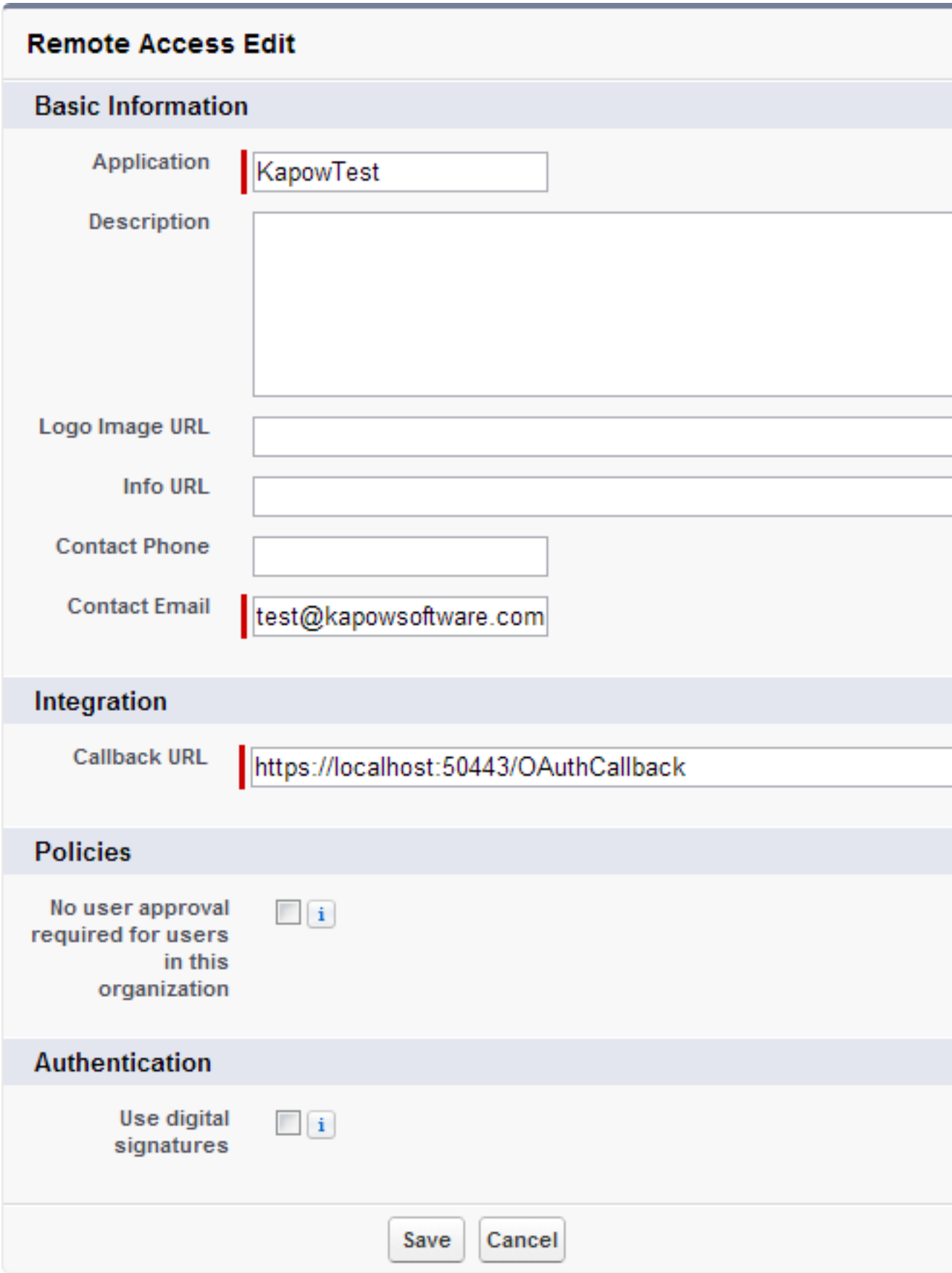
Then choose "Remote Access" from the menu on the left. It can be found under "App Setup" >> "Develop".



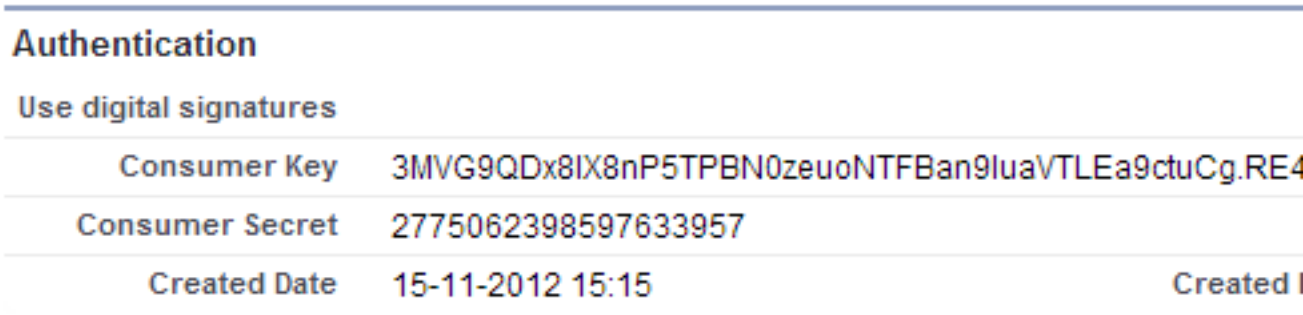
Now, create a new Remote Access Application. This application will give access to the APIs of Salesforce through OAuth.



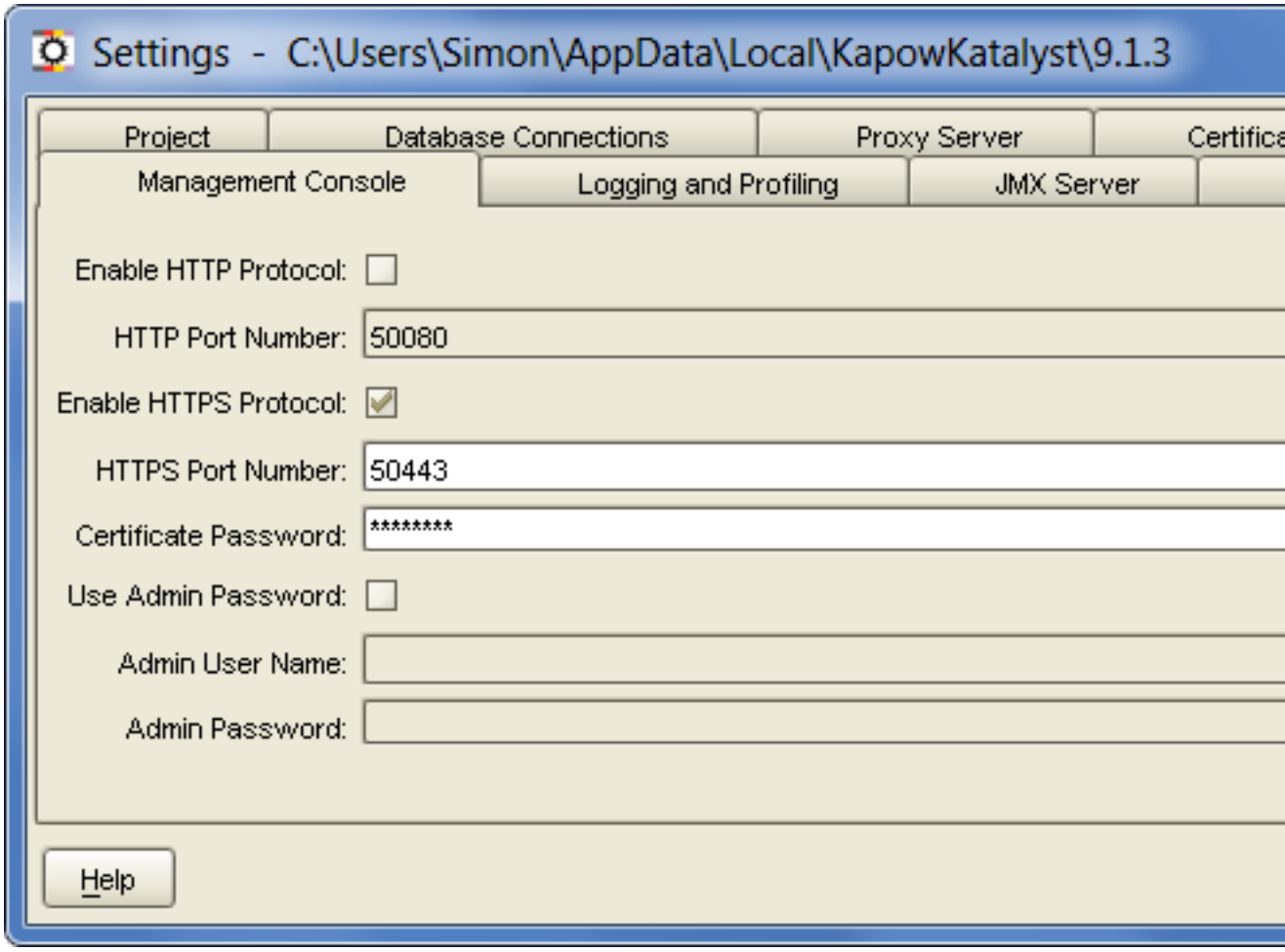
A configuration screen now appears for the new app. Fill in the form with relevant information. One of the fields is called "Callback URL". Write the OAuth callback URL in this field. Salesforce requires use of HTTPS so the callback URL is therefore directed to `https://localhost:50443/OAuthCallback`. As we will see in a moment, `50443` is the default port when setting up an embedded Management Console to use HTTPS.


Remote Access Edit
Basic Information
Application
KapowTest
Description
Logo Image URL
Info URL
Contact Phone
Contact Email
test@kapowsoftware.com
Integration
Callback URL
https://localhost:50443/OAuthCallback
Policies
No user approval required for users in this organization
Authentication
Use digital signatures
Save
Cancel

After clicking save, a page with application details is opened. On this page it is important to take note of the Consumer Key and Consumer Secret which will be used in the Management Console later on.

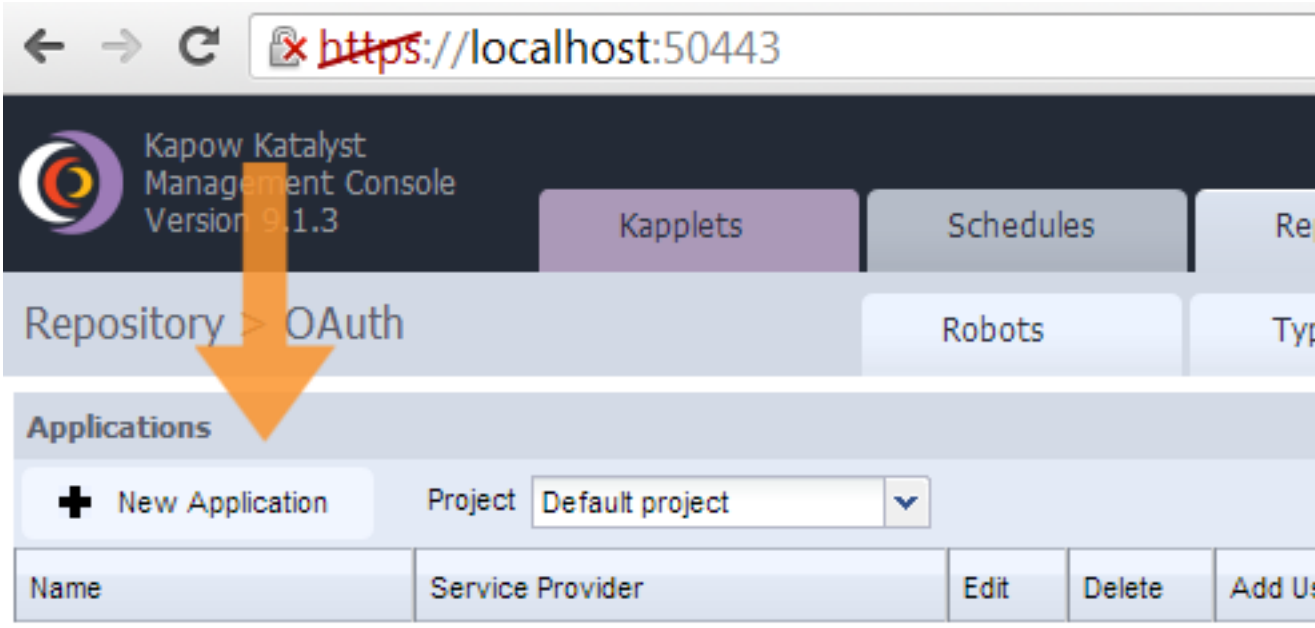


To set up a local Management Console to use the HTTPS protocol, open up Settings. On Windows, Settings can be found in the Start Menu, in the same folder as Design Studio. In Settings, go to the Management Console tab, disable the HTTP protocol and enable HTTPS instead.

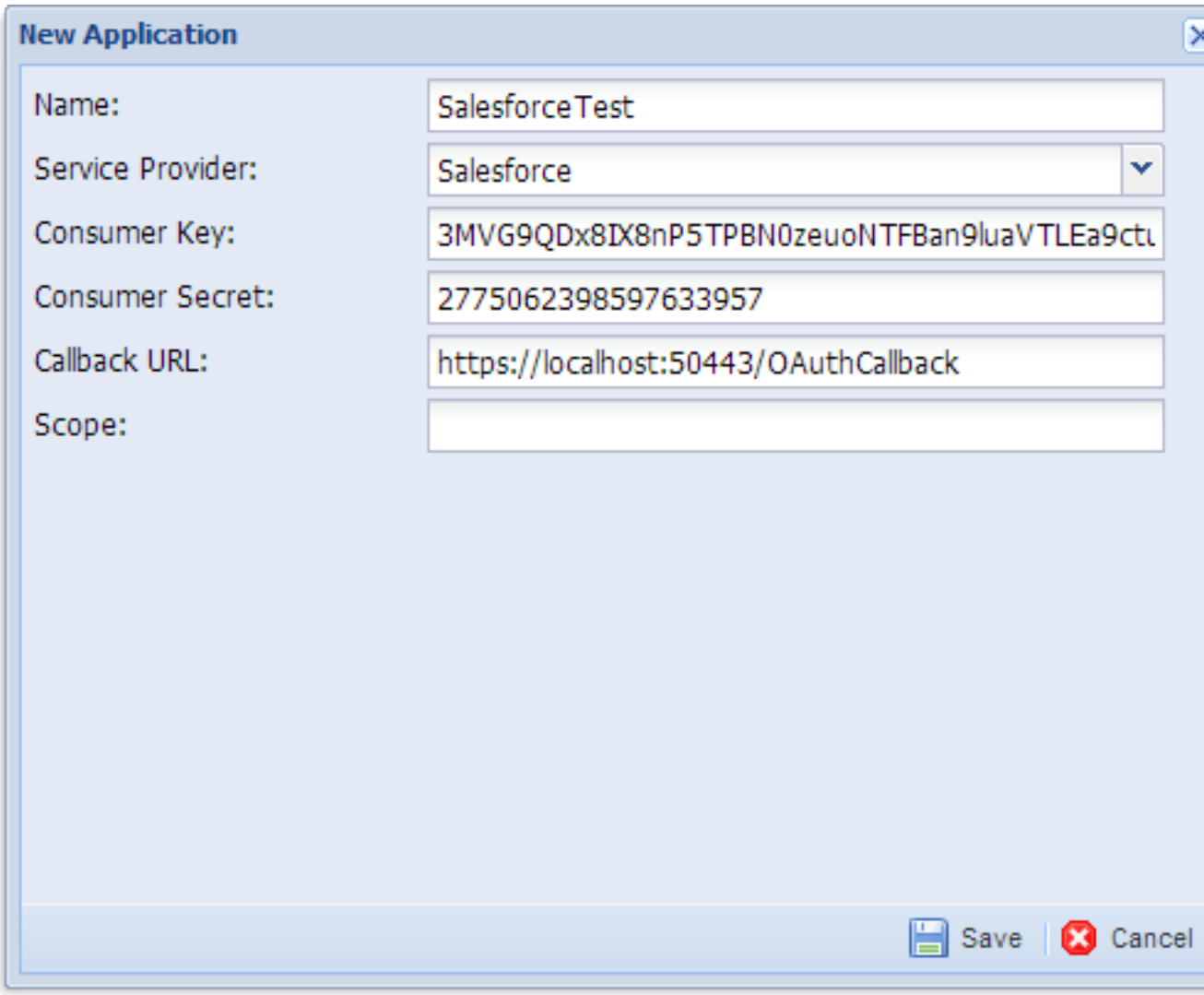


Click okay and start - or restart - the Management Console. Once started, open the Management Console. Make sure to open the Management Console at the URL specified for the app. That would be `https:/`

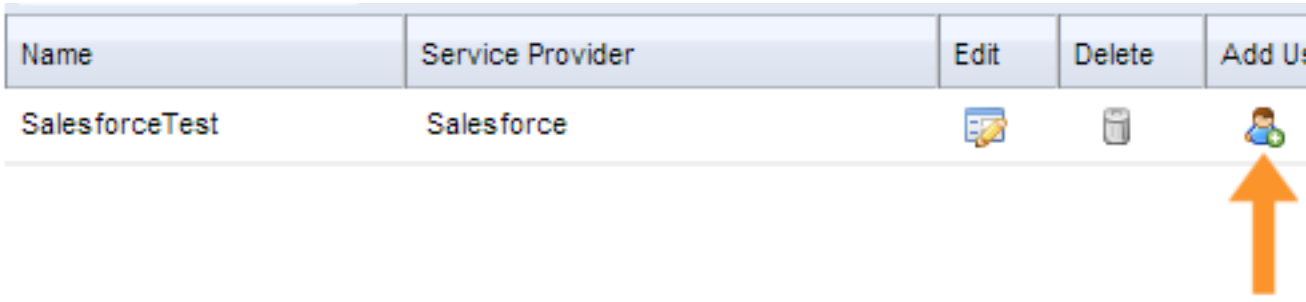
`/localhost:50443` for the information used above. Then navigate to the OAuth Repository and click "New Application".



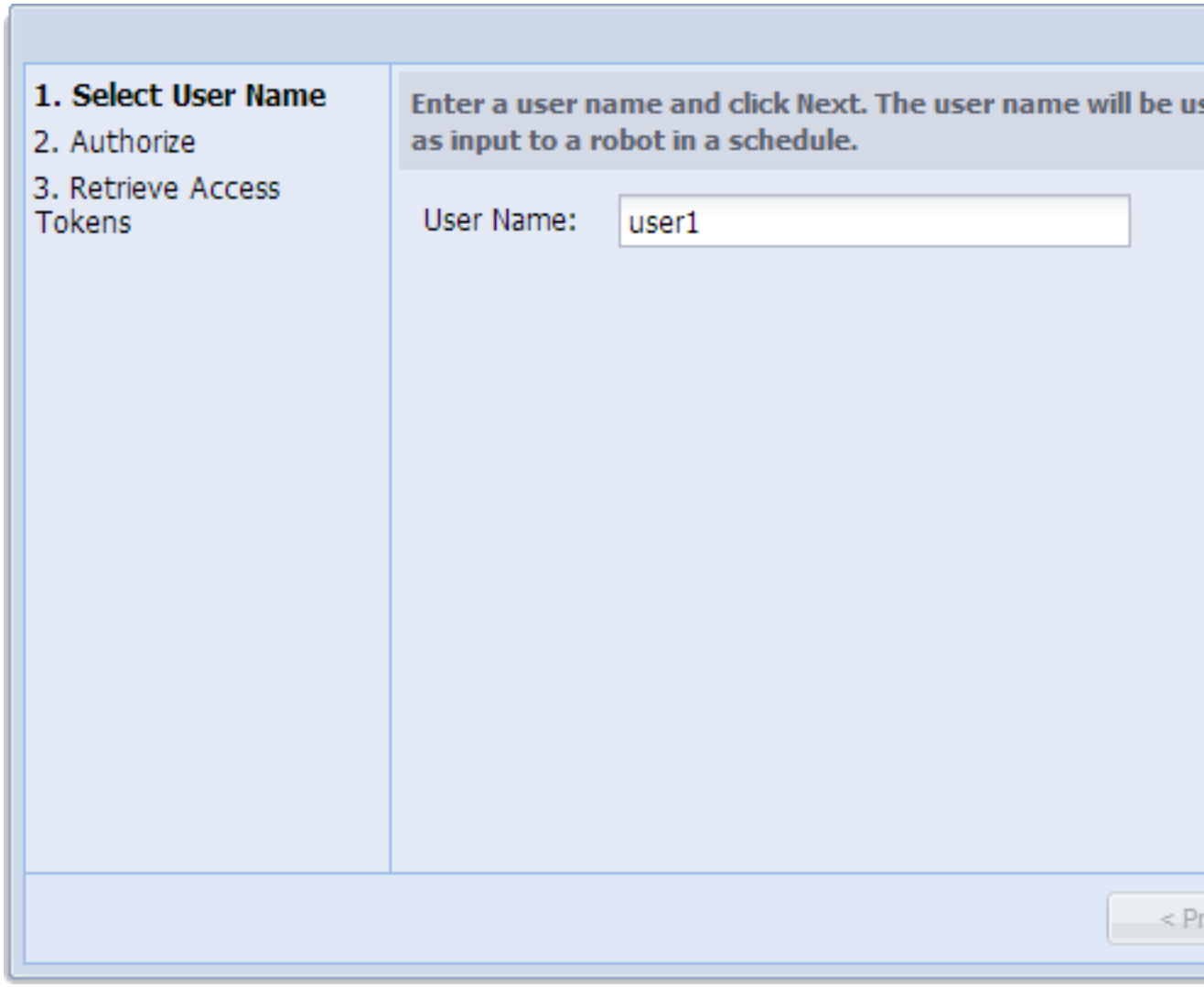
Select a name for the new application and choose Salesforce as the service provider. Enter the Consumer Key and Consumer Secret noted earlier. The Callback URL should already be correctly stated. If you would like to specify the user permissions requested by the app, you may fill in the Scope field using the parameters given on https://help.salesforce.com/help/doc/en/remoteaccess_oauth_scopes.htm. Separate scope parameters with spaces.



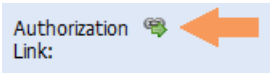
After being saved, the application appears in the repository. Click on the icon under "Add User" to add a user to the application.



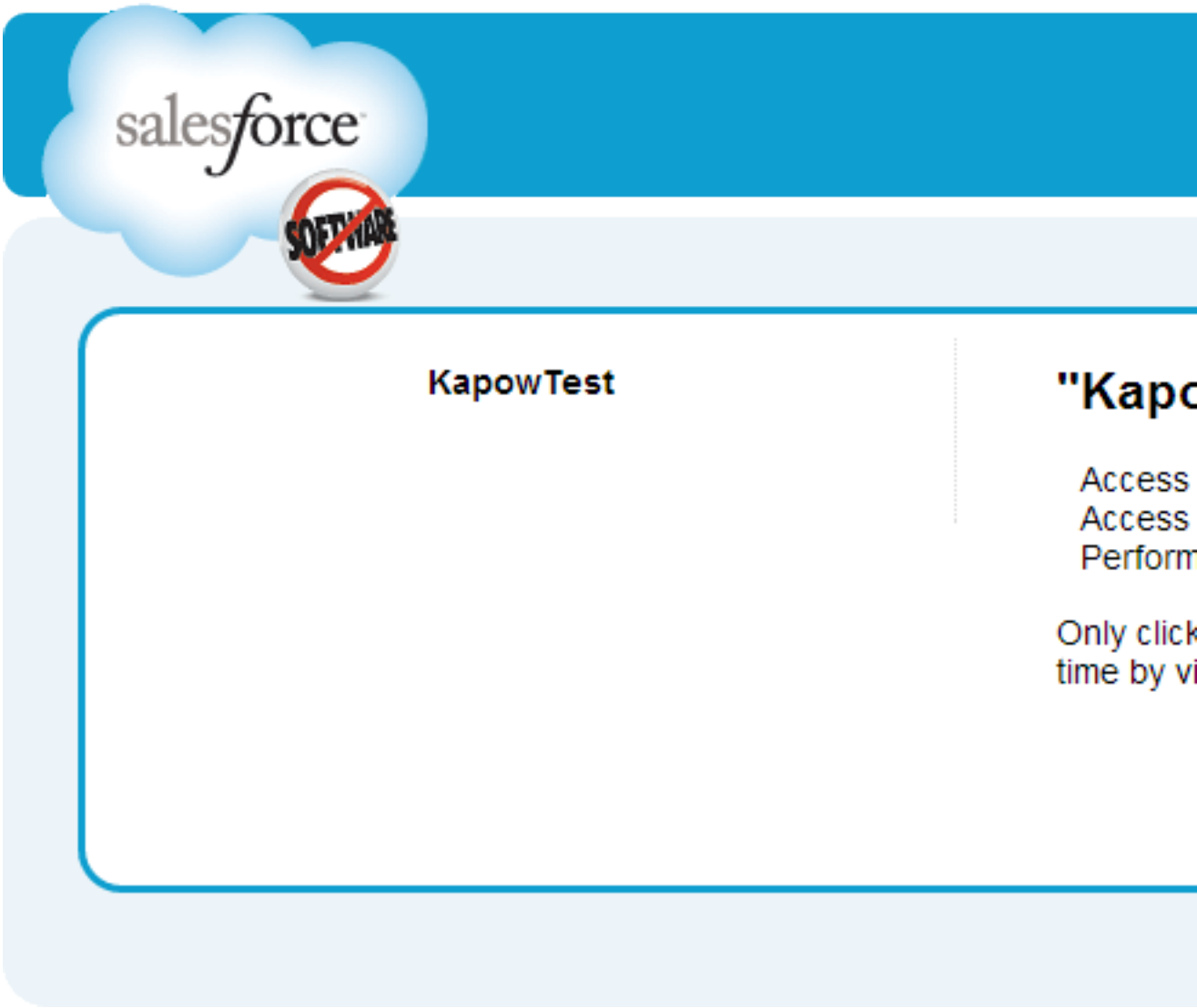
Choose a name for the user and click next.



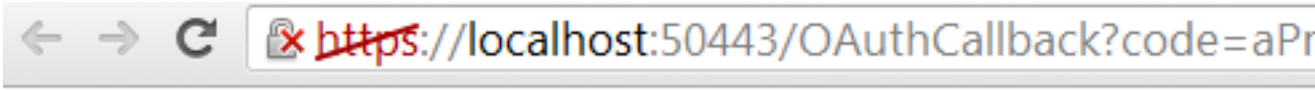
Now, click the authorization link.



This will open a page where the app will request permission to access your Salesforce account, specifying any special permissions requested. The permissions shown below are requested by default.

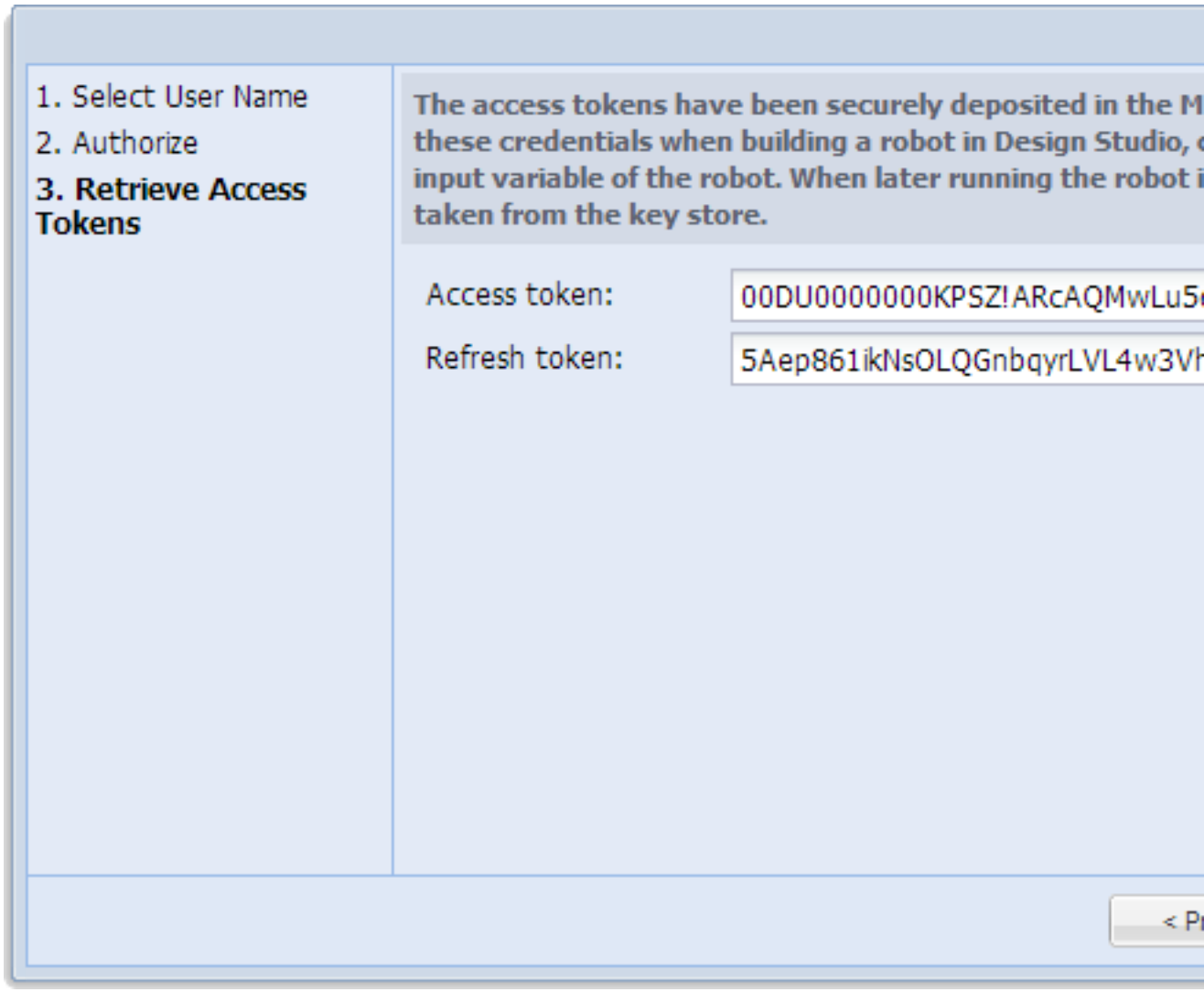


Authorizing the app will forward you to the OAuth callback page of the Management Console. At present time there is a bug in the HTTPS version of the Management Console, which causes the callback page to be scrambled. This does, however, not affect the authentication process.

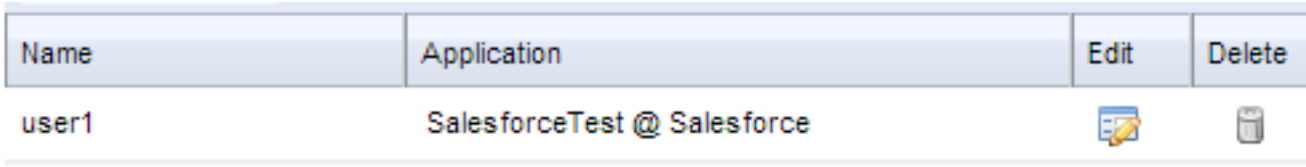



```
<!DOCTYPE html><html>
  <head><title>Success</title>
  <style type="text/css">
      body { background-color: #d2d9e5;
          color: black;
          font-family: tahoma, arial, helvetica;}
```

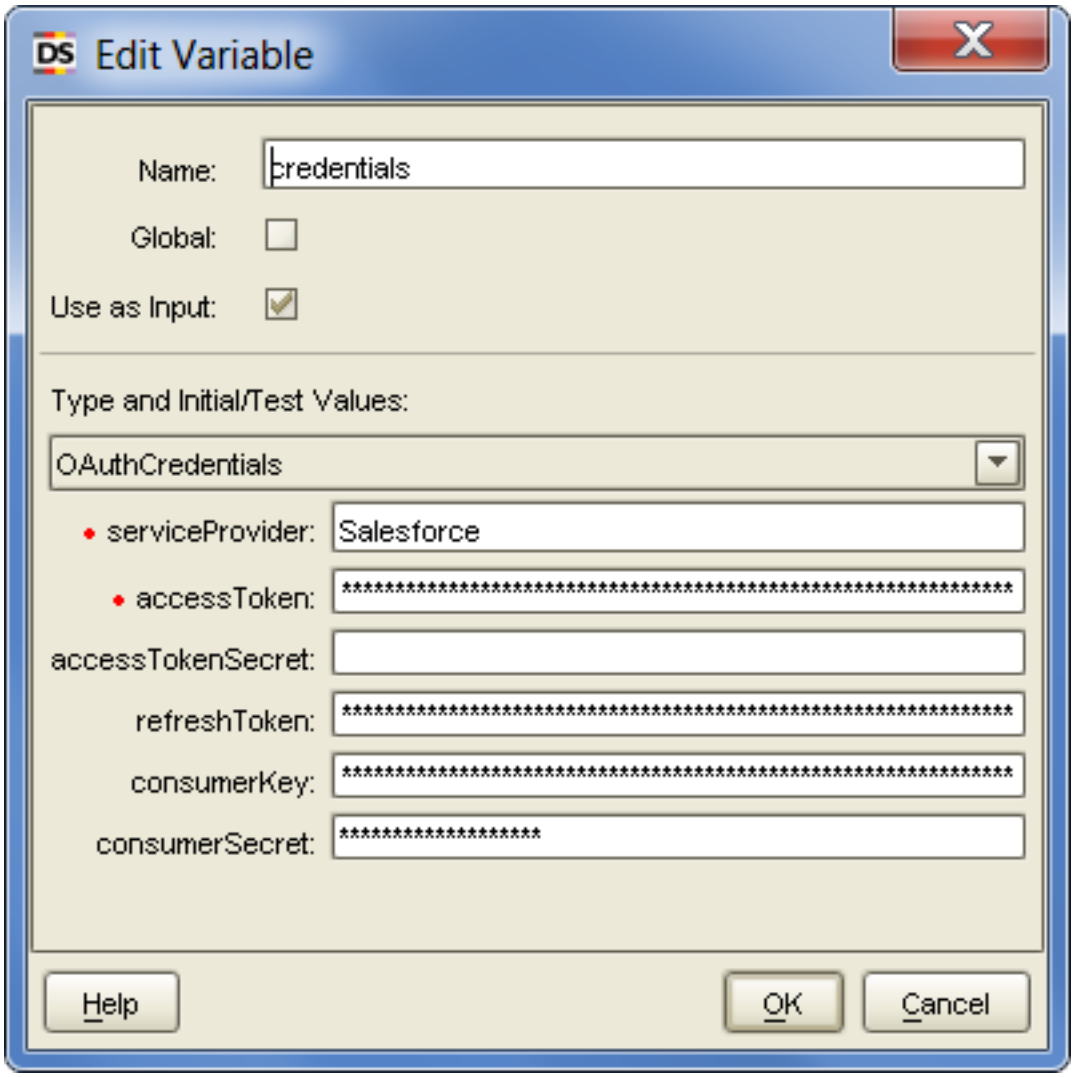
Close the OAuth callback page, returning to the Management Console. Click Next in the wizard and you will see the access token and refresh token granted for accessing Salesforce on the user's behalf. Copy these tokens into a notepad, since they will not be accessible later for security reasons.



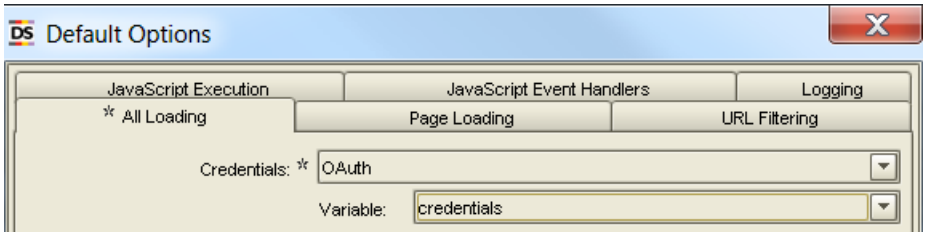
After clicking Finish, you should now see a user in the Users section of the OAuth tab.



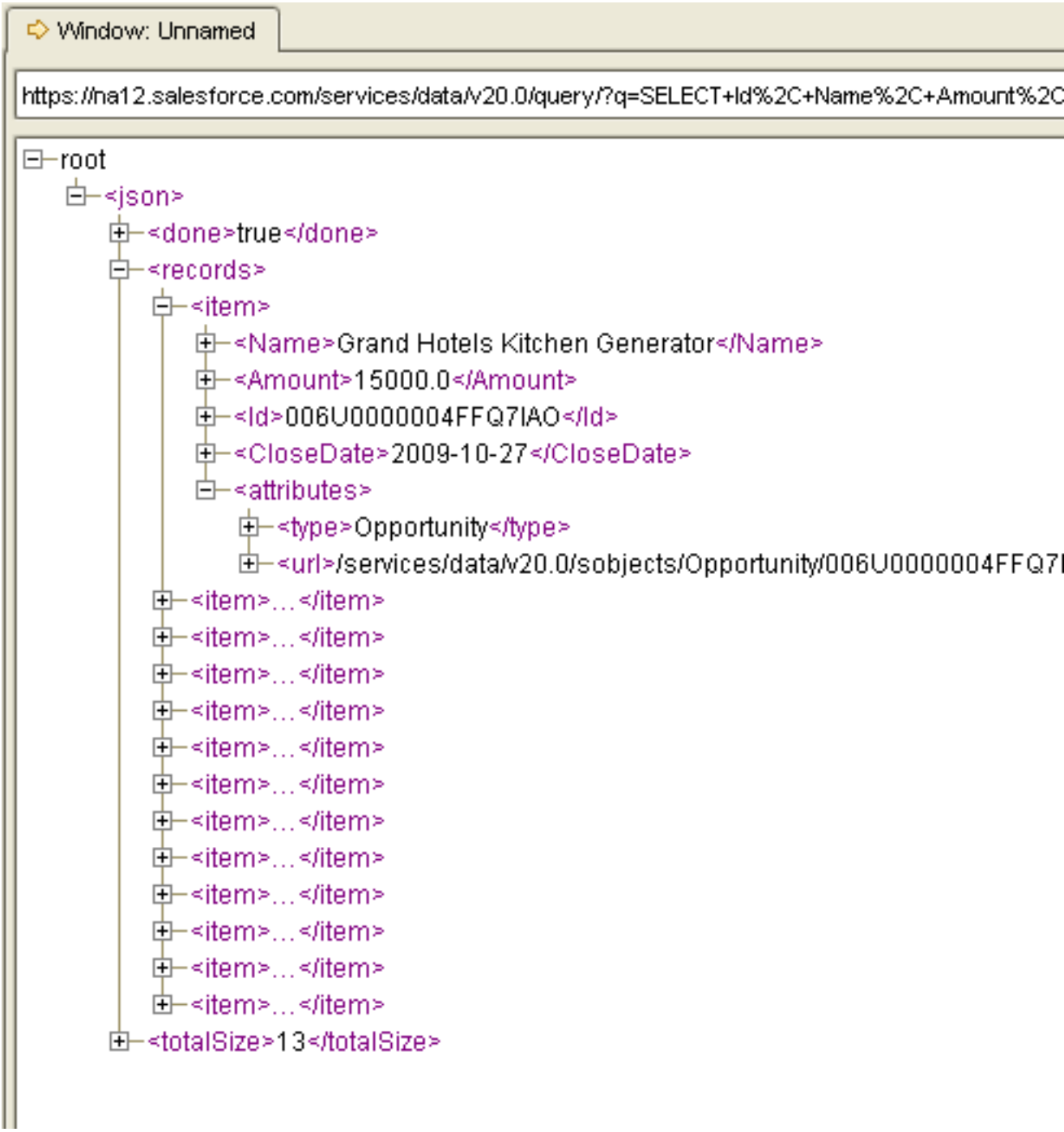
Now, open Design Studio and create a new robot or open the robot you would like to access the Salesforce API. For the robot to be able to use the Salesforce API, make sure that it has an input variable of the complex type "OAuthCredentials". The variable should have "Salesforce" as the service provider name. Also, input the access token and refresh token along with the consumer key and consumer secret noted earlier.



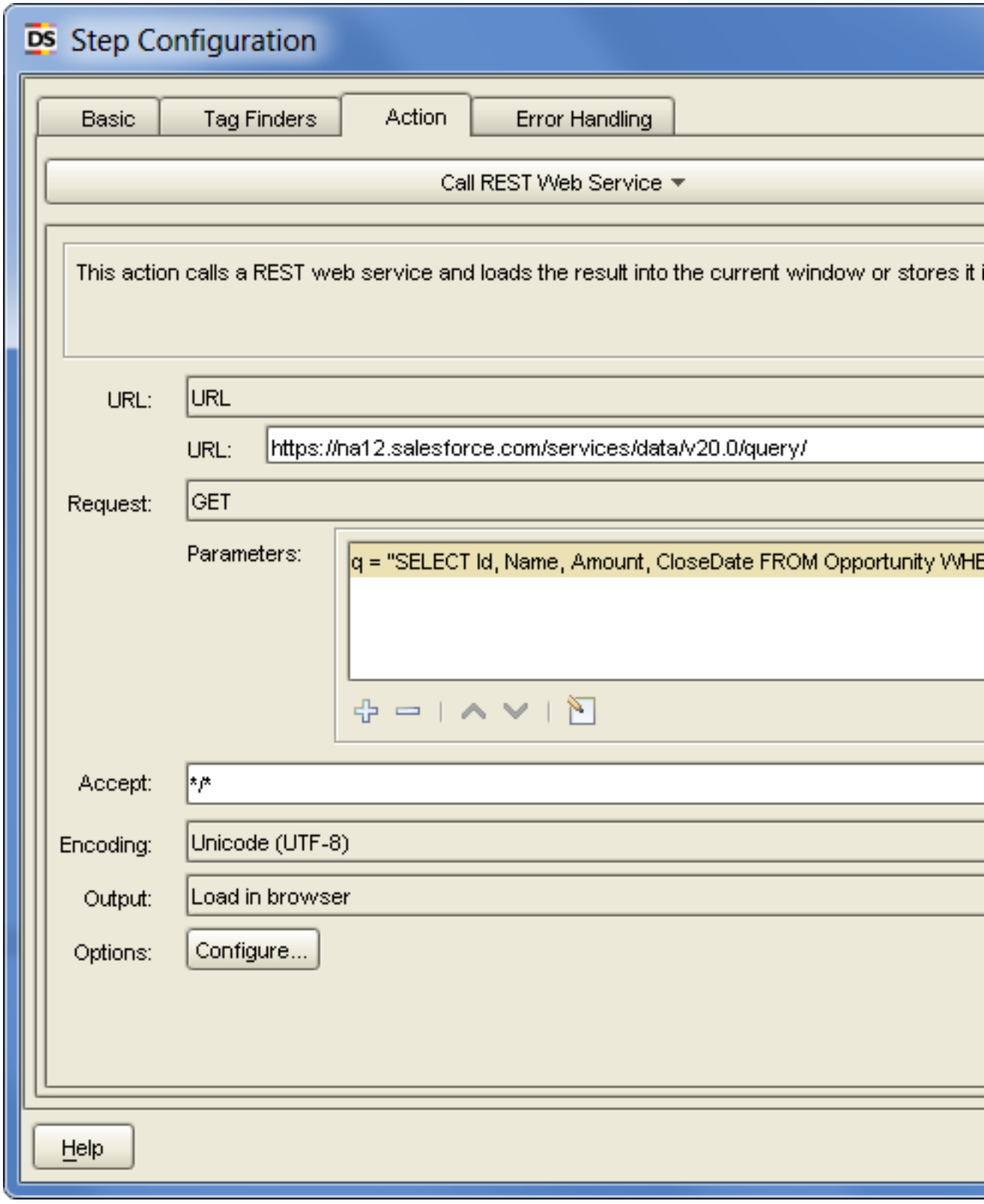
Then, make sure to configure the robot to use OAuth credentials.



The robot should now be able to access the Salesforce API. As an example try copying `https://instance.salesforce.com/services/data/v20.0/query/?q=SELECT+Id%2C+Name%2C+Amount%2C+CloseDate+FROM+Opportunity+WHERE+Probability%3C1` into the address bar of the browser view, replacing "instance" with your instance determined earlier, and pressing Enter. This should return a JSON page with opportunities.



To learn more about the Salesforce REST API visit http://www.salesforce.com/us/developer/docs/api_rest/. Since Salesforce uses a REST API it is possible to access the API using the Call REST Web Service [../ref/robomaker/reference/stepaction/CallRESTWebService2StepAction.html] step. This is especially helpful since it becomes much easier to specify parameters such as the query used in the above example.



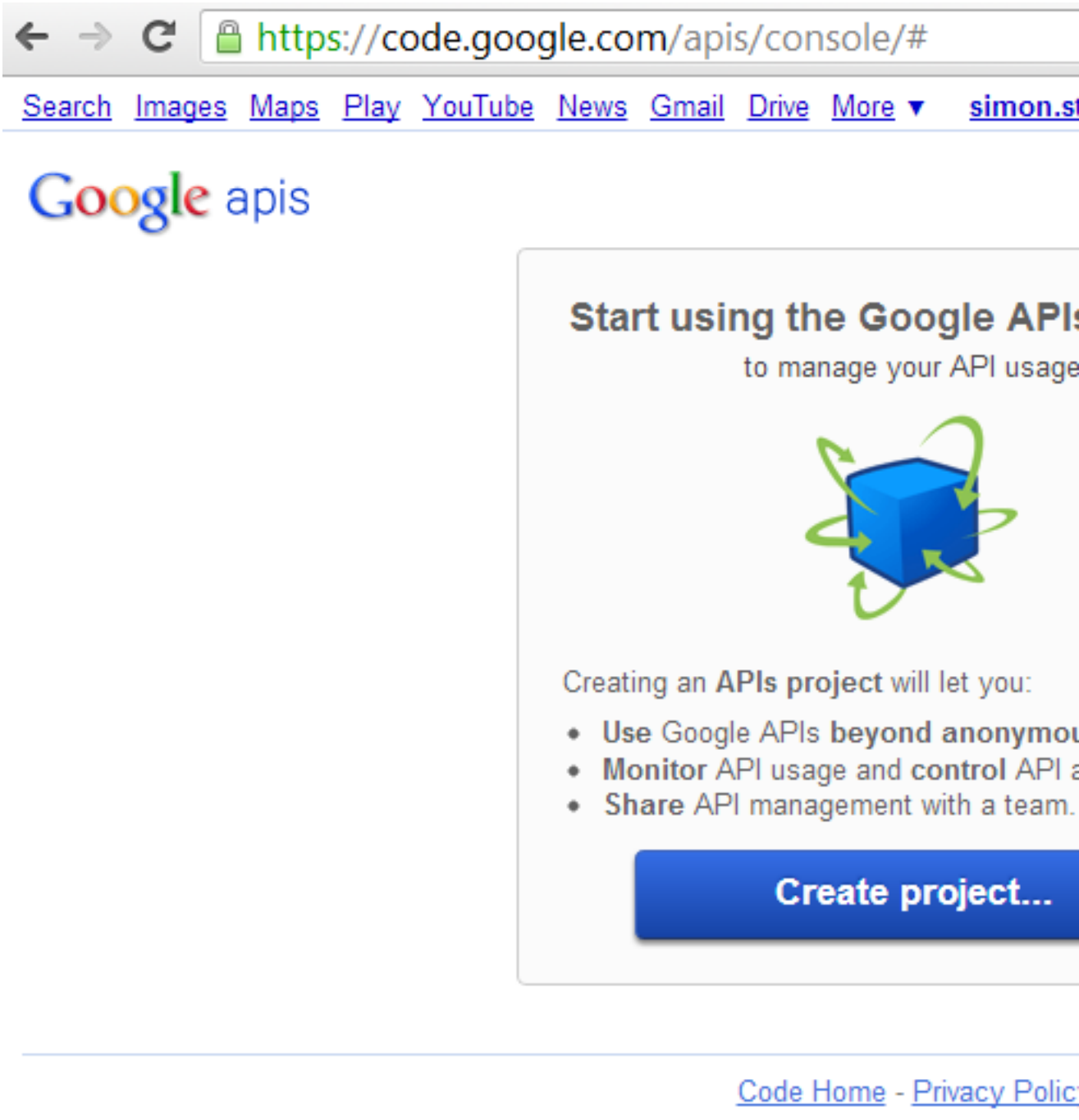
The query specified above is in the Salesforce Object Query Language (SOQL), which is very useful when accessing the Salesforce API. The documentation for SOQL is given at http://www.salesforce.com/us/

developer/docs/soql_sosl/index_Left.htm. Once a robot has been built, it is possible to schedule execution of the robot in the Management Console.

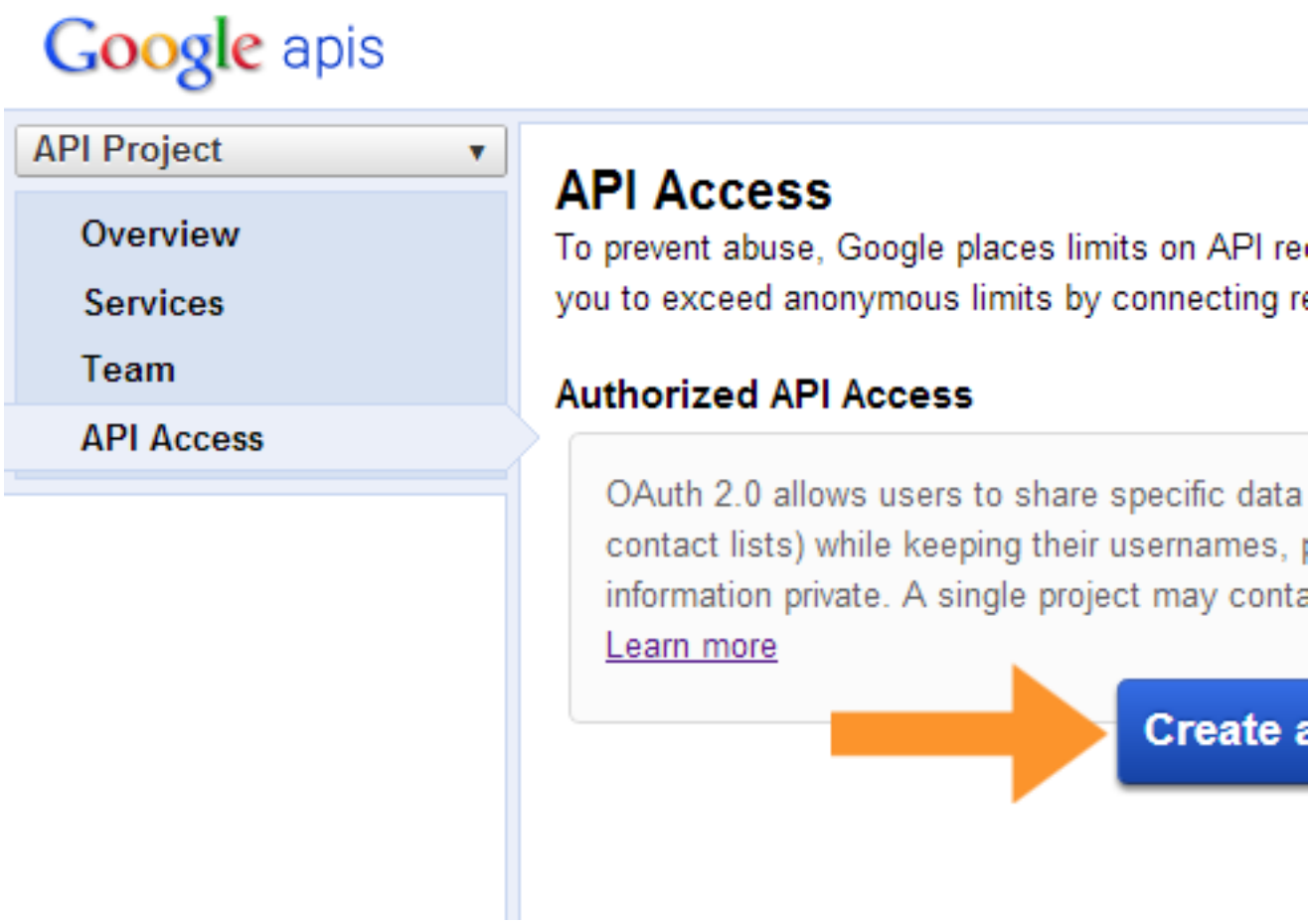
Google

This is a short guide on how to create a robot which accesses various Google APIs, which can be used to read and write to services by Google. This includes YouTube, Google Drive, Blogger, and many more.

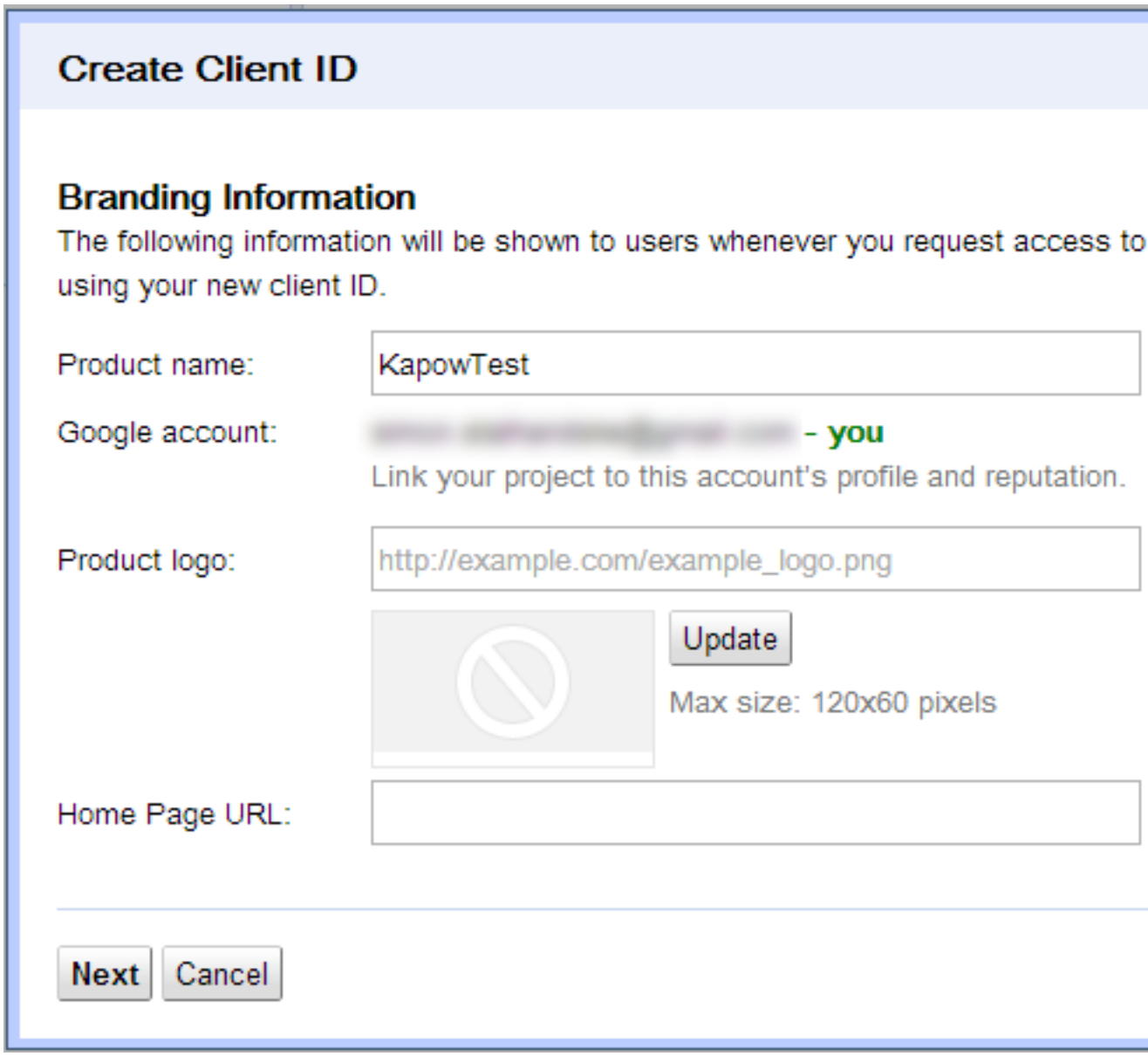
First, log into http://code.google.com/apis/console with your Google account, or create a new account if necessary. Create a new project by clicking "Create Project...".



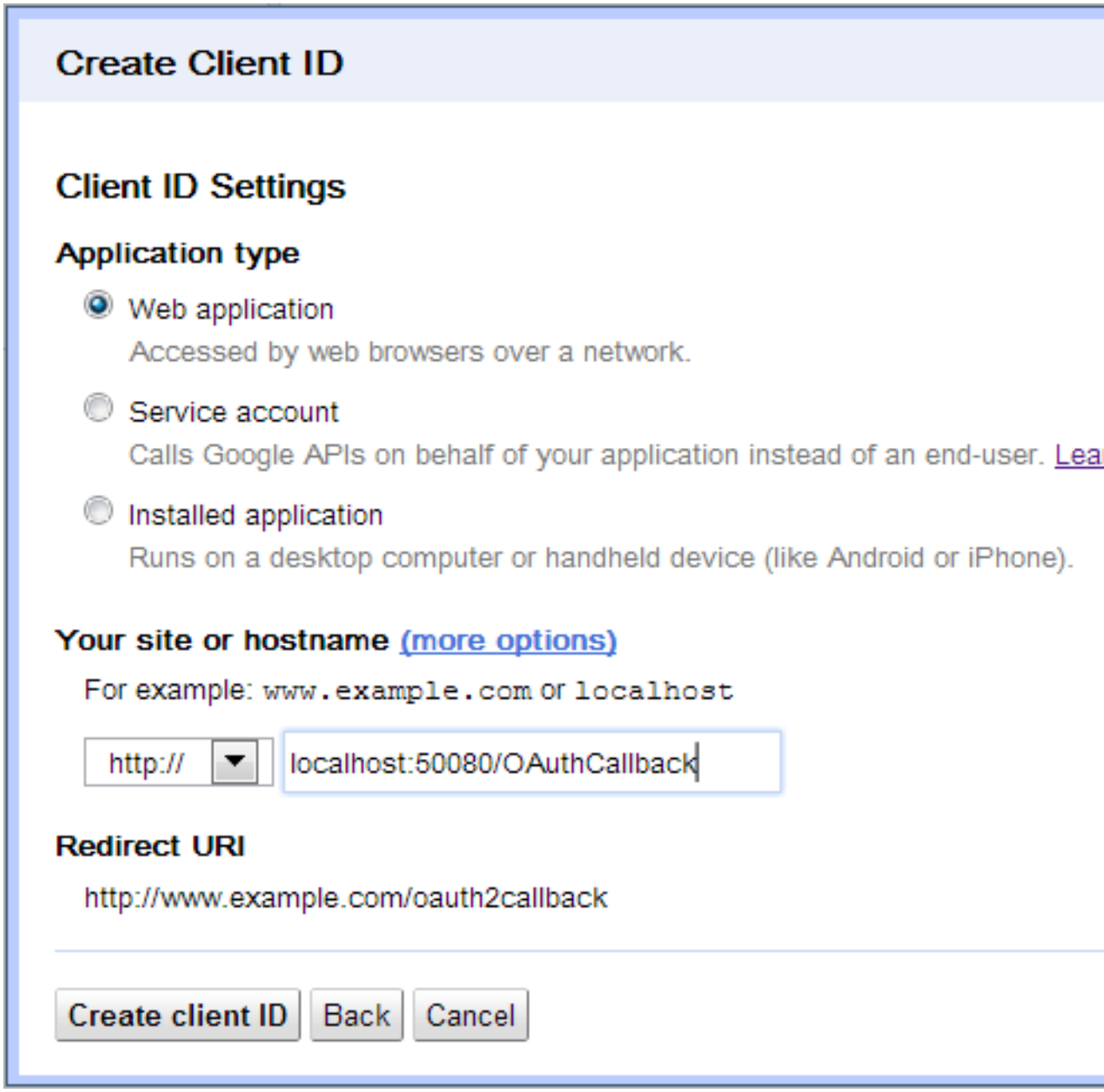
Choose API Access in the left-hand menu, then click "Create an OAuth 2.0 client ID...".



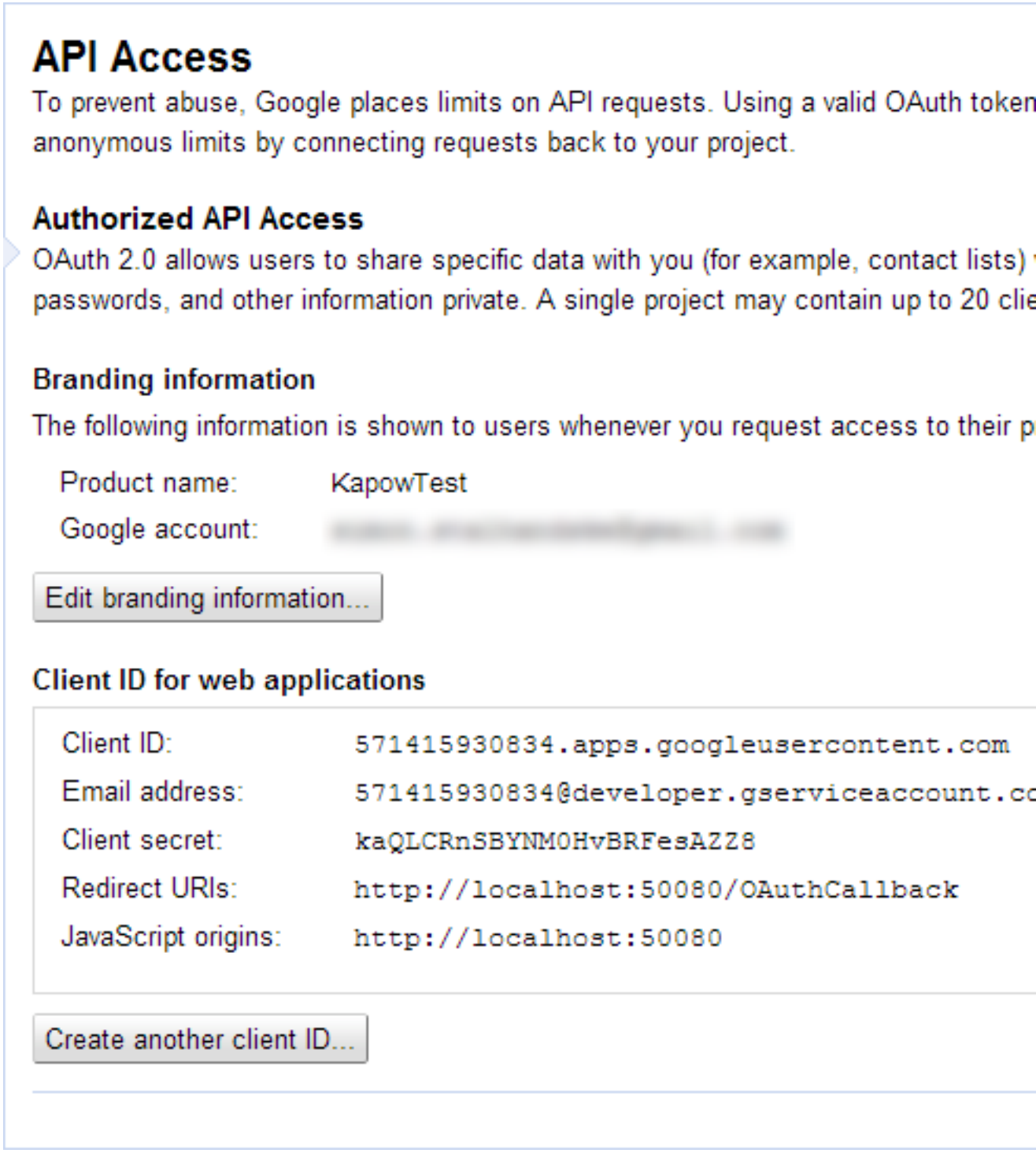
A configuration screen now appears. Input any relevant information; only a product name is mandatory.



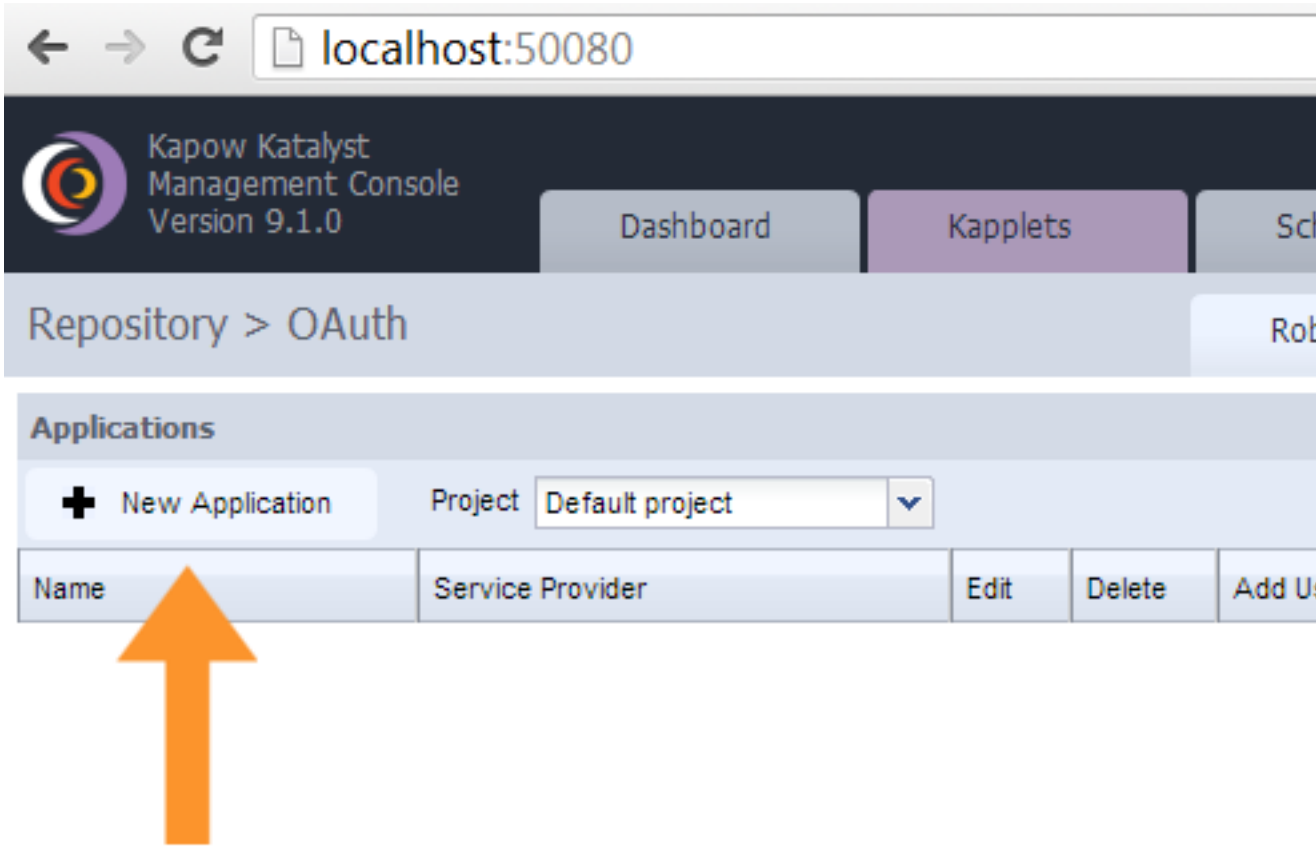
Click next and a new form will appear. Choose "Web Application" as the application type and write the OAuth callback URL in the field labeled "Your site or hostname". The callback URL is the path `OAuthCallback` under the folder in which the Management Console is deployed. For instance, if running with an embedded Management Console, it runs at `http://localhost:50080/` and the callback URL is `http://localhost:50080/OAuthCallback`. Press Enter after inputting your callback URL. The "Redirect URL" at the bottom of the form will now change to the URL you input. Remember to choose `http://` or `https://` from the drop down menu, depending on which protocol your Management Console is using.



After clicking "Create client ID", a page with Application details is opened. On this page it is important to take note of the Client ID and Client Secret which will be used as Consumer Key and Consumer Secret in the Management Console later on.

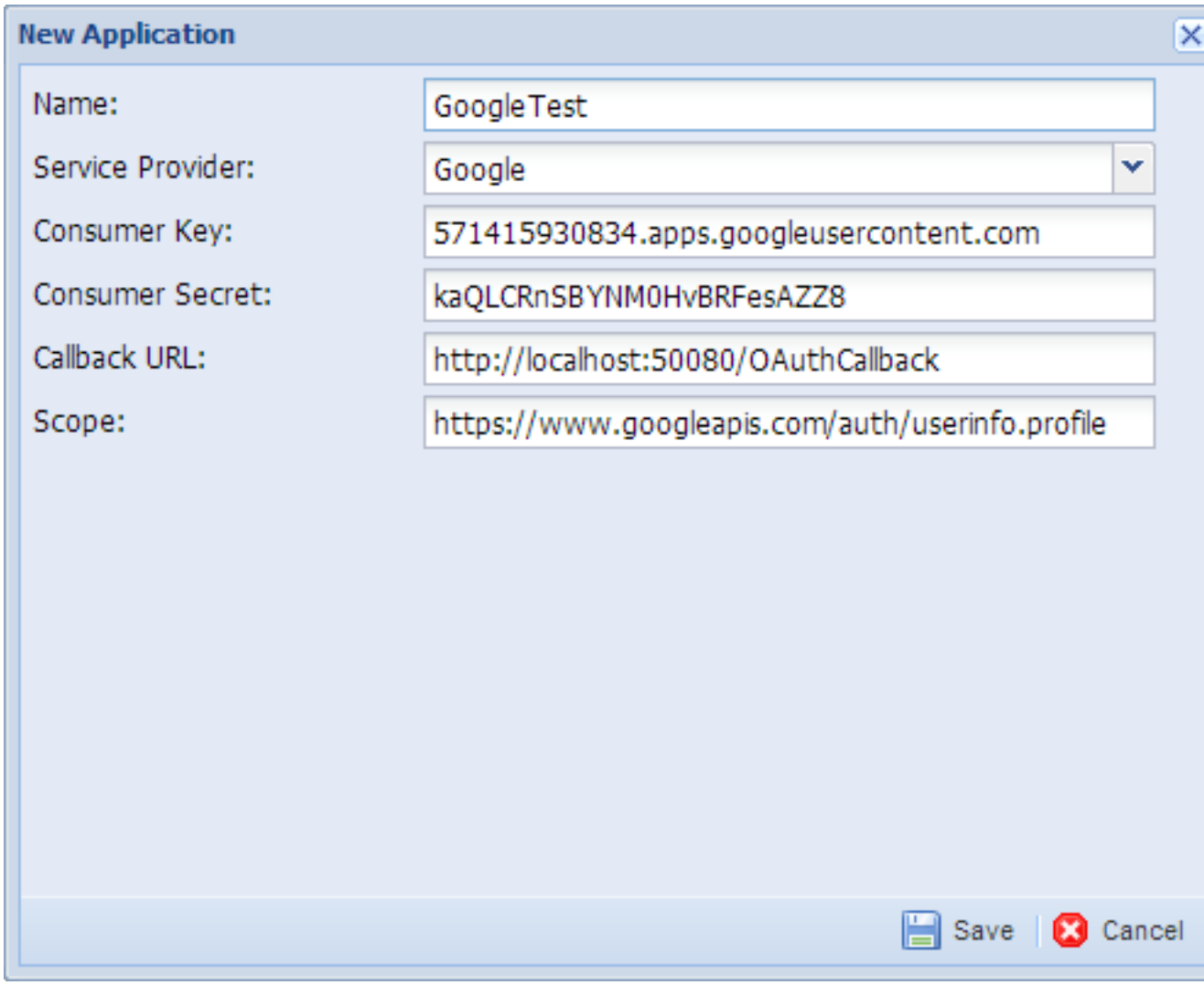


Now, open the Management Console. Make sure to open the Management Console at the URL specified for the app. That would be `http://localhost:50080` for the information used above. Then navigate to the OAuth Repository and click "New Application".

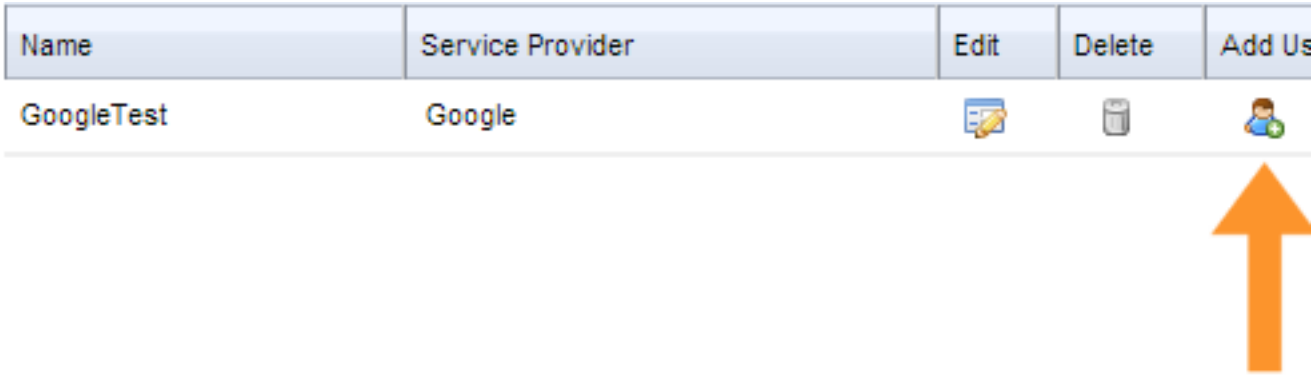


Select a name for the new application and choose Google as the service provider. Consumer Key and Consumer Secret should respectively be set to the Client ID and Client Secret noted earlier. The Callback URL should already be correctly stated.

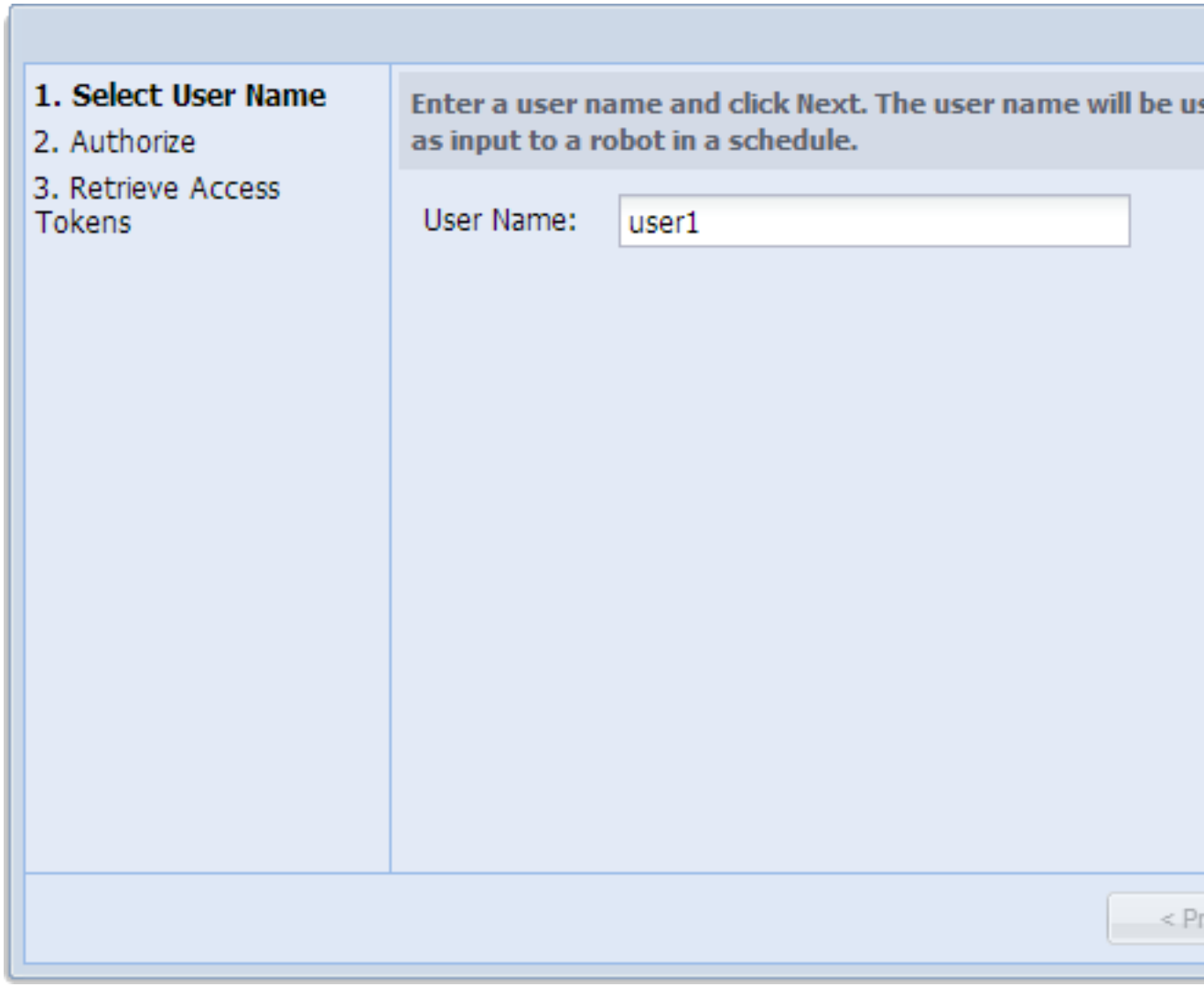
In the Scope field, we state the Google API permissions required from the app user. In the picture below the basic user information has been requested. A comprehensive list of APIs and the scope associated with each one is given through Google APIs Discovery Service [https://developers.google.com/discovery/]. A list of available APIs as of December 2012 is given in a table at the bottom of this page. The table gives a description of each API, a link to the documentation, as well as the scope URL.



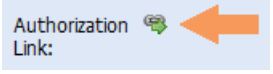
After being saved, the application appears in the repository. Click on the icon under "Add User" to add a user to the application.



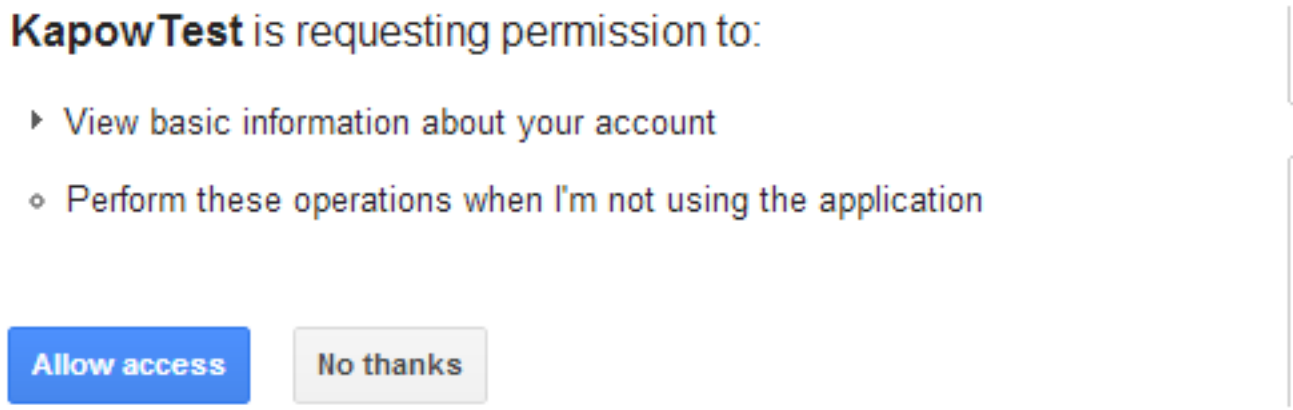
Choose a name for the user and click next.



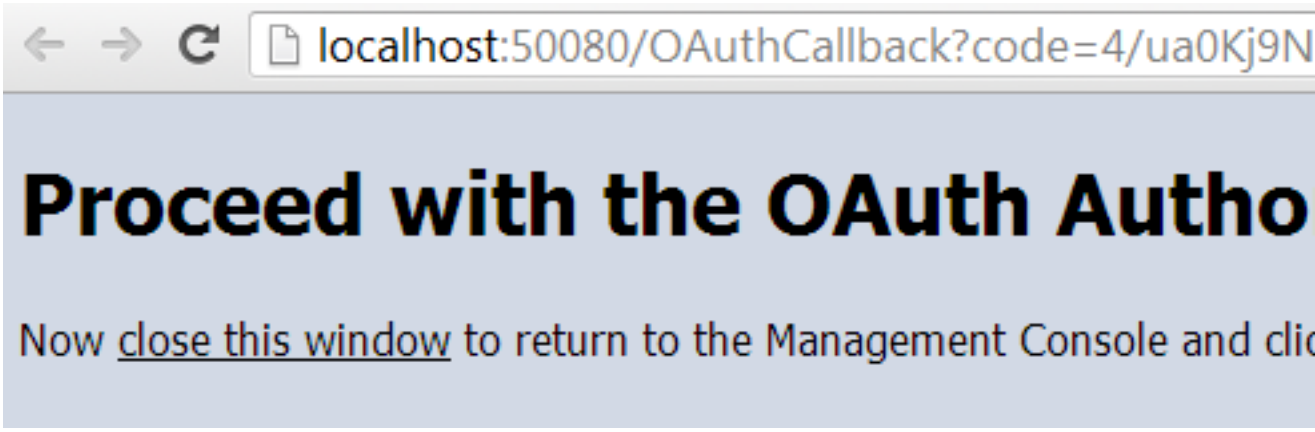
Now, click the authorization link.



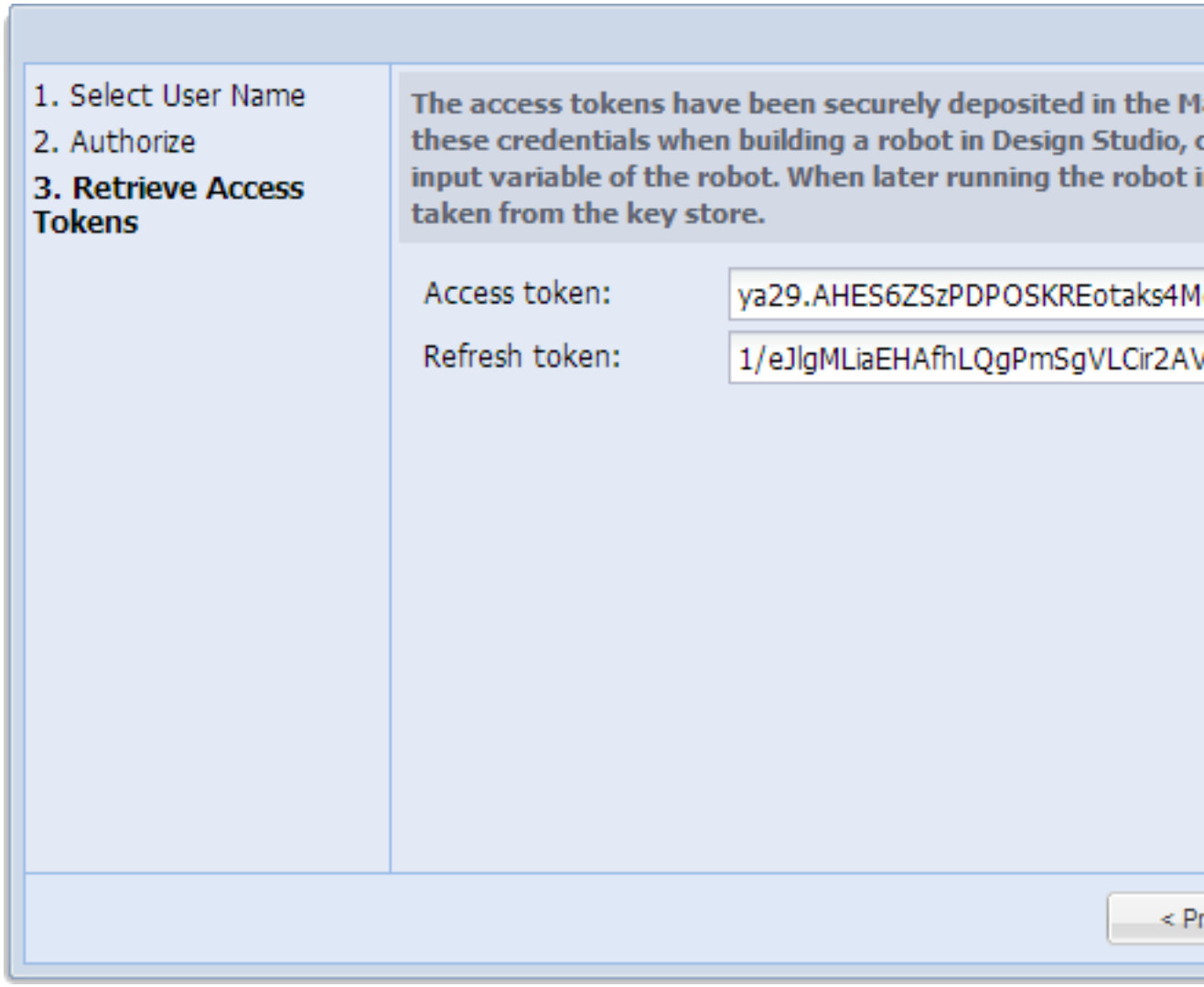
This will open a page where the app will request permission to access certain aspects of your Google account, as specified in through the scope of the application.



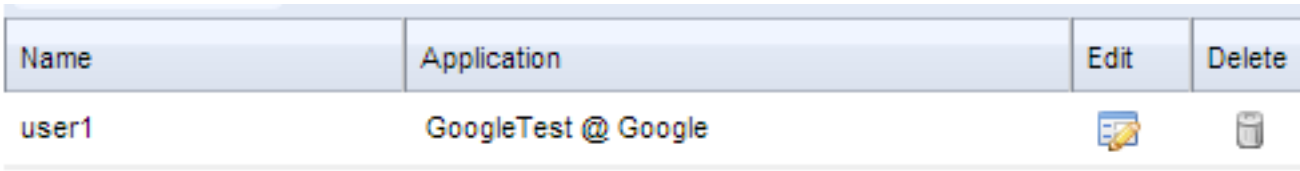
Authorizing the app will forward you to the OAuth callback page of the Management Console.



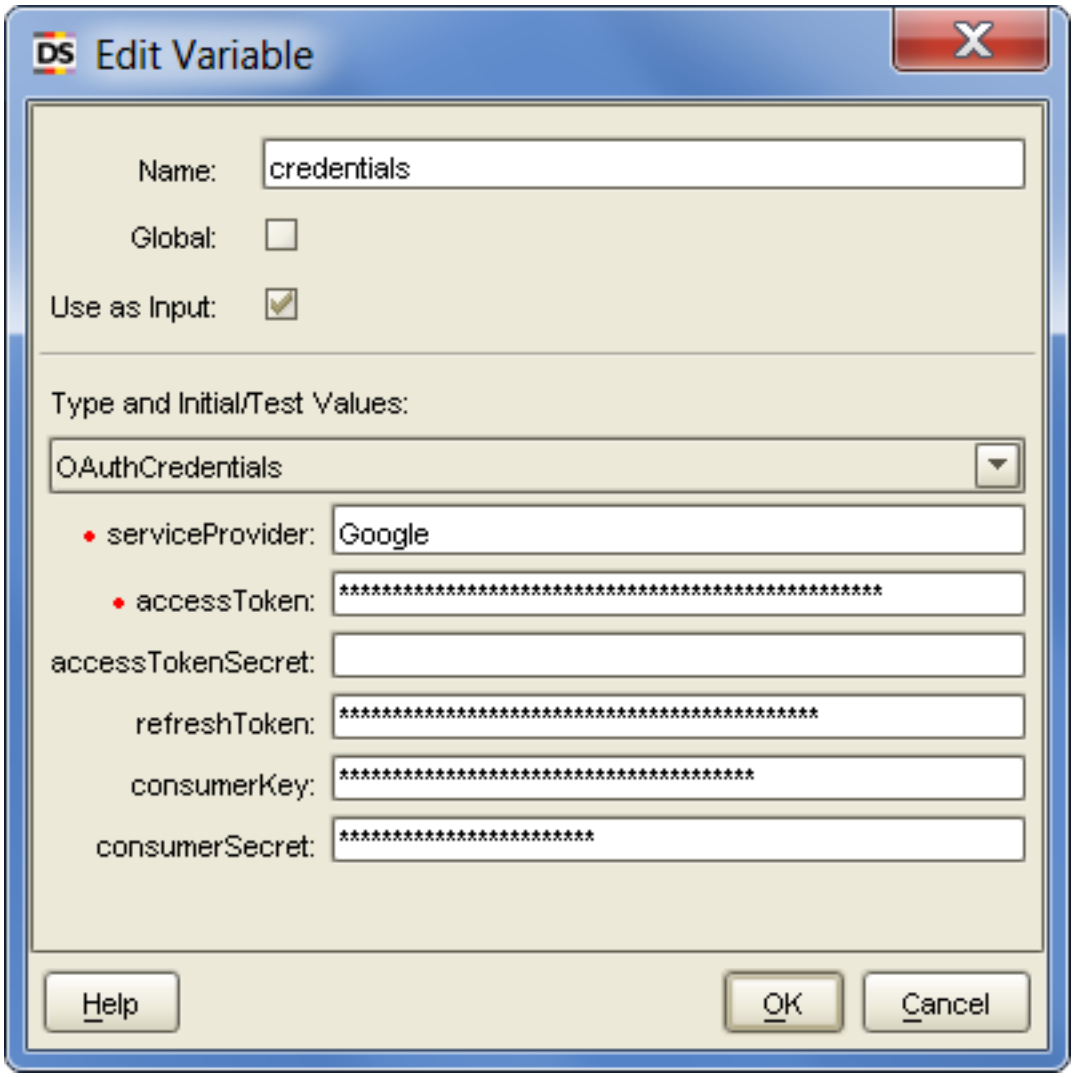
Close the OAuth callback page, returning to the Management Console. Click Next in the wizard and you will see the access token and refresh token granted for accessing Google on the user's behalf. Copy these tokens into a notepad, since they will not be accessible later for security reasons.



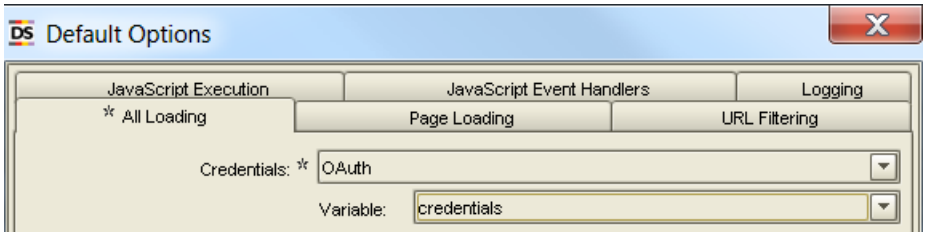
After clicking Finish, you should now see a user in the Users section of the OAuth tab.



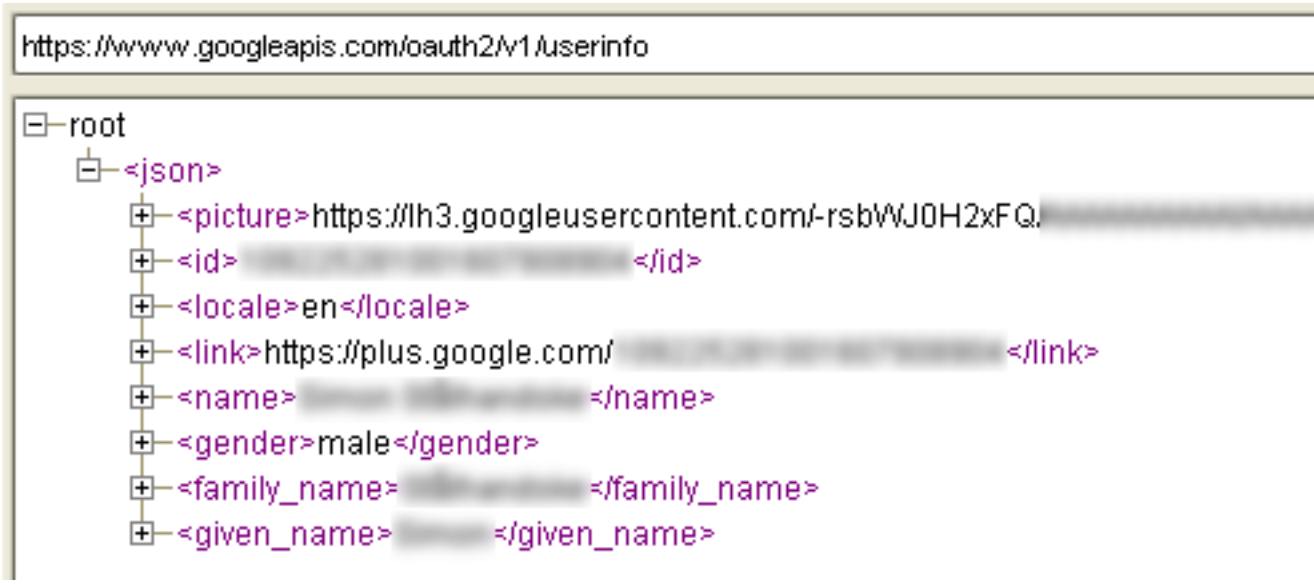
Now, open Design Studio and create a new robot or open the robot you would like to access the chosen Google API. For the robot to be able to use the Google API, make sure that it has an input variable of the complex type "OAuthCredentials". The variable should have "Google" as the service provider name. Also, input the access token and refresh token along with the Client Key and Client Secret noted earlier. The latter two should be used as consumer key and consumer secret respectively.



Then, make sure to configure the robot to use OAuth credentials.



The robot should now be able to access the chosen Google API. As an example try typing `https://www.googleapis.com/oauth2/v1/userinfo` in the address bar of the browser view and pressing Enter. This should return an JSON page with basic user information.



To learn more about the Google APIs, try experimenting with Google's OAuth 2.0 Playground [https://developers.google.com/oauthplayground/]. Also, the links to documentation in the table below are helpful resources to learn more about each API. Once a robot has been built, it is possible to schedule execution of the robot in the Management Console.

Table 40. An overview of the various Google APIs as of December 2012. Use Google APIs Discovery Service [https://developers.google.com/discovery/] to get an updated list.

API	Description	Documentation	Scope
Search API For Shopping	Manage your Ad Exchange buyer account configuration	https://developers.google.com/ad-exchange/buyer-rest	https://www.googleapis.com/auth/adexchange.buyer
AdSense Management API	View and manage your AdSense data	https://developers.google.com/adsense/management/	https://www.googleapis.com/auth/adsense
AdSense Management API	View your AdSense data	https://developers.google.com/adsense/management/	https://www.googleapis.com/auth/adsense.readonly
AdSense Host API	View and manage your AdSense host data and associated accounts	https://developers.google.com/adsense/host/	https://www.googleapis.com/auth/adsensehost
Google Analytics API	View and manage your Google Analytics data	https://developers.google.com/analytics/	https://www.googleapis.com/auth/analytics
Google Analytics API	View your Google Analytics data	https://developers.google.com/analytics/	https://www.googleapis.com/auth/analytics.readonly
BigQuery API	View and manage your data in Google BigQuery	https://developers.google.com/bigquery/docs/overview	https://www.googleapis.com/auth/bigquery
BigQuery API			

API	Description	Documentation	Scope
	Manage your data in Google Cloud Storage	https://developers.google.com/bigquery/docs/overview	https://www.googleapis.com/auth/devstorage.read_write
BigQuery API	Manage your data and permissions in Google Cloud Storage	https://developers.google.com/bigquery/docs/overview	https://www.googleapis.com/auth/devstorage.full_control
BigQuery API	View your data in Google Cloud Storage	https://developers.google.com/bigquery/docs/overview	https://www.googleapis.com/auth/devstorage.read_only
Blogger API	Manage your Blogger account	https://developers.google.com/blogger/docs/3.0/getting_started	https://www.googleapis.com/auth/blogger
Blogger API	View your Blogger account	https://developers.google.com/blogger/docs/3.0/getting_started	https://www.googleapis.com/auth/blogger.readonly
Books API	Manage your books	https://developers.google.com/books/docs/v1/getting_started	https://www.googleapis.com/auth/books
Calendar API	Manage your calendars	https://developers.google.com/google-apps/calendar/firstapp	https://www.googleapis.com/auth/calendar
Calendar API	View your calendars	https://developers.google.com/google-apps/calendar/firstapp	https://www.googleapis.com/auth/calendar.readonly
Compute Engine API	View and manage your Google Compute Engine resources	https://developers.google.com/compute/docs/reference/v1beta13	https://www.googleapis.com/auth/compute
Compute Engine API	View your Google Compute Engine resources	https://developers.google.com/compute/docs/reference/v1beta13	https://www.googleapis.com/auth/compute.readonly
Compute Engine API	View your data in Google Cloud Storage	https://developers.google.com/compute/docs/reference/v1beta13	https://www.googleapis.com/auth/devstorage.read_only
Google Maps Coordinate API	View your Google Coordinate jobs	https://developers.google.com/coordinate/	https://www.googleapis.com/auth/coordinate.readonly

API	Description	Documentation	Scope
Google Maps Coordinate API	View and manage your Google Maps Coordinate jobs	https://developers.google.com/coordinate/	https://www.googleapis.com/auth/coordinate
DFA Reporting API	View and manage DoubleClick for Advertisers reports	https://developers.google.com/doubleclick-advertisers/reporting/	https://www.googleapis.com/auth/dfareporting
Drive API	View the files and documents in your Google Drive	https://developers.google.com/drive/	https://www.googleapis.com/auth/drive.readonly
Drive API	View your Google Drive apps	https://developers.google.com/drive/	https://www.googleapis.com/auth/drive.apps.readonly
Drive API	View and manage the files and documents in your Google Drive	https://developers.google.com/drive/	https://www.googleapis.com/auth/drive
Drive API	View meta data for files and documents in your Google Drive	https://developers.google.com/drive/	https://www.googleapis.com/auth/drive.metadata.readonly
Drive API	View and manage Google Drive files that you have opened or created with this app	https://developers.google.com/drive/	https://www.googleapis.com/auth/drive.file
Freebase API	Sign in to Freebase with your account	http://wiki.freebase.com/wiki/API	https://www.googleapis.com/auth/freebase
Fusion Tables API	View your Fusion Tables	https://developers.google.com/fusiontables	https://www.googleapis.com/auth/fusiontables.readonly
Fusion Tables API	Manage your Fusion Tables	https://developers.google.com/fusiontables	https://www.googleapis.com/auth/fusiontables
Google Affiliate Network API	View your GAN data	https://developers.google.com/affiliate-network/	https://www.googleapis.com/auth/gan.readonly
Google Affiliate Network API	Manage your GAN data	https://developers.google.com/affiliate-network/	https://www.googleapis.com/auth/gan
Groups Settings API	View and manage the settings of a Google Apps Group	https://developers.google.com/google-apps/groups-settings/get_started	https://www.googleapis.com/auth/apps.groups.settings
Google Latitude API	Manage your city-level location	https://developers.google.com/latitude/v1/using	https://www.googleapis.com/auth/latitude.current.city

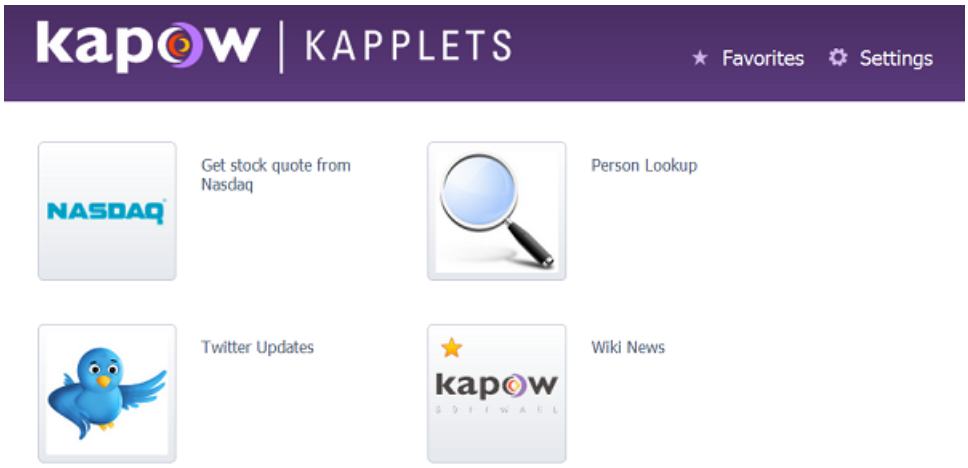
API	Description	Documentation	Scope
Google Latitude API	Manage your best-available location and location history	https://developers.google.com/latitude/v1/using	https://www.googleapis.com/auth/latitude.all.best
Google Latitude API	Manage your best-available location	https://developers.google.com/latitude/v1/using	https://www.googleapis.com/auth/latitude.current.best
Google Latitude API	Manage your city-level location and location history	https://developers.google.com/latitude/v1/using	https://www.googleapis.com/auth/latitude.all.city
Moderator API	Manage your activity in Google Moderator	http://code.google.com/apis/moderator/v1/using_rest.html	https://www.googleapis.com/auth/moderator
Google OAuth2 API	View basic information about your account	https://developers.google.com/accounts/docs/OAuth2	https://www.googleapis.com/auth/userinfo.profile
Google OAuth2 API	View your email address	https://developers.google.com/accounts/docs/OAuth2	https://www.googleapis.com/auth/userinfo.email
Google OAuth2 API	Know who you are on Google	https://developers.google.com/accounts/docs/OAuth2	https://www.googleapis.com/auth/plus.me
Orkut API	Manage your Orkut activity	http://code.google.com/apis/orkut/v2/reference.html	https://www.googleapis.com/auth/orkut
Orkut API	View your Orkut data	http://code.google.com/apis/orkut/v2/reference.html	https://www.googleapis.com/auth/orkut.readonly
Google+ API	Know who you are on Google	https://developers.google.com/+/api/	https://www.googleapis.com/auth/plus.me
Prediction API	Manage your data in the Google Prediction API	https://developers.google.com/prediction/docs/developer-guide	https://www.googleapis.com/auth/prediction
Prediction API	View your data in Google Cloud Storage	https://developers.google.com/prediction/docs/developer-guide	https://www.googleapis.com/auth/devstorage.read_only
Search API For Shopping	View your product data	https://developers.google.com/shopping-search/v1/getting_started	https://www.googleapis.com/auth/shoppingapi
Google Site Verification API	Manage your new site verifications with Google	http://code.google.com/apis/siteverification/	https://www.googleapis.com/auth/siteverification.verify_only

API	Description	Documentation	Scope
Google Site Verification API	Manage the list of sites and domains you control	http://code.google.com/apis/siteverification/	https://www.googleapis.com/auth/siteverification
Cloud Storage API	Manage your data in Google Cloud Storage	https://developers.google.com/storage/docs/json_api/	https://www.googleapis.com/auth/devstorage.read_write
Cloud Storage API	Manage your data and permissions in Google Cloud Storage	https://developers.google.com/storage/docs/json_api/	https://www.googleapis.com/auth/devstorage.full_control
Cloud Storage API	View your data in Google Cloud Storage	https://developers.google.com/storage/docs/json_api/	https://www.googleapis.com/auth/devstorage.read_only
TaskQueue API	Manage your Tasks and Task Queues	http://code.google.com/appengine/docs/python/taskqueue/rest.html	https://www.googleapis.com/auth/taskqueue
TaskQueue API	Consume Tasks from your Task Queues	http://code.google.com/appengine/docs/python/taskqueue/rest.html	https://www.googleapis.com/auth/taskqueue.consumer
Tasks API	Manage your tasks	http://code.google.com/apis/tasks/v1/using.html	https://www.googleapis.com/auth/tasks
Tasks API	View your tasks	http://code.google.com/apis/tasks/v1/using.html	https://www.googleapis.com/auth/tasks.readonly
URL Shortener API	Manage your goo.gl short URLs	http://code.google.com/apis/urlshortener/v1/getting_started.html	https://www.googleapis.com/auth/urlshortener
YouTube API	View and manage your assets and associated content on YouTube	https://developers.google.com/youtube	https://www.googleapis.com/auth/youtubepartner
YouTube API	Manage your YouTube videos	https://developers.google.com/youtube	https://www.googleapis.com/auth/youtube.upload
YouTube API	Manage your YouTube account	https://developers.google.com/youtube	https://www.googleapis.com/auth/youtube
YouTube API	View your YouTube account	https://developers.google.com/youtube	https://www.googleapis.com/auth/youtube.readonly
YouTube Analytics API	View YouTube Analytics reports for your YouTube content	http://developers.google.com/youtube/analytics/	https://www.googleapis.com/auth/yt-analytics.readonly

Kapow Kapplets User's Guide

Kapow Kapplets expose a friendly user interface to robots. A Kapplet administrator can make execution of one or more robots available to users who need not know anything about robots. They will interact with the robots through the Kapplet, which wraps the execution of one or more robots in parallel. A Kapplet can be customized to match the terminology of the end-users; an icon and a description can also be attached.

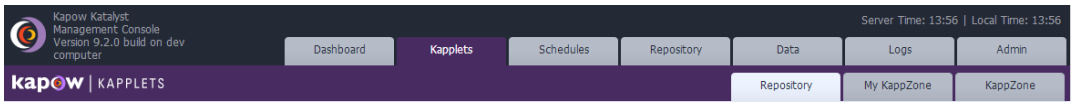
A Kapplet is build and maintained by a Kapplet administrator in the Kapplet builder. When the Kapplet is ready for use it is made available for all Kapplet users through the KappZone where the users can install the Kapplets into their individual My KappZone. Users keep their own list of installed Kapplets in My KappZone or bookmark Kapplets in their browser.



Kapplets build on the user/role management of the Management Console. With the Management Console deployed in a stand-alone application server, administrators can manage which users have access to Kapplet-exposed robots in which projects using LDAP.

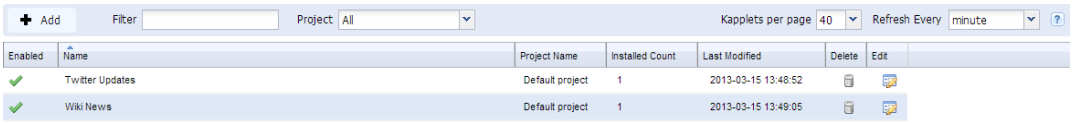
Building and Maintaining Kapplets

In the Management Console Kapplets figure as separate entities that can be managed from the tab called 'Kapplets'.

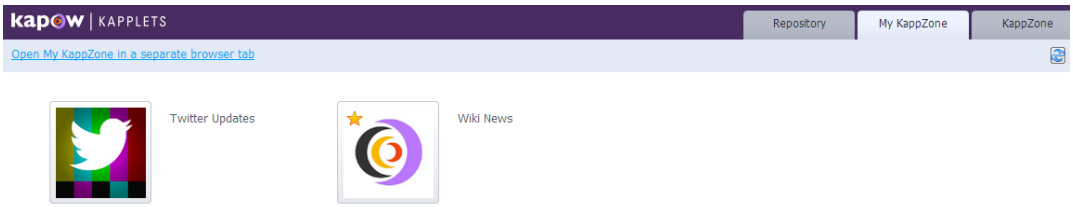


This part of the Management Console exposes three sub-sections:

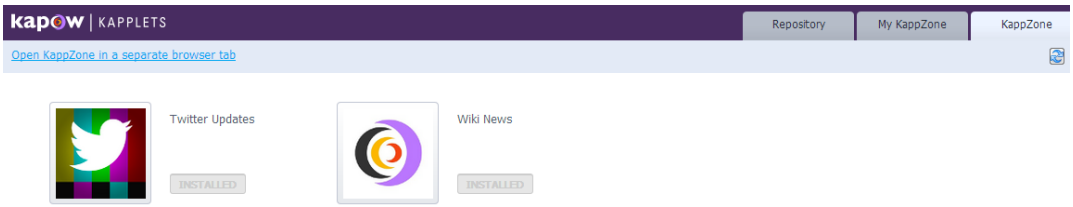
1. *The Repository* exposes a list of currently defined Kapplets. This is where you to add, delete, and edit Kapplet definitions. This sub-section will be further described below in Section Creating, Deleting, and Editing Kapplets.



2. *My KappZone* exposes the Kapplets that are currently in use by you. From here you can run Kapplets and select/deselect you current favorites. This sub-section is fully described in Installing and Using Kapplets and will not be further described here.



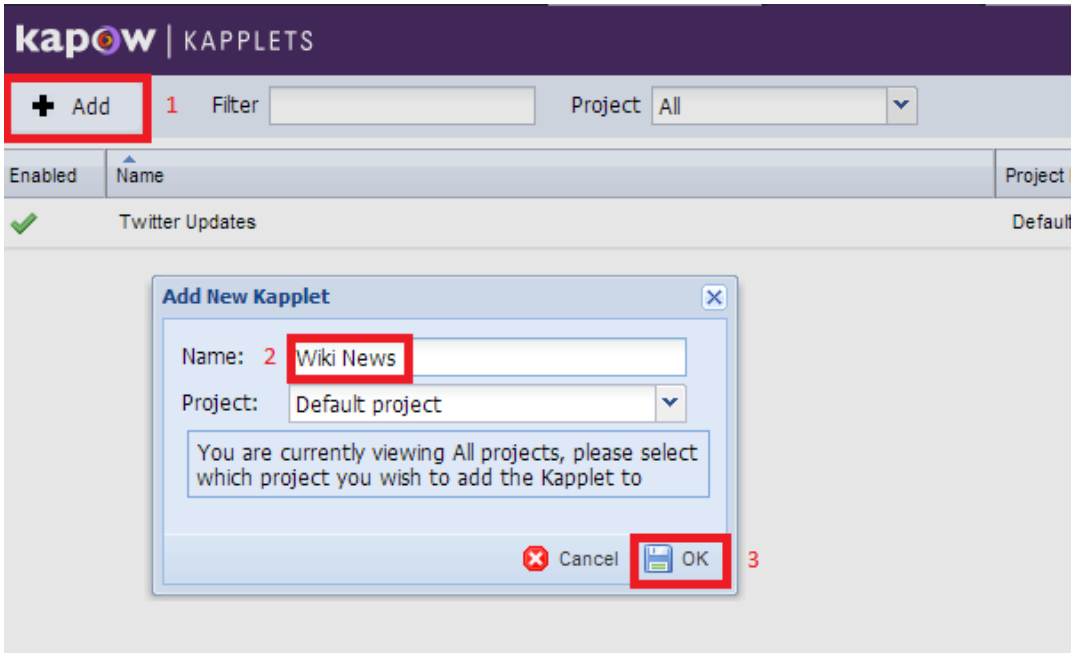
3. *The KappZone* exposes all Kapplets that you are currently allowed to use. From here you can install Kapplets to be available in My KappZone. This sub-section is fully described in Invoking Kapplets and will not be further described here.



For users of the Enterprise edition it is important to know that not all Management Console users can use, not to mention administrate, Kapplets. In order to assign such rights to a particular user you have to go to the Permissions tab of a project. Here you can configure which user groups should be Kapplet Administrators and Kapplet Users for this particular project. The security options of the Standard edition are coarser and all users have permission to view and edit Kapplets.

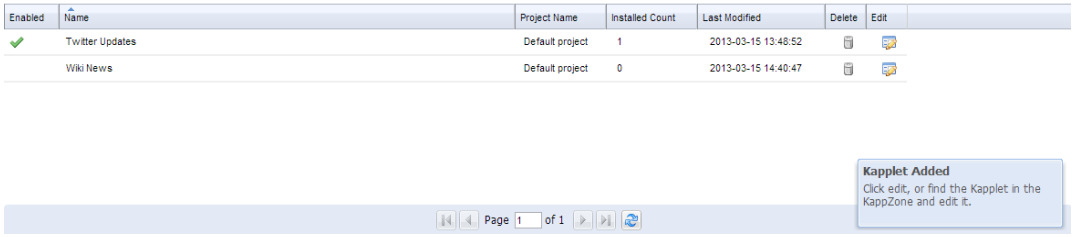
Creating, Deleting, and Editing Kapplets

In order to create a Kapplet a Kapplet Administrator must go to the Repository sub-section and click 'Add' in the top left corner. This will open a dialog, where the Kapplet Administrator must choose a project to host the new Kapplet and suggest a name for it.



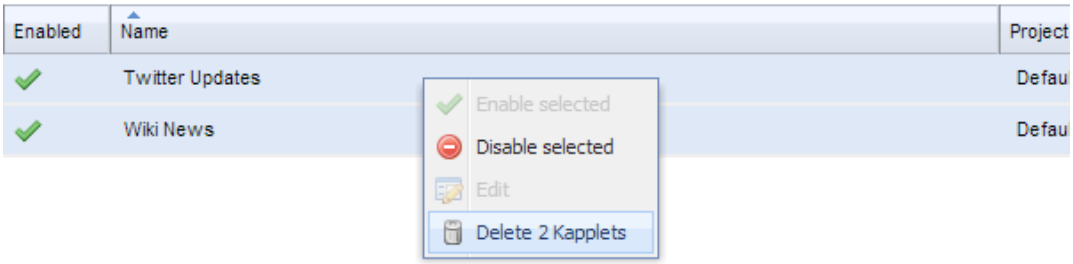
Once this information has been entered the new Kapplet is added to the list of Kapplets. Click the edit icon to open the Kapplet Designer where you can configure the details of the new Kapplet. The Kapplet Designer will be discussed in detail in Using the Kapplet Designer and will not be discussed further here.

When the new Kapplet has been created by completion of the dialog it is already stored and can be seen in the list exposed by the Kapplet Repository.

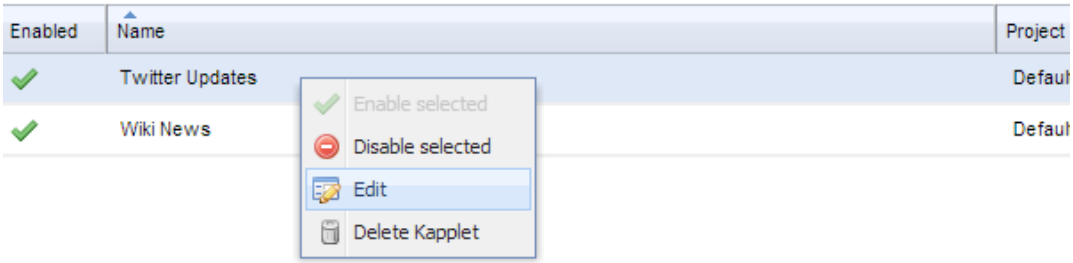


Note, however, that the new Kapplet is not yet available to any Kapplet user beyond the Kapplet creator herself. In order to change this the creator must right-click the Kapplet in the Repository and select the 'Enable selected' option, after which the Kapplet can be installed from the KappZone of other Kapplet users.

Should a Kapplet Administrator decide to remove a particular Kapplet from the system she can do so by clicking the trash icon ( ) next to the Kapplet in the Repository. The delete functionality can also be invoked by right-clicking the Kapplet and selecting the 'Delete Kapplet' option - if more than a single Kapplet is selected they will all be deleted.



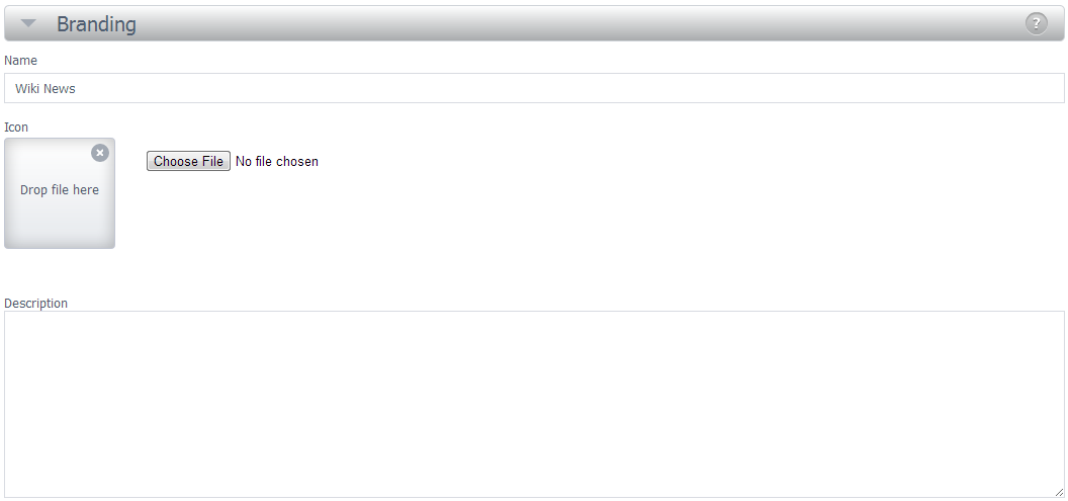
Similarly, the Kapplet can be reopened in the Kapplet Designer by either clicking the editor icon ( ) next to it or right-clicking and selecting the 'Edit' option.



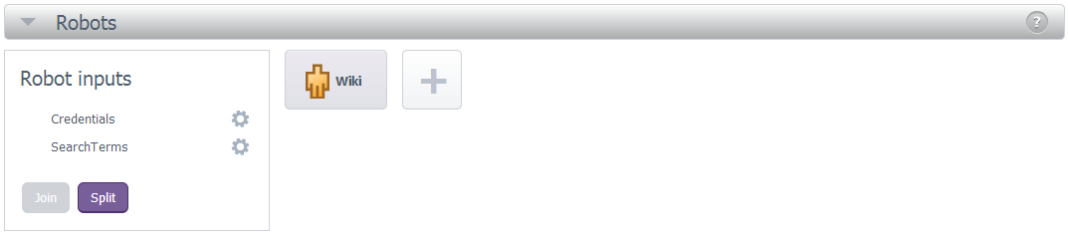
Using the Kapplet Designer

The Kapplet Designer is where the functionality and the user interface of Kapplets are determined. The designer has four sections, each allowing the configuration of specific details:

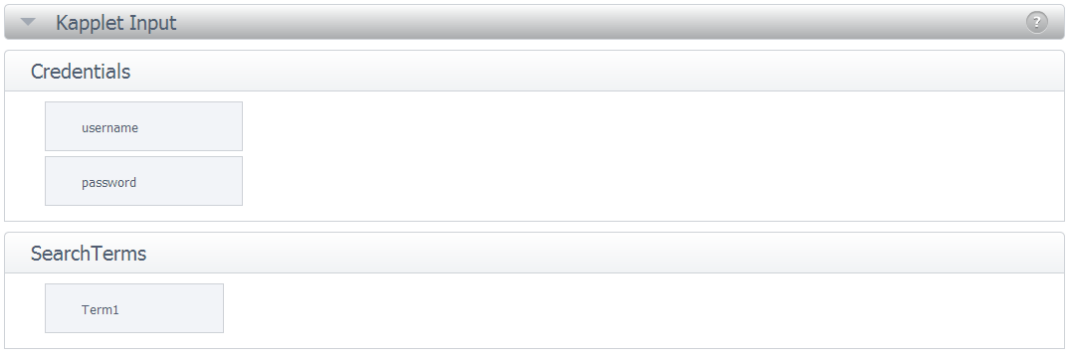
- The *Branding* section allows the association of an icon, by drag and drop or ordinary file upload, the change of Kapplet name, and the association of a descriptive or instructional text.



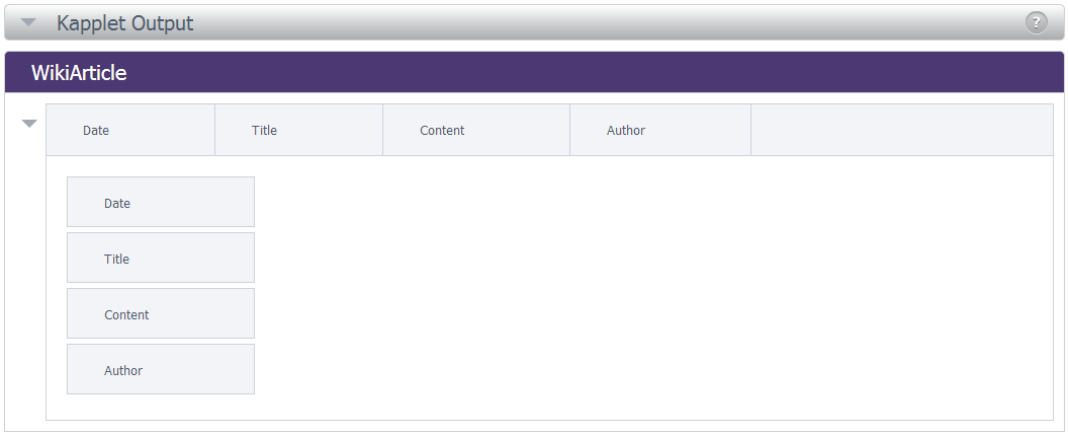
- The *Robots* section allows the association of multiple robots to be executed in parallel when the kapplet is run, the management of robot input variables, and assignment of default values to these variables. This section is further discussed in The Robots Section.



- The *Inputs* section allows the customization of the Kapplet input in regards to display order, display naming, and hiding/unhiding of variables and attributes. This section is further discussed in The Inputs Section.



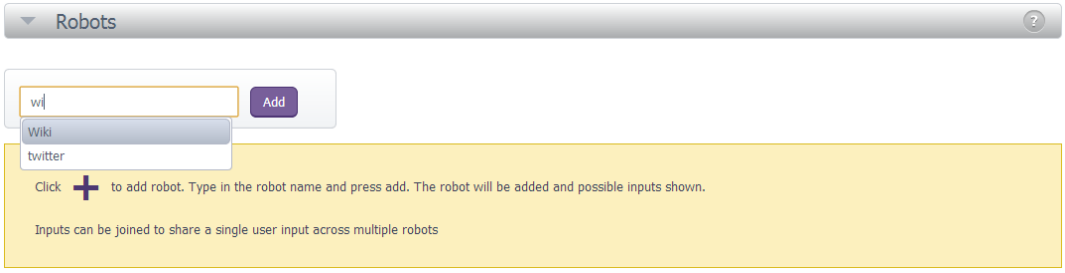
- Finally, the *Outputs* section allows the customization of the Kapplet output in regards to display order, display naming, and hiding/unhiding of types and attributes. This section is further discussed in The Output Section.



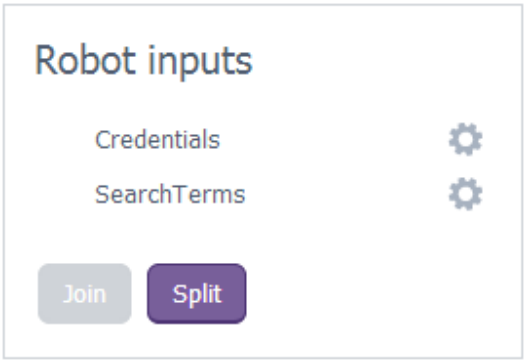
Note, that all changes made in the Kapplet Designer are persisted and effective immediately - even before you leave the Kapplet Designer. For this reason it is often a good idea to disable the Kapplet before opening it for maintenance.

The Robots Section

A Kapplet can be associated with any number of robots to be executed in parallel whenever the Kapplet is run by a user. Each such robot must be added by clicking the plus (+) button in the Robots section of the Kapplet Designer. This will expose an input field with auto-completion where any existing robot may be selected and added to the Kapplet.



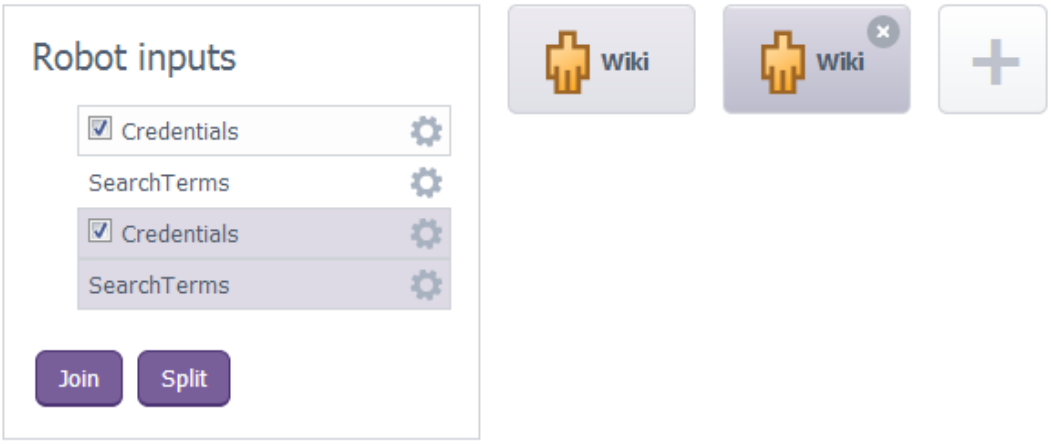
When the first robot is added the leftmost area of the Robots section changes into an Input Manager displaying the input variables currently required by the Kapplet.



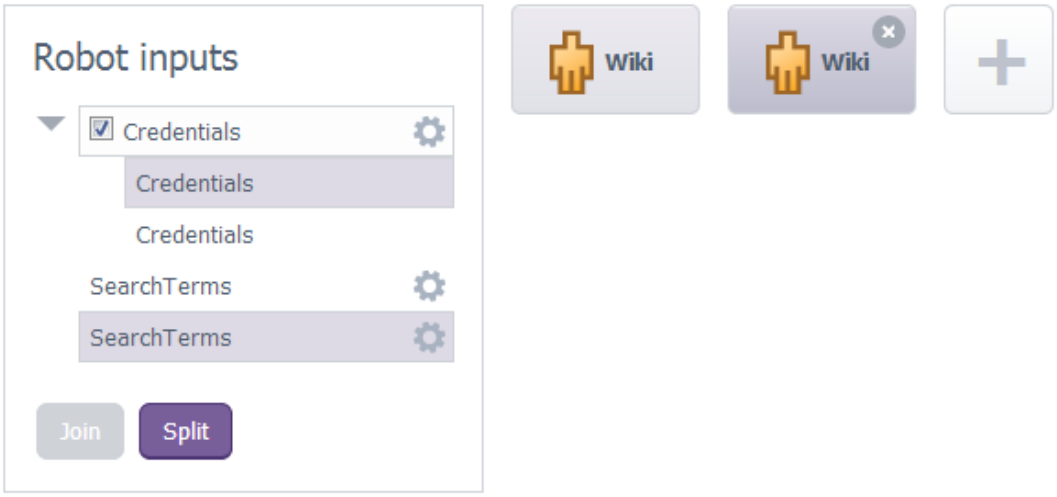
The rightmost area is updated to show a symbolic representation of the robot in addition to the plus button.

Every subsequent addition of a robot will update both the input manager and the robot list.

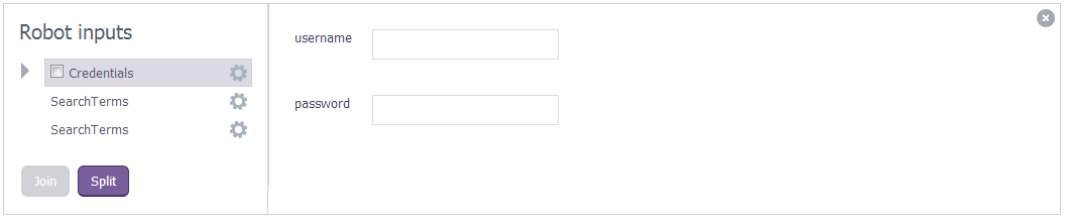
Initially, the input variables displayed in the leftmost list simply reflect the input requirements of the added robots. However, the robots added to a Kapplet are often related - they may, e.g., all collect data from the same site. Often some input variables are then also be related - it may, e.g., make sense to prime all robots with the same login credentials (but different search terms). Such related input variables can be joined into a single variable by selecting them in the input manager and clicking *join*.



Such a joined input may later be dissolved into its constituent inputs by selecting it and clicking *split*.



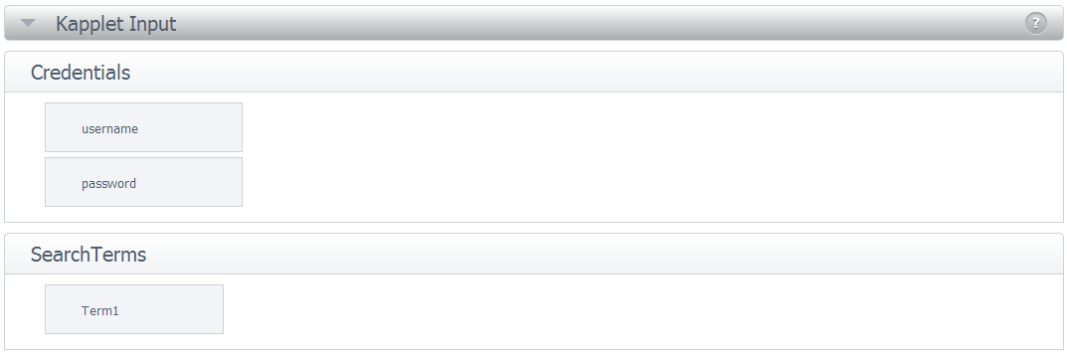
Other inputs may have a nature that mostly requires the same input for all Kapplet users. Such inputs can be given a default value by clicking the settings symbol ( ) on the variable and filling in the appropriate fields in the form that then opens over the robot list.



If no further changes are made the given default value is merely a suggestion to the user and may be changed at run-time. The Kapplet Administrator may decide, however, to hide the field from the user, as described in The Inputs Section, in which case the value is essentially a hidden constant.

The Inputs Section

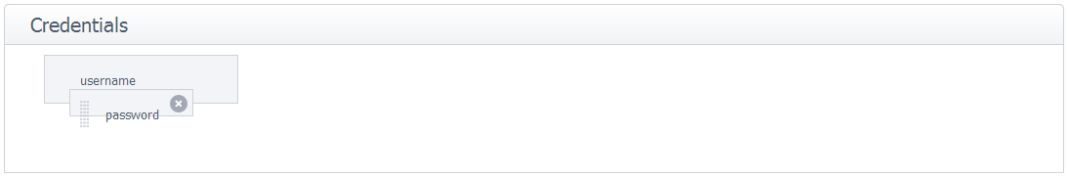
When robots have been added to a Kapplet (and the corresponding inputs have been managed appropriately as described in The Robots Section) the Kapplet Administrator can go on to the Inputs section and determine the appearance of the input form presented to the Kapplets Users when running the Kapplet.



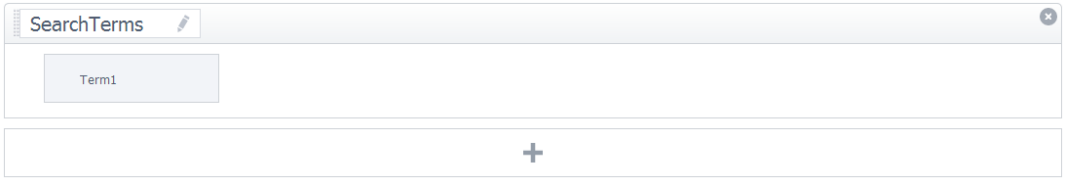
This section essentially consists of an ordered list of subsections, one for each of the required input variables, each comprising an ordered list of variable attributes. These lists may be manipulated in various ways in order to define the appearance of the runtime input form:

The order of variables, i.e. subsections, may be changed by drag and drop and doing so will change the order of presentation in the runtime input form correspondingly.

Similarly, attributes may be dragged and dropped in order to change the order in which they are presented to the Kapplet user.



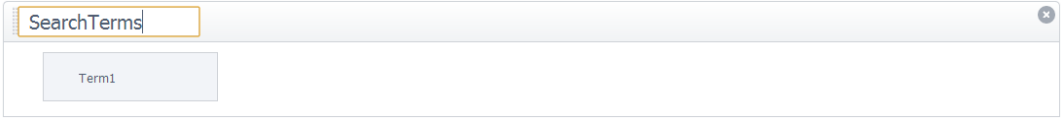
If the Kapplet Administrator wishes to hide an input variable from the users she may do so by clicking the remove symbol ( ) on the the title bar of the corresponding subsection. Note, that this usually requires that default input has been defined for the attributes of the variable as described in The Robots Section. In the same manner, specific variable attributes may be hidden by a click on their remove symbol ( ). If one or more variables are hidden a plus button will be present at the bottom of the Inputs section.



Clicking this button will allow you to un-hide a variable by selecting it from a list.

Similarly, hidden attributes can be un-hidden using the plus button of the appropriate subsections.

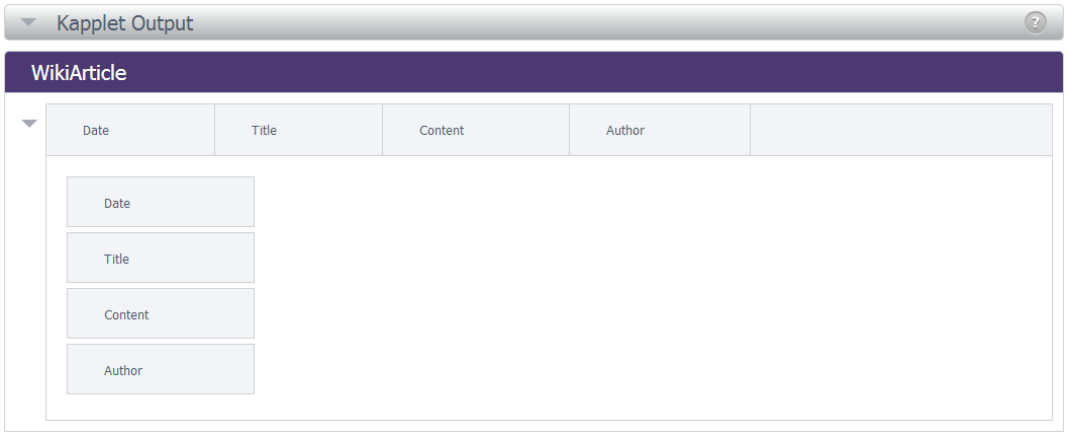
The display names of variables are initially the same as the variable names defined in the underlying robots. Similarly the display names of attributes are initially the same as the attribute names defined in the accompanying types. Both may be changed, however, simply by clicking on them. Doing so enables an underlying text field that may be edited.



Note, that changes to display names are effectuated immediately when focus leaves the text field or enter is pressed.

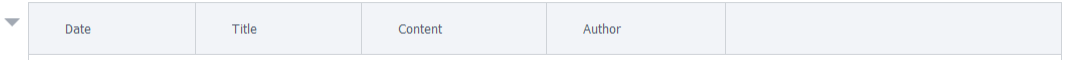
The Outputs Section

Like the input form the output presentation can also be determined by the Kapplet Administrator once robots have been added; this is the purpose of the Outputs section. Like the Inputs section it consists of an ordered list of subsections, one for each type of output that may be returned from one of the selected robots.



In contrast to the input section, however, each section comprises *two* ordered lists of attributes that may be configured independently:

- The horizontal list controls the overall display of the output tables.



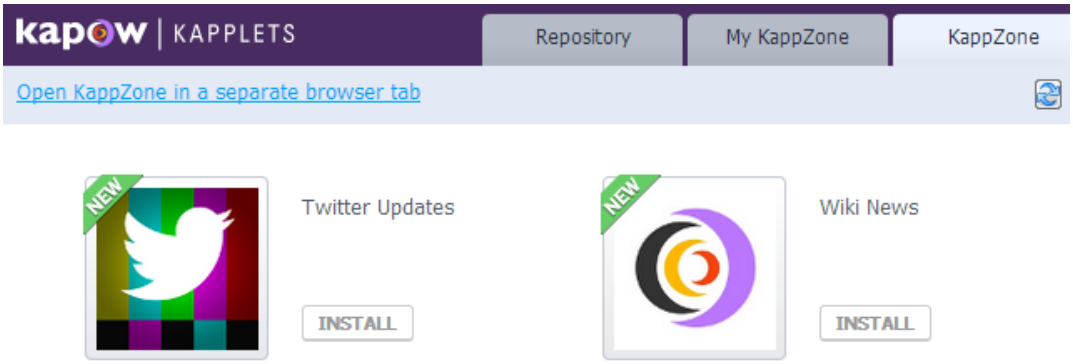
- The vertical list controls the details shown when a table row is opened up for further information. The vertical list also controls the output for the Excel download.

The customization options are essentially the same as described for the The Inputs Section. The one notable exception is that two display modes are configured simultaneously, which imposes only a single restriction:

It is not possible to give different display names to the same attribute in the two different view modes. All other settings, however, may be configured independently.

Installing and Using Kapplets

For non-administrative users, Kapow Kapplets will typically be accessed directly from the web browser by adding `kappzone` to the Management Console URL. For instance, if the Management Console is deployed at `http://localhost:50080/`, Kapplets can be accessed directly at `http://localhost:50080/kappzone`.

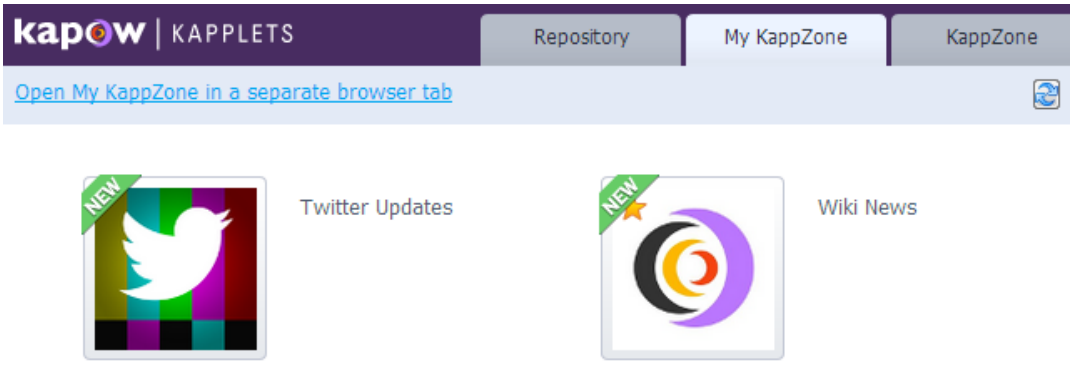


Available Kapplets; see how the Kapplet are marked as new if the user has not seen them before.

Users have access to Kapplets in the KappZone tab. When the user *Install* a Kapplet a instance of the Kapplet is made available in the users My KappZone. A Kapplet can be installed several times by a user - creating separate instances of the Kapplet (*installed Kapplets*) in the users My KappZone. Each of the installed Kapplets can then be configured with input values, scheduling options and notifications.

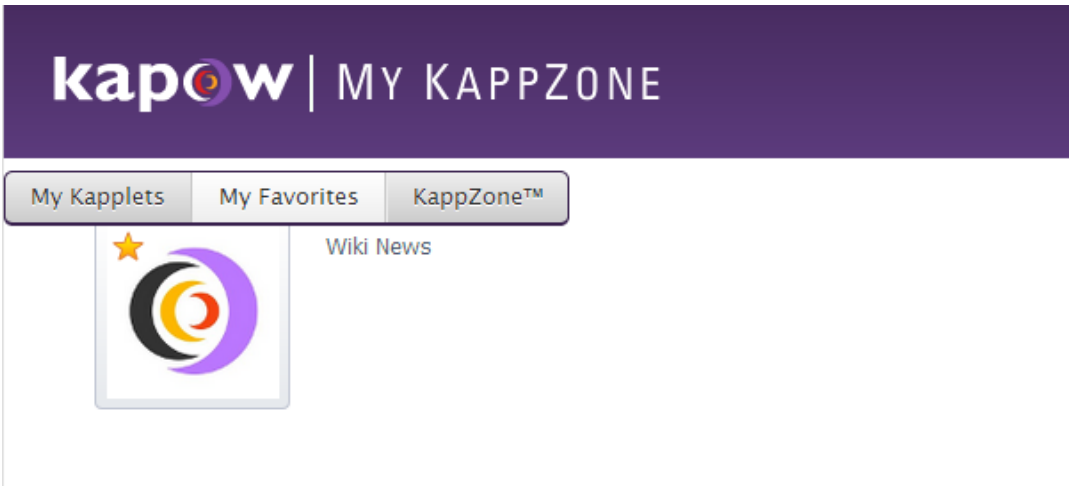
Users can bookmark installed Kapplets from My KappZone in their browser, and they can "star" installed Kapplet to add them to their display of favorites. In this way the user can navigate smoothly to the most often used Kapplets.

To bookmark a Kapplet the user has to identify the Kapplet URL. For users that are only Kapplet users the URL is shown on the browser and they can simply bookmark the current page. If the user is also a robot developer then the user can right click the installed Kapplet and copy the URL to bookmark the URL.



Installed Kapplets; see how the "Wiki News" Kapplet has been starred as a favorite by the user.

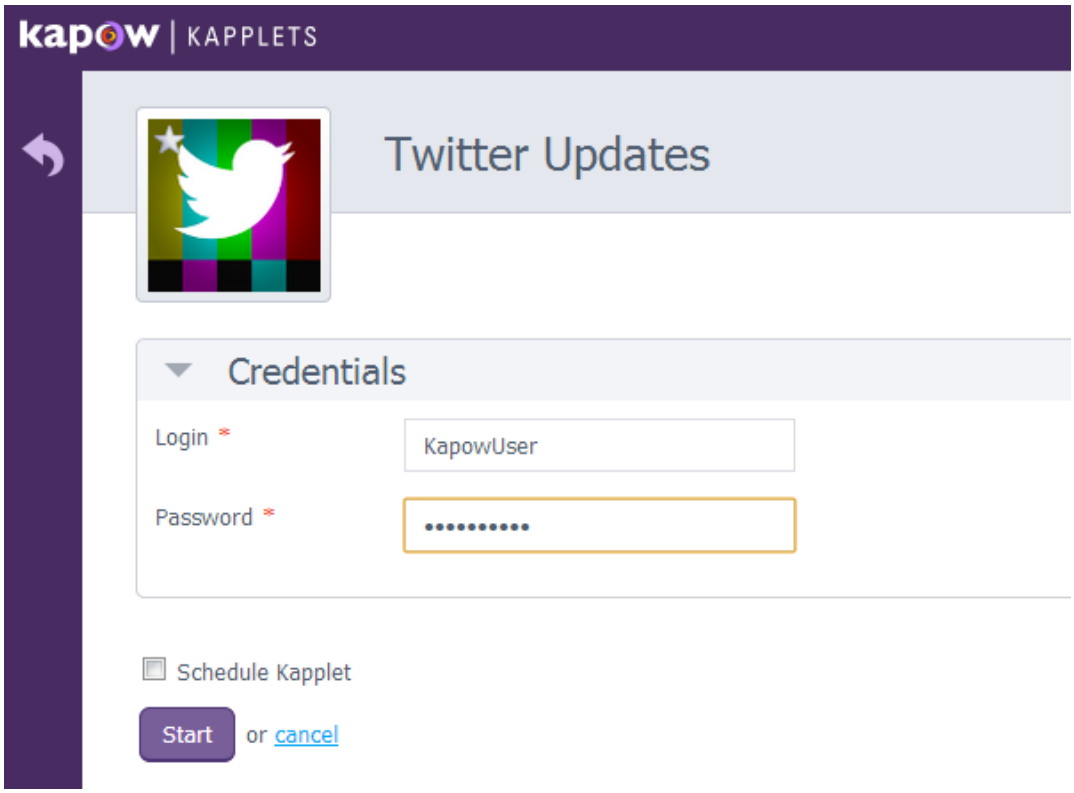
After opening the KappZone in a separate browser (see the link in the screen dump above) the user can click *My Favorites* that shows the user's favorite installed Kapplets.



Favorite Kapplets; see how only the "starred" Kapplets are shown.

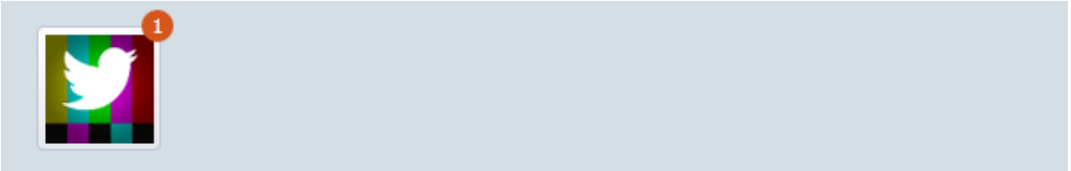
Invoking Kapplets

To run an installed Kapplet, simply click it - either in My KappZone or in My Favorites, and fill out all of the required fields and click the Start button. Required input fields for a Kapplet are marked with a star.



The main Kapplet dashboard will also indicate using a pulsating icon if a Kapplet is currently running. The latest Kapplet runs are always stored and can be re-found by clicking the "stored result(s)" link at the bottom when that Kapplet is open (see Viewing Results for a description of viewing the results).

Unread results will show up as notification icons on the Kapplet in the lower part of the screen for My KappZone.

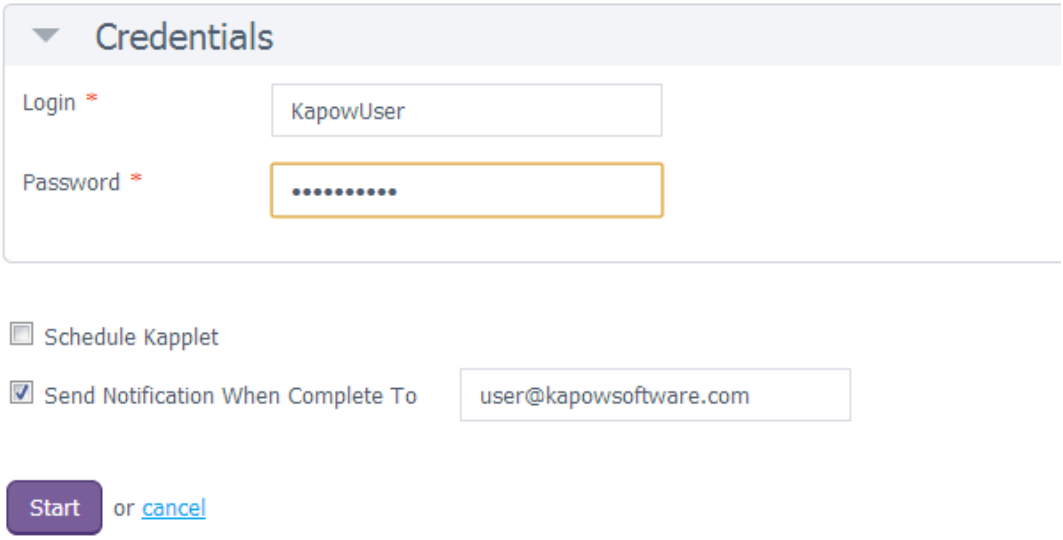


If a user tries to start a Kapplet without values in the required fields then the Kapplet will not start. Instead the user will be prompted for input.

Email Notifications from Kapplets

For long-running Kapplets, the user might not want to sit around waiting for the result page to populate. He or she can then enter their e-mail address when starting a Kapplet run and will receive an e-mail when the result is ready.

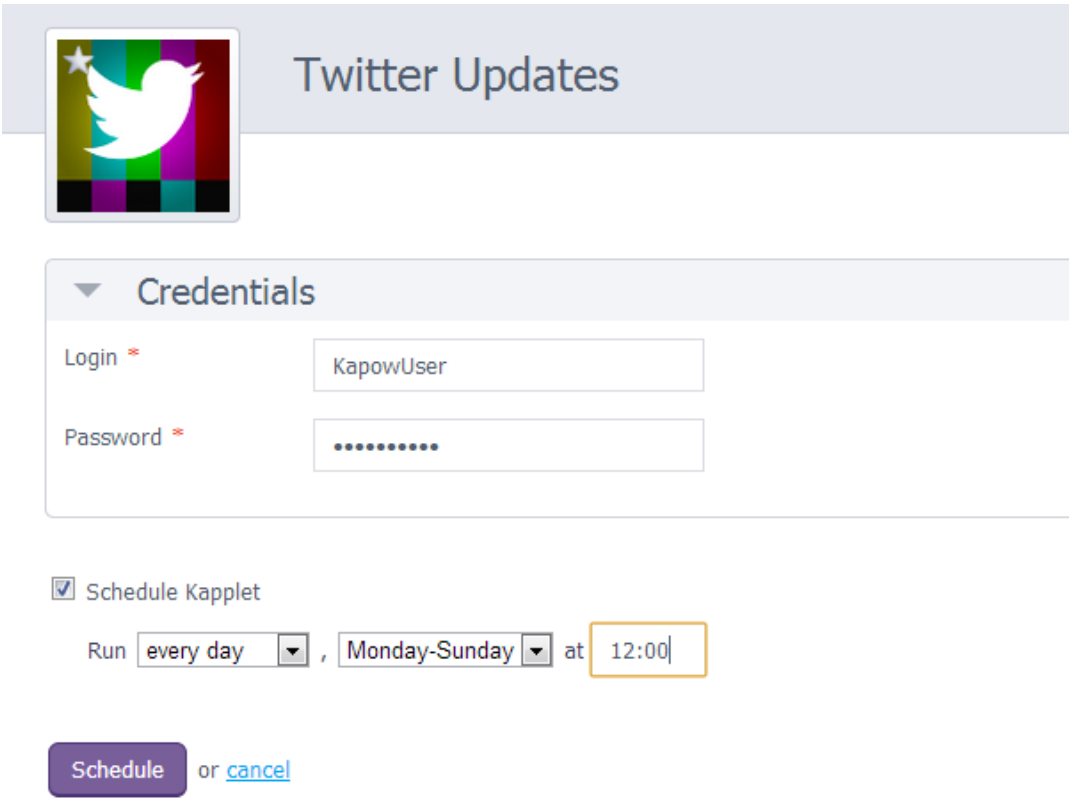
Before a user can configure notifications for Kapplets a (SMTP server must be configured in the Management Console). Below is shown an installed Kapplet where the SMTP server is configured and the users email address has been entered.



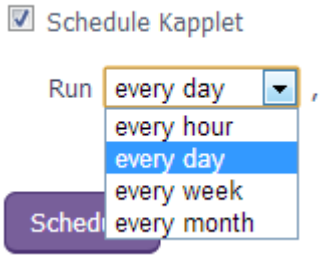
When the user click the *Start* button the Kapplet will run as usual at an email will be sent to the user when the Kapplet has finished.

Scheduling Kapplets

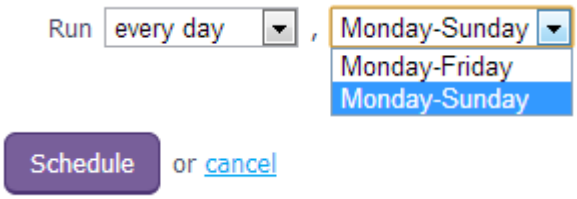
Some Kapplets should run with a predefined interval like every day at noon or every Friday at 4.50 pm. To schedule a Kapplet select the checkbox *Schedule Kapplet*



... select a period for the scheduling

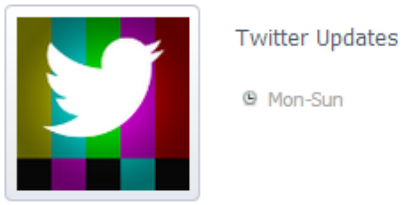


... and select possible weekdays.



Notice that the *Start* button chances to a *Schedule* button. When the Kapplet is scheduled it will be run periodically as specified by the user - which is indicated by a small note at the bottom of the page.

When the user lists the installed Kapplets the schedule is shown on the Kapplet.



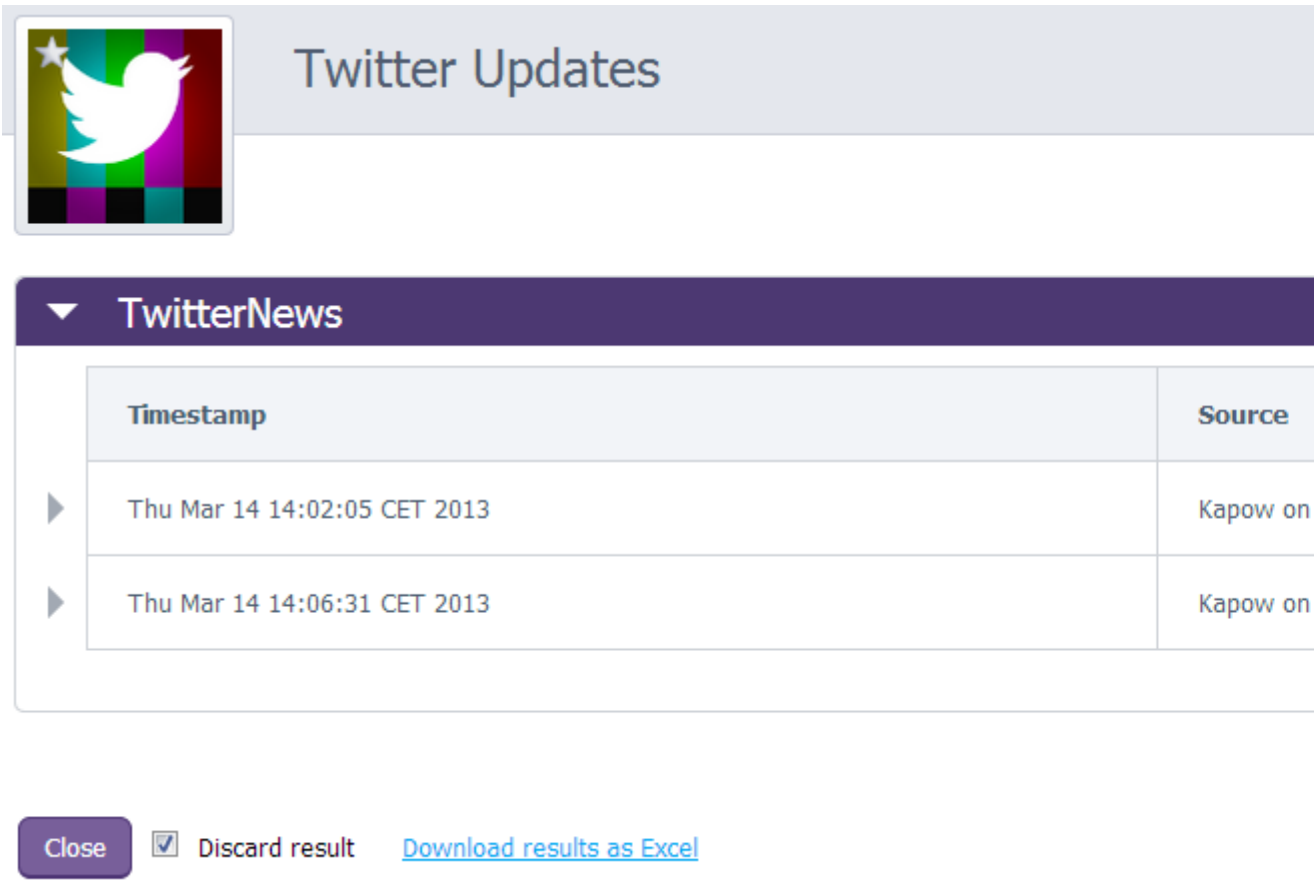
The user deletes a schedule by clicking the trash can symbol for the selected schedule.

Notice that scheduling and email notification can be combined for Kapplets.

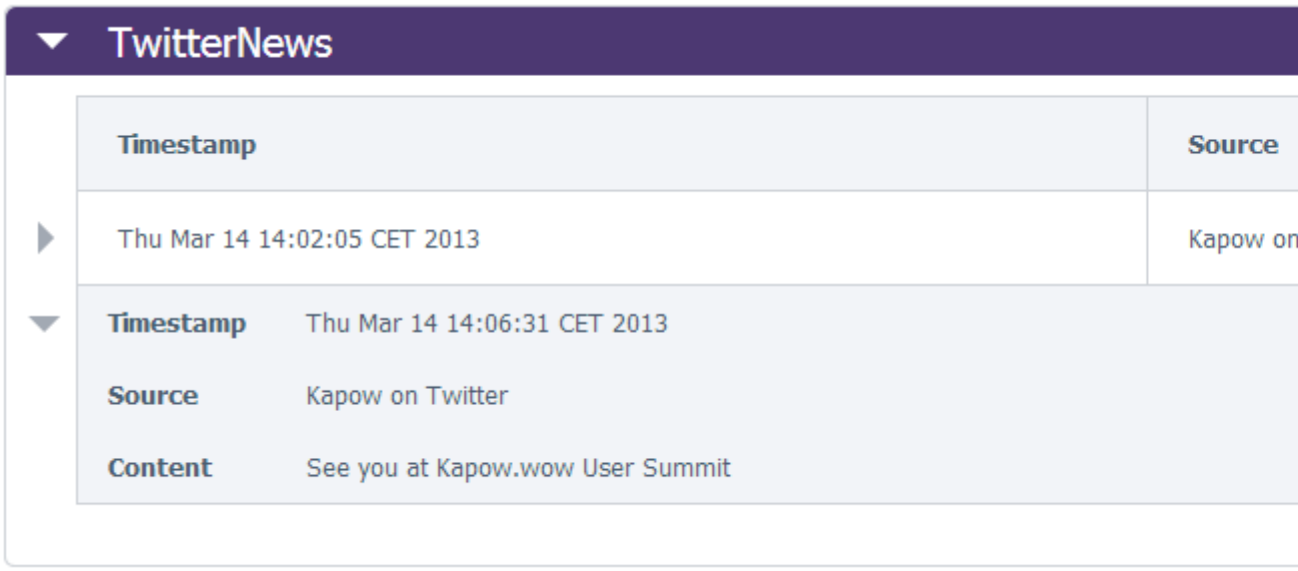
Viewing Results

Data will be displayed in a result page. The output is customized in the Kapplet Builder (see The Outputs Section for a description of customizing the results).

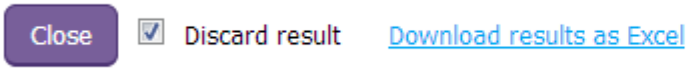
The output is initially presented in a column view that presents the user with an overview of the results from the Kapplet.



If the user needs to see details from a given result then pressing the *arrow* at the left of the result will expand the result to a detailed view of that specific result.



When the user has finished navigating through the result it is possible for the user to *Close* the result.



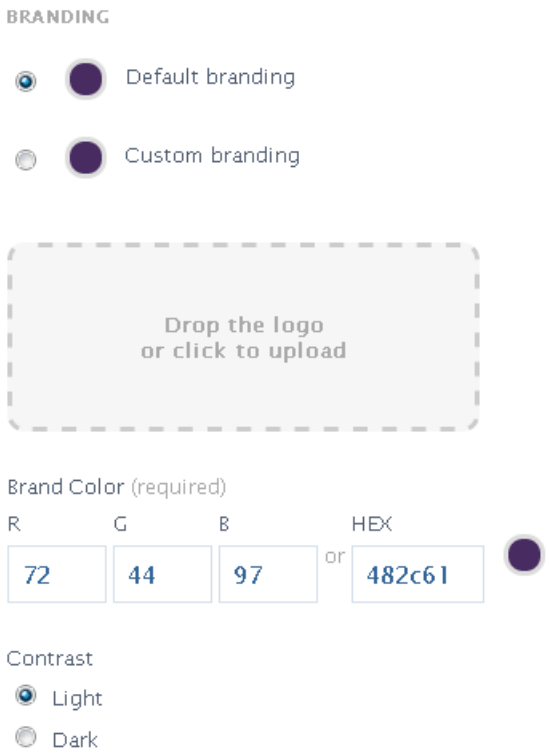
If checkbox *Discard Result* is checked then closing the result will also discard the result, otherwise the result will be stored for later access by the user.

If the user wish to work further with the Kapplet result in 3rd party programs then the Kapplet result can be downloaded in the Excel format.

Customizing the Branding

As of version 9.2.2, you can customize the branding of KappZone and My KappZone to comply with your corporate color scheme, as well as to replace the Kapow logo with your own.

To rebrand KappZone and My KappZone, go to the Branding tab in the Kapplets section of the Management Console. Initially, the default (Kapow) branding will be used.



To change the branding, first select "Custom branding". Then specify the color code that will be used as background for the top bar in the My KappZone section. If you have selected a dark color - such as the purple used in the default branding - you should pick "light" as the contrast color, which will be used for the "My KappZone" text as well as some icons that will be placed on top of your selected background color. If you have selected a light background color, a dark contrast is recommended.

The logo - which will replace the Kapow logo at the top of My KappZone and KappZone, as well as in a smaller version in some subpages can either be uploaded by dragging it to the gray drop zone or clicking the drop zone if you are using a browser that does not support drag and drop.

A preview of the selected logo and color will be shown on the right:

Programming With Robots

Robots are executed on RoboServer through an API (Java or .Net). You may use the API directly in your own application or indirectly when you execute robots using the Management Console.

This part of the documentation will look at the direct execution of Robot using the Java and .Net APIs

- The Java Programmer's Guide describes the API that can be used in Java programs.
- The .NET Programmer's Guide describes the API to use in .NET applications, including C# programs.

API reference documentation is available for Java, .NET (the Guides contain the appropriate links). The reference documentation is also installed as part of the installation, making it easy to use from your development environment.

Java Programmer's Guide

This guide describes how to execute Robots using the Kapow Katalyst Java API. The guide assumes that you have completed the Design Studio tutorials and know how to write simple Robots and that you are familiar with the Java programming language.

The programmer's guide has been completely rewritten for version 8.3, as large portions of the API has been deprecated, and a new execution API has been created. The API is still backwards compatible, but you should familiarize yourself with the new API and consider rewriting existing application to use the new API, as the deprecated classes will be removed in future releases.

The old `RobotExecutor` has been deprecated because of the following reasons

- Robots would continue to execute on RoboServer even after an RQLException was thrown by the API.
- Robots would continue to execute even if RoboServer lost connection to the client, resulting in log errors for every Return Value executed.
- The distribution policies didn't look at server capacity when distributing requests.
- It was cumbersome to implement object streaming, because there were many hidden pitfalls when implementing a custom `RQLHandler`

You can still find the old Java programmer's guide, at http://help.kapowtech.com/8.2/topic/doc/java/Top.html

Details about specific classes can be found in the JavaDoc [api/overview-summary.html]

Basics

Robots run by the Management Console are executed using the Java API. The Java API allows you to send requests to a RoboServer that instructs it to execute a particular robot. This is a classic client-server setup in which Management Console acts as the client and RoboServer as the server.

By using the API, any Java based application can become a client to RoboServer. In addition to running robots that store data in a database, you can also have the robots return data directly back to the client application. Here are some examples:

- Use multiple robots to do a federated search, which aggregates results from multiple sources in real time.
- Run a robot in response to an event on your application back-end. For instance run a robot when a new user signs up, to create accounts on web-based systems not integrated directly into your back-end.

The basic section of this guide will introduce the core classes, and how to use them for executing robots. We will also describe how to provide input to robots, and control their execution on RoboServer.

The Java API is a jar file, and it is located in `/API/robosuite-java-api/lib/robosuite-api.jar` inside the Kapow Katalyst installation folder, see important folders for details. All examples in this guide can also be found in `/API/robosuite-java-api/examples`. Located next to the Java API are 5 additional jar files which comprise the external dependencies of the API. Most basic API tasks such as executing robots can be done without using any of these 3rd party libraries, while some advanced features do require the usage of one or more of these 3rd party libraries. The examples in this guide will specify when such libraries are required.

First Example

Let's start by looking at the code required to execute the robot named `NewsMagazine.robot`, which is located in the Tutorials folder of the default project. The robot outputs its results using the *Return Value* step action, which makes it easy to handle the output programmatically using the API. Other robots (typically those run in a schedule by the Management Console) store their data directly in a database using the *Store in Database* step action, in which case data collected by the robot will not be returned to the API client.

In the following, we will look at how to execute the NewsMagazine robot and process the output programmatically.

Table 41. Execute a Robot without input

```
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

        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();
        }
    }
}
```

Let's start by looking at the classes involved and their responsibilities.

Table 42.

`RoboServer [api/com/kapowtech/robosuite/api/java/repository/construct/RoboServer.html]`	This is a simple value object that identifies a RoboServer which can execute robots. Each RoboServer must be activated by a Management Console and assigned KCU before use.
`Cluster [api/com/kapowtech/robosuite/api/java/repository/construct/Cluster.html]`	A cluster is a group of RoboServer functioning as a single logical unit.
`Request [api/com/kapowtech/robosuite/api/java/rql/Request.html]`	This class is used to construct the robot request. Before you can execute any requests you must register a cluster with the Request class.
`DefaultRobotLibrary [api/com/kapowtech/robosuite/api/java/rql/construct/DefaultRobotLibrary.html]`	A robot library instructs RoboServer where to find the robot identified in the request. Later examples will explore the various robot library types and when/how to use them.
`RQLResult [api/com/kapowtech/robosuite/api/java/rql/RQLResult.html]`	This contains the result of a robot execution. The result contains value responses, log and server messages.
`RQLObject [api/com/kapowtech/robosuite/api/java/rql/construct/RQLObject.html]`	Each value that is returned from a robot using the *Return Value* action can be accessed as an RQLObject.

Now let's go through each line in the example an look at the specifics.

This tells the API that our RoboServer is running on localhost port 50000.

Table 43.

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

This defines a cluster with a single RoboServer. The cluster is registered with the `Request` class, allowing you to execute request on this cluster. Each cluster can only be registered once.

Table 44. Registering a cluster

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

This creates a request that will execute the robot named *NewsMagazine.robot* located at *Library:/Tutorials*. *Library:/* refers to the robot Library configured for the request. Here the `DefaultRobotLibrary` is

used, which instructs RoboServer to look for the robot in the servers local file system, see Robot Libraries for details on how to use robot libraries.

Table 45.

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

This executes the robot on the cluster named `MyCluster` (the cluster we previously registered) and returns the result once the robot is done. By default execute will throw an exception if the robot generates an API exception.

Table 46.

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

Here we process the extracted values. First we get all extracted values of the type named *Post* and iterate through them. For each RQLObject we access the attributes of the Post type and print the result. We will look at attributes and mappings in a later section.

Table 47.

```
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);
}
```

Robot Input

Most robots executed through the API will be parameterized through input, such as a search keyword, or login credentials. Input to a robot is part of the request to RoboServer and is provided using the `createInputVariable` method on the request. Let us look at a short code fragment.

Table 48. Input using implicit `RQLObjectBuilder` [api/com/kapowtech/robosuite/api/java/rql/RQLObjectBuilder.html]

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

Here we create a `Request` and use `createInputVariable` to create an input variable named *userLogin*. We then use `setAttribute` to configure the username and password attributes of the input variable.

The above example is a common shorthand notation, but can also be expressed move verbosely by using the `RQLObjectBuilder`:

Table 49. Input using explicit `RQLObjectBuilder` [api/com/kapowtech/robosuite/api/java/rql/RQLObjectBuilder.html]

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

The two examples are identical. The first utilizes the cascading method invocation on the anonymous `RQLObjectBuilder` is therefore shorter.

When RoboServer receives this request the following occurs:

- RoboServers loads *Input.robot* (from whatever `RobotLibrary` is configured for the request).
- RoboServer verifies that the robot has a variable named *userLogin* and that this variable is marked as input.
- RoboServer now verifies that the attributes we have configured using `setAttribute` are compatible with the type of variable userLogin. Which means that the type must have attributes named username and password and that these must both be text based attributes (the next section will describe the mapping between API and Design Studio attributes).
- If all input variables are compatible, RoboServer will start executing the robot.

If a robot requires multiple input variables, you must create all of them in order to execute the robot. You only have to configure required attributes, any no-required attributes that you don't configure through the API will just have a null value. If we assume you have a robot that requires login to both Facebook and Twitter, you could define the input like this.

Table 50.

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

Attribute Types

When you define a new type in Design Studio you select an attribute type for each attribute. Some of these attributes can contain text, like Short text, Long Text, Password, HTML, XML, and when used inside a robot there may be requirements to the text store in these attributes. If you store text in a XML attribute, the text must be a valid XML document. This validation occurs when the type is used inside a robot, but since the API doesn't know the anything about the type it doesn't validate attribute values in the same manner. As a result the API only has 8 attribute types versus the 19 available in Design Studio This table shows the mapping between the API and Design Studio attribute types.

Table 51. API to Design Studio mapping

API Attribute Type	Design Studio Attribute Type
Text	`Short Text, Long Text, Password, HTML, XML, Properties, Language, Country, Currency, Refind Key`
Integer	`Integer`
Boolean	`Boolean`

API Attribute Type	Design Studio Attribute Type
Number	`Number`
Character	`Character`
Date	`Date`
Session	`Session`
Binary	`Binary, Image, PDF`

The API attribute types are then mapped to Java in the following way.

Table 52. Java types for attributes [api/com/kapowtech/robosuite/api/java/rql/construct/AttributeType.html]

Attribute Type	Java Class
Text	`java.lang.String`
Integer	`java.lang.Long`
Boolean	`java.lang.Boolean`
Number	`java.lang.Double`
Character	`java.lang.Character`
Date	`java.util.Date`
Session	`com.kapowtech.robosuite.api.construct.Session`
Binary	`com.kapowtech.robosuite.api.construct.Binary`

The `RQlObjectBuilder`'s `setAttribute` method is overloaded so you don't need to specify the attribute type explicitly when configuring an attribute through the API, as long as the right java class is used as argument. Here is an example that shows how to set the attributes for an object with all possible (Design Studio) attribute types.

Table 53. Recommended usage of setAttribute

```
Request request  = new Request("Library:/AllTypes.robot");
RQLObjectBuilder inputBuilder = request.createInputVariable("AllTypes");
inputBuilder.setAttribute("anInt", new Long(42L));
inputBuilder.setAttribute("aNumber", new Double(12.34));
inputBuilder.setAttribute("aBoolean", Boolean.TRUE);
inputBuilder.setAttribute("aCharacter", 'c');
inputBuilder.setAttribute("aShortText", "some text");
inputBuilder.setAttribute("aLongText", "a longer test");
inputBuilder.setAttribute("aPassword", "secret");
inputBuilder.setAttribute("aHTML", "<html>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");
```

The above example explicitly uses new Long(42L), and new Double(12.34), although 42L and 12.34 would be sufficient due to auto boxing. Also notice that we have to cast null values, because the Java compiler can't otherwise determine which of the overloaded `setAttribute` methods we want to call. However since since unconfigured attributes will automatically be null, you never need to set null explicitly.

It is possible to specify the Attribute and AttributeType explicitly when creating input using the API. This is approach is *not recommended*, but may be needed in rare cases, and would look like this.

Table 54. Not recommended usage of setAttribute

```
Request request  = new Request("Library:/AllTypes.robot");
RQLObjectBuilder inputBuilder = request.createInputVariable("AllTypes");
inputBuilder.setAttribute(new Attribute("anInt", "42", AttributeType.INTEGER));
inputBuilder.setAttribute(new Attribute("aNumber", "12.34", AttributeType.NUMBER))
inputBuilder.setAttribute(new Attribute("aBoolean", "true", AttributeType.BOOLEAN)
inputBuilder.setAttribute(new Attribute("aCharacter", "c", AttributeType.CHARACTER
inputBuilder.setAttribute(new Attribute("aShortText", "some text", AttributeType.T
inputBuilder.setAttribute(new Attribute("aLongText", "a longer test", AttributeTyp
inputBuilder.setAttribute(new Attribute("aPassword", "secret", AttributeType.TEXT)
inputBuilder.setAttribute(new Attribute("aHTML", "<html>bla</html>", AttributeType
inputBuilder.setAttribute(new Attribute("anXML", "<tag>text</tag>", AttributeType.
inputBuilder.setAttribute(new Attribute("aDate", "2012-01-15 23:59:59.123", Attrib
inputBuilder.setAttribute(new Attribute("aBinary", Base64Encoder.encode("some byte
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",
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", At
```

As we can see all attribute values must be provided in the form of strings. The string values are then converted to the appropriate Java objects based on the Attribute type provided. This is only useful if you build other generic APIs on top of the Kapow Katalyst Java API.

Execution Parameters

In addition to the `createInputVariable` method, the **`Request`** contains a number of methods that controls how the robot executes on RoboServer.

Table 55. Execution control methods on Request [api/com/kapowtech/robosuite/api/java/rql/Request.html]

`setMaxExecutionTime(int seconds)` [api/com/kapowtech/robosuite/api/java/rql/Request.html#setMaxExecutionTime(int)]	Controls the execution time of the robot. When this time has elapsed the robot will be stopped by RoboServer. The timer doesn't start until the robot begins to execute, so if the robot is queued on RoboServer this is not taken into account.

`setStopOnConnectionLost(boolean)` [api/com/kapowtech/robosuite/api/java/rql/Request.html#setStopOnConnectionLost(boolean)]	When true (default) the robot will stop if RoboServer discovers that the connection to the client application has been lost. You should have a very good reason for setting this value to false - if your code is not written to handle this, your application will not perform as expected.
`setStopRobotOnApiException(boolean)` [api/com/kapowtech/robosuite/api/java/rql/Request.html#setStopRobotOnApiException(boolean)]	When true (default) the robot will be stopped by RoboServer after the first API exception is raised. By default most steps in a Robot will raise an API exception if the step fails to execute - this is configured on the steps error handling tab. When set to false, the robot will continue to execute regardless of API exceptions, however unless your application is using the `RequestExecutor` streaming execution mode, an exception will still be thrown by execute(), so be extremely careful when setting this to false.
`setUsername(String)` [api/com/kapowtech/robosuite/api/java/rql/Request.html#setUsername(java.lang.String)], `setPassword(String)` [api/com/kapowtech/robosuite/api/java/rql/Request.html#setPassword(java.lang.String)]	Sets the RoboServer credentials. RoboServer can be configured to require authentication. When this option is enabled, the client must provide credentials or RoboServer will reject the request.
`setRobotLibrary(RobotLibrary)` [api/com/kapowtech/robosuite/api/java/rql/Request.html#setRobotLibrary(com.kapowtech.robosuite.api.java.rql.construct.RobotL	A robot library instructs RoboServer where to find the robot identified in the request. Later examples will explore the various robot library types and when/how to use them.
`setExecutionId(String)` [api/com/kapowtech/robosuite/api/java/rql/Request.html#setExecutionId(java.lang.String)]	Allows you to set the executionId for this request. If you don't provide one, RoboServer will generate one automatically. The execution ID is used for logging and also needed if your client needs to be able to stop the robot programmatically. The ID must be globally unique (in time). If two robots use the same execution ID, the logs will be inconsistent.
`setProject(String)` [api/com/kapowtech/robosuite/api/java/rql/Request.html#setProject(java.lang.String)]	This is used solely for logging purposes. The Management Console uses this field to link log messaged to project, so the log views can filter by project. If your application is not using the `RepositoryRobotLibrary` you should probably set this value to inform the RoboServer logging system which project (if any) this robot belongs to.

Robot Libraries

In Design Studio robots are grouped into projects. If you look in the file system you will see that these projects are identified by a folder named `Library`, see Libraries and Projects for details.

When you build the execute request for RoboServer, you identify the robot by a robot URL, like this:

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

Here, `Library:/` is a symbolic reference to a robot library, in which the RoboServer should look for the robot. The `RobotLibrary` is then specified on the builder like this:

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

There are three different robot library implementations, which one to select depends on you deployment environment.

Table 56. Robot Libraries

Library Type	Description
`DefaultRobotLibrary` [api/com/kapowtech/robosuite/api/java/rql/construct/DefaultRobotLibrary.html]	This configures RoboServer to look for the robot in the current project folder. This folder is defined in the Settings application. If you have multiple RoboServers you will have to deploy your robots to all RoboServers. This robot library is not cached, so the robot is reloaded from disk with every execution. This makes the library usable in a development environment where robots change often, but not suitable for a production environment.

`EmbeddedFileBasedRobotLibrary` [api/com/kapowtech/robosuite/api/java/rql/construct/EmbeddedFileBasedRobotLibrary.html]	This library is embedded in the execute request sent to RoboServer. To create this library you must create a zip file containing the robots and all its dependencies (types, snippets and resources). This can be done the *Tools->Create Robot Library File* menu in Design Studio.
	The library is sent with every request, which adds some overhead for large libraries, but the libraries cached on RoboServer, which offers best possible performance.
	
	One strength is that robots and code can be deployed as a single unit, which offers clean migration from QA environment to production environment. However, if the robots change often you will have to redeploy them often.
	
	You can use the following code to configure the embedded robot library for your request.
	
	`Request request = new Request("Library:/Tutorials/NewsMagaz`
	`RobotLibrary library = new EmbeddedFileBasedRobotLibrary(ne`
	`FileInputStream('`
	`request.setRobotLibrary(library);`

`RepositoryRobotLibrary` [api/com/kapowtech/robosuite/api/java/rql/construct/RepositoryRobotLibrary.html]	This is the most flexible `RobotLibrary`. This library uses the Management Console's built-in repository as a robot library. When you use this library, RoboServer will contact the Management Console which will send a robot library containing the robot and its dependencies. Caching occurs on a per robot basis, inside both Management Console and RoboServer. Inside Management Console, the generated library is cached based on the robot and its dependencies. On RoboServer, the cache is based on a timeout, so it doesn't have to ask the Management Console for each request. In addition, the library loading between

Library Type	Description

	RoboServer and Management Console uses HTTP public/private caching, to further reduce bandwidth.
	
	If NewsMagazine.robot has been uploaded to the Management Console we could use the repository robot library like this when executing the robot:
	
	`Request request = new Request("Library:/Tutorials/NewsMagaz`
	`RobotLibrary library = new RepositoryRobotLibrary("http://l`
	`request.setRobotLibrary(library);`
	
	This will instruct RoboServer to load the robot from a local Management Console and cache it for one minute before checking with the Management Console to see if a new version of the robot (it's type and snippets) has been changed.
	
	In addition any resource loaded through the `Library:/` protocol, will cause RoboServer request the resource directly from the Management Console.

Advanced

In this section we will look a little closer at some of the more advanced features on the API. These include output streaming, logging and SSL configuration, as well as parallel execution.

Load Distribution and Failover

Table 57. Load balanced executor

```
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);
```

Let's look a little closer at what happens inside the `RequestExecutor`. The executor is given an array of RoboServers. As the executor is constructed it tries to connect to each RoboServer. Once it is connected it sends a ping request to each RoboServer to discover how the server is configured.

Load is distributed to each online RoboServer in the cluster, based on the number of unused execution slots on the RoboServer. The next request is always distributed to the RoboServer with the most available slots. The number of available execution slots is obtained through the initial Ping response, and the executor keeps track of each robot it starts and when it completes. The number of execution slots on a RoboServer is determined by the max concurrent robots on the Servers tab.

If a RoboServer goes offline it will not receive any robot execution requests before it has successfully responded to the ping request.

Two client rule

You should only have one API client using a given cluster of RoboServer. If you have multiple JVMs running robots against the same RoboServers, this will result in reduced performance.

Executor Logger

When you execute a request, the execute method will throw an exception if a robot generates an error. Other types of errors and warnings are reported through the `ExecutorLogger` interface. In the previous examples, we have not provided any `ExecutionLogger` when executing robots, which means means we get the default implementation that will write to system out. Let's see how the `ExecutorLogger` will report if one of our RoboServers goes offline.

The example configures a cluster with a RoboServer which is not online.

Table 58. ExecutorLogger [api/com/kapowtech/robosuite/api/java/rql/engine/hotstandby/ExecutorLogger.html], offline server example

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

If you run this example if should print the following to the console:

Table 59. ExecutorLogger [api/com/kapowtech/robosuite/api/java/rql/engine/hotstandby/ExecutorLogger.html], offline RoboServer console output

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

Often you don't want to have your application writing directly to `System.out`, in that case you can provide a different `ExecutorLogger` implementation, you can do so when registering the cluster, like this

Table 60. Use `DebugExecutorLogger` [api/com/kapowtech/robosuite/api/java/rql/engine/hotstandby/DebugExecutorLogger.html]

```

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

This example uses the `DebugExecutorLogger`() which will also print to `System.out`, but only if the API debugging is enabled. Alternative you can provide your own implementation of the `ExecutorLogger`, to control how error messages should be handled. Check the `ExecutorLogger [api/com/kapowtech/robosuite/api/java/rql/engine/hotstandby/ExecutorLogger.html]` JavaDoc for additional details.

Data Streaming

Sometimes you need to present the results from a robot execution in real-time. In these cases you want the API to return the extracted values immediately instead of waiting for the robot to finish its execution and access the RQLResult.

The API offers the possibility to receive a callback every time the API receives a value that was returned by the Robot. This is done through the `RobotResponseHandler` interface

Table 61. Response streaming using `AbstractFailFastRobotResponseHandler` [api/com/kapowtech/robosuite/api/java/rql/engine/hotstandby/AbstractFailFastRobotResponseH

```
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}, fals
        Request.registerCluster(cluster);

        try {
            Request request = new Request("Library:/Tutorials/NewsMagazine.robot")
            RobotResponseHandler handler = new AbstractFailFastRobotResponseHandle
                public void handleReturnedValue(RobotOutputObjectResponse response
                    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();
        }
    }
}
```

The above example uses the second execute method of the `Request`, which expects a `RobotResponseHandler` in addition to the name of the cluster to execute the robot on. In this example we create a `RobotResponseHandler` by extending `AbstractFailFastRobotResponseHandler`, which provides default error handling, so we only need to handle the values returned by the robot.

The `handleReturnedValue` method is called whenever the API receives a returned value from RoboServer. The `AbstractFailFastRobotResponseHandler` used in this example, will throw exceptions in the same way as the non-streaming execute method. This means that an exception will be thrown in response to any API exceptions generated by the robot.

The `RobotResponseHandler` has several methods which can be grouped into 3 categories.

Robot life cycle events	Methods which are called when the robot's execution state change on RoboServer, such as when it starts and finishes its execution.
Robot data events	Methods which are called when the robot returns data or errors to the API.
Additional error handling	Methods which are called either due to an error inside RoboServer or in the API.

Table 62. RobotResponseHandler - robot life cycle events

Method name	Description
`void requestSent(RoboServer roboServer, ExecuteRequest request) [api/com/kapowtech/robosuite/api/java/rql/engine/hotstandby/RobotResponseHandler.html#requestSent(com.kapowtech.robosuite.api.java.repository.`	Called when the `RequestExecutor` has found the server which will execute the request.
`void requestAccepted(String executionId) [api/com/kapowtech/robosuite/api/java/rql/engine/hotstandby/RobotResponseHandler.html#requestAccepted(java.lang.String)]`	Called when the found RoboServer has accepted the request and put it into it queue.
`void robotStarted(Stoppable stoppable) [api/com/kapowtech/robosuite/api/java/rql/engine/hotstandby/RobotResponseHandler.html#robotStarted(com.kapowtech.robosuite.api.java.rql.engine`	Called when the RoboServer begins to execute the robot. This usually occurs immediately after the robot has been queued, unless the RoboServer is under heavy load, or used by multiple API clients.
`void robotDone(RobotDoneEvent reason) [api/com/kapowtech/robosuite/api/java/rql/engine/hotstandby/RobotResponseHandler.html#robotDone(com.kapowtech.robosuite.api.java.rql.engine.ho`	Called when the robot is done executing on RoboServer. The `RobotDoneEvent` is used to specify if the execution terminated normally, due to an error, or if it was stopped.

Table 63. RobotResponseHandler - robot data events

Method name	Description
`void handleReturnedValue(RobotOutputObjectResponse response, Stoppable stoppable) [api/com/kapowtech/robosuite/api/java/rql/engine/hotstandby/RobotResponseHandler.html#handleReturnedValue(com.kapowtech.robosuite.api.java.rql`	Called when the robot has executed a *Return Value* action and the value has been sent over the socket to the API.
`void handleRobotError(RobotErrorResponse response, Stoppable stoppable) [api/com/kapowtech/robosuite/api/java/rql/engine/`	Called when the robot raises an API exception. Under normal circumstances the robot will stop executing after the first API exception. This behavior can be overridden by using `Request.setStopRobotOnApiException(false)`, in which case this method will be called multiple times. This is useful

Method name	Description
`hotstandby/RobotResponseHandler.html#handleRobotError(com.kapowtech.robosuite.api.java.rql.co`	if you want a data streaming robot to continue to execute regardless of unhandled errors.
`void handleWriteLog(RobotMessageResponse response, Stoppable stoppable) [api/com/kapowtech/robosuite/api/java/rql/engine/hotstandby/RobotResponseHandler.html#handleWriteLog(com.kapowtech.robosuite.api.java.rql.cons`	Called if the robot executes the *Write Log* action. This is useful if you want to provide additional logging info from within a robot.

Table 64. RobotResponseHandler - additional error handling

Method name	Description
`void handleServerError(ServerErrorResponse response, Stoppable stoppable) [api/com/kapowtech/robosuite/api/java/rql/engine/hotstandby/RobotResponseHandler.html#handleServerError(com.kapowtech.robosuite.api.java.rql.c`	Called if RoboServer generates an error, for instance if the server too busy to process any requests, or if an error occurs inside RoboServer which prevents it from starting the robot.
`handleError(RQLException e, Stoppable stoppable) [api/com/kapowtech/robosuite/api/java/rql/engine/hotstandby/RobotResponseHandler.html#handleError(com.kapowtech.robosuite.api.java.rql.RQLExce`	Called if an error occurs inside the API. Most commonly if the client looses the connection to RoboServer.

Many of the methods will include a `Stoppable [api/com/kapowtech/robosuite/api/java/rql/engine/hotstandby/Stoppable.html]` object, this object can be used to stop for instance in response to a specific error or value returned.

Some of these methods allow you to throw an RQLException, if you do this you should be aware of the consequences. The thread that calls the handler is the thread the calls `Request.execute()`, this means that any exceptions thrown will bubble up the call stack and out the execute method. If you throw an exception in response to `handleReturnedValue`, `handleRobotError` or `handleWriteLog` it is your responsibility to invoke `Stoppable.stop()`, or the robot may continue to execute even though the call to `Request.execute()` has completed.

Data streaming is most often used in one of the following use cases.

- Ajax based web application, where results are presented to the user in real-time. If data was not streamed results could not be shown until the robot was done running.
- Robots that return so much data that the client would not be able to hold it all in memory throughout the robots execution.
- Processes that need to be optimized so the extracted values are processed in parallel with the robot execution.

- Processes that store data in databases in a custom format.
- Robots that should ignore or require custom handling of API exceptions (see below).

Table 65. Response and error collecting using `AbstractFailFastRobotResponseHandler` [api/com/kapowtech/robosuite/api/java/rql/engine/hotstandby/AbstractFailFastRobotResponseH

```
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}, fals
        Request.registerCluster(cluster);

        try {
            Request request = new Request("Library:/Tutorials/NewsMagazine.robot")
            request.setStopRobotOnApiException(false);  // IMPORTANT!!
            request.setRobotLibrary(new DefaultRobotLibrary());
            ErrorCollectingRobotResponseHandler handler = new ErrorCollectingRobot
            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.get
            }
        }
        catch (RQLException e) {
            e.printStackTrace();
        }
    }

    private static class ErrorCollectingRobotResponseHandler extends AbstractFailF

        private List<RobotErrorResponse> _errors = new LinkedList<RobotErrorRespon
        private List<RobotOutputObjectResponse> _output = new LinkedList<RobotOutp
        public void handleReturnedValue(RobotOutputObjectResponse response, Stoppa
            _output.add(response);
        }

        @Override
        public void handleRobotError(RobotErrorResponse response, Stoppable stoppa
            // 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;
        }
    }
}
```

The example above shows how to use a `RobotResponseHandler` that collects returned values and errors. This type of handler is useful if the robot should continue to execute even when error are encountered, this can be useful if the website is unstable and occasionally times out. Notice that only robot errors (API exceptions) are collected by the handler, if the connection to RoboServer is lost `Request.execute()` will still throw an `RQLException` (and the robot will be stopped by RoboServer).

For more details check the `RobotResponseHandler` JavaDoc.

SSL

The API communicates with RoboServer through an RQLService. The RQLService is a RoboServer component which listens for API requests on a specific network port. When you start a RoboServer you specify if the RoboServer should use the encrypted SSL service, or the plain socket service, or both (using two different ports). All RoboServers in a cluster must be running the same RQLService (although the port may be different).

Assuming we have started a RoboServer with the SSL RQLService on port 50043, like this

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

we can use the following code

Table 66. SSL configuration

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

All we need to do is to create the cluster as an SSL cluster and specify the SSL port used by each RoboServer. Now all communication between RoboServer and the API will be encrypted.

For this example to work you need `commons-ssl-0.3.8.jar` in you application classpath, you can find it next to the API jar file inside your kapow installation.

In addition to data encryption, SSL offers the possibility to verify the identity the remote party. This type of verification is very important on the Internet, as rouge websites could otherwise pretend to be someone they are not. Most often your API client and RoboServers will be on the same local network, so you rarely need to verify the identity of the other party, but the API supports this feature should it become necessary.

Because identity verification is almost never used we will not describe it in this guide. If you are interested, you should look at the SSL examples that are included with the Java API.

Parallel execution

Both execute methods of the `Request` are blocking, which means that a thread is required for each robot execution. The examples we have looked at until now have all executed the robot directly on the main thread, which is typically not preferable as you can only execute a single robot at a time in a sequential manner.

Let's look at an example that will execute two tutorial robots in parallel. This example uses the `java.util.concurrent` library for multithreading.

Table 67. Multithreading Example

```
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}, fals
        Request.registerCluster(cluster);

        int numRobots = 4;
        int numThreads = 2;
        ThreadPoolExecutor threadPool = new ThreadPoolExecutor(numThreads, numThre
        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();
            }
        }
    }
}
```

The above example creates a `ThreadPoolExecutor` with two threads, we then creates four `RobotRunnable`s and execute them on the thread pool. Since the thread pool has two threads, two robots will start to execute immediately, the remaining two will be parked in the `LinkedBlockingQueue` and executed in order as the two first robot finish their execution and the thread pool threads become available.

Please note that the `Request` is mutable, to avoid raise conditions the Request is cloned inside the execute method. Because `Request` is mutable you should never modify the same `Request` on separate threads.

Repository Integration

In the Management Console you also specify cluster of RoboServers, these are used to execute scheduled robots, as well as robots executed as REST services. The API allowed you to use the RepositoryClient to obtain cluster information from Management Console, check the RepositoryClient documentation for details.

Table 68. Repository Integration

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

        RepositoryClient client = RepositoryClientFactory.createRepositoryClient("
        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);
    }
}
```

The above example shows how to create a `RepositoryClient` `[api/com/kapowtech/robosuite/api/java/repository/engine/RepositoryClient.html]` which connects to a Management Console deployed on localhost. For this example to work, you must have `commons-logging-1.1.1.jar,commons-codec-1.4.jar,commons-httpclient-4.1.jar` included in your classpath.

Authentication is not enabled so `null` is passed as both username and password. When we register the `RepositoryClient` we specify the name of a cluster which exists on the Management Console, this

will then query the Management Console to get a list of RoboServers configured for this cluster, and check every 2 minutes to see if the cluster configuration has been updated on the Management Console.

This integration allows you to create a cluster on Management Console that you can change dynamically using the Management Console user interface. When you use a Management Console cluster with the API usage should be exclusive, and you should not use it for scheduling robot, as this would break the two client rule.

Under the hood

The section will explain what is going on *under the hood* when you register a cluster and execute a `Requests`.

When you register a `Cluster` with the `Request`, a `RequestExecutor` is created behind the scene. This `RequestExecutor` is stored in a `Map` using the cluster name as key. When a request is executed the provided cluster name is used to find the associated `RequestExecutor` and execute the request.

Let's look at a short example

Table 69. Normal execution

```
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");
    request.setRobotLibrary(new DefaultRobotLibrary());
    RQLResult result = request.execute("MyCluster");
    System.out.println(result);
}
```

Now let's write the same example by using the *hidden* `RequestExecutor` directly

Table 70. under the hood execution

```
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);
}
```

The reason the `RequestExecutor [api/com/kapowtech/robosuite/api/java/rql/engine/hotstandby/RequestExecutor.html]` is hidden by default, is so you don't have to keep track of it. You may only create one `RequestExecutor` per cluster, so

if you use it directly you need to store a reference to it throughout your application. Using `Request.registerCluster(cluster)` means that you can blissfully ignore the `RequestExecutor` and lifecycle rules.

The `RequestExecutor` contains the necessary state and logic which provides the load balancing and failover features. Using the `RequestExecutor` directly also offers a few extra features, which we will look at.

RequestExecutor Features

When the `RequestExecutor` is not connected to a repository, you can dynamically add remove RoboServers, by calling `addRoboServer(..)` and `removeRoboServer(..)`. These methods modifies the distribution list used inside the `RequestExecutor`.

`RequestExecutor.getTotalAvailableSlots()` returns the number of unused execution slots across all RoboServers in the internal distribution list.

By using these methods you can dynamically add RoboServers to your `RequestExecutor` once the number of available execution slots becomes low.

When you create the `RequestExecutor` you may optionally provide an `RQLEngineFactory` [api/com/kapowtech/robosuite/api/java/rql/engine/hotstandby/RQLEngineFactory.html]. The `RQLEngineFactory` allows you to customize which `RQLProtocol` [api/com/kapowtech/robosuite/api/java/rql/engine/remote/RQLProtocol.html] is used when connecting to a RoboServer. This is only needed under very rare circumstances, for instance if you want use a client certificate to increase security, check API Client Certificates for details.

Web applications

The `RequestExecutor` [api/com/kapowtech/robosuite/api/java/rql/engine/hotstandby/RequestExecutor.html] contains a number of internal threads used for sending and receiving requests to RoboServers, as well as pinging each known RoboServer at regular intervals. These threads are all marked as *daemon*, which means that they don't prevent the JVM from stopping when the main thread exists, check Thread JavaDoc [http://docs.oracle.com/javase/1.4.2/docs/api/java/lang/Thread.html] for details on daemon threads.

If you use the `RequestExecutor` inside a web application, the JVM has a longer lifespan than your web application, and you can deploy and un-deploy your web application while the web container is running. This means that a web application is responsible for stopping any threads that it has created, if it does not a memory leak will be created when you un-deploy the web application. The memory leak occurs because any objects referenced by running threads can't be garbage collected until the threads stop, so if these threads are not stopped when the application is un-deployed, they will never be garbage collected.

If you use the `RequestExecutor` inside a web application *your code* is responsible for shutting down these internal threads, this is done by calling `Request.shutdown()` [api/com/kapowtech/robosuite/api/java/rql/Request.html#shutdown()] or `RequestExecutor.shutdown()` [api/com/kapowtech/robosuite/api/java/rql/engine/hotstandby/RequestExecutor.html#shutdown()] if your code created the `RequestExecutor` explicitly.

This example show you how to use a `ServletContextListener` to shutdown the API correctly when a web application is un-deployed. You *must* define the context listener in your applications web.xml.

Table 71. proper shutdown in web application

```
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}, fals
        try {
            Request.registerCluster(cluster);
        }
        catch (ClusterAlreadyDefinedException e) {
            throw new RuntimeException(e);
        }
    }

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

`contextDestroyed` is called when the web container un-deploys the application. Here we call `Request.shutdown()` to ensure that all internal threads in the hidden `RequestExecutor` are stopped correctly.

Since `contextInitialized` can't throw any unchecked exceptions we have to wrap the `ClusterAlreadyDefinedException` in a `RunTimeException`. Developers may be tempted to ignore the `ClusterAlreadyDefinedException` at this location, because they claim that it can't be thrown, as our application has not defined any other clusters. However due to the class loader hierarchy in java web containers it is actually possible to get this exception if the application is deployed twice. It will however only occur if the API jar file was loaded by a common class loader and not by the individual application's class loader.

API Debugging

Although this is rarely needed, the API can provide additional information for debugging purposes. To enable API debugging you need to configure the system property `DEBUG_ON`. The value of this property must be a package/class name the API.

For instance, if you are interested in the data transmissions between the API and RoboServer, you could ask for debugging information for package `com.kapowtech.robosuite.api.java.rql.io`. While you are developing you can do this by directly setting the system property in code, like this:

Table 72. Enabling Debug

```
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);
```

If you are debugging an application in production, you would define the system property via the command line, like this

Table 73. Enabling Debug

```
java -DDEBUG_ON=com.kapowtech.robosuite.api.java.rql.io Tutorial1
```

If you are interested in debug from multiple packages, you separate the package names by `,` (comma). Instead of a package name, you can provide the argument `ALL`, to have debug from all packages printed.

Repository API

The Repository API allows you to query the Management Console's Repository, to get a list of projects, robots and the input required to call a robot. It also allows you to programmatically deploy robots, types and resource files.

Dependencies

To use the Repository API you need the following libraries, all libraries can be found in the `API/robosuite-java-api/lib` folder inside you Kapow Katalyst installation folder

- `commons-logging-1.1.1.jar`, or newer
- `commons-codec-1.4.jar`, or newer
- `commons-httpclient-4.1.jar`, or newer
- `commons-ssl-0.3.8.jar`, or newer - if your Management Console must be accessed through HTTPs

You also need to use Java 1.5 or newer.

Repository Client

Communication with the repository is achieved through the `RepositoryClient` found in the `com.kapowtech.robosuite.api.java.repository.engine`

Table 74. Create RepositoryClient

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

    String username = "admin";
    String password = "admin";
    try {
        RepositoryClient client = RepositoryClientFactory.createRepositoryClient("
        Project[] projects = client.getProjects();
        for (Project project : projects) {
            System.out.println(project.getName());
        }
    }
    catch (RepositoryClientException e) {
        e.printStackTrace();
```

```
    }
}
```

Here we see a `RepositoryClient` configured to connect to Management Console's repository on `http://localhost:50080/` , with a username and password.

Once the RepositoryClient is created, we use the `getProjects()` method to query the repository for a list of projects. Notice that when calling any of the `RepositoryClient` methods, a `RepositoryClientException` will be thrown if an error occurs.

The `RepositoryClient` has the following eleven methods

Table 75. Methods of the `RepositoryClient`

Method signature	description
`void deleteResource(String projectName, String resourceName, boolean silent)`	Deletes a resource from a project. If silent is true no error is generated if the resource doesn't exist
`void deleteRobot(String projectName, String robotName, boolean silent)`	Deletes a robot from a project
`void deleteSnippet(String projectName, String snippetName, boolean silent)`	Deletes a snippet from a project
`void deleteType(String projectName, String modelName, boolean silent)`	Deletes a type from a project
`void deployLibrary(String projectName, EmbeddedFileBasedRobotLibrary library, boolean failIfExists)`	Deploys a library to the server. Robots, types and resources will be overridden unless `failIfExists` is true.
`void deployResource(String projectName, String resourceName, byte[] resourceBytes, boolean failIfExists)`	Deploys a resource to a project. If a resource with the given name already exist it can be overridden by setting `failIfExists` to false.
`void deployRobot(String projectName, String robotName, byte[] robotBytes, boolean failIfExists)`	Deploys a robot to a project.If a robot with the given name already exist it can be overridden by setting `failIfExists` to false.
`void deploySnippet(String projectName, String snippetName, byte[] snippetBytes, boolean failIfExists)`	Deploys a robot to a project. If a snippet with the given name already exist it can be overridden by setting `failIfExists` to false.
`void deployType(String projectName, String typeName, byte[] typeBytes, boolean failIfExists)`	Deploys a type to a project. If a type with the given name already exist it can be overridden by setting `failIfExists` to false.
`Project[] getProjects()`	Returns the projects that exist in this repository
`Cluster[] getRoboServerClusters()`	Return the cluster configuration on the Management Console

Method signature	description
`Robot[] getRobotsInProject(String projectName)`	Returns the robots available in the project with the given name.
`RobotSignature getRobotSignature(String projectName, String robotName)`	Returns the robot signature, e.i. the input variables required to execute this robot and a list of the types it may return or store

Proxy servers must be specified explicitly when creating the RepositoryClient. Standard http proxy servers without authentication are supported. NTLM proxy servers with authentication is also supported.

Check the `RepositoryClient` [api/com/kapowtech/robosuite/api/java/repository/engine/RepositoryClient.html] JavaDoc for additional details

Deployment via Repository Client

The following example shows how to deploy a robot and a type from the local file system using the `RepositoryClient`

Table 76. Deployment using RepositoryClient

```
String user = "test";
String password = "test1234";
RepositoryClient client = new RepositoryClient("http://localhost:50080", user, pas
try {
    FileInputStream robotStream = new FileInputStream("c:\\MyRobots\\Library\\Test
    FileInputStream typeStream = new FileInputStream("c:\\MyRobots\\Library\\Test.

    // Use the Kapow Java APIs StreamUtil to convert InputStream to byte[].
    // For production we recommend IOUtils.toByteArray(InputStream i) in the commo
    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());
}
```

Repository REST API

The repository API is actually a group of restful services (and URLs where data can be posted).

All the Repository Client methods that retrieve information from the repository sends XML to the Repository, and the Repository responds with XML. All deploy methods post bytes to the Repository (information encoded in URL) and the Repository return XML to acknowledge. The format of the XML sent and received is governed by a DTD found here [http://www.kapowtech.com/robosuite/repository_1_3.dtd].

Here is an example of all the XML based requests, all messages must start with the following declaration

Table 77.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE repository-request PUBLIC "-//Kapow Technologies//DTD Repository 1.3
```

If the Management Console is deployed at `http://localhost:8080/ManagementConsole`, the requests must be posted to `http://localhost:8080/ManagementConsole/secure/RepositoryAPI?format=xml`

Table 78. REST operations

Method	Example request	Example response
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</	<repository-response><delete-successful/></repository-response>

Method	Example request	Example response
	project-name> <file-name>InputA.type</file-name> </delete-file> </repository-request>	
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></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-

Method	Example request	Example response
		attribute><input-attribute><attribute-name>aBoolean</attribute-name><attribute-type>Boolean</attribute-type></input-attribute><input-attribute><attribute-name>aDate</attribute-name><attribute-type>Date</attribute-type></input-attribute><input-attribute><attribute-name>aCharacter</attribute-name><attribute-type>Character</attribute-type></input-attribute><input-attribute><attribute-name>anImage</attribute-name><attribute-type>Image</attribute-type></input-attribute></input-attribute-list></input-object><input-object><variable-name>InputB</variable-name><type-name>InputB</type-name><input-attribute-list><input-attribute 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-

Method	Example request	Example response
		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>
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>

The deployment is done by posting the raw bytes (the `octet-stream` is sent as a post body) to the following URLs. Here is an example where the repository is deployed on `http://localhost:8080/ManagementConsole`

Table 79. Methods of the deploy operations

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

If authentication is enabled on Management Console the URL `http://localhost:8080/ManagementConsole/secure/RepositoryAPI` is protected by basic authentication. This allows you to include credentials in the URL in the following manner `http://username:password@localhost:8080/ManagementConsole/secure/RepositoryAPI`.

.NET Programmer's Guide

This guide describes how to execute Robots using the Kapow Katalyst .NET API. The guide assumes that you have completed the Design Studio tutorials and know how to write simple Robots, and that you are familiar with the C# programming language.

The programmer's guide has been completely rewritten for version 9.1, as large portions of the API has been deprecated, and a new execution API has been created. The API is still backwards compatible, but you should familiarize yourself with the new API and consider rewriting existing application to use the new API, as the deprecated classes will be removed in future releases.

The old `RobotExecuter` has been deprecated because of the following reasons

- Robots would continue to execute on RoboServer even after an RQLException was thrown by the API.
- Robots would continue to execute even if RoboServer lost connection to the client, resulting in log errors for every Return Value executed.
- The distribution policies didn't look at server capacity when distributing requests.
- It was cumbersome to implement object streaming, because there were many hidden pitfalls when implementing a custom `RQLHandler`

You can still find the old .NET programmer's guide, at http://help.kapowtech.com/8.2/topic/doc/dotnet/Top.html

Details about specific classes can be found in the compiled help, robosuite-dotnet-api.chm located in `\API\robosuite-dotnet-api\docs` inside the Kapow Katalyst installation folder.

Basics

By using the .NET API, any .NET based application (.NET 4.0 required) can become a client to RoboServer. In addition to running robots that store data in a database, you can also have the robots return data directly back to the client application. Here are some examples:

- Use multiple robots to do a federated search, which aggregates results from multiple sources in real time.
- Run a robot in response to an event on your application back-end. For instance run a robot when a new user signs up, to create accounts on web-based systems not integrated directly into your back-end.

The basic section of this guide will introduce the core classes, and how to use them for executing robots. We will also describe how to provide input to robots and control their execution on RoboServer.

The .NET API is a .dll file, and it is located in `/API/robosuite-dotnet-api/lib/robosuite-dotnet-api.dll` inside the Kapow Katalyst installation folder, see important folders for details. All examples in this guide can also be found in `/API/robosuite-dotnet-api/examples`. Located next to the .NET API is log4net.dll which is a required 3rd-party library.

First Example

Let's start by looking at the code required to execute the robot named `NewsMagazine.robot`, which is located in the Tutorials folder of the default project. The robot outputs its results using the *Return Value* step

action, which makes it easy to handle the output programmatically using the API. Other robots (typically those run in a schedule by the Management Console) store their data directly in a database using the *Store in Database* step action, in which case data collected by the robot will not be returned to the API client.

In the following, we will look at how to execute the NewsMagazine robot and process the output programmatically.

Table 80. Execute a Robot without input

```
#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 o

            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();
        }
    }
}
```

Let's start by looking at the classes involved and their responsibilities.

Table 81.

`RoboServer`	This is a simple value object that identifies a RoboServer which can execute robots. Each RoboServer must be activated by a Management Console and assigned KCU before use.
`Cluster`	A cluster is a group of RoboServer functioning as a single logical unit.
`Request`	This class is used to construct the robot request. Before you can execute any requests you must register a cluster with the Request class.

`DefaultRobotLibrary`	A robot library instructs RoboServer where to find the robot identified in the request. Later examples will explore the various robot library types and when/how to use them.
`RQLResult`	This contains the result of a robot execution. The result contains value responses, log and server messages.
`RQLObject`	Each value that is returned from a robot using the *Return Value* action can be accessed as an RQLObject.

Now let's go through each line in the example an look at the specifics.

The first line tells the API that our RoboServer is running on localhost port 50000.

Table 82.

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

The next three lines defines a cluster with a single RoboServer. The cluster is registered with the `Request` class, allowing you to execute request on this cluster. Each cluster may only be registered once per application, this is usually done during the initialization of the application.

Table 83. Registering a cluster

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

This is then followed by code that creates a request that will execute the robot named *NewsMagazine.robot* located at *Library:/Tutorials*. *Library:/* refers to the robot Library configured for the request. Here the `DefaultRobotLibrary` is used, which instructs RoboServer to look for the robot in the servers local file system, see Robot Libraries for details on how to use robot libraries.

Table 84.

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

The next line executes the robot on the cluster named `MyCluster` (the cluster we previously registered) and returns the result once the robot is done. If an error occurs while the robot is executing an exception will be thrown here.

Table 85.

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

Finally we process the extracted values. First we get all extracted values of the type named *Post*, and iterate through them. For each RQLObject we access the attributes of the Post type, and print the result. We will look at attributes and mappings in a later section.

Table 86.

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

```
    Console.WriteLine(title + ", " + preview);
}
```

Robot Input

Most robots executed through the API will be parametrized through input, such as a search keyword, or login credentials. Input to a robot is part of the request to RoboServer, and is provided using the `createInputVariable` method on the request. Let us look at a short code fragment.

Table 87. Input using implicit `RQLObjectBuilder`

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

Here we create a `Request` and use `CreateInputVariable` to create an input variable named *userLogin*. We then use `setAttribute` to configure the username and password attributes of the input variable.

The above example is a common shorthand notation, but can also be expressed move verbosely by using the `RqlObjectBuilder`:

Table 88. Input using explicit `RQLObjectBuilder`

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

The two examples are identical. The first utilizes the cascading method invocation on the anonymous `RqlObjectBuilder` and is therefore shorter.

When RoboServer receives this request the following occurs:

- RoboServers loads *Input.robot* (from whatever `RobotLibrary` is configured for the request).
- RoboServer verifies that the robot has a variable named *userLogin* and that this variable is marked as input.
- RoboServer now verifies that the attributes we have configured using `setAttribute` are compatible with the type of variable userLogin. Which means that the type must have attributes named username and password and that these must both be text based attributes (the next section will describe the mapping between API and Design Studio attributes).
- If all input variables are compatible, RoboServer will start executing the robot.

If a robot requires multiple input variables, you must create all of them in order to execute the robot. You only have to configure required attributes, any no-required attributes that you don't configure through the API will just have a null value. If we assume you have a robot that requires login to both Facebook and Twitter, you could define the input like this.

Table 89.

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

Attribute Types

When you define a new type in Design Studio you select an attribute type for each attribute. Some of these attributes can contain text, like Short text, Long Text, Password, HTML, XML, and when used inside a robot there may be requirements to the text stored in these attributes. If you store text in a XML attribute, the text must be a valid XML document. This validation occurs when the type is used inside a robot, but since the API doesn't know anything about the type it doesn't validate attribute values in the same manner. As a result the API only has 8 attribute types versus the 19 available in Design Studio This table shows the mapping between the API and Design Studio attribute types.

Table 90. API to Design Studio mapping

API Attribute Type	Design Studio Attribute Type
Text	`Short Text, Long Text, Password, HTML, XML, Properties, Language, Country, Currency, Refind Key`
Integer	`Integer`
Boolean	`Boolean`
Number	`Number`
Character	`Character`
Date	`Date`
Session	`Session`
Binary	`Binary, Image, PDF`

The API attribute types are then mapped to .NET in the following way.

Table 91. .NET types for attributes

Attribute Type	.NET Class
Text	`System.String (string)`
Integer	`System.Int64`
Boolean	`System.Boolean (bool)`
Number	`System.Double (double)`
Character	`System.Char (char)`
Date	`System.DateTime`
Session	`Com.Kapowtech.Robosuite.Api.Construct.Session`
Binary	`Com.Kapowtech.Robosuite.Api.Construct.Binary`

The `RqlObjectBuilder`'s `setAttribute` method is overloaded so you don't need to specify the attribute type explicitly when configuring an attribute through the API, as long as the right .NET class is used as argument. Here is an example that shows how to set the attributes for an object with all possible (Design Studio) attribute types.

Table 92. Recommended usage of setAttribute

```
RqlObjectBuilder inputBuilder = request.CreateInputVariable("AllTypes");
inputBuilder.SetAttributeEntry("anInt", 42L);
inputBuilder.SetAttributeEntry("aNumber", 12.34d);
inputBuilder.SetAttributeEntry("aBoolean", true);
```

```
inputBuilder.SetAttributeEntry("aCharacter", 'c');
inputBuilder.SetAttributeEntry("aShortText", "some text");
inputBuilder.SetAttributeEntry("aLongText", "a longer text");
inputBuilder.SetAttributeEntry("aPassword", "secret");
inputBuilder.SetAttributeEntry("aHTML", "<html>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");
```

Notice that in the above example we have to cast null values, because the C# compiler can't otherwise determine which of the overloaded version of `SetAttributeEntry` method we want to call. However since since unconfigured attributes will automatically be null, you never need to set null explicitly.

It is possible to specify the Attribute and AttributeType explicitly when creating input using the API. This is approach is *not recommended*, but may be needed in rare cases, and would look like this.

Table 93. Not recommended usage of setAttribute

```
RqlObjectBuilder inputBuilder = request.CreateInputVariable("alltypes");
inputBuilder.SetAttributeEntry(new AttributeEntry("anInt", "42", AttributeEntryTyp
inputBuilder.SetAttributeEntry(new AttributeEntry("aNumber", "12.34", AttributeEnt
inputBuilder.SetAttributeEntry(new AttributeEntry("aBoolean", "true", AttributeEnt
inputBuilder.SetAttributeEntry(new AttributeEntry("aCharacter", "c", AttributeEntr
inputBuilder.SetAttributeEntry(new AttributeEntry("aShortText", "some text", Attri
inputBuilder.SetAttributeEntry(new AttributeEntry("aLongText", "a longer text", At
inputBuilder.SetAttributeEntry(new AttributeEntry("aPassword", "secret", Attribute
inputBuilder.SetAttributeEntry(new AttributeEntry("aHTML", "<html>bla</html>", Att
inputBuilder.SetAttributeEntry(new AttributeEntry("anXML", "<tag>text</tag>", Attr
inputBuilder.SetAttributeEntry(new AttributeEntry("aDate", "2012-01-15 23:59:59.12

inputBuilder.SetAttributeEntry(new AttributeEntry("aBinary", null, AttributeEntryT
inputBuilder.SetAttributeEntry(new AttributeEntry("aPDF", null, AttributeEntryType
inputBuilder.SetAttributeEntry(new AttributeEntry("anImage", null, AttributeEntryT
inputBuilder.SetAttributeEntry(new AttributeEntry("aProperties", "name=value\nname
inputBuilder.SetAttributeEntry(new AttributeEntry("aCurrency", "USD", AttributeEnt
inputBuilder.SetAttributeEntry(new AttributeEntry("aCountry", "US", AttributeEntry
inputBuilder.SetAttributeEntry(new AttributeEntry("aLanguage", "en", AttributeEntr
inputBuilder.SetAttributeEntry(new AttributeEntry("aRefindKey", "Never use this as
```

As we can see all attribute values must be provided in the form of strings. The string values are then converted to the appropriate .NET objects based on the AttributeEntryType provided. This is only useful if you build other generic APIs on top of the Kapow Katalyst .NET API.

Execution Parameters

In addition to the `CreateInputVariable` method, the **`Request`** contains a number of properties that controls how the robot executes on RoboServer.

Table 94. Execution control properties on Request

`MaxExecutionTime`	This property controls the maximum number of seconds the robot can execute. When this time has elapsed the robot will be stopped by RoboServer. The timer doesn't start until the robot begins to execute, so if the robot is queued on RoboServer this is not taken into account.
`StopOnConnectionLost`	This property (true by default) controls if RoboServer will stop the robot if it discovers that the connection to the client application has been lost. You should have a very good reason for setting this value to false - if your code is not written to handle this, your application will not perform as expected.
`StopRobotOnApiException`	This property (true by default) instructs RoboServer to stop the robot when the first API exception is raised. By default most steps in a Robot will raise an API exception if the step fails to execute - this is configured on the steps error handling tab. When set to false, the robot will continue to execute regardless of API exceptions, however unless your application is using a `IRobotResponseHandler` for streaming the results, an exception will still be thrown by Execute(), so be extremely careful when setting this to false.
`Username, Password`	These properties are used to set the credentials. This is used when RoboServer is configured to require authentication. When this option is enabled, the client must provide credentials or RoboServer will reject the request.
`RobotLibrary`	This property is used to assign a RobotLibrary to the request. A robot library instructs RoboServer where to find the robot identified in the request. Later examples will explore the various robot library types and when/how to use them.
`ExecutionId`	Allows you to set the executionId for this request. If you don't provide one, RoboServer will generate one automatically. The execution ID is used for logging, and also needed if your client needs to be able to stop the robot programmatically. The ID must be globally unique (over time). If two robots use the same execution ID, the logs will be inconsistent. Setting this is useful if your robots are part of a larger workflow and you already have a unique identifier in your client application, as this allows you to easily join the robot logs with the rest of the system.
`setProject(String)`	This is used solely for logging purposes. The Management Console uses this field to link log messaged to project, so the log views can filter by project. If your application is not using the `RepositoryRobotLibrary` you should probably set this value to inform the RoboServer logging system which project (if any) this robot belongs to.

Robot Libraries

In Design Studio robots are grouped into projects. If you look in the file system you will see that these projects are represented by a folder with the only constraint that it must contain a folder named `Library`, see Libraries and Projects for details.

When you build the execute request for RoboServer, you identify the robot by a robot URL, like this:

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

Here, `Library:/` is a symbolic reference to a robot library, in which the RoboServer should look for the robot. The `RobotLibrary` is then specified on the builder like this:

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

There are three different robot library implementations, which one to select depends on you deployment environment.

Table 95. Robot Libraries

Library Type	Description
`DefaultRobotLibrary`	This configures RoboServer to look for the robot in the current project folder. This folder is defined in the Settings application. If you have multiple RoboServers you will have to deploy your robots on all RoboServers. This robot library is not cached, so the robot is reloaded from disk with every execution. This makes the library usable in a development environment where robots change often, but not suitable for a production environment.

`EmbeddedFileBasedRobotLibrary`	This library is embedded in the execute request sent to RoboServer. To create this library you must create a zip file containing the robots and all its dependencies (types, snippets and resources). This can be done the *Tools->Create Robot Library File* menu in Design Studio.
	
	The library is sent with every request, which adds some overhead for large libraries, but the libraries cached on RoboServer, which offers best possible performance.
	
	One strength is that robots and code can be deployed as a single unit, which offers clean migration from QA environment to production environment. However, if the robots change often you will have to redeploy them often.
	
	You can use the following code to configure the embedded robot library for your request.
	
	`var request = new Request("Library:/Tutorials/NewsMagazine.`
	`var stream = new FileStream("c:\\embeddedLibrary.robotlib",`
	`request.RobotLibrary = new EmbeddedFileBasedRobotLibrary(st`

`RepositoryRobotLibrary`	This is the most flexible `RobotLibrary`. This library uses the Management Console's built-in repository as a robot library. When you use this library, RoboServer will contact the Management Console which will send a robot library containing the robot and its dependencies. Caching occurs on a per robot basis, inside both Management Console and RoboServer. Inside Management Console, the generated library is cached based on the robot and its dependencies. On RoboServer, the

Library Type	Description

	cache is based on a timeout, so it doesn't have to ask the Management Console for each request. In addition, the library loading between RoboServer and Management Console uses HTTP public/private caching, to further reduce bandwidth.
	
	If NewsMagazine.robot has been uploaded to the Management Console we could use the repository robot library like this when executing the robot:
	
	`var request = new Request("Library:/Tutorials/NewsMagazine.`
	`request.RobotLibrary = new RepositoryRobotLibrary("http://l`
	`"Default`
	
	This will instruct RoboServer to load the robot from a local Management Console and cache it for one minute before checking with the Management Console to see if a new version of the robot (it's type and snippets) has been changed.
	
	In addition any resource loaded through the `Library:/` protocol, will cause RoboServer request the resource directly from the Management Console.

Advanced

In this section we will look a little closer at some of the more advanced features on the API. These include output streaming, logging and SSL configuration, as well as parallel execution.

Load Distribution

Table 96. Load balanced executor

```
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);
```

Lets look a little closer at what happens inside the `RequestExecutor`. The executor is given an array of RoboServers. As the executor is constructed it tries to connect to each RoboServer. Once it is connected it sends a ping request to each RoboServer to discover how the server is configured.

Load is distributed to each online RoboServer in the cluster, based on the number of unused execution slots on the RoboServer. The next request is always distributed to the RoboServer with the most available slots. The number of available execution slots is obtained through the initial Ping response, and the executor keeps track of each robot it starts, and when it completes. The number of execution slots on a RoboServer is determined by the max concurrent robots on the Servers tab.

If a RoboServer goes offline it will not receive any robot execution requests before it has successfully responded to the ping request.

Two client rule

You should only have one API client using a given cluster of RoboServer. If you have multiple .Net applications running robots against the same RoboServers, this will result in reduced performance.

Executor Logger

When you execute a request, the execute method will throw an exception if a robot generates an error. Other types of errors and warnings are reported through an *executor logger*, a class implementing the `IExecutorLogger` interface. In the previous examples, we have not provided any execution logger when executing robots, which means that the default implementation `ExecutorLogger` that will write to system out will be used. Let's see how the `ExecutorLogger` will report if one of our RoboServers goes offline.

The example configures a cluster with a RoboServer which is not online.

Table 97. ExecutorLogger offline server example

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

If you run this example if should print the following to the console:

Table 98. ExecutorLogger offline RoboServer console output

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

Often you don't want to have your application writing directly to `System.out`, in that case you can provide a different `IExecutorLogger` implementation, you can do so when registering the cluster, like this

Table 99. Use `DebugExecutorLogger`

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

This example uses the `DebugExecutorLogger`() which will also print to `System.out`, but only if the API debugging is enabled. Alternative you can provide your own implementation of the `ExecutorLogger`, to control how error messages should be handled.

Data Streaming

Sometimes you need to present the results from a robot execution in real-time, as the robot is executing. In these cases you want the API to return the extracted values immediately instead of waiting for the robot to finish its execution and access the RqlResult.

The API offers the possibility to receive a callback every time the API receives a value that was returned by the Robot. This is done through the `IRobotResponseHandler` interface

Table 100. Response streaming using`AbstractFailFastRobotResponseHandler`

```
using System;
```

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

namespace Examples
{

    public class DataStreaming {

        public static void Main(String[] args)  {

            var server = new RoboServer("localhost", 50000);
            var cluster = new Cluster("MyCluster", new RoboServer[] { server }, fa
            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 respons
        {
            var title = response.OutputObject["title"];
            var preview = response.OutputObject["preview"];
            Console.WriteLine(title + ", " + preview);
        }
    }
}
```

The above example uses the second execute method of the `Request`, which expects a `RobotResponseHandler` in addition to the name of the cluster to execute the robot on. In this example we create a `IRobotResponseHandler` by extending `AbstractFailFastRobotResponseHandler`, which provides default error handling, so we only need to handle the values returned by the robot.

The `handleReturnedValue` method is called whenever the API receives a returned value from RoboServer. The `AbstractFailFastRobotResponseHandler` used in this example, will throw exceptions in the same way as the non-streaming execute method. This means that an exception will be thrown in response to any API exceptions generated by the robot.

The `IRobotResponseHandler` has several methods which can be grouped into 3 categories.

Robot life cycle events	Methods which are called when the robot's execution state change on RoboServer, such as when it starts and finishes its execution.
Robot data events	Methods which are called when the robot returns data or errors to the API.

Additional error handling	Methods which are called either due to an error inside RoboServer or in the API.

Table 101. IRobotResponseHandler - robot life cycle events

Method name	Description
`void RequestSent(RoboServer roboServer, ExecuteRequest request)`	Called when the `RequestExecutor` has found the server which will execute the request.
`void RequestAccepted(String executionId)`	Called when the found RoboServer has accepted the request, and put it into it queue.
`void RobotStarted(IStoppable stoppable)`	Called when the RoboServer begins to execute the robot. This usually occurs immediately after the robot has been queued, unless the RoboServer is under heavy load, or used by multiple API clients.
`void RobotDone(RobotDoneEvent reason)`	Called when the robot is done executing on RoboServer. The `RobotDoneEvent` is used to specify if the execution terminated normally, due to an error, or if it was stopped.

Table 102. IRobotResponseHandler - robot data events

Method name	Description
`void HandleReturnedValue(RobotOutputObjectResponse response, IStoppable stoppable)`	Called when the robot has executed a *Return Value* action, and the value has been transmitted over the socket to the API.
`void HandleRobotError(RobotErrorResponse response, IStoppable stoppable)`	Called when the robot raises an API exception. Under normal circumstances the robot will stop executing after the first API exception. This behavior can be overridden by using `Request.StopRobotOnApiException = false`, in which case this method will be called multiple times. This is useful if you want a data streaming robot to continue to execute regardless of any generated errors.
`void HandleWriteLog(RobotMessageResponse response, IStoppable stoppable)`	Called if the robot executes the *Write Log* action. This is useful if you want to provide additional logging info from within a robot.

Table 103. IRobotResponseHandler - additional error handling

Method name	Description
`void HandleServerError(ServerErrorResponse response, IStoppable stoppable)`	Called if RoboServer generates an error, for instance if the server too busy to process any requests, or if an error occurs inside RoboServer which prevents it from starting the robot.
`void handleError(RQLException`	Called if an error occurs inside the API. Most commonly if the client looses the connection to RoboServer.

Method name	Description
e, IStoppable stoppable)	

Many of the methods will include a `IStoppable` object, this object can be used to stop for instance in response to a specific error or value returned.

Some of these methods allow you to throw an RQLException, if you do this you should be aware of the consequences. The thread that calls the handler is the thread that calls `Request.Execute()`, this means that any exceptions thrown will bubble up the call stack and out the execute method. If you throw an exception in response to `handleReturnedValue`, `handleRobotError` or `handleWriteLog` it is your responsibility to invoke `Stoppable.stop()`, or the robot may continue to execute even though the call to `Request.Execute()` has completed.

Data streaming is most often used in one of the following use cases.

- Ajax based web application, where results are presented to the user in real-time. If data was not streamed results could not be shown until the robot was done running.
- Robots that return so much data that the client would not be able to hold it all in memory throughout the robots execution.
- Processes that need to be optimized so the extracted values are processed in parallel with the robot execution.
- Processes that store data in databases in a custom format.
- Robots that should ignore or require custom handling of API exceptions (see below).

Table 104. Response and error collecting using `AbstractFailFastRobotResponseHandler`

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

namespace Examples
{
    public class DataStreaming
    {

        public static void Main(String[] args)
        {

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

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

```
            request.StopRobotOnApiException = false;  // IMPORTANT!!

            ErrorCollectingRobotResponseHandler handler = new ErrorCollectingRobot
            request.Execute("MyCluster", handler); // blocks until robot is done,

            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.ErrorMess
            }
        }
    }


    public class ErrorCollectingRobotResponseHandler : AbstractFailFastRobotRespon

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

        override public void HandleReturnedValue(RobotOutputObjectResponse respons
            _output.Add(response);
        }


        override public void HandleRobotError(RobotErrorResponse response, IStoppa
            // 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;
        }
    }
}
```

The example above shows how to use a `IRobotResponseHandler` that collects returned values and errors. This type of handler is useful if the robot should continue to execute even when error are encountered, which can be useful if the website is unstable and occasionally times out. Notice that only robot errors (API exceptions) are collected by the handler, if the connection to RoboServer is lost `Request.Execute()` will still throw an `RQLException` (and the robot will be stopped by RoboServer).

For more details check the `IRobotResponseHandler` chm documentation in the `/docs` folder .

SSL

The API communicates with RoboServer through an RQLService. The RQLService is a RoboServer component which listens for API requests on a specific network port. When you start a RoboServer you specify if the RoboServer should use the encrypted SSL service, or the plain socket service, or both (using two different ports). All RoboServers in a cluster must be running the same RQLService (although the port may be different).

Assuming we have started a RoboServer with the SSL RQLService on port 50043, like this

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

we can use the following code

Table 105. SSL configuration

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

All we need to do is to create the cluster as an SSL cluster and specify the SSL port used by each RoboServer. Now all communication between RoboServer and the API will be encrypted.

In addition to data encryption, SSL offers the possibility to verify the identity the remote party. This type of verification is very important on the Internet, as rouge websites could otherwise pretend to be someone they are not. Most often your API client and RoboServers will be on the same local network, so you rarely need to verify the identity of the other party, but the API supports this feature should it become necessary.

Check here to find out how to compile and run the included SSL example.

Repository Integration

In the Management Console you also specify cluster of RoboServers, these are used to execute scheduled robots, as well as robots executed as REST services. The API allowed you to use the RepositoryClient to obtain cluster information from Management Console, check the RepositoryClient documentation for details.

Table 106. Repository Integration

```

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

The above example shows how to create a `RepositoryClient` which connects to a Management Console deployed on localhost port 50080.

If the Management Console requires authentication you will need to pass a username and password, otherwise you may pass null for both. When we register the `RepositoryClient` we specify the name of a cluster which exists on the Management Console, this will then query the Management Console to get a list of RoboServers configured for this cluster, and check every 2 minutes to see if the cluster configuration has been updated on the Management Console

This integration allows you to create a cluster on Management Console that you can change dynamically using the Management Console user interface. When you use a Management Console cluster with the API usage should be exclusive, and you should not use it for scheduling robot, as this would break the two client rule.

Under the hood

The section will explain what is going on *under the hood* when you register a cluster and execute a `Request`.

When you register a `Cluster` with the `Request`, a `RequestExecutor` is created behind the scene. This `RequestExecutor` is stored in a `Map` using the cluster name as key. When a request is executed the provided cluster name is used to find the associated `RequestExecutor` and execute the request.

Lets look at a short example

Table 107. Normal execution

```
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);
}
```

Now lets write the same example by using the *hidden* `RequestExecutor` directly

Table 108. under the hood execution

```
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);
}
```

The reason the `RequestExecutor` is hidden by default, is so you don't have to keep track of it. You may only create one `RequestExecutor` per cluster, so if you use it directly you need to store a reference to it throughout your application. Using `Request.RegisterCluster(cluster)` means that you can blissfully ignore the `RequestExecutor` and lifecycle rules.

The `RequestExecutor` contains the necessary state and logic which provides the load balancing and failover features. Using the `RequestExecutor` directly also offers a few extra features, which we will look at.

RequestExecutor Features

When the `RequestExecutor` is not connected to a repository, you can dynamically add remove RoboServers, by calling `AddRoboServer(..)` and `RemoveRoboServer(..)`. These methods modifies the distribution list used inside the `RequestExecutor`.

`RequestExecutor.TotalAvailableSlots` property contains the number of unused execution slots across all RoboServers in the internal distribution list.

By using these methods you can dynamically add RoboServers to your `RequestExecutor` once the number of available execution slots becomes low.

When you create the `RequestExecutor` you may optionally provide an `IRqlEngineFactory`. The `IRqlEngineFactory` allows you to customize which `RQLProtocol` is used when connecting to a RoboServer. This is only needed under very rare circumstances, for instance if you want use a client certificate to increase security, check API Client Certificates for details.

Repository API

The Repository API allows you to query the Management Console's Repository to get a list of projects, robots and the input required to call a robot. It also allows you to programmatically deploy robots, types and resource files.

Repository Client

Communication with the repository is achieved through the `RepositoryClient` found in the namespace `Com.KapowTech.RoboSuite.Api.Repository.Engine`

Let's look at an example

Table 109. Get Projects from Repository

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

Here we see a `RepositoryClient` configured to connect to Management Console's repository on `http://localhost:50080/` with a username and password. If the Management Console is not password protected, you must supply `null` for user name and password.

Once the RepositoryClient is created, we use the `GetProjects()` method to query the repository for a list of projects. Notice that when calling any of the `RepositoryClient` methods, a `RepositoryClientException` will be thrown if an error occurs.

The `RepositoryClient` has the following eleven methods

Table 110. Methods of the `RepositoryClient`

Method signature	description
`void DeleteResource(string projectName, string resourceName, boolean silent)`	Deletes a resource from a project.
`void DeleteRobot(string projectName, string robotName, boolean silent)`	Deletes a robot from a project
`void DeleteType(string projectName, string typeName, boolean silent)`	Deletes a type from a project
`void DeleteSnippet(string projectName, string SnippetName, boolean silent)`	Deletes a snippet from a project
`void DeployLibrary(string projectName, EmbeddedFileBasedRobotLibrary library, boolean failIfExists)`	Deploys a library to the server. Robots, types and resources will be overridden unless `failIfExists` is true.
`void DeployResource(string projectName, string resourceName, byte[] resourceBytes, boolean failIfExists)`	Deploys a resource to a project. If a resource with the given name already exist it can be overridden by setting `failIfExists` to false.
`void DeployRobot(string projectName, string robotName, byte[] robotBytes, boolean failIfExists)`	Deploys a robot to a project. If a robot with the given name already exist it can be overridden by setting `failIfExists` to false.
`void DeployType(string projectName, string typeName,`	Deploys a type to a project. If a type with the given name already exist it can be overridden by setting `failIfExists` to false.

Method signature	description
`byte[] typeBytes, boolean failIfExists)`	
`void DeploySnippet(string projectName, string snippetName, byte[] snippetBytes, boolean failIfExists)`	Deploys a snippet to a project. If a snippet with the given name already exist it can be overridden by setting `failIfExists` to false.
`Project[] GetProjects()`	Returns the projects that exist in this repository
`Cluster[] GetRoboServerClusters()`	Return the cluster configuration on the Management Console
`Robot[] GetRobotsInProject(string projectName)`	Returns the robots available in the project with the given name.
`RobotSignature GetRobotSignature(string projectName, string robotName)`	Returns the robot signature, e.i. the input variables required to execute this robot and a list of the types it may return

Check the .Net documentation for details. The .Net documentation is located inside you Kapow Katalyst installation at `/API/robosuite-dotnet-api/docs/RoboSuite .NET API.chm`.

If authentication is enabled on the repository, the request may be declined if the credentials given doesn't have sufficient access.

The repository is accessed via http. When using the .Net version of the Repository API, any proxy servers configured for Internet Explorer will be used by the Repository API.

Deployment via Repository Client

The following example shows how to deploy a robot and a type from the local file system using the `RepositoryClient`

Table 111. Deploying to Repository

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

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);
```

Repository API as REST

The repository can also be accessed via restful services

Examples

The Kapow Katalyst installation contains six additional API code examples, these examples are found in `API\robosuite-dotnet-api\example`

Compiling & Running the Examples

To compile the examples run the build.bat from a command prompt. This will cause six .exe files to be produced, which can be run directly.

The .exe files are relying on the `robosuite-dotnet-api.dll` and the `log4net.dll` both of which are located in the examples directory. Both files are identical copies of the ones located in the bin folder and are copied in here to make it easier to run the examples.

Each example program will print a small usage text when run without any arguments.

C# Compiler Issues

The build.bat file assumes that the C# compiler is available in the path.

.NET Framework 4.0

The API and accompanying log4net is build targeting the .net framework 4.0 client profile. For details on the .net framework 4.0 client profile see http://msdn.microsoft.com/en-us/library/cc656912.aspx

Ssl example

To run the ssl example RunSslRobot.exe the RoboServer has to be configured to use ssl and the certificate has to be imported on the client machine. This guide will show you how to configure ssl using a self-signed certificate on a windows PC running a local RoboServer.

Creating the self-signed certificate

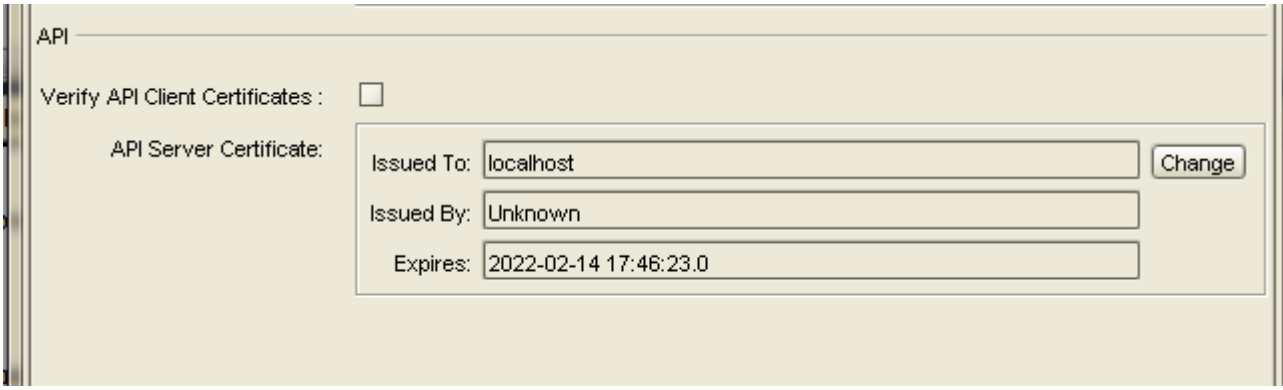
The bat file `MakeCertificates.bat` included in the examples directory will create a self-signed certificate for your .net API client. You can use `MakeCertificates.bat` as follows

1. Start a Visual Studio Command Prompt and go do the examples directory
2. Type `MakeCertificates.bat`
3. You will now be prompted to type the password for the certificate in this case we simply type "123"
4. You will now be prompted to type a hostname that the certificate. For a local RoboServer we type "localhost" It is important that this is the same hostname that you use in your client code when creating the protocol.
5. Now you will be prompted 3 times for the password again - type "123" in each dialog

Certificates can also be created with Keytool as described in API Client Certificates

Configure the RoboServer

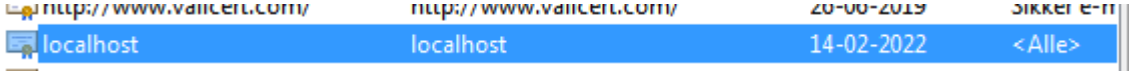
1. On the RoboServer start the **Settings** application located in **Start -> All Programs -> Kapow Katalyst**
2. In the **Settings** application goto the **certificates** tab
3. Click **change** and select the file `API\robosuite-dotnet-api\example\server.pfx`
4. When prompted for a password type "123".



5. the RoboServer is now fully configured and can be started using the switch `-service ssl:50443` to use the ssl configuration on port 50443

Configure the API client

1. Run the command `mmc.exe`
2. On the Console menu, click **Add/Remove Snap-in**
3. Under **Snap-in**, double-click **Certificates**, and select to manage certificates for the Local computer and click **Finish**
4. With the certificates snap-in loaded expand the node **Certificates -> Trusted root Certification Authorities** and right click the Certificates node and click the menu item **All tasks -> Import**
5. This will start the **Certificate Import Wizard**. When prompted to pick the certificate file browse to `API\robosuite-dotnet-api\example\server.pub.cer` and complete the import



After completing these steps, both the server and client are configured to use ssl and running the example RunSslRobot.exe can be used to verify the configuration.

Clipping Programmer's Guide

This guide describes the Kapow Clipping API. The API defines a number of extension points and allows advanced users of the Kapow Clipping technology to configure and control the use and execution of Clipping robots.

A Clipping robot is a special robot type that will be called from a web clip. Web clips will be deployed to either a web server (servlet engine) or a portal application.

The Kapow Clipping API provides Java programmers with a means of modifying and extending the standard functionality of clipping robots. The extension points allow you to:

- Provide, modify or filter input or output to your robot, e.g. by accessing data from an external source. This includes modifying the robot's input and output values, adding properties, and filtering cookies and header information.
- Provide user credentials, e.g. by retrieving them from an external credential vault.
- Program your own scheduling algorithm for balancing the load on multiple RoboServers.
- Controlling session restart.

The guide is intended for Servlet programmers integrating Kapow Clips into their portal or web application. We assume that the programmer is familiar with the following:

- Java.
- Java Portlet standard (JSR-168).

While this guide covers many basic issues, it does not go into detail on subjects that are covered in other Kapow Katalyst guides, such as the Design Studio and Java Programmer's Guides. The reader is advised to consult these other guides as needed.

Organization

The Guide is divided into the following sections.

- An introduction, containing a discussion of required items, notational conventions, and references.
- A brief overview of how to use the API, describing the clipping architecture and how extension points are called. Furthermore, this section gives an introduction to how the extension points are defined. Different kinds of extension points are discussed in subsequent sections.
- A discussion of Providers, which make it possible to provide extra information to the clipping robot.
- A discussion of Handlers, which make it possible to modify the output from the clipping robot.
- A discussion of Filters, which can modify information transferred to and from the clipping robot.
- A discussion of Controllers, which are used to control the behavior of the RoboServer.
- Finally, some tips and tricks.

The guide can be read in one stretch. However, it is not necessary to read through the entire guide to be able to get started with the API. We recommend that you read the Introduction and Using the API sections first. After that you should be able to jump to the section describing the extension point that you need for your purpose.

Introduction

This section contains:

- A list of items you need in order to use the Kapow .NET API, and instructions on how to get them.
- An overview of the notation that is used in the description of the API.
- References.

Required Items

The API is written for Java 2 Standard Edition (J2SE) 1.3 through 6. Earlier versions of Java are not supported. As the API is intended to be used within the context of a web container, the necessary Java 2 Enterprise Edition (J2EE) API's (in particular the Servlet API) will automatically be available.

Another requirement is that you have a license that allows clipping.

Notation

The notation used in the guide will probably be familiar, if you have read any of the other guides. Here is a listing of the important types of notation:

Code examples:	Being a programming guide, this document contains several small code examples. We use mostly code fragments for brevity. This means that the code fragments cannot simply be copy-pasted into a Java source file and compiled. Furthermore, in the examples brevity has been preferred to good coding style.
Inline code:	Written in fixed-width font. Inline code is used to refer to Java classes, keywords, or file names. For example, this is how we would refer to the `java.lang.String` class.
New terms or important features:	Are **emphasized** so as to stand out.

References

The API uses an XML document for configuration, this document is called a clip descriptor, and has the .clip file extension. The format of the clip descriptor is defined in the following DTD

Kapow Clipping Deployment Descriptor DTD:	http://www.kapowtech.com/robosuite/rql/dtd/clip-deployment-descriptor_1_3.dtd

Using the API

This section provides an overview of how the clipping API works and how to use it.

The Clip Configuration File

A clip is deployed as a clip configuration file. This is an XML file with file extension `.clip`. The clip configuration file contains all information needed to configure the clip.

The Clipping API consists of a number of interfaces that defines extension points to the clipping logic. Developers may provide implementations of these interfaces and register them in the clip configuration file by specifying the class name of the implementation. In some cases default implementations are provided, and these can be chosen in the clip configuration file without specifying a class name.

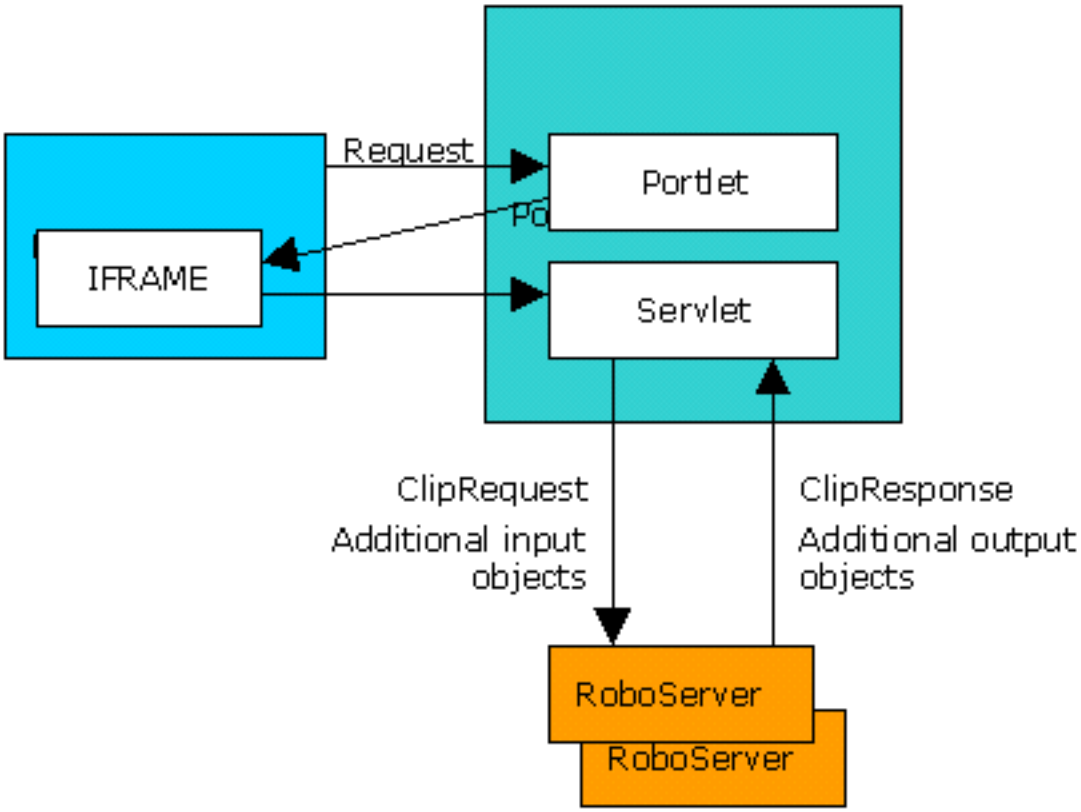
Be aware that the order of the elements in the clip descriptor must follow the order defined in the DTD mentioned in References. In the clip descriptor example listed in this guide, we have tried to add an element before and/or after, to make it clear where the element fits in.

Note that deploying a robot from Design Studio will overwrite the clip configuration file, so in this case modifications to the clip configuration file will have to be redone. To avoid doing this often, you should choose to use a robot library at a URL. Furthermore, you could use a Property Provider to retrieve configuration which might change often from another source than the clip configuration file.

The Clipping Logic

When a request is made for a clip e.g. in a portlet, the HTML of the clip will not be returned directly, but instead a document with an IFRAME will be returned (see the figure below). This IFRAME will make a new request targeting the clip configuration file; that is its source (`src`) URL is pointing at the corresponding clip configuration file. The request for the clip configuration file is then handled by a servlet. In other words all requests for clip configuration files are mapped to a servlet which means that there must be a servlet mapping in the web application's `web.xml` file for this (this is automatically placed there when you run a clipping wizard). The specific class name of the servlet may depend on where the clip is deployed e.g. on a standard servlet engine, in a BEA WebLogic Portal, etc. The servlet contains code (in the following called clipping logic) that will be deployed to a web server or portal and this code will be responsible for contacting a RoboServer to run the clipping robot and for presenting the clip on the web page. The clipping logic will send request to a RoboServer and will receive responses back from this. Requests may take various forms, e.g. a request to restart a robot, a request telling the RoboServer that the user has clicked on a link in the clip, etc. Request may also contain extra information to the robot e.g. extra input object. The responses can also take many forms, e.g. content to present in the clip, an error message, etc.

The figure below illustrates how this is done if the clip is deployed to a portal application, but clips deployed to a web server will work in the same way.



The Clipping Logic

Calling RoboServer

The servlet sends a set of `RQLObjects` to a RoboServer, like when a robot is normally run from the API. The values of these `RQLObjects` are mapped into the corresponding input variables that must be defined in the robot in the Variables View in Design Studio. One variable — the ClipRequest — is predefined. You can define and declare additional input variables by defining types in Design Studio and adding variables in the Robot Editor's Variables View.

The ClipRequest consists of a command specification and some data. The command typically reflects an action performed by the user, e.g. clicking on a link, changing focus, retrieving an attachment or navigating in the history. In addition, a command might contain instructions to the RoboServer, e.g. to start and stop the session.

The data sent as part of the ClipRequest includes

SessionId:	An identifier uniquely identifying the current session.
Cookies:	An abstract representation of the cookies in the servlet request.
Header:	An abstract representation of the HTTP header sent in the servlet request.
User credentials:	User name and password used in the robot to login to the site being clipped

RoboServer credentials: If authentication is enabled on RoboServer, this is the credentials used to authenticate the portal against the RoboServer

Properties: Input values, which can be retrieved within the robot using the Get Properties converter.

Response from RoboServer

When RoboServer finishes processing the clipping robot, it will send back a response to the Servlet. Like in the call to the RoboServer, this response consists of a number of `RQLObject`s representing outputted variable values, where one variable — ClipResponse — is predefined. You can define and declare additional variables.

A ClipResponse contains

Changes: A specification of the changes to the servlet resulting from executing the robot.

SessionId: An identifier uniquely identifying the current session.

Cookies: An abstract representation of the cookies set by the clipped site.

Header: An abstract representation of the HTTP header returned from the clipped site.

Alert message: If the clipping robot execution resulted in a JavaScript alert box being shown, the message of this alert will be present in this attribute.

Extension Points

We are now able to describe the extension points in more detail. If you want to insert code at an extension point, you must provide a Java class implementing the Java interface for the extension point. This class must be specified in the clip configuration file.

All providers, handlers and controller have access to the servlet request (`HttpServletRequest`) and response (`HttpServletResponse`) and can therefore access various resources, e.g. portlet preferences to get access to data from which to create the values it provides.

Providers

A provider is an extension point, which is called before the servlet calls the RoboServer. The provider will then provide, modify or add information that will determine how the RoboServer is called. You can implement the following providers:

RoboServer credentials provider: If authentication is enabled on RoboServer, this provider controls what username and password is used when executing the clipping robot

Additional Input Objects provider: Adds additional input to the call to RoboServer.

User Credentials provider: Modify username and password in the ClipRequest sent to RoboServer.

Properties provider: Modify or add properties in the ClipRequest sent to RoboServer.

User Message Localization provider: Provide localization of Kapow messages presented by the servlet.

Distribution Policy provider: Modify the scheduling algorithm for load balancing multiple RoboServers

Handlers

A handler is an extension point, which is called after RoboServer has executed the clipping robot. Hence, handlers have access to the response from the robot. There are two handler interfaces, which you might implement:

Additional Output Objects handler: Have the ability to access additional output values returned from the call to RoboServer. These extra values can e.g. be stored in the session or a database.

Error handler: Allows you to write your own error handling strategy; e.g. forwarding to a portlet specific error page.

Filters

A filter corresponds to two extension points: the filter is called both before RoboServer is called and after RoboServer has responded. The filter is capable of modifying properties, which are common to the ClipRequest and the ClipResponse:

Header filter: Modifies the list of headers by filtering, rewriting or adding headers.

Controllers

A controller makes it possible to control the command sent to RoboServer as part of the ClipRequest. Only one controller is available:

Restart Session controller: This extension point is before a call to RoboServer and allows you to discard the command that would normally be sent and instead send a restart session command. This can be useful if the user has changed his credentials and the clipping robot therefore should log out and log in again.

Technical Issues

A couple of technical issues relating to all extension points are mentioned here.

Correlation between Portlet and Servlet

In a portal there might be several instances of the same portlet (also called portlet windows). For a clipping portlet this means that several clipping portlets in a portal may use the same clip configuration file. This also means that the servlet needs a way to distinguish between calls from these. If a user requests that a given clipping portlet should restart its session then it is this portlet that should restart and not some other instance. In order for the servlet to know which portlet it is delivering content to, the request contains a portlet id and this can be accessed in the following way:

```
request.getParameter(ClipServlet.PORTLET_PARAMETER_NAME);
```

If one extends the clipping portlet class `ClipJavaPortlet` to provide one's own functionality, e.g. an edit mode, then the same portlet id may be obtained by calling the method:

```
String getPortletID(RenderResponse response)
```

See Communication between Portlet and Servlet, where this example is explored in depth. Note, that the `getPortletID` method can only be called during rendering, e.g. from `doEdit`, `doView` or `doHelp` and not when handling an action from `processAction`.

Thread Safety

All classes implementing providers, handles, controllers and filters are required to be thread-safe.

Providers

This section describes the provider extension points in the clipping servlet. Providers are called by the servlet before RoboServer is called, and they are able to intervene in this calling mechanism: either by choosing a RoboServer in an environment with several RoboServers (the Distribution Policy provider) or by modifying the data sent as part of the request.

RoboServer credentials Provider

Authentication can be enabled on RoboServer to prevent unauthorized access. If this is the case the portal running the clip must include credentials when running a clipping robot, the RoboServer credentials provider allows you to specify these programmatically.

Clip Configuration File Specification

The clip logic will look in the clip descriptor to determine which (if any) credentials to send to RoboServer. The credentials my be hard-coded, or provided through code by implementing the `RoboServerCredentialsProvider`

If your implementation of `RoboServerCredentialsProvider` is named `MyRoboServerCredentialsProvider` you can add it to the clip descriptor using the following markup

Table 112. An Example of a Clip Configuration File

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE clip PUBLIC
  "-//Kapow Technologies//DTD Clip Deployment Descriptor 1.3//EN"
  "http://www.kapowtech.com/robosuite/rql/dtd/clip-deployment-descriptor_1_3.dtd">
<clip>
  <robot-url>library:/my.robot</robot-url>
  <robot-library><default/></robot-library>
  <robot-server>
    <socket-object-protocol host="localhost" port="50000"/>
  </robot-server>
  <robot-server-credentials>
    <class-name>
      com.mycompany.myproject.MyRoboServerCredentialsProvider
    </class-name>
  </robot-server-credentials>
  <user-credentials>
    <class-name>
      com.mycompany.myproject.MyCredentialsProvider
    </class-name>
  </user-credentials>
```

```
</clip>
```

If you want to hard-code the credentials the syntax is.

```
<robot-server-credentials>
  <username>Joe Smith</username>
  <password>abcdefgh</password>
</robot-server-credentials>
```

As described, the order of the elements in the clip descriptor must follow the order defined in the DTD (References). The following diagram shows the location of the `robot-server-credentials` element in the order of these (not all elements may be present):

```
robot-url, robot-library, robot-server, robot-server-credentials,
portlet, popups-required, popups-blocked-text, action-header-filter,
portal-cookie-filter, user-credentials, properties, additional-input-
objects, additional-output-objects, user-messages, restart-session-
controller, error-handler, wait-page-text, wait-page-style
```

Implementation

To pass RoboServer credentials programmatically you must write a class which implements the following interface:

```
com.kapowtech.robosuite.client.clip.version1.api.RoboServerCredentialsProvid
```

This interface has one method named `provide`:

```
RoboServerCredentials provide(HttpServletRequest request, HttpServletRespons
                              RoboServerCredentials defaultCredentials);
```

The `provide` method returns a `RoboServerCredentials` object which contains the credentials used for authentication when the clipping robot is run.

The `RoboServerCredentials` contains a userName and password, which must match a user configured on the RoboServer, and the user must have the 'Start Robot' privilege.

If the user doesn't exist or the credentials are invalid this results in an error, which is displayed directly to the end-user of the clip, unless an error-handler is defined for the clip.

Example

Below, we show what the code of such a provider could look like.

This provider gets the username and password from session scope. The userName and password could then be configured in the portal, and passed along to the clip through the Portlet/Servlet Session

Table 113. Code Example of a RoboServer credentials Provider

```
package com.mycompany.myproject;
import javax.servlet.http.*;

public class RoboServerCredentialsProviderImpl implements RoboServerCredentialsPro
    public static final String USER_NAME_KEY = "com.kapow.roboserver.username";
    public static final String PASSWORD_KEY = "com.kapow.roboserver.password";
```

```
    public RoboServerCredentials provide(HttpServletRequest request, HttpServletRe
                                         RoboServerCredentials defaultCredentials)

        String userName = (String) request.getSession().getAttribute(USER_NAME_KEY
        String password = (String) request.getSession().getAttribute(PASSWORD_KEY)
        return new RoboServerCredentials(userName, password);
    }
}
```

Additional Input Objects Provider

Additional Input Objects providers make it possible to pass data from the servlet's environment to the clipping robot running on RoboServer.

Purpose and Use

Additional Input Objects providers are particularly well suited for passing structured data to the RoboServer. This can be useful for e.g. transmitting binary data or for passing database records to the server. To use an Additional Input Objects provider, the corresponding variables accepting the input values must be defined in Design Studio, which makes this option a bit cumbersome.

Hence, if the data is less structured and of string type, using a Properties provider is encouraged: a common example of this is passing portal user preferences to the robot.

The Additional Input Objects provider functionality has a natural counterpart in the Additional Output Objects handler, which makes it possible for the clipping robot to return a modified version of its input as output.

Clip Configuration File Specification

In order for the clipping logic code to know that it should use the given Additional Input Objects provider this must be declared in the clip configuration file (.clip-file). An example of such a clip configuration file is shown below. It tells the clip to use a given Additional Input Objects provider called `MyAdditionalInputObjectsProvider`.

Table 114. An Example of a Clip Configuration File

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE clip PUBLIC
  "-//Kapow Technologies//DTD Clip Deployment Descriptor 1.3//EN"
  "http://www.kapowtech.com/robosuite/rql/dtd/clip-deployment-descriptor_1_3.dtd">
<clip>
  <robot-url>library:/my.robot</robot-url>
  <robot-library><default/></robot-library>
  <robot-server>
    <socket-object-protocol host="localhost" port="50000"/>
  </robot-server>
  <user-credentials>
    <class-name>
      com.mycompany.myproject.MyCredentialsProvider
    </class-name>
  </user-credentials>
  <additional-input-objects>
    <class-name>
```

```
      com.mycompany.myproject.MyAdditionalInputObjectsProvider
    </class-name>
  </additional-input-objects>
</clip>
```

The `additional-input-objects` tag can contain either a specification of values to use or a class specification. (see the DTD [http://www.kapowtech.com/robosuite/rql/dtd/clip-deployment-descriptor_1_3.dtd] for details).

As described, the order of the elements in the clip descriptor must follow the order defined in the DTD (References). The following diagram shows the location of the `additional-input-objects` element in the order of these (not all elements may be present):

```
robot-url, robot-library, robot-server, robot-server-credentials,
portlet, popups-required, popups-blocked-text, action-header-filter,
portal-cookie-filter, user-credentials, properties, additional-input-
objects, additional-output-objects, user-messages, restart-session-
controller, error-handler, wait-page-text, wait-page-style
```

Implementation

Additional input objects are passed from the clip to RoboServer using an Additional Input Objects provider. This is a class that you write yourself that must implement the interface:

```
com.kapowtech.robosuite.client.clip.version1.api.
     AdditionalInputObjectsProvider
```

This interface has one method named `provide`:

```
RQLObjects provide(HttpServletRequest request,
                   HttpServletResponse response,
                   RQLObjects defaultObjects);
```

The `provide` method returns an `RQLObjects` object that contains the additional input objects (each being an `RQLObject`). Each `RQLObject` contains a number of `Attribute`s.

The types `RQLObject`, `RQLObjects` and `Attribute` are all from the Kapow Java API and documentation on these can be found in the Java Programmer's Guide. The `RQLObject`s in `RQLObjects` created by the method must correspond to input variables in the clipping robot. Object and attribute names are case-sensitive.

The provider has access to the `HttpServletRequest` which makes it possible to retrieve important information about the context, such as the current URL or user preferences.

The `provide` method also takes an `HttpServletResponse` as parameter — this makes the clipping API consistent with servlet and portlet APIs (such as the servlet filter) and is therefore useful if you need to call methods in these APIs (e.g. forwarding to a request dispatcher).

The third parameter `defaultObjects` of the provider is included for future use, and its value is left unspecified.

Example

Below, we show what the code of such a provider could look like. This example provider will add an `RQLObject` named `SearchInput` which have two attributes (`text` and `matchCase`) to the input objects that are being sent to the robot.

Table 115. Code Example of an Additional Input Objects Provider

```
package com.mycompany.myproject;

import com.kapowtech.robosuite.api.java.rql.construct.Attribute;
import com.kapowtech.robosuite.api.java.rql.construct.RQLObject;
import com.kapowtech.robosuite.api.java.rql.construct.RQLObjects;
import com.kapowtech.robosuite.client.clip.version1.
       api.AdditionalInputObjectsProvider;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.util.ArrayList;
import java.util.List;

public class MyAdditionalInputObjectsProvider
  implements AdditionalInputObjectsProvider {

  public RQLObjects provide(HttpServletRequest request,
                            HttpServletResponse response,
                            RQLObjects defaultObjects) {
    List rqlObjectsList = new ArrayList();
    List attributes = new ArrayList();
    attributes.add(new Attribute("text", "Kapow"));
    attributes.add(new Attribute("matchCase", "true"));
    RQLObject rqlObject = new RQLObject("SearchInput", attributes)
    rqlObjectsList.add(rqlObject);
    return new RQLObjects(rqlObjectsList);
  }
}
```

The provider will create an `RQLObject` object representing the `SearchInput` variable in the robot and add this to an `RQLObjects` object. This `RQLObjects` object will be returned from the `provide` method of the provider. The value that is assigned to the attributes will of course not normally be fixed as in the example, but may be obtained from some external source, e.g. from the portal preferences of the user.

Properties Provider

Property providers provide an alternative way of passing data from the servlet's environment to the clipping robot. The data is passed via the string attribute `properties` on the ClipRequest. Special classes and methods are available to the Properties provider for creating such values, and the properties are parsed by the robot and made available via a converter.

Purpose and Use

The property provider gives you a simple way of providing extra input to the clipping robot. It is convenient if the data to pass to the robot is simple strings, since properties are defined as simple name-value pairs, where both the name and the value are represented as a string.

The most obvious use case is to retrieve business domain values from the session and pass these to the robot, for instance to pass a customer number that the robot should enter into a field. A less obvious but very useful use of property providers is to make your robot more general: if a value of your robot is likely to change, you can replace this value by a property and have your property provider look up the value in a configuration file or database. This way it is not necessary to change the robot (and redeploy or create a

robot library) and you can just change your configuration instead. An example of this would be retrieving the start URL as a property.

Within the clipping robot, the data of a property is retrieved using the Get Property converter.[1]

Clip Configuration File Specification

The Properties provider must be declared in the clip configuration file (.clip-file). An example of such a clip configuration file is shown below. It tells the clip to use a given Additional Input Objects provider called `MyPropertiesProvider`.

Table 116. An Example of a Clip Configuration File

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE clip PUBLIC
  "-//Kapow Technologies//DTD Clip Deployment Descriptor 1.3//EN"
  "http://www.kapowtech.com/robosuite/rql/dtd/clip-deployment-descriptor_1_3.dtd">
<clip>
  <robot-url>library:/my.robot</robot-url>
  <robot-library><default/></robot-library>
  <robot-server>
    <socket-object-protocol host="localhost" port="50000"/>
  </robot-server>
  <user-credentials>
      <class-name>com.mycompany.MyCredentialsProvider</class-name>
  </user-credentials>
  <properties>
      <class-name>com.mycompany.MyPropertiesProvider</class-name>
  </properties>
  <additional-input-objects>
      <class-name>com.mycompany.MyInputProvide</class-name>
  </additional-input-objects>
</clip>
```

The `properties` tag contains either a `class-name` tag (as in the example) or a number of `property` tags which have the following syntax:

```
<property name="propertyName">value</property>
```

As described, the order of the elements in the clip descriptor must follow the order defined in the DTD (References). The following diagram shows the location of the `properties` element in the order of these (not all elements may be present):

```
robot-url, robot-library, robot-server, robot-server-credentials,
portlet, popups-required, popups-blocked-text, action-header-filter,
portal-cookie-filter, user-credentials, properties, additional-input-
objects, additional-output-objects, user-messages, restart-session-
controller, error-handler, wait-page-text, wait-page-style
```

Implementation

Your property provider must implement the interface

[1]The list of properties is also available as a text attribute on the ClipRequest and can in principle be retrieved from there. This, however, requires parsing the string, and hence it is not a very useful option.

```
com.kapowtech.robosuite.client.clip.version1.api.
     PropertiesProvider
```

This interface has one method:

```
Properties provide(HttpServletRequest request,
                   HttpServletResponse response,
                   Properties defaultProperties);
```

The `provide` method will then return a `Properties` object that contains properties.

Property names are case-sensitive.

The provider has access to the `HTTPServletRequest` and `HTTPServletResponse` as parameters — see Additional Input Objects Provider for further explanation.

The third parameter `defaultProperties` is intended for future use, and its value is left unspecified.

You can create a `Properties` object in two ways:

- Create a map from names (strings) to values (strings). Then call the `Properties` constructor with the map as argument.

  For example:

  ```
  Map<String,String> propertyMap = new HashMap<String, String>();
  propertyMap.put("property1", "value1");
  propertyMap.put("property2", "value2");
  Properties properties = new Properties(propertyMap);
  ```

- Create a list of `Property` objects and call the `Properties` constructor with this list as argument. A `Property` is created by calling its constructor with a name and a value. You should not add more than one `Property` object with the same property name, as this will cause an error when the clipping robot is executed.

  For example:

  ```
  List<Property> propertiesList = new ArrayList<Property>();
  propertiesList.add(new Property("property1", "value1"));
  propertiesList.add(new Property("property2", "value2"));
  Properties properties = new Properties(propertiesList);
  ```

Example

In the following example, an URL is chosen randomly between Google and Kapow and passed as a property.

Table 117. Code Example of a Properties Provider

```
package com.mycompany.myproject;

import java.util.ArrayList;
import java.util.List;
import java.util.Random;
```

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.kapowtech.robosuite.client.clip.version1.api.Properties;
import com.kapowtech.robosuite.client.clip.version1.api.
       PropertiesProvider;
import com.kapowtech.robosuite.client.clip.version1.api.Property;

public class MyPropertiesProvider implements PropertiesProvider {
  public Properties provide(HttpServletRequest request,
                            HttpServletResponse response,
                            Properties defaultProperties) {
    Random generator = new Random();
    int i = generator.nextInt(2);
    Property property =
      new Property("startUrl",
          "http://" + (i==0 ?
              "www.google.com":
          "www.kapowsoftware.com"));
    List<Property> properties = new ArrayList<Property>();
    properties.add(property);
    return new Properties(properties);
  }
}
```

User Credentials Provider

User Credentials providers are used to supply user credentials to the clipping robot, which in turn will use this for login on the clipped web site. User credentials are passed as attributes on the ClipRequest object, and can be retrieved by the robot from this object and assigned to the corresponding ClipRequest variable.

If the clipping robot is set up to use HTTP-based login, the robot will automatically retrieve the username and password from these attributes. If form based login is used, the robot developer must ensure that the attributes are entered into the correct field on the login form.

Purpose and Use

The User Credentials provider is used for supporting automatic login in clipped web sites and is hence instrumental in using clipped web sites in a **single sign on** portal solution.

Typically, an SSO solution will be based on some sort of credentials vault. In this case, the User Credentials provider should access this vault to retrieve the username and password. In another use case, the user might login in a special portlet, which would store the username and password (encrypted) in the session. This could then be retrieved by the User Credentials provider.

Clip Configuration File Specification

The User Credentials provider must be declared in the clip configuration file (.clip-file). The clip configuration file shown below makes the clip use a given User Credentials provider named `MyUserCredentialsProvider`.

Table 118. An Example of a Clip Configuration File

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!DOCTYPE clip PUBLIC
  "-//Kapow Technologies//DTD Clip Deployment Descriptor 1.3//EN"
  "http://www.kapowtech.com/robosuite/rql/dtd/clip-deployment-descriptor_1_3.dtd">
<clip>
  <robot-url>library:/my.robot</robot-url>
  <robot-library><default/></robot-library>
  <robot-server>
    <socket-object-protocol host="localhost" port="50000"/>
  </robot-server>
  <additional-input-objects>
    <class-name>com.mycompany.MyInputProvide</class-name>
  </additional-input-objects>
  <user-credentials>
    <class-name>
      com.mycompany.myproject.MyCredentialsProvider
    </class-name>
  </user-credentials>
</clip>
```

The `user-credentials` tag contains either a `class-name` tag (as in the example) or a pair of `username` and `password`. E.g.

```
<user-credentials>
  <username>Joe Smith</username>
  <password>abcdefgh</password>
</user-credentials>
```

As described, the order of the elements in the clip descriptor must follow the order defined in the DTD (References). The following diagram shows the location of the `user-credentials` element in the order of these (not all elements may be present):

```
robot-url, robot-library, robot-server, robot-server-credentials,
portlet, popups-required, popups-blocked-text, action-header-filter,
portal-cookie-filter, user-credentials, properties, additional-input-
objects, additional-output-objects, user-messages, restart-session-
controller, error-handler, wait-page-text, wait-page-style
```

Implementation

To create a User Credentials provider, you must implement the interface

```
com.kapowtech.robosuite.client.clip.version1.api.
     UserCredentialsProvider
```

This interface has one method:

```
UserCredentials provide(HttpServletRequest request,
                        HttpServletResponse response,
                        UserCredentials defaultCredentials);
```

This provider follows the well known pattern from previous sections on providers by taking an `HttpServletRequest` and `HttpServletResponse` as input. In addition — as in previous sections — it takes default credentials, which will always be empty in practice (i.e. user credentials with

empty username and empty password), as default credentials cannot be specified in the clip configuration file together with a class name specifying a User Credentials provider.

The provider returns an object of type `UserCredentials`. Such an object is created by calling the constructor with username and password (see the example below).

Example

A trivial example of a User Credentials provider is given below. Here the username “demo” and password “demo” is passed to the RoboServer — in practice you would retrieve the username and password from some external source such as a credentials vault or the session.

Table 119. Code Example of a User Credentials Provider

```
package com.mycompany.myproject;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.kapowtech.robosuite.client.clip.version1.api.
       UserCredentials;
import com.kapowtech.robosuite.client.clip.version1.api.
       UserCredentialsProvider;

public class MyUserCredentialsProvider
implements UserCredentialsProvider {

  public UserCredentials provide(HttpServletRequest request,
      HttpServletResponse response,
      UserCredentials defaultCredentials)
  {
    // place your code to fetch username and password here:
    final String username = "demo";
    final String password = "demo";

    return new UserCredentials(username, password);
  }
}
```

User Message Localization Provider

Kapow clipping technology pipes HTML from the clipped page to the consumer (portal or web server). This means that correct localization and internationalization of the actual content is the responsibility of the clipped pages, and that your main task is to ensure that the correct user settings are transferred to the clipped page.

However, a few user messages exist which originate solely from Kapow. Hence, to make your portal truly international, these user messages need to be localized.

Purpose and Use

The user message localization is useful in multi-language environments, where the messages should depend on the user or the browser's locale. If you just need to use clipping in a non-English speaking environment, it is easier to specify the localized texts in Design Studio.

A User Message Localization provider should provide a localized version of the static Kapow user messages, which will then be shown in the end-user's browser. Currently two such messages exist: the wait page text, and the message shown when a popup is blocked (i.e. the same messages that can be modified in the robot configuration window in Design Studio).

Dynamic user message localization can be specified in two ways. The first way of specifying user message localization is to use **resource bundles**. If several languages are involved and localization should be dependent on the computer's configuration (i.e. dependent on system and browser settings in the standard Java way), using this fundamental standard Java localization functionality will often be preferred. This is done by specifying a resource bundle in the deployment descriptor. The second and most general way is to specify a User Message Localization provider. You specify in the clip configuration file which provider to use. This can be useful if the localization is dependent on other things (e.g. if localization is user configurable to ensure that the same language is used no matter where the user logs in).

Clip Configuration File Specification

As we have seen, localization can be specified in three different ways. First, you can specify static localized texts directly in the clip configuration file. The following example shows how to localize the messages to a purely Danish system — if you specify the texts in Design Studio (the User's Browser tab in the Robot Configuration dialog) the values will be written to the clip configuration file this way.

Table 120. Clip Configuration File with Static Messages

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE clip PUBLIC
  "-//Kapow Technologies//DTD Clip Deployment Descriptor 1.3//EN"
  "http://www.kapowtech.com/robosuite/rql/dtd/clip-deployment-descriptor_1_3.dtd">
<clip>
  <robot-url>library:/my.robot</robot-url>
  <robot-library><default/></robot-library>
  <robot-server>
    <socket-object-protocol host="localhost" port="50000"/>
  </robot-server>
  <additional-input-objects>
    <class-name>com.mycompany.MyInputProvide</class-name>
  </additional-input-objects>
  <wait-page-text>
  <!--Please wait in Danish -->
    Vent venligst!
  </wait-page-text>
  <popups-blocked-text>
    <!--Popus blocked in Danish -->
    Denne applikation kræver popups.
    Slå popup blockere fra og tryk Refresh i din browser.
  </popups-blocked-text>
</clip>
```

The second way is to provide Resource Bundles; this is the standard way of specifying localization in Java, so we will not go into details here. The localization will be according to the system locale setup (operating system and browser).

The clip configuration file given below specifies a resource bundle. In the example, `com.mycompany.myproject.MyResourceBundle` is a class extending `java.util.ResourceBundle`.

Table 121. Clip Configuration File with a Resource Bundle

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE clip PUBLIC
  "-//Kapow Technologies//DTD Clip Deployment Descriptor 1.3//EN"
  "http://www.kapowtech.com/robosuite/rql/dtd/clip-deployment-descriptor_1_3.dtd">
<clip>
  <robot-url>library:/my.robot</robot-url>
  <robot-library><default/></robot-library>
  <robot-server>
    <socket-object-protocol host="localhost" port="50000"/>
  </robot-server>
  <additional-input-objects>
    <class-name>com.mycompany.MyInputProvide</class-name>
  </additional-input-objects>
  <user-messages>
    <resource-bundle>
      com.mycompany.myproject.MyResourceBundle
    </resource-bundle>
  </user-messages>
</clip>
```

The resource bundle should specify a translation of the texts named

- `UserMessageLocalizationProvider.WAIT_PAGE_TEXT_KEY`
- `UserMessageLocalizationProvider.POPUPS_BLOCKED_KEY`

An example of a root resource bundle is given below. Specific locales are specified in the same way.

Table 122. Resource Bundle Root

```
import java.util.*;

import com.kapowtech.robosuite.client.clip.version1.api.
UserMessageLocalizationProvider;

public class MyResourceBundle extends ListResourceBundle {

  private static final Object[][] localization =
    new Object[][] {
      {UserMessageLocalizationProvider.WAIT_PAGE_TEXT_KEY,
       "Please wait."},
      {UserMessageLocalizationProvider.POPUPS_BLOCKED_KEY,
       "Popups blocked. Please unblock."}
    };

  protected Object[][] getContents() {
      return localization;
  }
}
```

The name of the resource bundle can be specified in Design Studio and will then be written to the clip configuration file. Note that the `user-messages` tag can be used together with the `wait-page-text` and/or `popups-blocked-text` tags, which then provide default values.

The third way of specifying the localization is to provide a class which performs the localization. The class name is specified in the standard fashion as illustrated below.

Table 123. Clip Configuration File with a User Message Localization Provider

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE clip PUBLIC
  "-//Kapow Technologies//DTD Clip Deployment Descriptor 1.3//EN"
  "http://www.kapowtech.com/robosuite/rql/dtd/clip-deployment-descriptor_1_3.dtd">
<clip>
  <robot-url>library:/my.robot</robot-url>
  <robot-library><default/></robot-library>
  <robot-server>
    <socket-object-protocol host="localhost" port="50000"/>
  </robot-server>
  <additional-input-objects>
    <class-name>com.mycompany.MyInputProvide</class-name>
  </additional-input-objects>
  <user-messages>
    <class-name>
      com.mycompany.myproject.MyMessageLocalizationProvider
    </class-name>
  </user-messages>
</clip>
```

As described, the order of the elements in the clip descriptor must follow the order defined in the DTD (References). The following diagram shows the location of the `user-messages` element in the order of these (not all elements may be present):

```
robot-url, robot-library, robot-server, robot-server-credentials,
portlet, popups-required, popups-blocked-text, action-header-filter,
portal-cookie-filter, user-credentials, properties, additional-input-
objects, additional-output-objects, user-messages, restart-session-
controller, error-handler, wait-page-text, wait-page-style
```

Implementation

We will not describe how to implement resource bundles here — instead you should consult Sun's document on localization.

A user message provider must implement the interface

```
com.kapowtech.robosuite.client.clip.version1.api.
     UserMessageLocalizationProvider
```

This interface has one method:

```
void modifyMessageMap(HttpServletRequest request,
                      HttpServletResponse response,
                      Map messageMap);
```

This provider follows the well known pattern from previous sections on providers by taking an `HttpServletRequest` and `HttpServletResponse` as parameters.

The last parameter is a map mapping user message keys (strings) to localized messages. This provider works by modifying this map. The keys in the map are the same as we saw with resource bundles:

- `UserMessageLocalizationProvider.WAIT_PAGE_TEXT_KEY`
- `UserMessageLocalizationProvider.POPUPS_BLOCKED_KEY`

Example

A simple example of a User Message Localization provider is given below.

Table 124. An Example of a User Message Localization Provider

```
package com.mycompany.myproject;

import java.util.Enumeration;
import java.util.Locale;
import java.util.Map;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.kapowtech.robosuite.client.clip.version1.api.
       UserMessageLocalizationProvider;

public class MyUserMessageLocalizationProvider
implements UserMessageLocalizationProvider {

  public void modifyMessageMap(HttpServletRequest request,
                               HttpServletResponse response,
                               Map messageMap) {
    Enumeration locales = request.getLocales();
    if (locales.hasMoreElements()) {
      final Locale locale = (Locale) locales.nextElement();
      if (locale.equals(Locale.FRENCH))
        messageMap.put(WAIT_PAGE_TEXT_KEY,
        "Attendéz si'l vous plait");
    }
  }
}
```

In this example the wait text will be in French if the locale is French and the default wait text (as given in the robot configuration) will be used otherwise.

Distribution Policy Provider

A Distribution Policy provider makes it possible to write your own policy for contacting RoboServers in an environment with multiple RoboServers.

Purpose and Use

A Distribution Policy determines how clipping sessions should be distributed to the available RoboServers. This is, in effect, a coarse-grained client-side load balancing policy. The Distribution Policy for a particular clip is used for all instances of the clip portlet. The policy is normally not shared among different clips in a portal.

The `DistributionPolicyProvider` interface allows you to specify which `DistributionPolicy` instance should be used for the clip.

Two different distribution policies are provided by Kapow and can be used without having to use the API: Round Robin and Random. In both cases, you specify a list of RoboServers in the clip configuration file. For each session, the servlet will then choose a RoboServer from the list according to the policy specified. In addition, it will make sure that the RoboServer is available, and otherwise choose another one.

The Round Robin Distribution Policy will choose RoboServers successively from the list of servers, i.e. when one instance of the portlet has contacted some RoboServer, the next portlet instance will contact the next in the list etc. When the last RoboServer in the list has been contacted, the next instance will contact the first in the list.

The Random Distribution Policy will contact a random RoboServer each time a session starts. You should consult standard text books on the subject for an introduction to scheduling algorithms and their pros and cons.

In many cases, choosing one of these will be sufficient (Round Robin is the default if you do not modify the clip configuration file).

Defining your own Distribution Policy can, however, be useful in a number of situations. E.g.:

- It is often convenient to separate the list of available servers from the clip configuration file. This makes it possible to change the list of servers without having to redeploy. The available servers might be retrieved from a directory service such as LDAP.
- Round Robin and Random distribution policies are not always optimal: They behave best if the processes allocated to the different RoboServers result in a similar load. In addition, the implementation of Round Robin has one state (the next RoboServer to use) for each clip, which means that different clips might accidentally favor the same RoboServers. An introduction to other scheduling policies is beyond the scope of this manual.
- You might want to implement more advanced fail-over than provided in the default distribution policies. The default implementation is, however, fairly advanced: If a RoboServer fails, it is marked as unavailable. Every time a new RoboServer is requested, it is picked from the list of available servers. If a certain time has elapsed since the last check, the unavailable server is pinged and potentially marked as available. The time interval is increased every time pinging the server fails. This solution is often sufficient, but can be tuned to the actual situation (e.g. pinging more often might be efficient in a low latency-low bandwidth network, and just trying to send the request might be useful in a high latency-high bandwidth network).

Clip Configuration File Specification

Round Robin and Random distribution policies are set up in the clip configuration file by specifying a policy tag (one of `random-distribution-policy` and `round-robin-distribution-policy`) in the `robot-server` tag. These tags both contain a list of RoboServers. The following example uses `random-distribution-policy`.

Table 125. Clip Configuration File Specifying Random Distribution Policy

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE clip PUBLIC
  "-//Kapow Technologies//DTD Clip Deployment Descriptor 1.3//EN"
  "http://www.kapowtech.com/robosuite/rql/dtd/clip-deployment-descriptor_1_3.dtd">
```

```
<clip>
  <robot-url>library:/my.robot</robot-url>
  <robot-library><default/></robot-library>
  <robot-server>
    <random-distribution-policy>
      <socket-object-protocol host="localhost" port="50000"/>
      <socket-object-protocol host="localhost" port="50001"/>
    </random-distribution-policy>
  </robot-server>
  <user-credentials>
      <class-name>com.mycompany.MyCredentialsProvider</class-name>
  </user-credentials>
</clip>
```

If you want to implement your own Distribution Policy, you should specify a class name instead as shown below.

Table 126. Clip Configuration File Specifying Random Distribution Policy

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE clip PUBLIC
  "-//Kapow Technologies//DTD Clip Deployment Descriptor 1.3//EN"
  "http://www.kapowtech.com/robosuite/rql/dtd/clip-deployment-descriptor_1_3.dtd">
<clip>
  <robot-url>library:/my.robot</robot-url>
  <robot-library><default/></robot-library>
  <robot-server>
    <class-name>
      com.mycompany.myproject.MyDistributionPolicyProvider
    </class-name>
  </robot-server>
  <user-credentials>
      <class-name>com.mycompany.MyCredentialsProvider</class-name>
  </user-credentials>
</clip>
```

As described, the order of the elements in the clip descriptor must follow the order defined in the DTD (References). The following diagram shows the location of the `robot-server` element in the order of these (not all elements may be present):

```
robot-url, robot-library, robot-server, robot-server-credentials,
portlet, popups-required, popups-blocked-text, action-header-filter,
portal-cookie-filter, user-credentials, properties, additional-input-
objects, additional-output-objects, user-messages, restart-session-
controller, error-handler, wait-page-text, wait-page-style
```

Implementation

Your Distribution Policy provider must implement the interface:

```
com.kapowtech.robosuite.client.clip.version1.api.
     DistributionPolicyProvider
```

This interface has one method:

```
DistributionPolicy provide(HttpServletRequest request,
                           HttpServletResponse response);
```

The arguments are the usual request and response, and are available to access the context and call servlet API methods.

The `provide` method returns a

```
com.kapowtech.robosuite.api.java.rql.engine.dist.
     DistributionPolicy

RQLEngine nextRQLEngine();

List selectAvailableRQLEngines();
```

The first method is at the core of the Distribution Policy: Whenever a session is started it must return a new `RQLEngine`.

The second method returns a list of all available `RQLEngine`s. This will often be a natural part of the implementation of the `DistributionPolicy`, but has been made public: It can be used to allow `RQLEngine`s to multicast special requests to all RoboServers.

All relevant state is kept in the `DistributionPolicy` instance. For example, the `RoundRobinDistributionPolicy` keeps track of which server instance was returned last, and also which servers are known to be unavailable. The `DistributionPolicy` implementation is accessed from multiple threads, so it must be thread-safe. In practice, this means that all state must be protected by a mutual exclusion lock.

Note: It is important that you only create one Distribution Policy inside your provider, and return references to that instance on subsequent calls. If you return a new instance every time the provider is called, the load will not be distributed between server, as the RQLEngine returned by the policy will only be used for the one request, and the policy will then be discarded.

Example

It is important to understand that implementing your own Distribution Policy is a difficult task. To give a complete and useful example which goes beyond the pre-implemented distribution algorithms is therefore beyond the scope of this manual. The example given below is therefore not a useful one, and while it is fully functional, it is less efficient than the built-in Random Distribution Policy. It is, however, sufficiently complex to illustrate the central concerns that you must address when you implement your own Distribution Policy.

An example of a Distribution Policy provider is given below. It defines a `DistributionPolicy` as a final field (`_distributionPolicy`) which is returned every time the provide method is called — note that there are no guarantees concerning how often the `provide` method is called. The Distribution Policy is a `MyRandomDistributionPolicy`, which is described in detail below. The `MyRandomDistributionPolicy` is initialized with a static list of `RQLEngine`s — each of these are created in `addRoboServer` as a `RemoteRQLEngine` communicating via `SocketBasedObjectRQLProtocol`. The method `getRoboServers` is meant to illustrate that the list of servers is retrieved from an external source.

The example also presents the implementation of `MyRandomDistributionPolicy`. The policy works by choosing a random `RQLEngine` among the available engines.

The Distribution Policy tries to ensure that only available engines are returned in the `nextRQLEngine` and `selectAvailableRQLEngines` methods (which implement the `DistributionPolicy` interface). It does so in a naive (some would say stupid) way, by always pinging all potentially available servers to see if they are available. This is implemented in `updateAvailableEngines`, which pings each server using a special handler (the `myPingRQLHandler`) which will throw an exception if pinging errs — hence adding the engine will be skipped and the exception caught and ignored.

Pinging all servers every time is of course highly inefficient, but is included to illustrate the importance of availability.

Table 127. An Example of a Distribution Policy Provider

```
package com.mycompany.myproject;

import java.util.ArrayList;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.kapowtech.robosuite.api.java.rql.engine.RQLEngine;
import com.kapowtech.robosuite.api.java.rql.engine.dist.
       DistributionPolicy;
import com.kapowtech.robosuite.api.java.rql.engine.remote.
       RQLProtocol;
import com.kapowtech.robosuite.api.java.rql.engine.remote.
       RemoteRQLEngine;
import com.kapowtech.robosuite.api.java.rql.engine.remote.socket.
       SocketBasedObjectRQLProtocol;
import com.kapowtech.robosuite.client.clip.version1.api.
       DistributionPolicyProvider;

public class MyDistributionPolicyProvider
implements DistributionPolicyProvider {
  private interface RoboServerSpec {
    String getHost();
    int getPort();
  }

  private final DistributionPolicy _distributionPolicy
  = new MyRandomDistributionPolicy(getEngineList());

  public DistributionPolicy provide(HttpServletRequest request,
                                    HttpServletResponse response) {
    return _distributionPolicy;
  }

  private List<RQLEngine> getEngineList() {
    List<RQLEngine> engineList = new LinkedList<RQLEngine>();
    List<RoboServerSpec> servers = getRoboServers();
    for (Iterator<RoboServerSpec> r = servers.iterator();
         r.hasNext();) {
      addRoboServer(engineList, r.next());
    }
```

```
    return engineList;
  }

  private void addRoboServer(List<RQLEngine> engineList,
      RoboServerSpec host) {
    RQLProtocol protocol
      = new SocketBasedObjectRQLProtocol(host.getHost(),
        host.getPort());
    RQLEngine engine = new RemoteRQLEngine(protocol);
    engineList.add(engine);
  }

  private List<RoboServerSpec> getRoboServers() {
    //  implementation left to the reader
  }
}
```

Table 128. An Example of a Distribution Policy

```
package com.mycompany.myproject;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;
import java.util.Random;
import com.kapowtech.robosuite.api.java.rql.RQLException;
import com.kapowtech.robosuite.api.java.rql.construct.PingRequest;
import com.kapowtech.robosuite.api.java.rql.construct.RQLRequest;
import com.kapowtech.robosuite.api.java.rql.engine.RQLEngine;
import com.kapowtech.robosuite.api.java.rql.engine.RQLHandler;
import com.kapowtech.robosuite.api.java.rql.engine.dist.
       DistributionPolicy;
import com.kapowtech.robosuite.api.java.rql.engine.dist.
       NoRQLEngineAvailableException;

class MyRandomDistributionPolicy implements DistributionPolicy {
  static final RQLRequest _PING_REQUEST = new PingRequest();
  static final RQLHandler _PING_HANDLER = new myPingRQLHandler();
  private final List<RQLEngine> _engines;
  private final List<RQLEngine> _availableEngines
    = new LinkedList<RQLEngine>();
  private final Random _random = new Random();

  public MyRandomDistributionPolicy(List<RQLEngine> engines) {
    _engines = engines;
    updateAvailableEngines();
  }

  public RQLEngine nextRQLEngine()
      throws NoRQLEngineAvailableException {
    updateAvailableEngines();
    int noOfEngines = _availableEngines.size();
    if (noOfEngines == 0)
      throw new NoRQLEngineAvailableException();
    return _availableEngines.get(_random.nextInt(noOfEngines));
```

```
  }

  public List selectAvailableRQLEngines() {
    updateAvailableEngines();
    return _availableEngines;
  }

  public void updateAvailableEngines() {
    _availableEngines.clear();
    for (Iterator<RQLEngine> eng = _engines.iterator();
    eng.hasNext();) {
      RQLEngine engine = eng.next();
      try {
        engine.request(_PING_REQUEST, _PING_HANDLER);
        _availableEngines.add(engine);
      } catch (RQLException e) {
        // do nothing
      }
    }
  }
}
```

Table 129. A Simple RQLHandler for Handling the Result of Ping

```
package com.mycompany.myproject;

import com.kapowtech.robosuite.api.java.rql.RQLException;
import com.kapowtech.robosuite.api.java.rql.construct.RQLResponse;
import com.kapowtech.robosuite.api.java.rql.engine.RQLHandler;

public class myPingRQLHandler implements RQLHandler {
    public void handleError(RQLException e) throws RQLException {
        throw e;
    }

    public void handleFatal(RQLException e) throws RQLException {
        throw e;
    }

    public void handleResponse(RQLResponse response)
             throws RQLException {
        // do nothing
    }
}
```

In the implementations of `RoundRobinDistributionPolicy` and `RandomDistributionPolicy`, the knowledge of availability is put in the hands of the `RQLEngine` itself (an engine wrapping another engine as described in the Java Programmer's Guide). This means that you do not have to ping the server to know whether it is available, but can do it at each request to the `RQLEngine`. It is then the responsibility of the Distribution Policy to return the engines to service. This is done by pinging each unavailable engine after a certain interval — if the engine is still not available this is repeated after increasingly longer intervals. While the implementation given above is significantly simpler than this, it is hoped that it illustrates the principles, so you will be able to write your own Distribution Policy.

Handlers

This section describes the handler extension points in the clipping servlet. Handlers are called when execution of the clipping robot has terminated on the RoboServer — either successfully or by returning an error. Handlers can intercept the returned values (an Additional Output Objects handler) or the exception thrown in case of error (an error handler).

Additional Output Objects Handler

Additional Output Objects handlers are called by the clipping logic if the robot returns additional output objects.

Purpose and Use

An Additional Output Objects handler gives access to values returned from the Robot. These values come from variables in the robot returned in return steps. The returned values can be used for a variety of purposes like being written to log files or databases, but most often they will be put in a session attribute to be read by other servlets or portlets. The data could be a key such that other portlets can retrieve the information and show relevant information. E.g. if you are clipping a list of invoices due for a given customer, you might want to return the customer number, so other portlets can show other information — perhaps clipped from other systems — concerning the same customer. Or it could be information that you navigated past in the clipping robot — in the invoices due example, you might have navigated past a page with the customer's address, which could be shown in another portlet.

One thing that one has to be aware of with respect to clipping portlets returning additional output objects is that there is no way for the portlet to alert other portlets (clipping or non clipping) that the portlet has returned an object and that these other portlets might want to update their view to reflect the change. The reason for this is that a clip in a clipping portlet is actually an IFrame that communicates directly with a servlet without the portlet “knowing” about this. So the clipping portlet cannot decide to refresh the portal to let other portlets refresh their views. So if one wants to write a portlet that adjusts its view based on objects returned by a clip portlet then one has to put in some kind of regular updating mechanism into the portlet, such that this will update itself regularly, e.g. by using JavaScript and possibly AJAX.

Clip Configuration File Specification

To add an Additional Output Objects handler, you need to specify a class name within an `additional-output-objects` tag. The following example shows how this can be done.

Table 130. Example of a Clip Configuration File with an Additional Output Objects Handler

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE clip PUBLIC "-//Kapow Technologies//DTD Clip Deployment Descriptor 1.3//
<clip>
  <robot-url>library:/my.robot</robot-url>
  <robot-library><default/></robot-library>
  <robot-server>
    <socket-object-protocol host="localhost" port="50000"/>
  </robot-server>
  <additional-input-objects>
    <class-name>com.mycompany.MyInputProvide</class-name>
  </additional-input-objects>
  <additional-output-objects>
```

```
    <class-name>
      com.mycompany.MyAdditionalOutputObjectHandler
    </class-name>
  </additional-output-objects>
</clip>
```

As described, the order of the elements in the clip descriptor must follow the order defined in the DTD (References). The following diagram shows the location of the `additional-output-objects` element in the order of these (not all elements may be present):

```
robot-url, robot-library, robot-server, robot-server-credentials,
portlet, popups-required, popups-blocked-text, action-header-filter,
portal-cookie-filter, user-credentials, properties, additional-input-
objects, additional-output-objects, user-messages, restart-session-
controller, error-handler, wait-page-text, wait-page-style
```

Implementation

An Additional Output Objects handler is a Java class that must implement the following interface:

```
com.kapowtech.robosuite.client.clip.version1.api.
     AdditionalOutputObjectsHandler
```

This interface has one method:

```
void handle(HttpServletRequest request,
            HttpServletResponse response,
            RQLObjects outputObjects);
```

`RQLObjects` are mentioned in Additional Input Objects Provider and further described in the Java Programmer's Guide. It is generated from all values returned explicitly in the clipping robot. In each object you can then access the attribute values.

Example

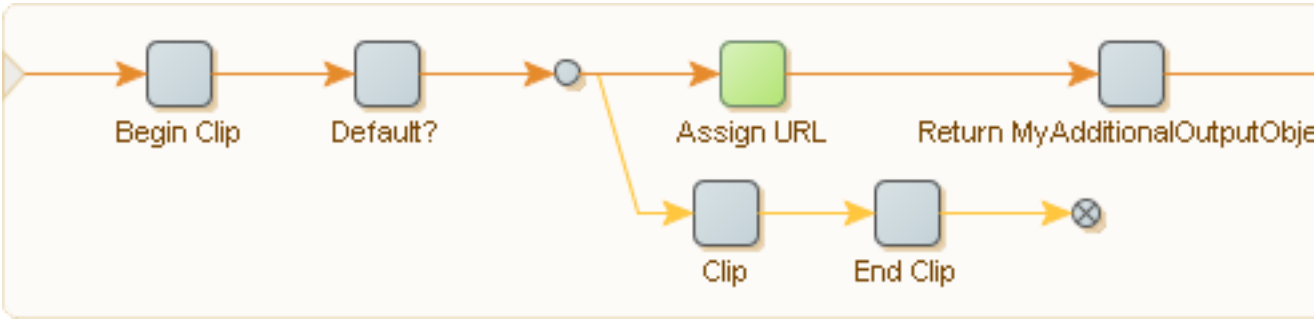
The example below shows an Additional Output Objects handler that takes an attribute value from an output object and places this in a session attribute. The output object is called `MyAdditionalOutputObject` and this has one attribute `url`.

Table 131. Code Example of an Additional Output Objects Handler

```
package com.mycompany.myproject;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.kapowtech.robosuite.api.java.rql.construct.RQLObject;
import com.kapowtech.robosuite.api.java.rql.construct.RQLObjects;
import com.kapowtech.robosuite.client.clip.version1.api.
       AdditionalOutputObjectsHandler;

public class MyAdditionalOutputObjectsHandler
implements AdditionalOutputObjectsHandler {

  public void handle(HttpServletRequest request,
      HttpServletResponse response,
```

```
      RQLObjects objects) {
    RQLObject rqlObject
      = (RQLObject) objects.getByName("MyAdditionalOutputObject").get(0);
    String url = (String) rqlObject.getAttribute("url").getValue();
    request.getSession().setAttribute("com.mycompany.url", url);
  }
}
```

The robot is a simple clipping robot that besides returning the normal `ClipResponse` variable also returns additional output containing the URL of the clipped document, as shown below.



Clipping Robot Returning Additional Output

The session attribute written by the Additional Output Objects handler may be read by other portlets using the following code:

```
request.getPortletSession().
        getAttribute("com.mycompany.url",
                     PortletSession.APPLICATION_SCOPE)
```

or in a JSP page by using the following code:

```
<%= request.getSession().getAttribute("com.mycompany.url") %>
```

Error Handler

Error handlers make it possible to intercept errors and deal with them in a suitable manner.

Purpose and Use

An error handler defines a strategy for handling errors in a clip. The custom error handler is only invoked after all other handlers have failed and the clipping logic is about to throw a `ServletException` causing the current request to fail. The custom error handler can forward to a portlet-specific error page or provide localized error messages, but should not otherwise attempt to recover from the error.

Note that most portal systems (among them BEA WebLogic Portal 8.1 to 10.0) allow the user to specify a portlet specific error page to handle exceptions, so implementing this class might not be necessary.

Clip Configuration File Specification

The example below shows how to add a reference to an error handler in a clip configuration file.

Table 132. An Example of a Clip Configuration File with an Error Handler

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<!DOCTYPE clip PUBLIC "-//Kapow Technologies//DTD Clip Deployment Descriptor 1.3//
<clip>
  <robot-url>library:/my.robot</robot-url>
  <robot-library><default/></robot-library>
  <robot-server>
    <socket-object-protocol host="localhost" port="50000"/>
  </robot-server>
  <additional-output-objects>
    <class-name>
      com.mycompany.MyAdditionalOutputObjectHandler
    </class-name>
  </additional-output-objects>
  <error-handler>
   <class-name>com.mycompany.MyErrorHandler</class-name>
  </error-handler>
</clip>
```

As described, the order of the elements in the clip descriptor must follow the order defined in the DTD (References). The following diagram shows the location of the `error-handler` element in the order of these (not all elements may be present):

```
robot-url, robot-library, robot-server, robot-server-credentials,
portlet, popups-required, popups-blocked-text, action-header-filter,
portal-cookie-filter, user-credentials, properties, additional-input-
objects, additional-output-objects, user-messages, restart-session-
controller, error-handler, wait-page-text, wait-page-style
```

Implementation

A custom error handler must implement the following interface:

```
com.kapowtech.robosuite.client.clip.version1.api.
     ErrorHandler
```

This interface has one method:

```
void handleError(HttpServletRequest request,
                 HttpServletResponse response,
                 Exception e) throws Exception;
```

The exception (`e`) in this method signature may be used to find the cause of the error to be able to produce an appropriate error message.

Example

An example of an error handler class is given below. It will forward the request to a JSP page that will present an error message. The handler will place the message in a request attribute such that the JSP page can get access to this. Further below there is a simple example of how such a JSP page can be constructed. One might decide not to handle the exception in an error handler and re-throw the exception such that the portal or web application's default method for handling error may be invoked.

Table 133. Code Example of an Error Handler

```
package com.mycompany.myproject;
```

```
import javax.servlet.RequestDispatcher;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.kapowtech.robosuite.client.clip.version1.ClipException;
import com.kapowtech.robosuite.client.clip.version1.api.ErrorHandler;

public class MyErrorHandler implements ErrorHandler {

  private static final String _ERROR_JSP = "error.jsp";

  public void handleError(HttpServletRequest request,
                          HttpServletResponse response,
                          Exception clipException)
  throws Exception {
    request.setAttribute("com.mycompany.error",
                         clipException.getMessage());
    RequestDispatcher dispatcher
      = request.getRequestDispatcher(_ERROR_JSP);
    try {
      dispatcher.forward(request, response);
    } catch (Exception e1) {
      throw new ClipException(
          "ErrorHandler failed to redirect to jsp: " +
          _ERROR_JSP, e1);
    }
  }
}
```

The JSP error page `error.jsp` could then look something like what is shown below. The page `error.jsp` must be located in the same folder as the clip configuration file for the example to work.

Table 134. JSP Error Page Example

```
<head>
  <title>Error</title>
</head>
<body>
  <p>
    An error has occurred: <%= request.getAttribute("com.mycompany.error") %>
  </p>
</body>
```

Filters

Filters intercept the communication between the client and RoboServer. They are able to modify the content of the communication: Both requests sent to RoboServer and responses from RoboServer can be modified.

Header Filter

With a header filter, you can exclude, rewrite or even add to the list of HTTP headers sent to RoboServer and returned from RoboServer.

Purpose and Use

Some single sign on solutions (SSO) work by adding a token to the HTTP header of all calls. Suppose that you are clipping from one internal site using this SSO solution and want to insert the clip into a portal using the same SSO solution. Now, the SSO will append the special header to the request from the user's browser to the servlet and you need to make sure that the header is forwarded to RoboServer and on to the site from which you are clipping. On the other hand, the header should not be forwarded to other sites which are not under the SSO solution.

The SSO problem is one of the key examples of using a header filter, and the hope is that it serves to illustrate the purpose of the filter. On the other hand, a word of caution is needed: altering or adding headers or even just allowing headers from the user's browser to be forwarded to RoboServer might result in unexpected behavior. E.g. consider the situation where the `user-agent` is allowed to filter through: Then the clipped site will begin to function according to the end-user's browser and not according to the browser emulated by RoboServer (where most of the site is processed).

For this reason the default filter will exclude all headers.

Clip Configuration File Specification

If nothing is specified in the clip configuration file, no headers will be passed on. If you instead want to allow all unrecognized extension headers to pass, you should specify the preconfigured `extension-headers` filter. The following example shows how to do this.

Table 135. Header Filter Element

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE clip PUBLIC "-//Kapow Technologies//DTD Clip Deployment Descriptor 1.3//
<clip>
  <robot-url>library:/my.robot</robot-url>
  <robot-library><default/></robot-library>
  <robot-server>
    <socket-object-protocol host="localhost" port="50000"/>
  </robot-server>
  <user-credentials>
    <class-name>com.mycompany.MyCredentialsProvider</class-name>
  </user-credentials>
  <action-header-filter>
    <extension-headers/>
  </action-header-filter>
</clip>
```

The `extension-headers` filter passes on all unrecognized extension headers. A header is an unrecognized extension header if it is not a general, request, response, or entity header as defined in RFC 2616 (HTTP/1.1), and is not a cookie-related header (as defined in RFC 2109 and RFC 2965). This filter will be useful in the SSO example presented above.

If you wish to write your own header filter, you specify the class name of your implementation as follows:

```
<action-header-filter>
    <class-name>
      com.mycompany.MyHeaderFilter
    </class-name>
  </action-header-filter>
```

As described, the order of the elements in the clip descriptor must follow the order defined in the DTD (References). The following diagram shows the location of the `action-header-filter` element in the order of these (not all elements may be present):

```
robot-url, robot-library, robot-server, robot-server-credentials,
portlet, popups-required, popups-blocked-text, action-header-filter,
portal-cookie-filter, user-credentials, properties, additional-input-
objects, additional-output-objects, user-messages, restart-session-
controller, error-handler, wait-page-text, wait-page-style
```

Implementation

If you wish to write your own header filter, your must implement the interface

```
com.kapowtech.robosuite.client.clip.version1.api.
     HeaderFilter
```

This interface has one method:

```
public List filterHeaders(List headers)
```

Both the argument and the result of this method is a list of headers:

```
com.kapowtech.robosuite.client.clip.version1.api.
     Header
```

A header cannot be modified, but new headers are easily constructed.

As described, the order of the elements in the clip descriptor must follow the order defined in the DTD (References). The following diagram shows the location of the `action-header-filter` element in the order of these (not all elements may be present):

```
robot-url, robot-library, robot-server, robot-server-credentials,
portlet, popups-required, popups-blocked-text, action-header-filter,
portal-cookie-filter, user-credentials, properties, additional-input-
objects, additional-output-objects, user-messages, restart-session-
controller, error-handler, wait-page-text, wait-page-style
```

Example

The example below gives the implementation of a header filter, which corresponds to the default blocking filter which excludes all headers.

Table 136. Code Example of a Header Filter

```
package com.mycompany.myproject;

import java.util.Collections;
import java.util.List;
import com.kapowtech.robosuite.client.clip.version1.api.
       HeaderFilter;

public class MyHeaderFilter implements HeaderFilter {
  public List filterHeaders(List headers) {
    return Collections.EMPTY_LIST;
```

```
  }
}
```

Controllers

A controller is an extension point which allows the programmer direct control over the execution of the robot on the server. Only one controller is available, namely the Restart Session controller, which allows the programmer to restart the robot session.

Restart Session Controller

A Restart Session controller determines whether to restart the clipping session for a robot on a RoboServer.

Purpose and Use

It is often useful to restart the session if some of the input values to a robot have changed. An example could be a clip that uses credentials to login to the remote site. If a user changes the credentials, the robot should logout and login again. This would happen if the robot is told to restart its clipping session, because this would make the robot run its logout branch (if it has one) and then run its login branch again with the new credentials.

Clip Configuration File Specification

The following example of a clip configuration file shows how to refer to an error handler.

Table 137. Example of a Clip Configuration File with a Restart Session Controller

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE clip PUBLIC "-//Kapow Technologies//DTD Clip Deployment Descriptor 1.3//
<clip>
  <robot-url>library:/my.robot</robot-url>
  <robot-library><default/></robot-library>
  <socket-object-protocol host="localhost" port="50000"/>
  </robot-server>
  <restart-session-controller>
    <class-name>
      com.mycompany.MyRestartSessionController
    </class-name>
  </restart-session-controller>
<error-handler>
   <class-name>com.mycompany.MyErrorHandler</class-name>
  </error-handler>
</clip>
```

As described, the order of the elements in the clip descriptor must follow the order defined in the DTD (References). The following diagram shows the location of the `restart-session-controller` element in the order of these (not all elements may be present):

```
robot-url, robot-library, robot-server, robot-server-credentials,
portlet, popups-required, popups-blocked-text, action-header-filter,
portal-cookie-filter, user-credentials, properties, additional-input-
objects, additional-output-objects, user-messages, restart-session-
controller, error-handler, wait-page-text, wait-page-style
```

Implementation

A Restart Session controller must implement the interface:

```
com.kapowtech.robosuite.client.clip.version1.api.
     RestartSessionController
```

This interface has one method:

```
public boolean shouldRestartSession(HttpServletRequest request,
                                    HttpServletResponse response)
```

The method should return `true` when the robot session should be restarted.

Example

In this section, we just give a fragment of an example of the use: We leave unspecified how to decide when to restart. In Communication between Portlet and Servlet, we will see how we can communicate this information from the surrounding portlet to the servlet and hence on to the controller. The next section will also illustrate how to extend the portlet, and these two techniques in combination will allow us to extend the portlet with an edit page where the end-user can control the restart.

A prototype of a Restart Session controller is shown below.

Table 138. Example of a Restart Session Controller

```
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.kapowtech.robosuite.client.clip.version1.api.RestartSessionController;

public class MyRestartSessionController
    implements RestartSessionController {
  public boolean shouldRestartSession(HttpServletRequest request,
                                      HttpServletResponse response)
  {
    final Boolean shouldRestart = ...
    return shouldRestart;
  }
}
```

Tips and Tricks

This section discusses some common use cases and present ideas of how to solve them.

Extending the Standard Portlet

You can extend the functionality of your clipping portlet by extending the `ClipJavaPortlet` class, which by default is used for all clipping portlets. Remember to change the `<portlet-class>` in `portlet.xml` to point to your implementation. E.g.:

```
<portlet-class>
  clippingapi.ClipJavaPortletWithHelp
```

```
</portlet-class>
```

Example

A very simple example of extending the clipping portlet is to add a help page. First, you must enable the help-mode of your portlet (again in `portlet.xml`):

```
<supports>
  <mime-type>text/html</mime-type>
  <portlet-mode>help</portlet-mode>
</supports>
```

You then extend the `ClipJavaPortlet` class and implement the `doHelp` method, for example as follows.

Table 139. Extending the Clipping Portlet with a Help Page

```
package com.mycompany.myproject;

import javax.portlet.PortletException;
import javax.portlet.PortletRequestDispatcher;
import javax.portlet.RenderRequest;
import javax.portlet.RenderResponse;
import com.kapowtech.robosuite.client.clip.version1.adapter.
       platform.portlet10.ClipJavaPortlet;

public class ClipJavaPortletWithHelp extends ClipJavaPortlet {

  protected void doHelp(RenderRequest request,
                        RenderResponse response)
  throws PortletException, java.io.IOException {
    PortletRequestDispatcher portletRequestDispatcher
      = getPortletContext().getRequestDispatcher("/help.html");
    portletRequestDispatcher.include(request, response);
    response.flushBuffer();
  }
}
```

In this example, the `doHelp` method redirects to a simple html page displaying the help.

Another common use case is to add an edit page. We will return to this example in the next section.

Communication between Portlet and Servlet

If clipping is used in a Portal, the portlet shown will contain an IFRAME with a servlet (see The Clipping Logic). Often information from the portlet needs to be communicated from the portlet to the servlet. Assume for instance that you have created an edit page for your portlet, and that this page allows the user to restart the clipping session. Now you need to communicate the user's action from the portlet to the clipping servlet, such that a `RestartSessionController` can retrieve the information and return true.

Example

As an example let us assume that we want to extend our clipping portlet with an edit page on which there is a button to restart the portlet. That is, if at a given point a user goes to the edit page and presses this

button the clip will restart its session and the clip will display its start URL instead of whatever it was displaying before.

This example is meant to capture the essence of a common situation where the clip portlet has an edit page allowing the user to change the configuration of the robot. After modifying the configuration and pressing “save”, the clip has to be restarted for the configuration changes to take effect.

For this to work we need to find a way to get the information that the portlet wants to restart its session to the servlet.

The example below shows a portlet class that extends the clipping portlet class `ClipJavaPortlet`. First, it adds an edit page to the portlet: This is done in the `doEdit` method, which redirects to a JSP page called `edit.jsp`, given below.

Table 140. The Edit JSP Page

```
<%@ taglib uri="http://java.sun.com/portlet" prefix="portlet"%>
<form name="edit" method='POST'
      action="<portlet:actionURL portletMode='edit'/>">
   Press here to restart the clip:
   <input name="restart" type="submit" value="restart"/>
</form>
```

Table 141. Portlet Extending ClipJavaPortlet

```
package com.mycompany.myproject;

import ...

public class ClipJavaPortletWithEdit extends ClipJavaPortlet {
  public static final String SHOULD_RESTART = "ShouldRestart";
  private static final String LAST_RESTART_ATTRIBUTE_NAME
    = "lastRestart";
  private static final Random random = new Random();

  public void processAction(ActionRequest request,
                            ActionResponse response)
      throws PortletException, IOException {
    final boolean isEditMode
      = PortletMode.EDIT.equals(request.getPortletMode());
    if (isEditMode) {
      final String restart = request.getParameter("restart");
      if (restart != null) {
        response.setRenderParameter("restart",
                                    "" + random.nextInt());
        response.setPortletMode(PortletMode.VIEW);
      }
    }
  }

  protected void doEdit(RenderRequest request,
                        RenderResponse response)
      throws PortletException, java.io.IOException {
    PortletRequestDispatcher portletRequestDispatcher =
```

```
      getPortletContext().getRequestDispatcher("/edit.jsp");
    portletRequestDispatcher.include(request, response);
    response.flushBuffer();
  }

  public void doView(RenderRequest request,
                     RenderResponse response)
      throws PortletException, IOException  {
    final String restartParameter = request.getParameter("restart");
    final String lastRestartParameter
      = (String) getSessionAttribute(request, response,
                                     LAST_RESTART_ATTRIBUTE_NAME);
    final boolean shouldRestart
      = restartParameter != null &&
        !restartParameter.equals(lastRestartParameter);
    if (shouldRestart) {
      setSessionAttribute(request, response,
                          LAST_RESTART_ATTRIBUTE_NAME,
                          restartParameter);
    }
    setSessionAttribute(request, response,
                        SHOULD_RESTART, shouldRestart);
    super.doView(request, response);
  }
}
```

The edit page has a button ‘restart’. When the user presses this, the method `processAction` will be called. If the user has pressed ‘restart’, the `processAction` method will set a RenderParameter `restart` to a random value. The `doView` method can read this parameter and check whether it is equal to the last value of the `restart` parameter (which is saved as a session attribute). Hence `processAction` ensures that a new value is generated whenever it is responsible for calling `doView`, but when `doView` is called for other reasons, the value will be unchanged. Thus the random value makes it possible for `doView` to distinguish calls due to the user pressing ‘restart’, and calls due to portal refreshes (this method works well in practice, but is not 100% bulletproof in principle, as `random` might return the same value twice).

If `doView` is called due to a restart, we should make this fact available to the Restart Session controller: The key to transferring data from the portlet to the servlet (where the Restart Session controller is called) is to let the portlet write a session attribute and let the servlet read it.

The Restart Session controller then reads the same attribute and resets it.

So to summarize:

1. The JSP edit page communicates that a restart is required by adding a parameter.

2. The `processAction` method reads this parameter, sets a render parameter to a random value and changes the portlet mode to `VIEW`.

3. The `doView` method reads the render parameter, and if it is set and different from the saved value, it sets a session attribute.

4. The Restart Session controller reads the session attribute and restarts the session if necessary.

In the above example, setting and reading session attributes is done using method calls. The implementation of these is in the `ClipJavaPortletWithEdit` class and is presented below. There are two

setter methods and two getter methods, making it possible to call them from `doView` and the Restart Session controller (i.e. the servlet), using a `RenderRequest`/`RenderResponse` pair resp. a `HttpServletRequest`.

It is important to note that session attributes are shared among all portlets, so if you just write a session attribute “ShouldRestart”, you might risk that another portlet restarts its session.

To avoid this, we augment the attribute name with an identifier uniquely identifying the portlet instance: The `clipJavaPortlet` class offers a method `getPortletID` for exactly this purpose so we can use this method in the session attribute methods taking a `RenderRequest`/`RenderResponse` pair.

Unfortunately, the `getPortletID` method requires a `RenderRequest` (because it uses `getNameSpace`) so we cannot call this from the Restart Session controller, but only from `doView`. This problem has been solved in the clipping framework in the following way: When the `ClipJavaPortlet` class embeds the servlet in an IFrame it calls the `getPortletID` method to retrieve an id, and passes this id as a parameter to the servlet. This means that we are able to retrieve the id in the methods, which take an `HttpServletRequest` by getting this parameter.

Table 142. Session Attribute Methods

```
public static Object getSessionAttr(HttpServletRequest request,
                                     String name) {
  final HttpSession session = request.getSession();
  final String portletId
    = request.getParameter(ClipServlet.PORTLET_PARAMETER_NAME);
  return session.getAttribute(name + portletId);
}

public static void setSessionAttr(HttpServletRequest request,
                                  String name,
                                  Object value) {
  final HttpSession session = request.getSession();
  final String portletId
    = request.getParameter(ClipServlet.PORTLET_PARAMETER_NAME);
  session.setAttribute(name + portletId, value);
}

protected Object getSessionAttribute(RenderRequest request,
                                     RenderResponse response,
                                     String name) {
  final PortletSession session = request.getPortletSession();
  final String portletId = getPortletID(response);
  return session.getAttribute(name + portletId,
                              PortletSession.APPLICATION_SCOPE);
}

protected void setSessionAttribute(RenderRequest request,
                                   RenderResponse response,
                                   String name,
                                   Object value) {
  final PortletSession session = request.getPortletSession();
  final String portletId = getPortletID(response);
  session.setAttribute(name + portletId, value,
                       PortletSession.APPLICATION_SCOPE);
```

```
  }
```

The Restart Session controller is presented below. It gets the attribute from the session by calling the `getSessionAttribute`, which we saw above. If this returns `true`, it will reset the attributes value to `false` and return `true`, which will tell the servlet to restart the clip.

Table 143. The Restart Session Controller

```
package clippingapi;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import com.kapowtech.robosuite.client.clip.version1.api.RestartSessionController;

public class MyRestartSessionController
    implements RestartSessionController {
  public boolean shouldRestartSession(HttpServletRequest request,
                                      HttpServletResponse response){
    final Boolean shouldRestart
      = (Boolean) ClipJavaPortletWithEdit.getSessionAttribute(
                     request,
                     ClipJavaPortletWithEdit.SHOULD_RESTART);
    if (shouldRestart) {
      ClipJavaPortletWithEdit.setSessionAttribute(
          request,
          ClipJavaPortletWithEdit.SHOULD_RESTART,
          Boolean.FALSE);
    }
    return shouldRestart;
  }
}
```

Runtime

Kapow Katalyst offers a number of tools for executing the robots you have developed. The following sections describe these tools:

- RoboServer is a server application that allows remote clients to execute robots. It is configured using both the Management Console and the RoboServer Settings application (for advanced configuration, such as security and authentication).
- The Management Console allows you to schedule execution of Robots, view logs and extracted data. Also provides a dashboard for monitoring system health, and a centralized place where settings for clusters of RoboServers can be configured.
- Control Center allows you to remotely monitor RoboServers and their robots.

RoboServer User's Guide

RoboServer runs robots created with Design Studio. Robots can be started in various ways; either scheduled to run at specific times by the Management Console, called via a REST web service, through the Java or .NET APIs or from a Kapplet.

In order for RoboServer to be able to execute robots, it must be activated by a Management Console. A RoboServer is active when it belongs to a cluster in a Management Console with a valid license, and sufficient KCUs have been assigned to the cluster. The RoboServer also receives settings from the Management Console where they are configured on the clusters. Please refer to the Management Console User's Guide for more information on the administration of RoboServers and clusters of these.

Starting RoboServer

RoboServer can be started in several different ways:

- By clicking the RoboServer program icon (or the 'Start Management Console' program icon which starts both the Management Console and RoboServer).
- By invoking it from the command line, which is described in detail below.
- By running it as a service. For more information about running RoboServer as a service, see Starting Servers Automatically.

To invoke RoboServer from the command line, open up a command window and type:

```
RoboServer
```

followed by Enter. If you have installed RoboServer correctly as described in the Installation Guide, then the following is printed in the command window after which RoboServer terminates:

Table 144. RoboServer Help Text

```
Detected the following plugins:
...

Usage: RoboServer [-maxClippingSessions <num>] [-verbose] [-version] [-MC] [-port]

Available services:
...
```

The dots indicate information that depends on your installation. If something different than the above is printed in the command window, then please confirm that you have installed RoboServer as described in the Installation Guide.

Example

The command below starts a RoboServer and that accepts socket connections to the Socket-based RQL Service listening on port number 50000.

```
RoboServer -p 50000
```

RoboServer Parameters

No matter how RoboServer is started, it accepts the following parameters:

Table 145. RoboServer Parameters

Parameter	Description
`-c <num> | -maxClippingSessions <num>`	This parameter specifies the maximum number of clipping sessions that can exist on this RoboServer. This parameter is optional. The default value is 50. The minimum value is 0. Example: `-maxClippingSessions 20`
`-v | -verbose`	This optional parameter causes RoboServer to output status and runtime events.
`-V | -version`	This optional parameter causes RoboServer to output the version number, and then exit.
`-MC`	This optional parameter triggers the Management Console to be started as part of RoboServer. The Management Console runs on an embedded web server configured through the Settings application.
`-s <service-name:service-parameters> | -service <service-name:service-parameters>`	This parameter specifies a RQL or JMX service that RoboServer should start. This parameter must be specified at least once, and may be specified multiple times to start multiple services in the same RoboServer. The available services depend on your installation. Example: `-service socket:50000` Example: `-service jmx:50100`
`-p <port-number> | -port <port-number>`	This is a shorthand for calling `-s socket:<port-number>` Example: `-port 50000`
`-P <port-number> | -sslPort <port-number>`	This is a shorthand for calling `-s ssl:<port-number>` Example: `-sslPort 50001`

There is no required ordering of the parameters. The JMX service provides management information from the RoboServer on the specified port.

Shutting Down RoboServer

RoboServer can be shut down using the command line tool ShutDownRoboServer. Run ShutDownRoboServer without arguments to see the various options for how to shut down the server, in particular how to handle any robots that are currently running on the server.

Production Configuration

In order to get a stable and performing production environment, you may have to tweak some of the default RoboServer parameters. We will look at the following configuration options:

- Number of RoboServer instances
- Memory allocation
- Number of concurrent robots
- Automatic memory overload detection

RoboServer runs on Oracle's Java Virtual Machine (JVM), this in turn runs on top of an operating system (OS), which runs on top of your hardware. JVM's and OS's are patched, hardware architecture change, and each new iteration aims to bring better performance; so, although we can give some general guidelines about performance, the only way to make sure you have the optimal configuration is to test it.

As a general rule you get a little more performance by starting two instances of RoboServer. The JVM uses memory management known as garbage collection (GC). On most hardware only a single CPU core is active during GC, which leaves 75% of the CPU idle on a quad-core CPU. If you start two instances of RoboServer, one instance can still use the full CPU while the other in running GC.

The amount of concurrent robots a RoboServer can run depends on the amount of CPU available, and how fast you can get the data RoboServer needs to process. The number of concurrent robots is configured in the Management Console cluster settings. A robot running against a slow website will use a lot less CPU than a robot running against a website with a fast response time, and here is why. The amount of CPU used by a program can be described with the following formula

```
CPU (core)% = 1 - WaitTime/TotalTime
```

If a robot takes 20 seconds to execute, but 15 seconds are spent waiting for the website, it is only executing for 5 seconds, thus during the 20 seconds it is using an average of 25% (of a CPU core). The steps in a robot are executed in sequence, which means that a single executing robot will only be able to utilize one CPU core at a time. Most modern CPUs have multiple cores, so a robot that executes in 20 seconds, but waits for 15 seconds, will in fact only use about 6% of a quad-core CPU.

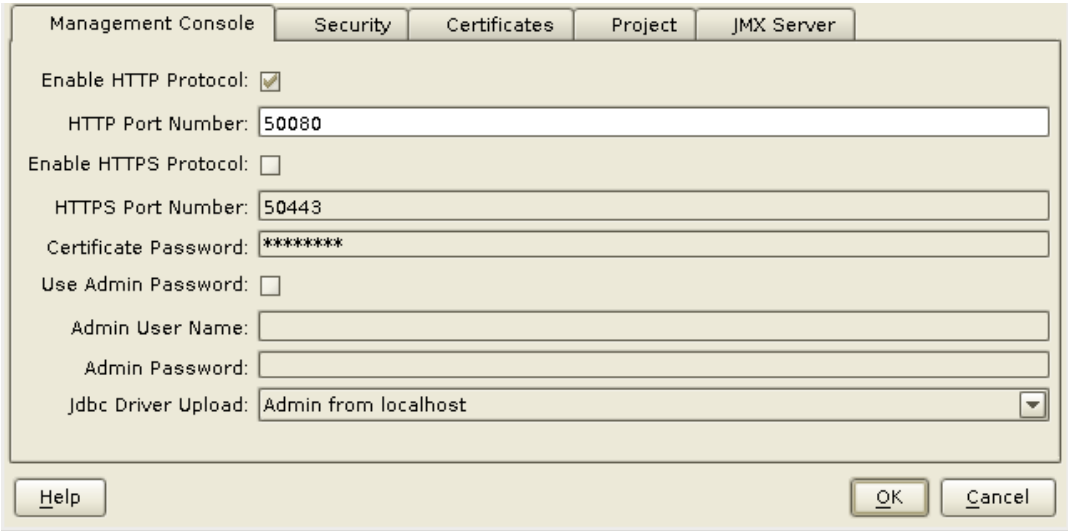
By default RoboServer is configured to maximally run 20 robots concurrently. The number of concurrent robots is configured in the Management Console cluster settings. If all your robots use 6% CPU, the CPU will be fully utilized when you are running 16-17 robots concurrently. If you start 33 of these 6% robots concurrently, you will overload RoboServer; because the amount of CPU available is constant, the result is that each robot will take twice as long to finish. In the real world the CPU utilization of a robot may be anywhere between 5-95% of a CPU core, depending on robot logic and the website it interacts with. As a result it is hard to guess or calculate the correct value for the max concurrent robots, the only way to be sure you have the right value is to do a load test and monitor the RoboServer CPU utilization, as well as the robot runtime as load increases.

Another parameter that may affect the number of concurrent robots each RoboServer can handle is the amount of memory. The amount of memory used by robots can vary from a few megabytes (MB) to hundreds of MB. By default RoboServer is configured with 1024MB of memory, this is often not enough if you are actually executing 20 robots concurrently; check Changing the RAM Allocation to see how to control memory allocation. If you don't provide enough memory to RoboServer, you run the risk of crashing it with an out of memory error. The easiest way to ensure proper memory allocation is to monitor memory utilization during your load tests. If you allocate 2048MB memory to a RoboServer, the JVM will not allocate all of it up front, but it will reserve it from the OS (this is why allocating more than 1200MB frequently fails on 32bit Windows). Once the JVM starts to use the memory, it will not be given back to the OS. To find the optimal memory allocation you often have to run a series of load tests that push the CPU to 100%. After each test is complete, you check how much of the reserved memory was actually used by the JVM (the java.exe process). If all 1024MB (default) were used, increase (usually double) the memory and run the test again. At some point the JVM will not have used all of the reserved memory, and whatever it did use reflects the actual memory requirement and should be used to configure RoboServer.

Since RoboServer will crash if it runs out of memory, RoboServer tries to prevent this from occurring. Before RoboServer starts a new robot it will check the memory utilization. If it is above 80% it will queue the robot instead of starting it; this greatly reduces the risk of crashing RoboServer if the memory allocation is configured incorrectly. This mechanism is often referred to as the 80% memory threshold. The threshold value is configurable through the system property `kapow.memoryThreshold=80`.

Configuring RoboServer

This section describes how to configure RoboServer through the RoboServer Settings application. RoboServer Settings may be started as a separate application, e.g. from the Start menu on Windows, or as a dialog opened from Control Center.



RoboServer Settings Main Window

Using this application, you can configure the following (covered in individual subsections):

- Management Console: Configuration of the embedded Management Console.
- Security: Security settings such as authentication and permissions.
- Certificates: The use of certificates.
- Project: The location of the default RoboServer project.
- JMX Server: The JMX server.

After changing any of the settings, you should click “OK” to store the new settings, and then restart any RoboServers that you have running, to make the changes take effect.

If you need to change the maximum amount of RAM that RoboServer can use, consult the section Changing the RAM Allocation.

Configuring the Embedded Management Console

RoboServer contains an embedded web server which runs the Management Console. The web server is part of RoboServer, but is activated only when RoboServer is started with the -MC option enabled. By default, the web server will listen on port 50080, and thus the Management Console web interface is available on:

```
http://host:50080/
```

Protocols and Ports

The web server can be configured to be accessible through HTTP and HTTPS on separate ports. If a protocol is enabled, a port number for it must be chosen; the defaults are port 50080 (HTTP) and port 50443 (HTTPS).

In order to enable HTTPS a server certificate in JKS format must be stored in a file called `tomcat.keystore` in the `Certificates/Web` folder in the installation. If a certificate password other than the default one (‘changeit’) must be used, it can be entered in the Certificate Password field.

Enabling Administration Security

The Management Console can be accessed not only from the same computer (localhost), but also from others. One of the points of having a Management Console is that it coordinates execution of robots, and thus it typically must be accessible to many clients.

To mitigate the potential security risk of having access to the Management Console from other machines, you can enable an administrator password. Select the Use Admin Password option and enter the desired administrator user name and password. You must use these credentials both when you publish a robot to the Management Console from Design Studio and when you access the web interface from a browser.

You can also restrict who is allowed to upload JDBC driver to the embedded Management Console (see more about uploading JDBC drivers here). Possible choices are 'Not Allowed' where no one can upload JDBC drivers, 'Admin from localhost' which means that the admin user can upload drivers if he is accessing the Management Console from the local machine, and finally, 'Admin from any host' which means that the admin user can always upload JDBC drivers.

Security

On the Security tab of the RoboServer settings, you specify general security restrictions and whether authentication is required for accessing the RoboServer. Audit logging can also be enabled here.

Restrictions

You can specify whether the RoboServer is allowed file system and command line access. By default, this is not allowed. If you enable it, however, robots running on RoboServer are allowed to access the file system and, using the Execute Command Line step, execute arbitrary commands on the machine running RoboServer. **WARNING:** enabling file system and command line access IS a security risk, and you should carefully consider if it is necessary. If you do enable it, you should make sure the machine is not accessible from outside the local network, and/or you should require user authentication. Having a RoboServer with file system and command line access running on a machine accessible from the Internet and not requiring authentication, basically opens up the machine to the outside, and anyone can, for instance, modify the file system in a way corresponding with the access rights of the user running RoboServer.

You can also disable accepting JDBC drivers from the Management Console. When activating RoboServers, the Management Console also sends settings to them. By default, this includes any JDBC drivers that have been uploaded to the Management Console. If a malicious user has gained administrator access to the Management Console, he could upload equally malicious jar files which would then be sent to the RoboServers. If the admin Management Console user is only allowed to upload JDBC drivers from the localhost, the above would only happen if the attacker is in fact sitting in front of the machine running the Management Console, or has gained access to, for instance, a VPN (in which case you probably have bigger problems), so in general it should not be necessary to disable accepting JDBC drivers. If you do,

however, you can make JDBC drivers available to the RoboServer by manually putting them into the `lib/jdbc` directory of the installation folder as described here.

Requiring Authentication

If you want to protect your RoboServer against unauthorized access you can turn on authentication. This has effect on all RoboServers you run from your Kapow Katalyst installation, that is both a RoboServer you start as a service and a RoboServer you start from a command-line.

To turn on authentication you put a check mark the checkbox Require RoboServer Authentication. In order to be able to run robots on a RoboServer with authentication turned on, you have to add some users. You do this by clicking on the add button. This will insert a new unnamed user. You can then fill out the information about the user including the username which will then be shown in the list of users.

A user is configured using the following properties:

Table 146. Users Properties

Property	Description
Username	This is the username used by the user when accessing the RoboServer.
Password	This is the password used by the user when accessing the RoboServer.
Comments	Here you can write a comment about the user.
Start Robot	This enables the user to start robots on the RoboServer.
Stop Robot	This enables the user to stop robots on the RoboServer.
View Execution	This enables the user to connect to the RoboServer from the ControlCenter application.
View / Edit Settings	This enables the user to view and edit the configuration of the Kapow Katalyst from the ControlCenter application.
Shutdown RoboServer	This enables the user to shutdown the RoboServer from the ControlCenter application.

In order to run robots on a RoboServer that has been configured with authorization, the caller must provide proper credentials. In the Management Console, this is done in the settings. When running a robot via the Java API, credentials are provided as explained in Execution Parameters. For a clipping robot, credentials may be provided with a RoboServer credentials Provider.

Configuring Audit Logging

To automatically have every HTTP and FTP request made by RoboServer logged, check the "Log HTTP/FTP Traffic" option. This will log HTTP and FTP requests using the Log4J logger specified by `log4j.logger.kapow.auditlog`. Log4J is configured by the `log4j.properties` file in the `Configuration` folder in the application data folder.

The audit log will log all requests to both web pages and all of its resources. Here's an excerpt from the log produced by loading the front page of Google:

```
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/
02-29 13:51:22  INFO kapow.auditlog - google www.google.com http://www.google.com/
02-29 13:51:22  INFO kapow.auditlog - google www.google.com http://www.google.com/
```

```
02-29 13:51:23  INFO kapow.auditlog - google ssl.gstatic.com http://ssl.gstatic.co
```

The log contains the following information:

- Timestamp
- Log level (INFO)
- Logger (kapow.auditlog)
- Robot name
- Hostname
- Request URL
- HTTP/FTP response code
- Number of bytes loaded from the response body

Certificates

A key problem in establishing secure communications is to make sure that you're communicating with the right party. It is not sufficient just to request identity information from the other party - there must also be a way to verify this information before you can trust it. Certificates provide a solution to this, as they embody both a party's identity information (including that party's hostname and public key) and a signature from a trusted third party who vouches for the correctness of the identity information. A certificate should be trusted only:

- If the hostname stated within the certificate matches the hostname of the site that it comes from (otherwise it is a record of someone else's identity, which amounts to using false credentials).
- And if the certificate is signed by a third party that you trust.
- (Additionally, the certificate expiration date and the like should be checked, but we will not be concerned with these details here.)

We will not go into the technical details of the signing process, other than mentioning that it is based on public/private key cryptography. A signature for a certificate is the (fairly compact) result of a complex calculation that involves both the contents of the certificate and the signer's private key, and which cannot be reproduced without that same private key. The signature can be verified by doing another calculation that involves the contents of the certificate, the signature and the signer's public key. This calculation will tell whether the certificate matches the signature and thus is genuine. Note that the public key only enables you to verify a signature, not generate one. Thus the public key will not enable anyone to fake a certificate.

The signing party must never give the private key to anyone, but will distribute the public key as widely as possible. However, one issue still remains before you can trust a signature: You must be sure that you have a genuine copy of the signing party's public key. Public keys for well-known signing authorities like VeriSign are distributed with every browser and Java Runtime, and your trust in the signature (and thus the certificate) is in fact based on your trust in the way the browser or Java Runtime was installed.

It is possible to create your own signing authority by creating a public/private key pair and distributing the public key. This is done by embedding the public key in a certificate (a so-called self-signed certificate). Of course a party receiving such a certificate will not trust it based on its signature, but because he trusts you and the way that the certificate was communicated to him.

In order to make this scheme more flexible in actual implementation, it is possible to delegate the authority to sign. That is, the signature on a certificate may not actually be from a third party that you trust, but rather from yet another party who can display a certificate that you will trust. This may be extended to any number of levels, representing a chain of trust. To make verification practical, the signed certificate contains copies of all the certificates associated with the chain of trust (the last one being a self-signed root certificate). At least one of the certificates in this chain should be previously trusted by you.

Certificates are used in four different ways on RoboServer, corresponding to the four properties on the "Certificates" tab. Two of these have to do with how robots access web servers as part of their execution:

Verify HTTPS Certificates:
: A robot may need to verify the identity of a web server that it accesses (via HTTPS). Such a verification is routinely (and invisibly) done by ordinary browsers in order to detect phishing attacks. However, the verification often is not necessary when robots collect information, because the robots only access those well-known web sites that they have been written for. Thus the verification it is not enabled by default.

  If enabled, verification is done in the same way a browser does it: The web server's certificate is checked based on an installed set of trusted HTTPS certificates similar to those you can configure in a browser.

HTTPS Client Certificates:
: This is almost the same as described above, but in the opposite direction. A robot may need to access a web server which wants to verify the identity of the client (robot) that accesses it. Presumably the web server contains confidential or commercial data that should be passed on only to clients with a proven identity. This identity is represented by a HTTPS client certificate.

Two other properties have to do with authentication in the communication between RoboServer and API clients that want robots to be executed on RoboServer. These properties apply only when clients connect to RoboServer via the secure socket based RQL protocol. The main purpose of the secure protocol is to encrypt communication, but with a little configuration it will also provide authentication, i.e., identity verification:

Verify API Client Certificates:
: RoboServer is able to verify the identity of any client that connects to it in order to execute robots. This verification is disabled by default.

  If enabled, the mechanics of verification is the same as for HTTPS certificates even though the purpose is quite different. The connecting client's certificate is checked based on an installed set of trusted API client certificates.

API Server Certificate:
: RoboServer also has a server certificate that it will present to connecting clients. This certificate has a dual purpose: It makes the encryption side of SSL work (for this reason RoboServer comes with a default self-signed certificate), and it identifies this particular RoboServer to the clients.

  The default API server certificate is the same for all RoboServers and thus is not any good for identification. If your clients need to verify the identity of the RoboServer they connect to, as described in SSL, you must create and install a unique API Server Certificate for each RoboServer.

HTTPS Certificates

When a robot accesses a web site over HTTPS, it will verify the site's certificate (provided that the "Verify HTTPS Certificates" checkbox is checked). Verification is done based two sets of trusted certificates: The set of root certificates and an additional set of server certificates.

The root certificates are installed with Kapow Katalyst just as root certificates are installed with your browser. They are found in the `Certificates/Root` folder in the application data folder.

Some HTTPS sites may use certificate authorities that are not included by default. In this case, you need to install the appropriate certificates for Kapow Katalyst to load from these sites. Most often, these would be installed in the `Certificates/Server` folder in the application data folder.

To be precise, it does not matter - for the purpose of handling HTTPS sites - whether you add certificates to the set of root certificates or to the set of server certificates. However, please note that the root certificates have a broader scope, as they are also used when checking API client certificates.

To install a certificate, you need to obtain the certificate as a PKCS#7 certificate chain, as a Netscape certificate chain, or as a DER-encoded certificate. You install the certificate by copying it to either of the two folders mentioned above. The name of the file containing the certificate does not matter.

The following example explains how to install a server certificate for the website https://www.foo.com. The example is based on Internet Explorer.

- Open https://www.foo.com in Internet Explorer.
- Select “Properties” under the “File” menu, and then select “Certificates”. This will open a Certificate dialog.
- Click “Install Certificate”.
- Click “Next” twice, “Finish” once, and exit the certificate dialog. Now you have installed the certificate in your browser. The next step is to export it to a file.
- Click “Internet Options...” in the Tools menu. This opens the “Internet Options” dialog.
- Go to the “Content” tab and click on the “Certificates” button.
- Now you need to locate the installed certificate. It is most likely in the tab called “Other People”.
- Select the certificate and press “Export”.
- Press “Next” twice.
- Save the certificate in the `Certificates/Server` folder in the application data folder as `foo.cer`.
- Restart all your running RoboServers.

Note

You must install the certificates on all installations that need to load from the particular HTTPS sites.

HTTPS Client Certificates

When a robot accesses a HTTPS site, it may need to provide its own certificate to the web server. This is set up in the robot's Default Options [../ref/robomaker/reference/stepaction/support/BrowserEngine.html]

or in the step-specific Options. One way to provide the certificate is to reference one of those that have been configured into the Kapow Katalyst installation.

To add a new client certificate, in the Certificates tab of RoboServer Settings click the icon below the list of HTTPS client certificates. You will be prompted to select a certificate file which must be in PKCS12 format, and you must enter the password used to encrypt the file. When the certificate is first entered into the list, it will be given a unique ID, which is later used in the robot to select the certificate. You can change the ID by clicking the icon. Remember that the certificates must be configured into all Kapow Katalyst installations that need to run the robot.

API Client Certificates

When API clients connect to RoboServer over SSL, RoboServer will verify the certificates presented to it (provided that the "Verify API Client Certificates" checkbox is checked). Verification means that RoboServer will reject connections from clients that fail verification, and is done based on two sets of trusted certificates: The set of root certificates and an additional set of API client certificates.

The root certificates are installed with Kapow Katalyst just as root certificates are installed with your browser. They are found in the `Certificates/Root` folder in the application data folder. These are the same root certificates which are used for checking HTTPS certificates; however, root certificates probably will play a much smaller role when verifying API clients.

This is because in most cases, you will create your own self-signed API client certificates rather than use (expensive) certificates issued by official signing authorities. You should install your API client certificates in the `Certificates/API/TrustedClients` folder in the application data folder so that RoboServer will recognize them.

Technically speaking, it does not matter - for the purpose of verifying connecting API clients - whether you add API client certificates to the set of root certificates or to the set of API client certificates. However the guidelines given above will help you avoid problems caused by the fact that the root certificates are also (even mainly) used when checking HTTPS certificates.

You can generate a self-signed certificate for your API client with the Java `keytool` command as follows:

```
keytool -genkey -keystore client.p12 -alias client -keyalg RSA -storetype "P
```

You will be prompted for the following information: Name (domain), name of Organizational Unit, Organization, City, State, Country and password. Do not forget the password, there is no way to retrieve it if lost. This call of `keytool` will put the certificate into the keystore `client.p12`. You then must extract it into a separate file:

```
keytool -export -keystore client.p12 -alias client -storetype "PKCS12" -file
```

You will be prompted for the password used when the certificate was generated. The output file `client.pub.cer` is what should be copied into the `Certificates/API/TrustedClients` folder in the application data folder.

API Server Certificate

For technical reasons, RoboServer must have a server certificate that it can present to API clients when they connect to it using SSL. During installation, a default self-signed certificate is installed. This certificate is no good for identification purposes since it is the same for all RoboServers, and should be replaced if the API clients need to verify the identity of the RoboServer.

If you want to use a certificate issued by a signing authority like VeriSign, you can import it by clicking the "Change" button in the Certificates tab of RoboServer Settings. This will open a file selection dialog

and once the certificate is located, you will be prompted for the password. If the password is correct the certificate will be imported, and you should see the certificate's Issued To, Issued By, and Expires properties.

You can generate a new self-signed certificate for RoboServer by using the following Java `keytool` command:

```
keytool -genkey -keystore server.p12 -alias server -keyalg RSA -storetype "P
```

You will be prompted for the following information: Name (domain), name of Organizational Unit, Organization, City, State, Country and password. Do not forget the password, there is no way to retrieve it if lost. The certificate will be saved in the keystore file `server.p12`, which you can then import as described.

Setting the Default RoboServer Project

You can set the location of the default RoboServer project folder in the 'Project' tab of RoboServer Settings. By default, the folder will be set to the default robot project created in the installation folder by the installation process. See the Design Studio User's Guide for more information on robot projects.

The RoboServer default project is used only by the API. When executing a robot from the API, any references it has to types, snippets or other resources are resolved by looking in the default project.

Configuring the JMX Server

The embedded JMX server can be used to monitor the running RoboServer through tools like JConsole. It is enabled by providing an argument on the RoboServer command line (see the RoboServer User's Guide for further information.)

Hiding Sensitive Robot Input

The Show Inputs option controls whether robot input parameters are shown in the management interface. This makes it possible to hide security sensitive information like passwords.

JMX Server Access

By default the JMX server can be accessed by all clients with access to the correct port on the server. By selecting the Use Password option the chosen user name and password must be provided when connecting.

Heartbeat Notifications

If an interval (in seconds) greater than 0 is specified the JMX server sends out a heartbeat notification with the given interval as long as the RoboServer is running and responding to queries.

Changing the RAM Allocation

As installed, each Kapow Katalyst application is configured with a maximum amount of RAM that it may use. This amount usually is plenty for ordinary work, but if you run many robots in parallel on RoboServer, or if some robots use much RAM, it may be necessary to increase the allocation.

You can change the allocation for any of the applications by editing its `.conf` file, found in the `bin` subfolder of the installation. For RoboServer, the file to edit is named:

```
bin/RoboServer.conf
```

You need add a line with:

```
wrapper.java.maxmemory=1024
```

For example, in order to permit RoboServer to use up to 4GB of RAM, add this line:

```
wrapper.java.maxmemory=4096
```

(Note that an allocation this high is possible only on the 64-bit version of Kapow Katalyst).

Also note that the `.conf` file can be edited only by the user who installed Kapow Katalyst (e.g., typically the administrator on Windows).

If necessary, the maximum RAM allocation for other Kapow Katalyst applications can be changed in the same way.

Control Center User's Guide

Control Center is a legacy application for managing RoboServers. Most monitoring and administration of RoboServers and robots are today done using the Management Console. For old setups that do not take advantage of the Management Console, Control Center can be used. Control Center also allows monitoring of robots runs that have been initiated by communication from a Java or .NET client directly to RoboServer and provides progress information on the robot runs.

Organization

The guide is divided into sections covering:

- Basic concepts.
- The user interface.
- How to use Control Center to manage RoboServers.
- How to use Control Center to manage robots.
- Miscellaneous other Control Center features.

Control Center Basics

This section describes the basic concepts used in the Control Center.

The Control Center has the following main tasks:

- Monitoring and shutting down RoboServers.
- Monitoring and stopping robots on RoboServers.
- Various administrative tasks.

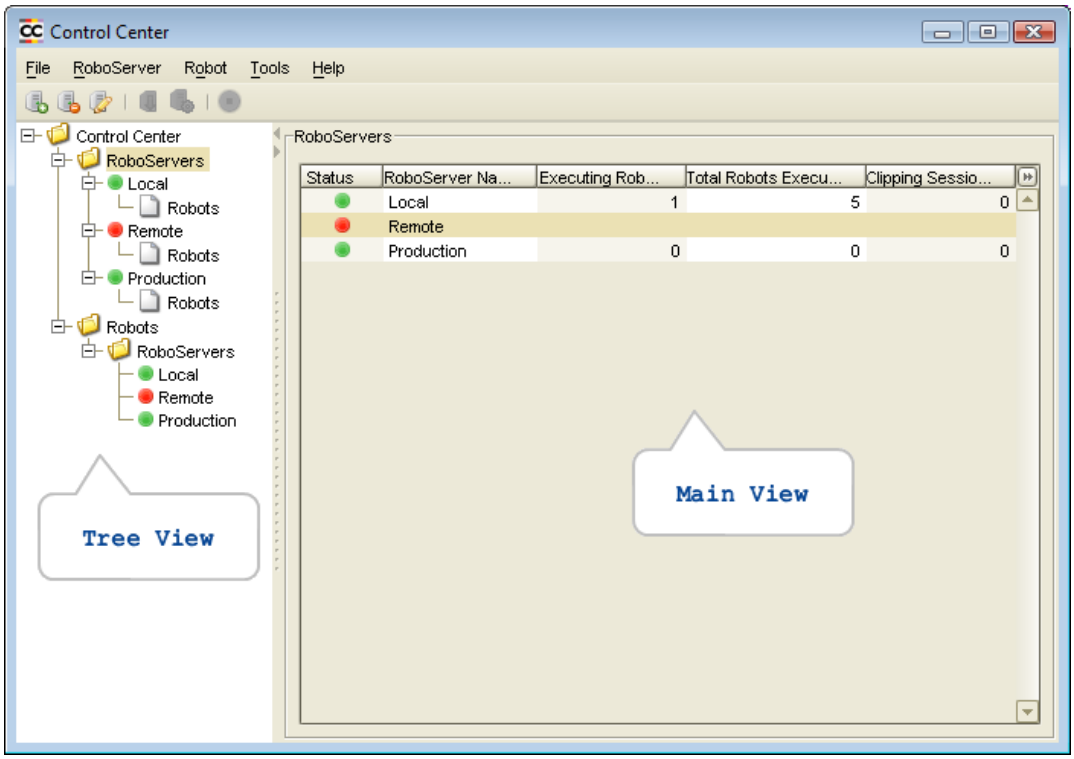
In order for the Control Center to manage a RoboServer, you must **register** the RoboServer with the Control Center. Registering a RoboServer simply means adding it to the list of RoboServers that can be managed by the Control Center.

Control Center User Interface

This section gets you started with using the Control Center by introducing you to the user interface of the application.

The Main Window

The main window of the Control Center is shown below:



Control Center Main Window

To the left in the window is the **tree view**. The root of the tree (labeled Control Center) has two children that are the two main categories:

- The **RoboServers** category (selected above), which shows the RoboServers that are currently registered in the Control Center.
- The **Robots** category, which shows the robots on all registered RoboServers.

As seen in the tree view above, the **Robots** category has a subcategory called **RoboServers**, which shows the robots on the individual RoboServers.

To the right in the window is the **main view**. The main view shows details about the current selection in the tree view. In, an overview of the registered RoboServers is shown (corresponding to the category **RoboServers** selected in the tree view).

Topmost, you see the **menu line** and **tool bar**. The menu line and tool bar contain the actions that are relevant for the current selection in the tree and main view.

Navigating the Control Center

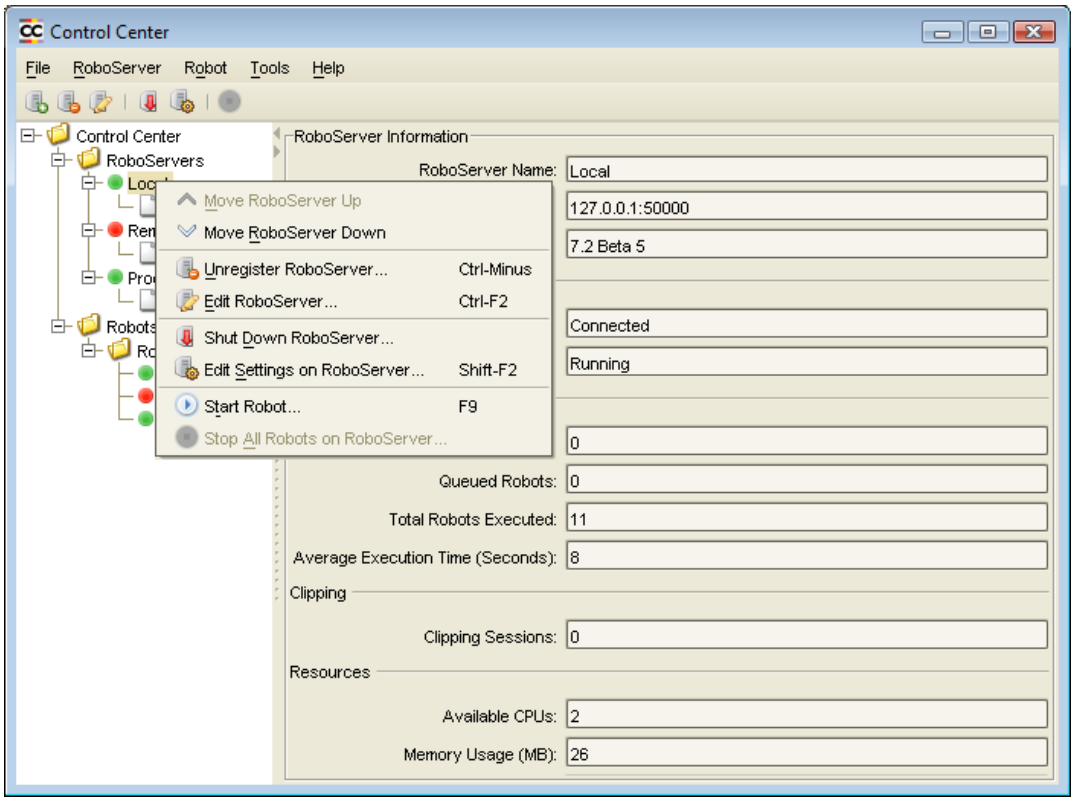
Changing the Selection

When you use the Control Center, the selection in the tree view determines what you currently view. Use the left mouse button to make a selection in the tree view. When the selection changes, the main view as well as the menu line and the tool bar change accordingly to show whatever is relevant for the current selection.

Actions and Popup Menus

As already mentioned, the menu line and tool bar contain actions relevant for the current selection. Some of these actions also have keyboard shortcuts (for example, you can edit a RoboServer by pressing Ctrl-F2).

If you right click in the tree view or in a table in the main view, a popup menu will open. The popup menu contains actions that apply to the item that you clicked:



A Popup Menu

In general, there are several ways to perform an action:

- Select the action in the tool bar.
- Select the action in the appropriate menu.
- Right click (in the tree view or the main view) and select the action in the popup menu.
- Press the keyboard shortcut (if any).

For a selection in the tree view and in the main view, the menu always contains all applicable actions, while the tool bar contains the most commonly used actions.

Tables in the Main View

If you double click an item in a table in the main view, the selection changes to show details about the item that you clicked. For example, if you are currently viewing a table of RoboServers and double click a RoboServer in the table, this RoboServer is selected in the tree view and the main view shows the details about this RoboServer.

By left clicking a column header of a table in the main view, the table is sorted by the column that you clicked.

By right clicking the header of a table in the main view, you can select which columns to view. Note that this selection only applies to the current view.

Moving RoboServers

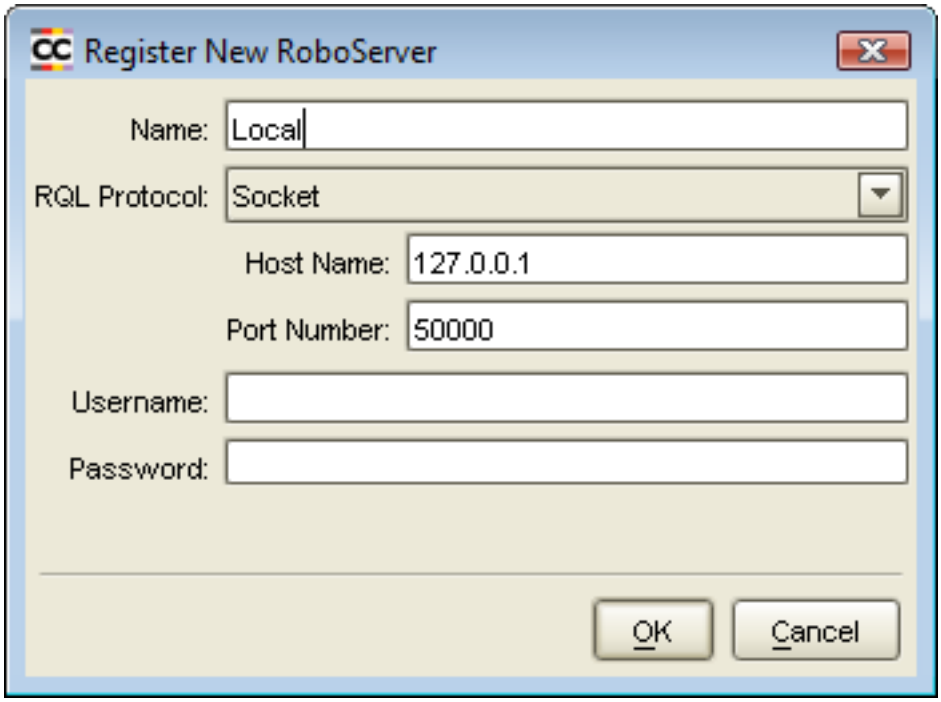
As shown in the popup menu example, by right clicking a RoboServer in the tree view, you can move the RoboServer by selecting "Move RoboServer Up" or "Move RoboServer Down". Note that the RoboServer is moved in the **RoboServers** category as well as in the **RoboServers** subcategory of the **Robots** category.

Managing RoboServers

This section explains how to manage RoboServers using the Control Center.

Registering RoboServers

A RoboServer is registered by clicking the icon, which brings up the RoboServer Registration Dialog shown below:



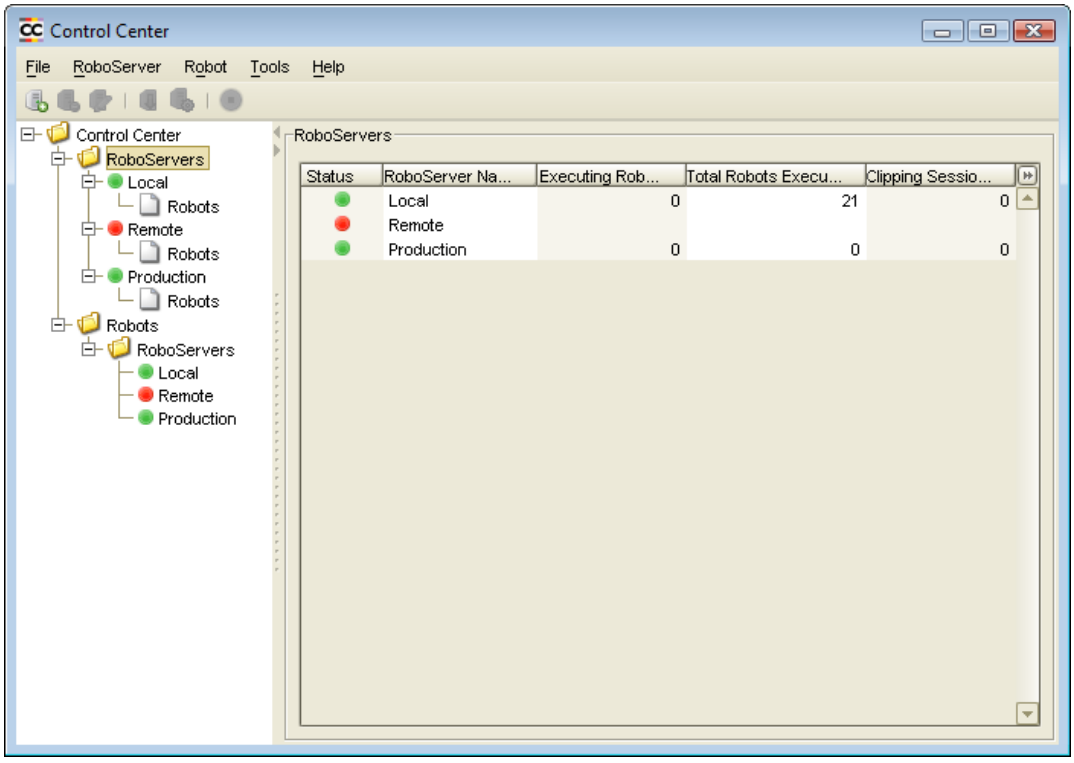
RoboServer Registration Dialog

Here, you give your RoboServer a name that will be shown in the Tree view. You also has to select and configure the protocol for the RoboServer that you want to register and add credentials for the RoboServer if needed. When you are done, press "OK" to register the RoboServer.

Note that you cannot register a RoboServer that is already registered.

Monitoring RoboServers

The registered RoboServers can be monitored by selecting the **RoboServers** category in the tree view. The main view will then show an overview of the registered RoboServers as shown below:



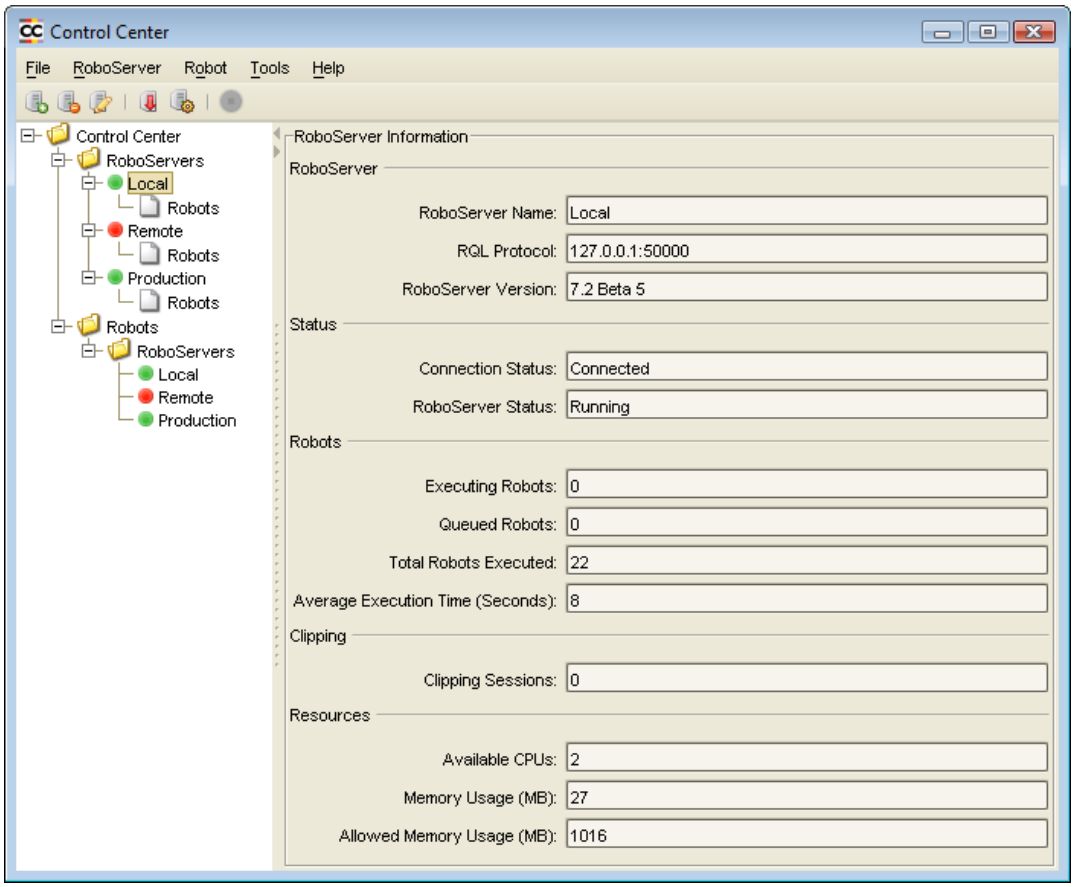
RoboServers Overview

The **status** of a RoboServer is shown as an icon in the tree view and in the table in the main view.

The status is one of the following:

- *Connected and running* — indicated by the icon .
- *Connected and shutting down* — indicated by the icon .
- *Unavailable* — indicated by the icon .
- *Unknown* — indicated by the icon .

If a RoboServer is selected in the tree view (or if you double click on the RoboServer in the table), the main view shows details about this particular RoboServer:



Viewing a RoboServer

In this view, you see information about the following:

RoboServer: The name and the protocol of the RoboServer.

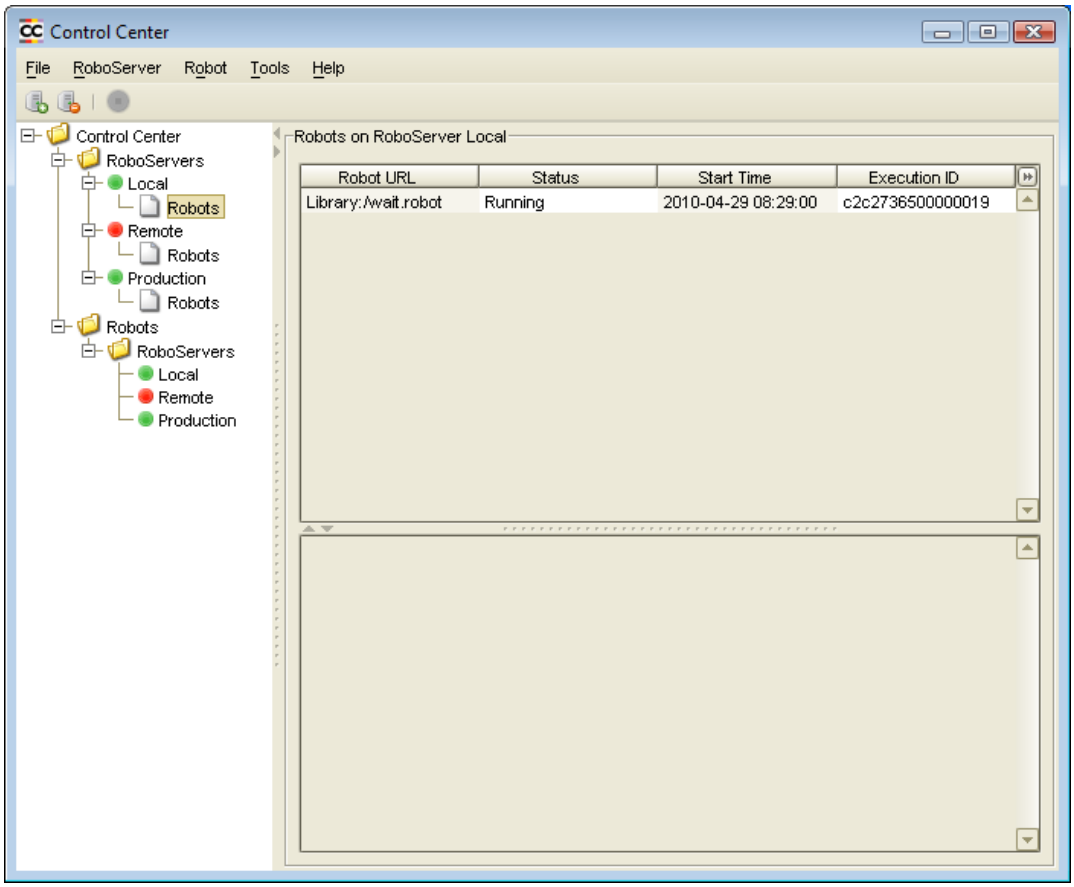
Status: The connection status (connected, unavailable or unknown) and, if connected, the RoboServer status (running or shutting down).

Robots: The number of currently executing robots, the number of queued robots, the total number of executed robots and the average execution time.

Clipping: The number of clipping sessions on the RoboServer.

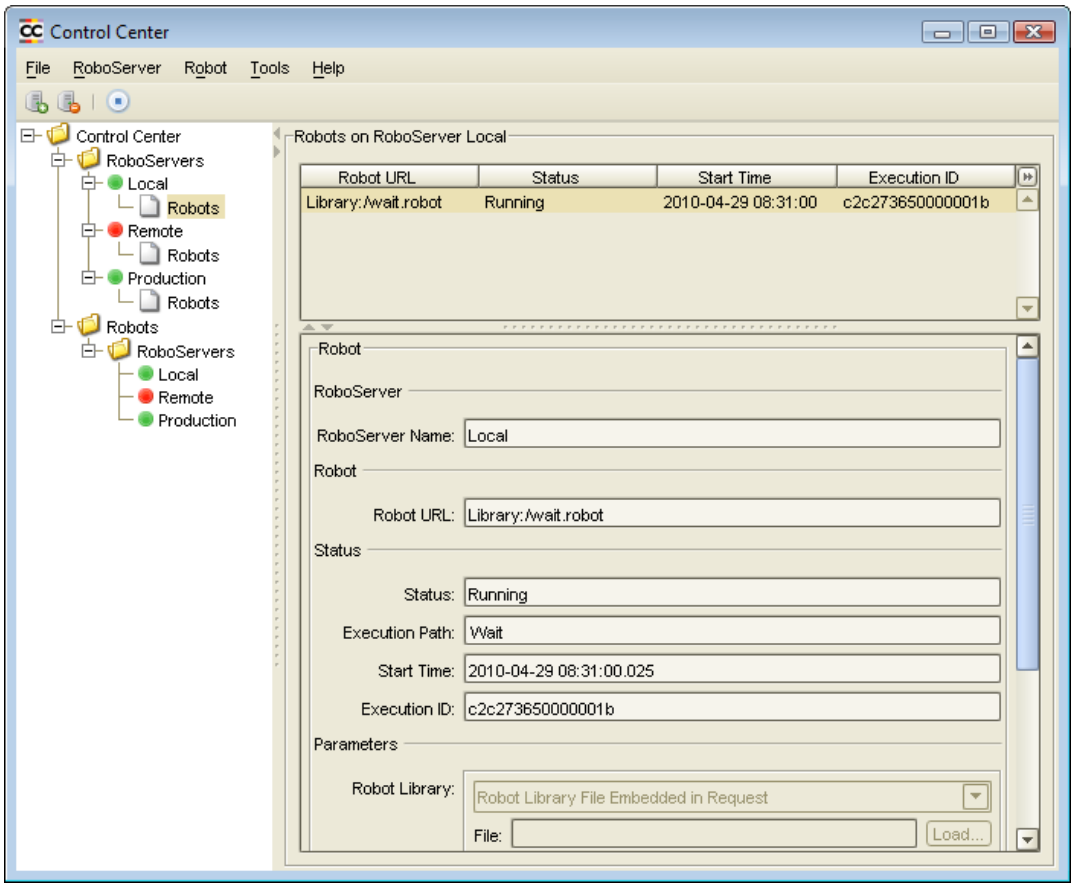
Resources: Information about available and used resources on the RoboServer.

If the **Robots** subcategory is selected, the main view shows an overview of the robots on this particular RoboServer:



Viewing the Robots of a RoboServer

If you select a robot in the table in the main view, details about this particular robot are shown below the table:



Viewing the Details of a Robot

In this view, you see information about the following:

RoboServer: The name of the RoboServer.

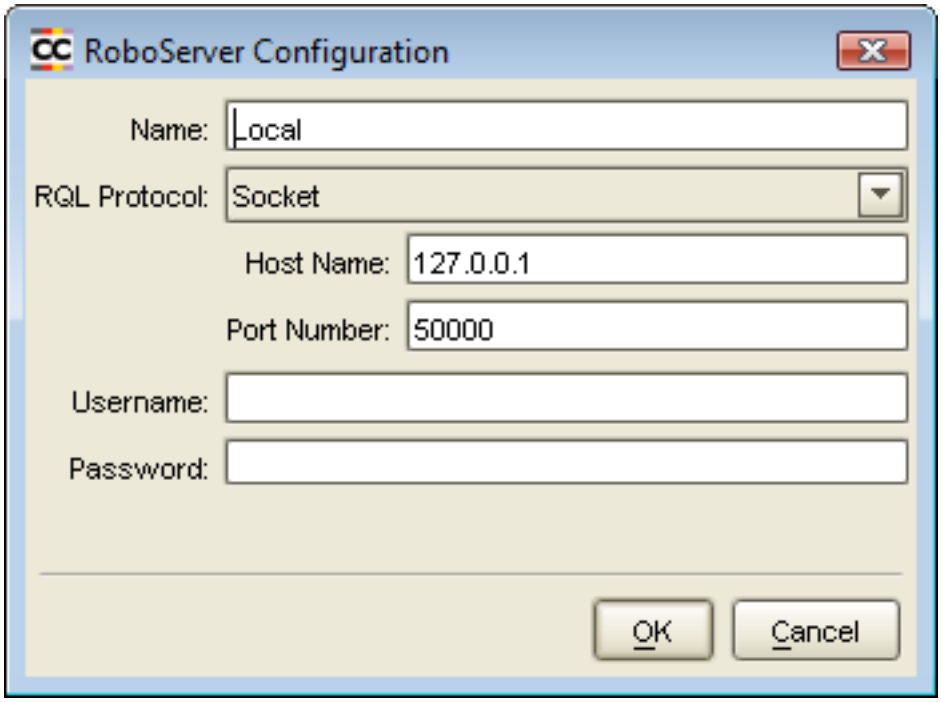
Robot: The Robot URL.

Status: The robot status (running or queued), the start time and the execution ID.

Parameters: The robot library and environments.

Editing RoboServers

You can edit the name and protocol of a RoboServer by selecting the RoboServer and clicking the icon. This brings up a dialog similar to the one shown below:

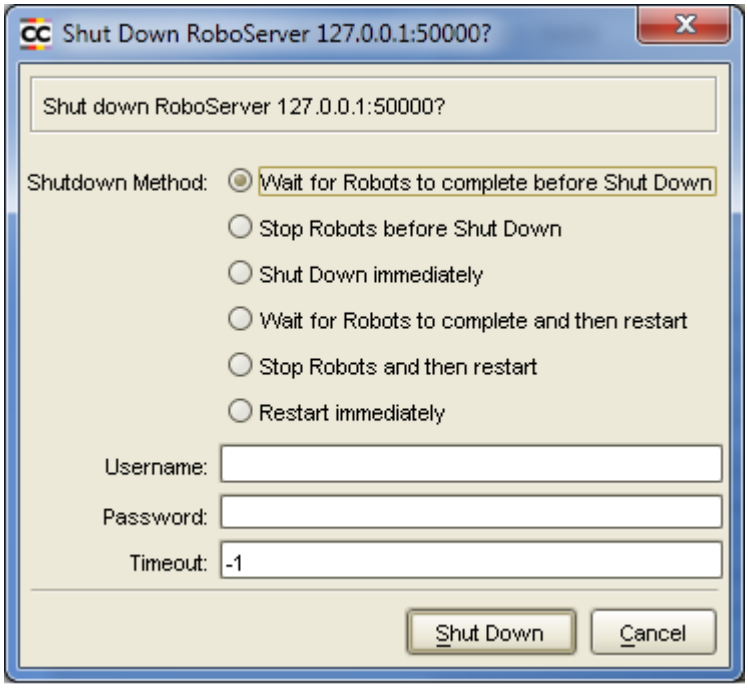


Editing a RoboServer

The name of a RoboServer can be anything you like, but you cannot change the protocol to one that is already used by another RoboServer registered in the Control Center.

Shutting Down RoboServers

A RoboServer is shut down by selecting a connected RoboServer and clicking the icon. This brings up a dialog similar to the one shown below:



Shutting Down a RoboServer

When shutting down a RoboServer, you have to choose a **shutdown method**. There are six shutdown methods:

Wait for Robots to Complete before Shut Down:	The RoboServer is not shut down until all robots (if any) on the RoboServer have completed.
Stop Robots before Shut Down:	All robots on the RoboServer are stopped immediately, and the RoboServer is subsequently shut down.
Shut Down immediately:	The RoboServer is shut down immediately without stopping the robots first.
Wait for Robots to complete and then restart:	The RoboServer is not restarted until all robots (if any) on the RoboServer have completed.
Stop Robots and then restart:	All robots on the RoboServer are stopped immediately, and the RoboServer is subsequently restarted.
Restart immediately:	The RoboServer is restarted immediately without stopping the robots first.

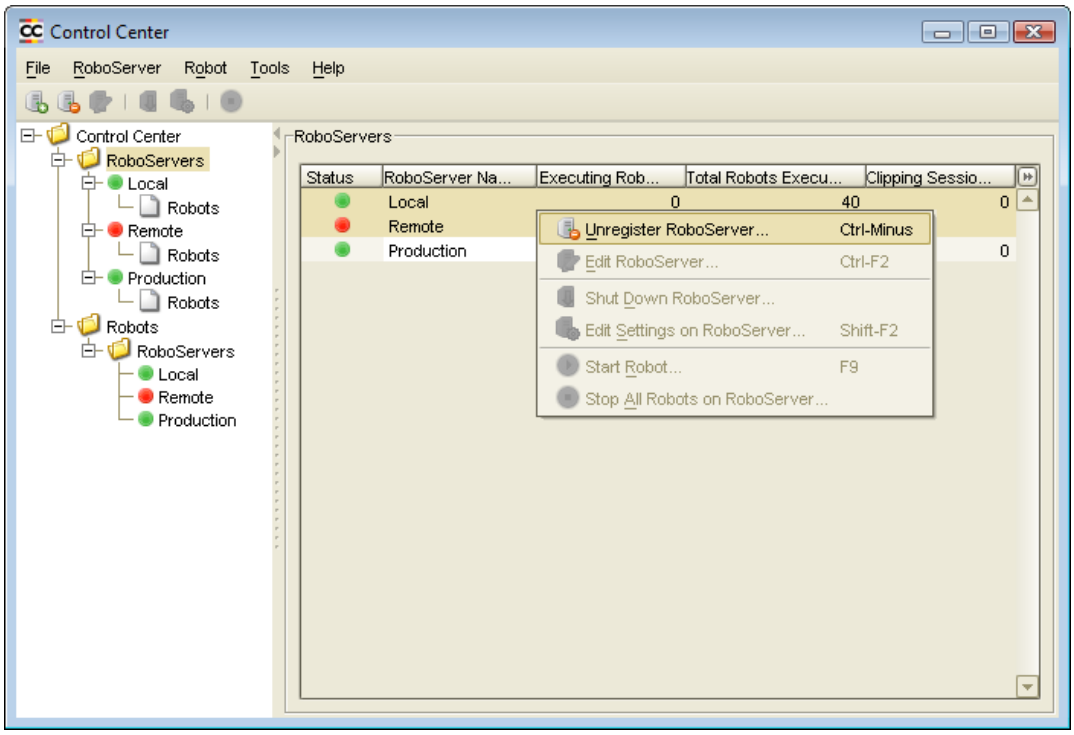
If the RoboServer requires authentication, you must enter the username and password.

Additionally, you can specify a timeout (in seconds). When this timeout expires, the server will shut down or restart (depending on your selection) regardless of whether all robots have been completed or not. Set the field value to -1 (the default) to disable the use of timeouts.

Unregistering RoboServers

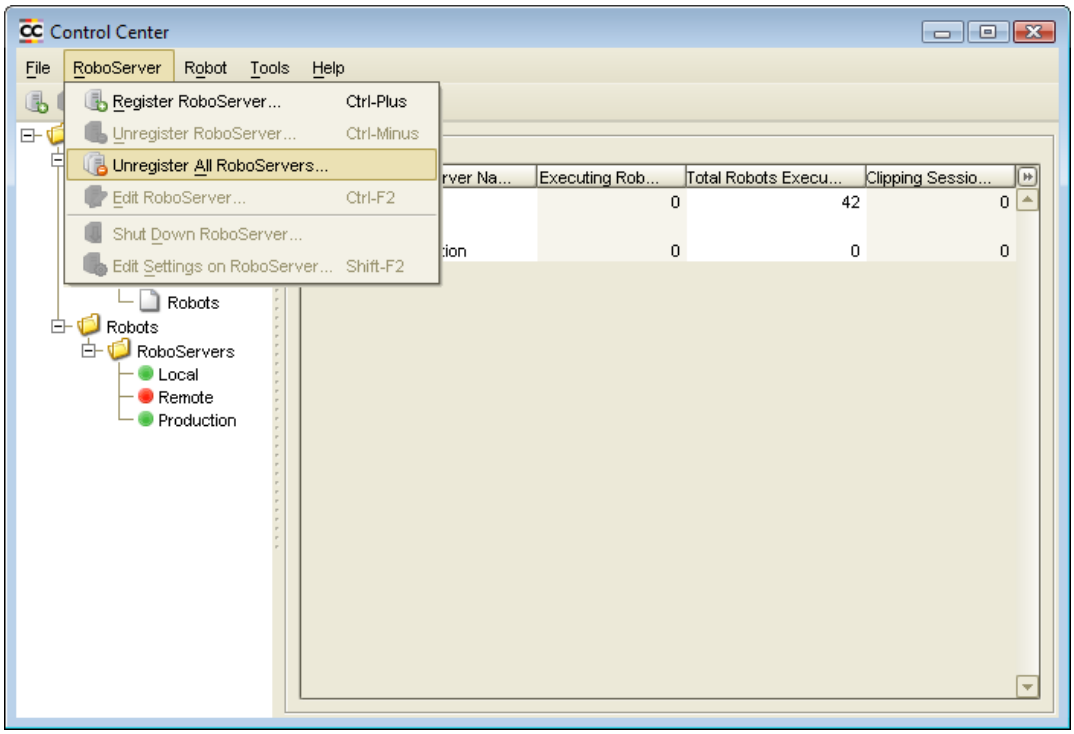
A RoboServer is unregistered by selecting the RoboServer and clicking the icon. This brings up a dialog asking you to confirm the unregistering.

You can shut down more than one RoboServer by selecting them in the table in the main view and either do as described above or right click the selection and choose "Unregister RoboServer..." as shown below:



Unregistering Multiple RoboServers

You can unregister *all* RoboServers by opening the RoboServer menu and selecting "Unregister All RoboServers..." as shown below:



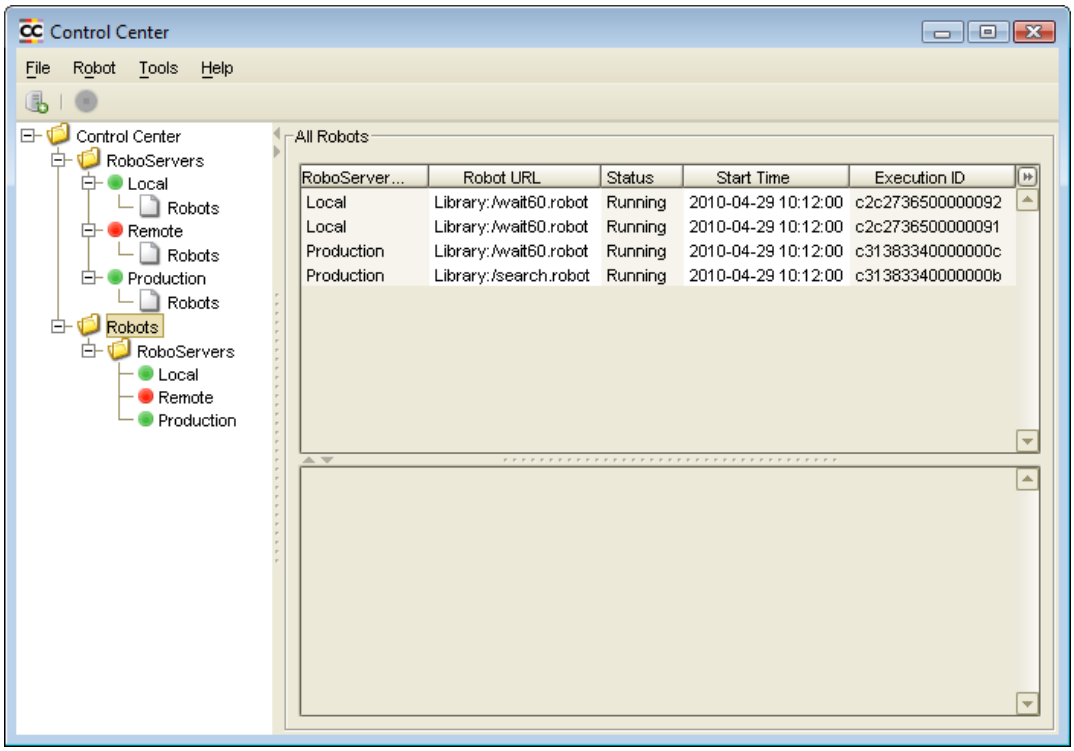
Unregistering All RoboServers

Managing Robots

This section explains how to manage robots using the Control Center.

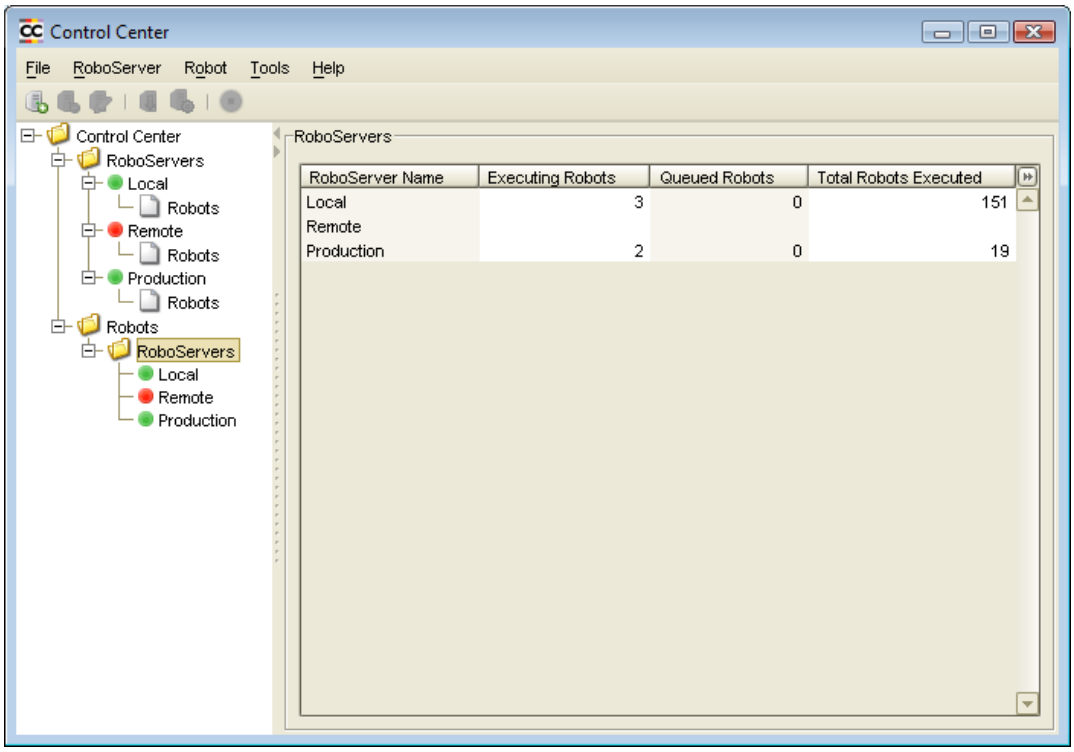
Monitoring Robots

The robots can be monitored by selecting the **Robots** category in the tree view. The main view will then show an overview of the robots on *all* registered RoboServers as shown below:



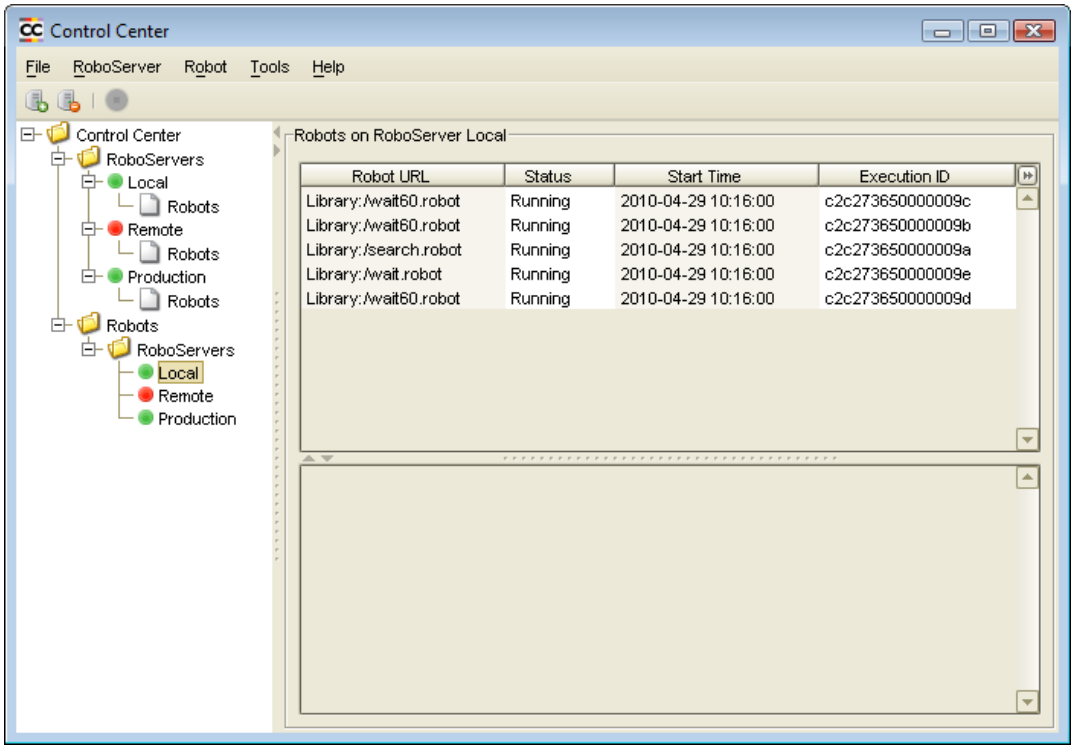
Monitoring All Robots

If the **RoboServers** subcategory is selected, the main view shows an overview of the robots on the individual RoboServers:



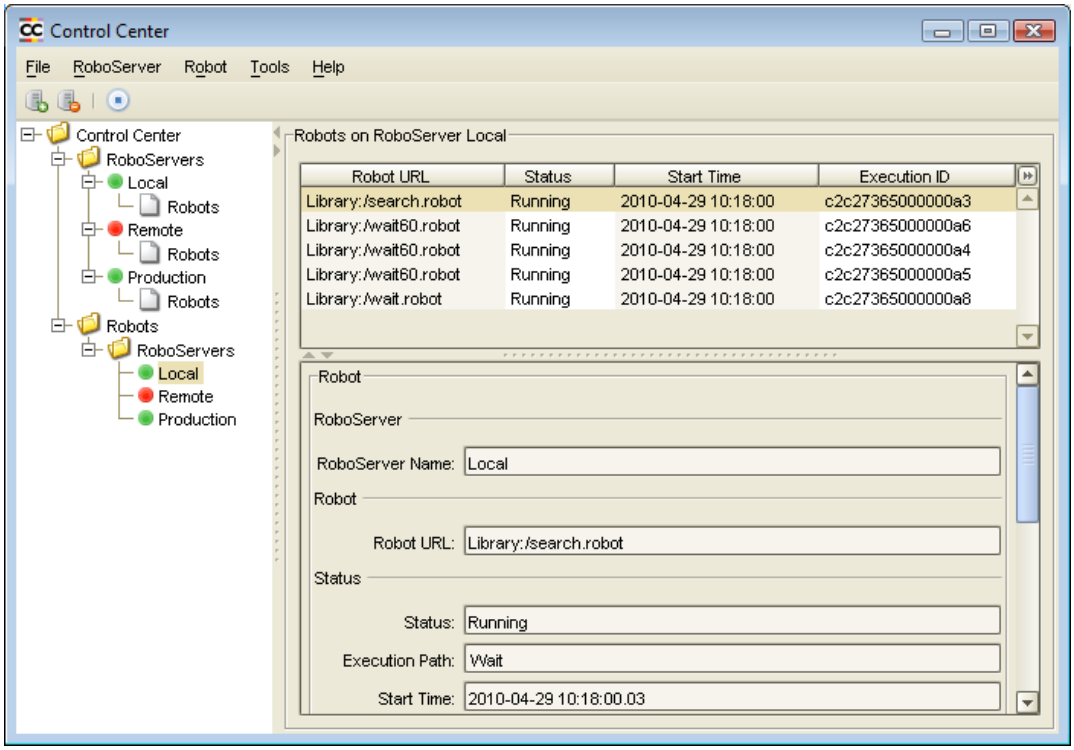
Monitoring Robots on RoboServers

If you select a single RoboServer in the tree view (or if you double click on the RoboServer in the table), the main view shows an overview of the robots on this particular RoboServer:



Monitoring Robots on a RoboServer

If you select a robot in the table in the main view, details about this particular robot are shown below the table:



Monitoring a Robot

Stopping Robots

Robots are stopped by selecting a robot and clicking the icon. This brings up a dialog asking you to confirm the stopping of the robot. Note that this should only be done if you do not want the robot to stop by itself.

You can stop more than one robot by selecting the robots in a table in the main view and do as described above, or by right clicking the selected robot and choosing "Stop Robot...".

You can stop all robots on a particular RoboServer by selecting a RoboServer and clicking the icon.

You can stop all robots on all RoboServers by selecting the **RoboServers** category, the **Robots** category, or the **RoboServers** subcategory and clicking the icon.

Other Control Center Features

This section explains how to use the features of the Control Center that are not directly related to managing RoboServers and robots.

Exporting and Importing the RoboServer List

Whenever you shut down the Control Center, the list of registered RoboServers is automatically saved, and when you start the Control Center again, the list is loaded again.

However, you can also export and import the RoboServer list manually. Once a RoboServer list is exported to a file, it can be imported on other Control Center installations. This allows you to avoid registering your RoboServers on each installation, and instead easily export and import the settings.

Exporting the RoboServer List

The current RoboServer list can be exported by opening the File menu and selecting "Export RoboServers...".

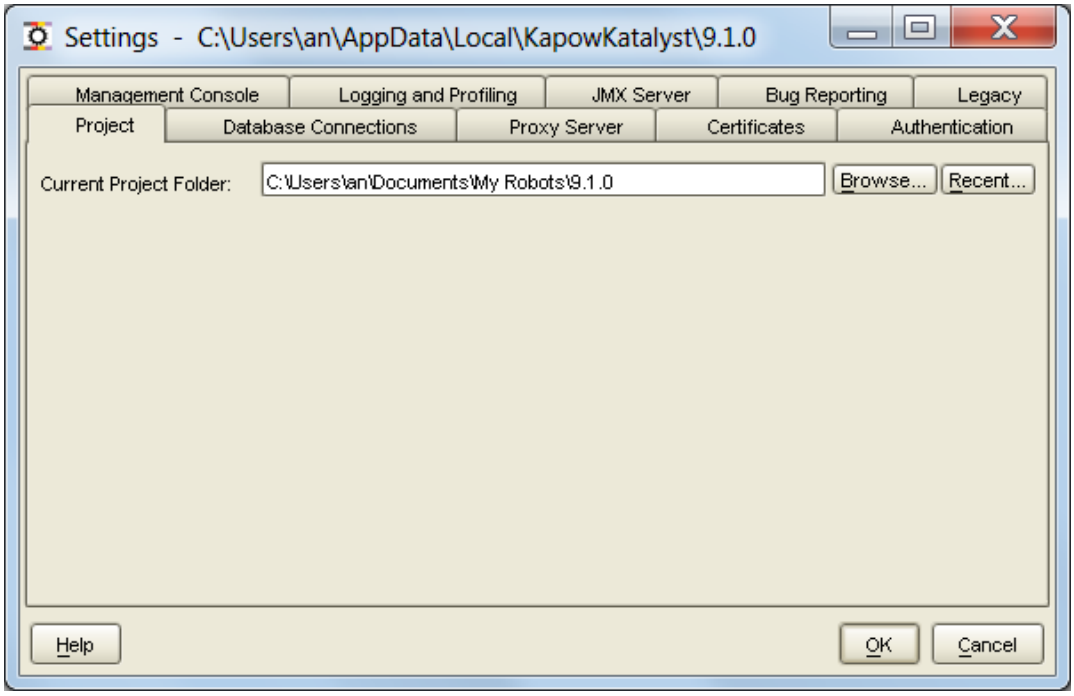
Importing a RoboServer List

A RoboServer list can be imported by opening the File menu and selecting "Import RoboServers...".

When you import a RoboServer list, you register all RoboServers in the list that are not already registered. RoboServers that use the same protocol as a RoboServer already registered (but possibly with a different name) are *not* registered again.

Editing Settings on a Remote RoboServer

From the Control Center, you can edit the settings of a connected RoboServer by clicking the icon. This brings up a dialog similar to the one shown below:



Editing Settings

After changing the settings, you must restart the RoboServer in order for the changes to take effect.

Management Console in Tomcat

By default, Management Console is run as an embedded component inside RoboServer, which makes for easy installation. As an alternative, it can be deployed as a regular web application on a standalone Tomcat 5.5, 6 or 7 web server. The following table lists the differences in feature set.

Table 147. Management Console features and configuration

Feature	Embedded	Standalone J2SE Web Container (Enterprise)
Authentication	Single user defined in Settings	Role based security by project. Roles are obtained through LDAP or other provider
Management Console data store	Embedded Derby database	Container managed Data Source (supported platforms)

Instructions on configuring an embedded Management Console can be found in the RoboServer Configuration section. An embedded Management Console is started as described in Starting the Management Console.

Detailed instructions on deploying Management Console in a Tomcat web container are found on the following pages.

Notes on Upgrading

If you have been running a previous version of Management Console this section will inform you how to upgrade to 9.2.0

Upgrading involves the following steps

- Create a backup of your data from the previous version of Management Console
- Install Management Console
- Update the Management Console configuration
- Restore the backup

Create a backup

Check the online documentation of your previous version of Management Console to learn how to create a backup. 9.1 [http://help.kapowtech.com/9.1/topic/doc/sc/OptionsTab.html] 9.0 [http://help.kapowtech.com/9.0/topic/doc/sc/OptionsTab.html] 8.2 [http://help.kapowtech.com/8.2/topic/doc/sc/OptionsTab.html] 8.1 [http://help.kapowtech.com/8.1/topic/doc/sc/OptionsTab.html] 8.0 [http://help.kapowtech.com/8.0/topic/doc/sc/OptionsTab.html] 7.2 [http://help.kapowtech.com/7.2/topic/doc/sc/OptionsTab.html] 7.1 [http://help.kapowtech.com/7.1/topic/doc/sc/OptionsTab.html] 7.0 [http://help.kapowtech.com/7.0/topic/doc/sc/OptionsTab.html]

Install Management Console

Follow the next steps in this guide to learn how to install the Management Console. If you are upgrading from 8.1 or later you can reuse your old `Configuration.xml`, see next section.

Upgrade Management Console configuration

If you are upgrading from version 8.0 or earlier, you will have to re-configure your new Management Console from scratch. Prior to version 8.1 some configuration was done in `web.xml` this is no longer the case. You may no longer copy or modify the `web.xml`.

Most configuration is done in `Configuration.xml`. This file may be copied from previous versions. Copy the `Configuration.xml` from the previous version into `WEB-INF/` overriding the version already there. Once you start Management Console it will create a new `Configuration.xml` based on your old version, you must then copy the upgraded version into `WEB-INF/` overriding your old configuration. Instruction should display when you access Management Console web interface.

As of version 8.2 , LDAP integration is no longer container managed. If you used LDAP for authentication in a previous version, you should remove the `Realm` definition in the `ManagementConsole.xml` inside Tomcat's `conf/Catalina/localhost/`. LDAP integration is now configured in `WEB-INF/login.xml`, as described in LDAP Integration

Restore the backup

Check the Management Console user's guide for details on how to restore a backup.

Deploying into Tomcat

We will now detail how to install the Management Console on a stand alone J2SE web container. For this guide we have chosen Tomcat, the distribution has been tested on Tomcat 5.5, 6.0 and 7.0. Your J2SE web container *must* be using the Java 1.6 runtime or newer.

Configuring ManagementConsole.war

The Management Console application comes in the form of a Web Application Archive (a WAR file) named `ManagementConsole.war` and is located inside the `/WebApps` folder in the Kapow Katalyst installation folder.

The version of ManagementConsole.war that ships with Kapow Katalyst is configured to run embedded inside RoboServer, before you can deploy it as a standalone application on Tomcat it must be reconfigured to fit your environment.

A WAR-file is compressed using zip, in order to access the configuration files you will have to explode (unpack) it. Once the configuration files are updated you re-zip and deploy ManagementConsole.war to you Tomcat server.

The table below contains a list of the configuration files relative to the root of the un-zipped ManagementConsole.war

Table 148. Configuration files

File	Configures	Notes
WEB-INF/Configuration.xml	Clustering, password encryption, REST-Plugin	If you copy the version of the file from 8.1, it will automatically be upgraded once you start the Management Console

File	Configures	Notes
WEB-INF/login.xml	Administrators and users, this is where you integrate with LDAP	
WEB-INF/roles.xml	Permissions for various application roles	
WEB-INF/classes/log4j.properties	application logging	

Spring configuration files

`Configuration.xml`, `login.xml` and `roles.xml` are all Spring configuration files (www.springsource.org [http://www.springsource.org/]) and share the same general syntax which we will outline here.

Spring is configured through a series of beans, each bean has a number of properties which configures a piece of code inside the application. The general syntax is:

```
<bean id="id" class="SomeClass">
    <property name="myName" value="myValue"/>
</bean>
```

Table 149. Part

part	Configures	
id="id"	The id of the bean is an internal handle, that the application use to refer to the bean. It is also referred to as the beans name	
`class="SomeClass"`	The class identifies the code component which the bean configures.	
`<property name="myName" value="myValue"/>`	Defines a property with the name `myName` and the value `myValue`. This configures a property on the code component defined by the class attribute	

The only part of spring beans that you may need to change is the value attribute of the `<property name="myName" value="myValue"/>`, for example if you want to change the admin user defined in `login.xml`;

```
 <bean class="com.kapowtech.mc.config.HardCodedUser">
    <property name="userName" value="admin"/>
    <property name="password" value="admin"/>
    <property name="groups" value="ADMINISTRATOR"/>
</bean>
```

You can change the values of the userName, password, and groups properties to define a user with a different userName, password, and group association.

Sometimes the property value of a bean is a list of nested beans, like the list of hard coded users in `login.xml`, which is defined like this

```
<bean id="hardCoded" class="com.kapowtech.mc.config.HardCodedLogin" lazy-init=
```

```
        <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>
```

In case you need more hard coded users, you can add them by copying an existing `HardCodedUser` bean in the list and modifying the userName, password and groups property values. Notice that nested beans often don't have any id. Lists may also contain simple values like the `adminGroups` bean in `login.xml`

```
    <bean id="adminGroups" class="com.kapowtech.mc.config.AdminGroups">
        <property name="adminGroups">
            <list>
                <value>ADMINISTRATOR</value>
            </list>
        </property>
    </bean>
```

This defines a simple list where each value is just a text string.

Troubleshooting

If you have any problems during the installation, you should check the Tomcat log in the `/logs` folder in your Tomcat installation. During the configuration process it is often easier to run Tomcat from the command line, as it will then print error messages directly in the command line window.

Creating a new Database

We strongly recommend that you create a new database for the tables used by the Management Console. There are two requirements to the database.

- Unicode support
- Case-sensitive comparison

Unicode support is needed because non-ASCII characters, like Danish Æ, German ß or Cyrillic # may be given as input to robots. This input is stored in the database, and without Unicode support these characters may be stored incorrectly.

Case-sensitive comparison is needed because it is possible to upload a robot named `a.robot` and another named `A.robot`. Without case-sensitive comparison uploading the latter would override the first.

Database servers handles Unicode and case-sensitive comparison very differently. The following list contains recommendations for the supported database systems.

Table 150. Recommendations for Unicode support and case-sensitive comparison

Database	Recommendations
IBM DB2	Create the database using `CODESET UTF-8`
MySQL 5.x	Create the database with `utf8_bin` collation: `CREATE DATABASE KAPOW_MC COLLATE utf8_bin`
Oracle	We use `NVARCHAR2`, `NCLOB` types for Unicode. For case-sensitive comparison ensure that `NLS_COMP` is set to `BINARY`
Microsoft SQL Server 2005/ 2008	We use `NVARCAR`, `NTEXT` types for Unicode. For case-sensitive comparison create the database with a Case-Sensitive collation like `SQL_Latin1_General_CP1_CS_AS`: `CREATE DATABASE KAPOW_MC COLLATE SQL_Latin1_General_CP1_CS_AS`
Sybase Adaptive Server Enterprise	We use `NVARCAR`, `NTEXT` types for Unicode. For case-sensitive comparison ensure that the sort order is binary (see `sp_helpsort`). The scripts below use "NVARCHAR(255) NOT NULL UNIQUE", due to index limitations this will not work on an ASE with 2K page size, unless you modify the scripts to use NVARCHAR(200) NOT NULL UNIQUE (or install on a 4/8/16K page server)

The tables used by the Management Console can be grouped into 3 categories: Platform tables, logging tables, and data view tables. The platform tables hold information exclusive to the Management Console such as the uploaded robots and their scheduling information, while the logging and data view tables are shared with RoboServer.

User privileges

When the Management Console starts, it will automatically try to create the required platform tables and logging tables (if not already created by RoboServer). This means that the user account used to access the database must have the CREATE TABLE and ALTER TABLE privilege. Oracle users also needs the CREATE SEQUENCE privilege. If this is not possible you can have your Database administrator create the tables using the scripts below.

Additionally the user must be allowed to SELECT, INSERT, UPDATE, DELETE for the system to work properly.

SQL scripts for Management Console tables

Table 151. SQL scripts for Management Console tables (right-click and choose "Save As")

Database	Script to Create Tables	Script to Drop Tables
IBM DB2	create [sql/platform/db2_create.sql]	drop [sql/platform/db2_drop.sql]
MySQL 5.x	create [sql/platform/mysql_create.sql]	drop [sql/platform/mysql_drop.sql]
Oracle		

Database	Script to Create Tables	Script to Drop Tables
	create [sql/platform/oracle_create.sql]	drop [sql/platform/oracle_drop.sql]
Microsoft SQL Server 2005/2008	create [sql/platform/sqlserver_create.sql]	drop [sql/platform/sqlserver_drop.sql]
Sybase Adaptive server	create [sql/platform/sybase_create.sql]	drop [sql/platform/sybase_drop.sql]

The Management Console uses a 3rd party scheduling component called Quartz. Quartz also requires a number of tables which must reside among the other platform tables. These tables are also created automatically when the Management Console starts, or may be created manually using the scripts below.

SQL scripts for Quartz tables

Table 152. SQL scripts for Quartz tables (right-click and choose "Save As")

Database	Script to Create Tables	Script to Drop Tables
IBM DB2	create [sql/quartz/quartz_db2_create.sql]	drop [sql/quartz/quartz_db2_drop.sql]
MySQL 5.x	create [sql/quartz/quartz_mysql_create.sql]	drop [sql/quartz/quartz_mysql_drop.sql]
Oracle	create [sql/quartz/quartz_oracle_create.sql]	drop [sql/quartz/quartz_oracle_drop.sql]
Microsoft SQL Server 2005/2008	create [sql/quartz/quartz_sqlserver_create.sql]	drop [sql/quartz/quartz_sqlserver_drop.sql]
Sybase Adaptive server	create [sql/quartz/quartz_sybase_create.sql]	drop [sql/quartz/quartz_sybase_drop.sql]

Creating a Tomcat context file

In an enterprise environment databases are often accessed through a data source. This guide will show you how to configure your Tomcat with a data source that connects to a local MySQL database server.

In Tomcat data sources are defined within the applications context. The context may be declared either embedded or external to the application. When the context is embedded it is defined in the file `context.xml`, which must be located inside the WAR file inside the `META-INF` folder. When declared externally the file must be located in Tomcat's `/conf/Catalina/localhost` folder and the name of the file must be ManagementConsole.xml (same name as the deployed WAR file). Although Tomcat recommends deploying with an embedded context, as it provides a single deployment unit, we will use an external context definition in this guide, as it makes modifying the file easier. Once you have refined your configuration, you can embed the context file and deploy the War file to your production environment.

Adding platform data source

Create the file `ManagementConsole.xml` inside Tomcat's`/conf/Catalina/localhost` folder and add the following content:

```
<Context>
    <!-- Default set of monitored resources -->
```

```
        <WatchedResource>WEB-INF/web.xml</WatchedResource>

        <Resource name="jdbc/kapow/platform" auth="Container" type="javax.sql.Data
               maxActive="100" maxIdle="30" maxWait="-1"
               validationQuery="/* ping */" testOnBorrow="true"
               username="MyUser" password="MyPassword" driverClassName="com.mysql.
               url="jdbc:mysql://localhost:3306/KAPOW_MC?useUnicode=yes&amp;charac

    </Context>
```

The `url` parameter above is a JDBC URL. The `username` and `password` attributes are used by Tomcat to create a connection pool used when connecting to the database.

The data sources are defined differently for other databases. For instance, if you are using Microsoft SQL Server 2005/2008, the relevant three lines above should instead be:

```
username="MyUser" password="MyPassword"
driverClassName="com.microsoft.sqlserver.jdbc.SQLServerDriv
validationQuery="SELECT 1" testOnBorrow="true"
url="jdbc:sqlserver://localhost:1433;DatabaseName=MyDbName"
```

Note that if you are using Microsoft SQL Server, you need to configure it to use *mixed mode* authentication. Generally, you should consult the JDBC documentation to identify which values to use in the data sources.

The URL `jdbc:mysql://localhost:3306/KAPOW_MC?useUnicode=yes&characterEncoding=UTF-8` refers to a database named `KAPOW_MC` in our local MySQL. For MySQL it is recommended that you add `?useUnicode=yes&characterEncoding=UTF-8` to all connection strings, otherwise the JDBC driver will not handle Chinese, Japanese or other 3-byte utf-8 characters correctly, since we can't have & directly inside the context xml file, we must encode it as &

The `driverClassName` parameter controls which JDBC driver is used; each database vendor provides a JDBC driver for their database, which you will have to download. The JDBC driver, typically a single .jar file, must be copied into the /lib folder on Tomcat 6/7, or commons/lib on Tomcat 5.5.

The `validationQuery` is used by Tomcat to verify that the connection obtained from the connection pool is still valid (as the database server may have closed the connection). The validation query is lightweight and uses very few resources on the database server, this list contains validation queries for the supported databases.

Table 153. Validation queries

Database	Query
MySQL	/* ping */
Microsoft SQL Server 2005/2008	SELECT 1
Sybase Adaptive Server Enterprise	SELECT 1
IBM DB2	VALUES(1)
Oracle	SELECT 1 FROM DUAL

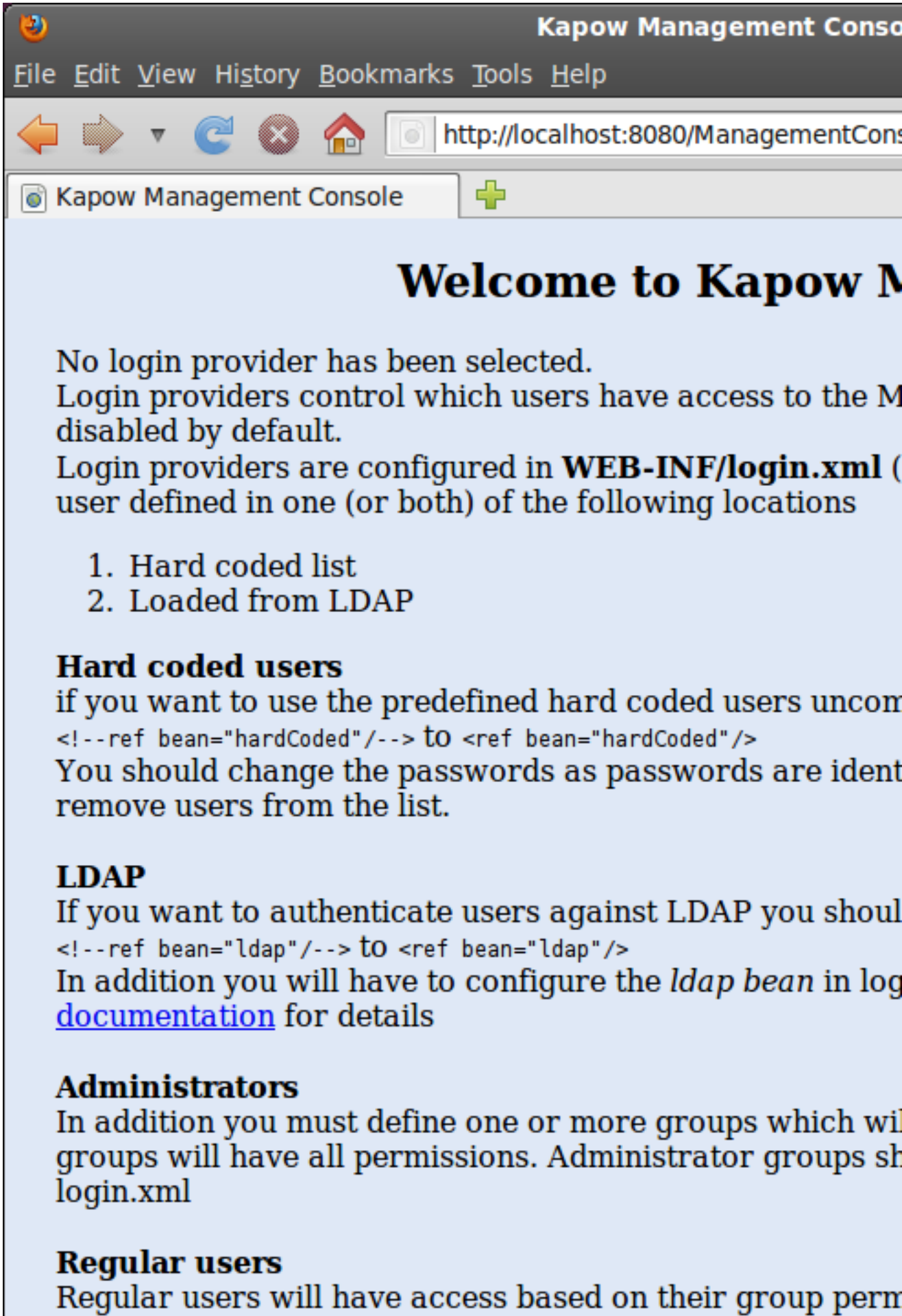
Note that the MySQL JDBC driver supports a special lightweight `/* ping */` 'request', check JConnector manual [http://downloads.mysql.com/docs/connector-j-en.a4.pdf] section 6.1 for details

For more information on context configuration and data sources, see JNDI Resources HOW-TO [http://tomcat.apache.org/tomcat-6.0-doc/jndi-resources-howto.html] and JNDI data source HOW-TO [http://tomcat.apache.org/tomcat-6.0-doc/jndi-datasource-examples-howto.html].

We are now ready to start the Tomcat server

Starting Tomcat

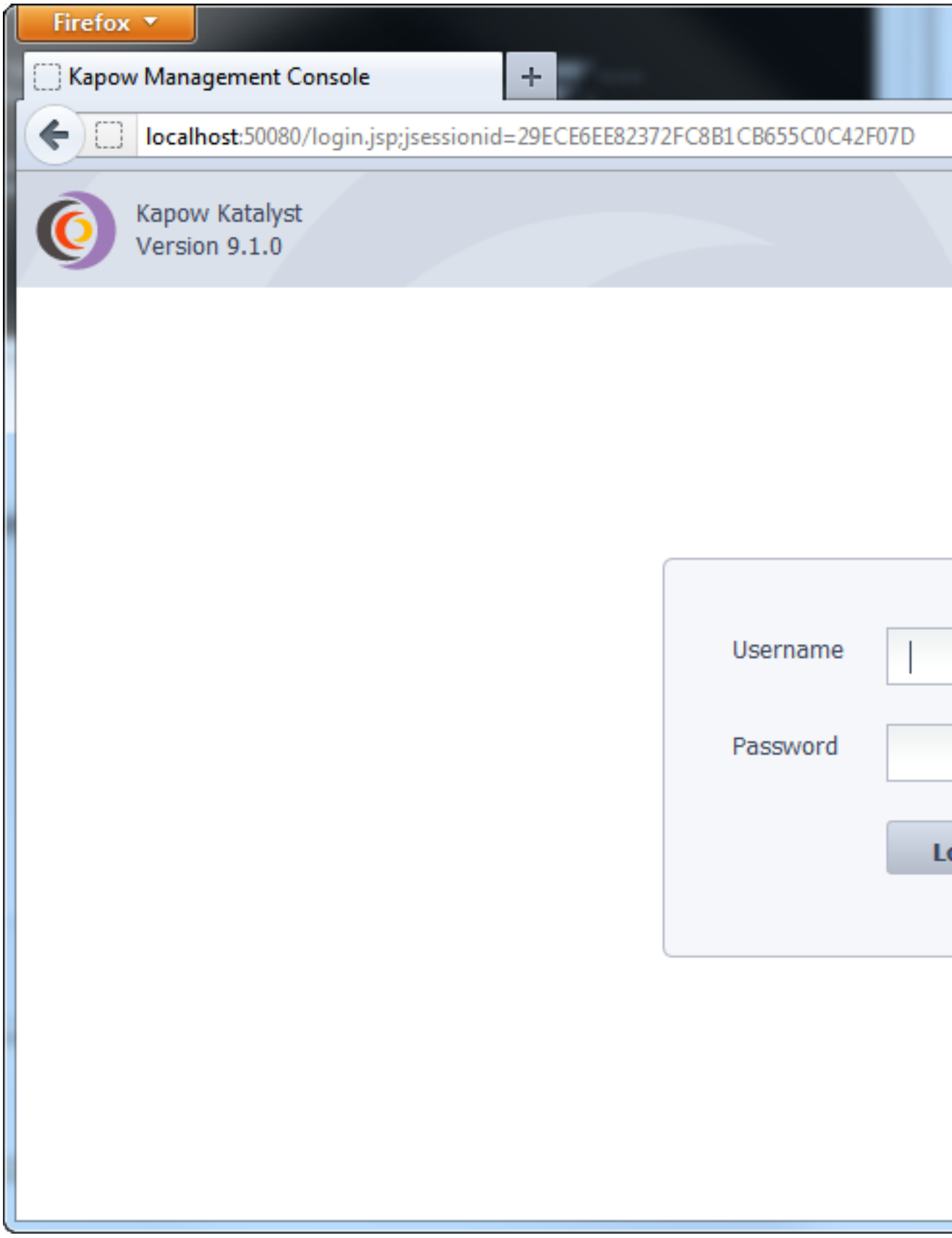
Start your Tomcat server, wait a couple of seconds for the application to be deployed, then navigate to http://localhost:8080/ManagementConsole [http://localhost:8080/ManagementConsole]. You should see the following page.


Kapow Management Conso
File Edit View History Bookmarks Tools Help
http://localhost:8080/ManagementCons
Kapow Management Console
Welcome to Kapow M
No login provider has been selected.
Login providers control which users have access to the M
disabled by default.
Login providers are configured in WEB-INF/login.xml (
user defined in one (or both) of the following locations
1. Hard coded list
2. Loaded from LDAP
Hard coded users
if you want to use the predefined hard coded users uncom
<!--ref bean="hardCoded"/--> to <ref bean="hardCoded"/>
You should change the passwords as passwords are ident
remove users from the list.
LDAP
If you want to authenticate users against LDAP you shoul
<!--ref bean="ldap"/--> to <ref bean="ldap"/>
In addition you will have to configure the ldap bean in log
documentation for details
Administrators
In addition you must define one or more groups which wi
groups will have all permissions. Administrator groups sh
login.xml
Regular users
Regular users will have access based on their group perm

No Providers

For this guide, we will use the predefined hard coded users. Go to `WEB-INF/login.xml` and uncomment `<!--ref bean="hardCoded"/-->` Then restart Tomcat

After restart, press F5 to refresh you browser, you should no be redirected the http://localhost:8080/ManagementConsole [http://localhost:8080/ManagementConsole] you should see the following login screen.

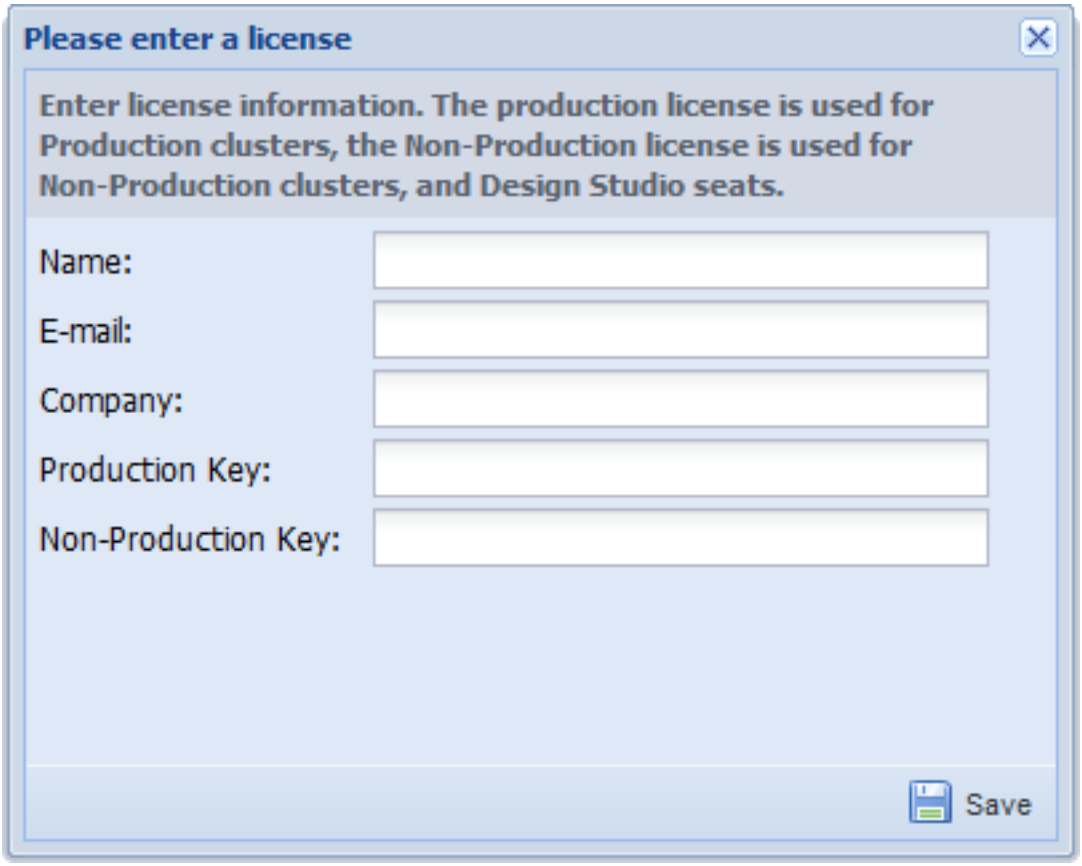

Firefox
Kapow Management Console
localhost:50080/login.jsp;jsessionid=29ECE6EE82372FC8B1CB655C0C42F07D
Kapow Katalyst
Version 9.1.0
Username
Password

Login page

Enter "admin" as username and password and click the login button.

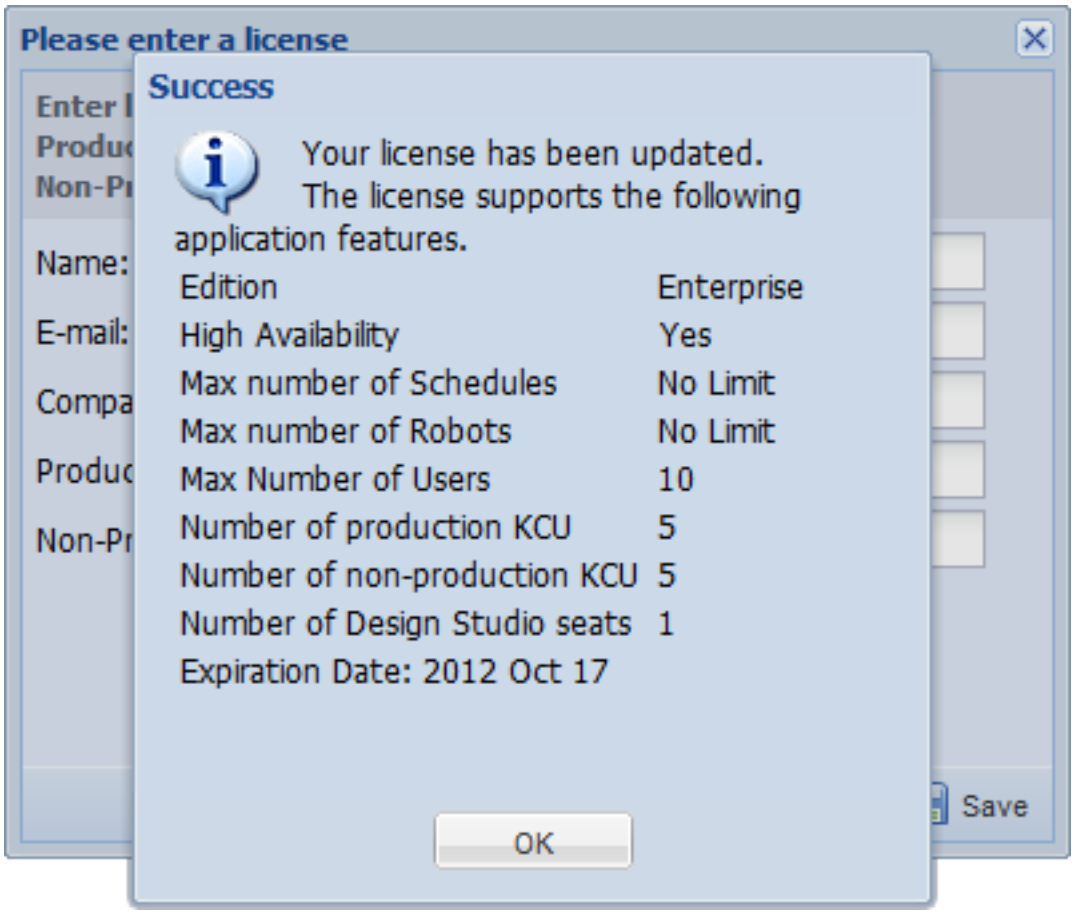
Enter License

After logging in, the following dialog is displayed:



No License

Now enter the Kapow Katalyst license information and click the Save button. You should see the following popup displaying which features your license key enables.



License updated

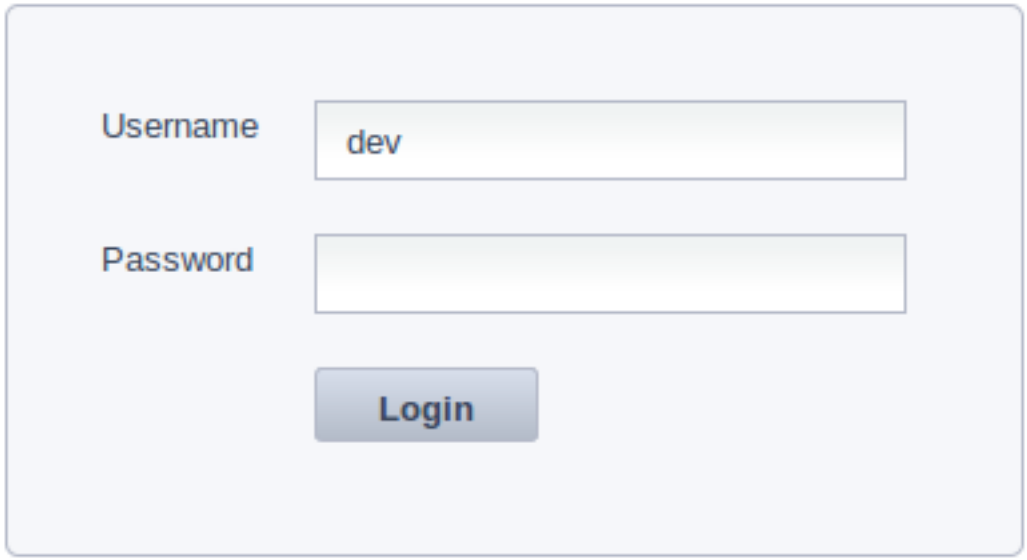
Project Permissions

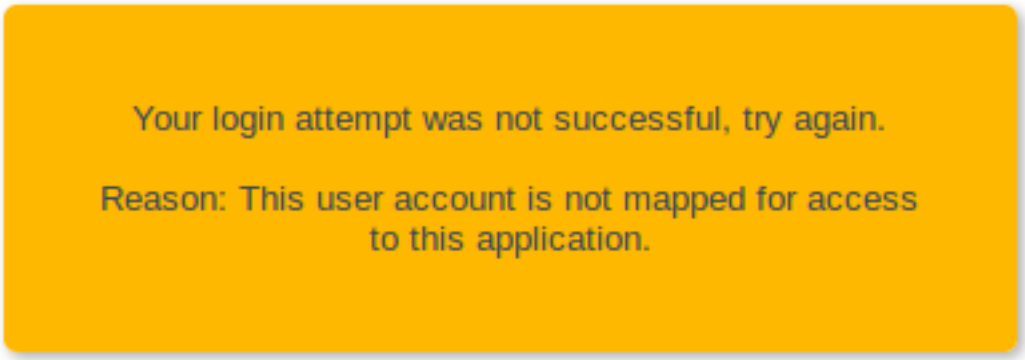
The admin user we logged in with bypasses the normal project permission applied to regular users. The admin user is a member of the group named ADMINISTRATOR. This group name is listed as one of the values in the list of `adminGroups`.

```
<bean id="adminGroups" class="com.kapowtech.mc.config.AdminGroups">
    <property name="adminGroups">
        <list>
            <value>ADMINISTRATOR</value>
        </list>
    </property>
</bean>
```

Any group listed here is considered an administrator group, and members of these groups will always be able to login, and have permission to perform all actions. They are the only users who can export and restore backups, and change the license keys.

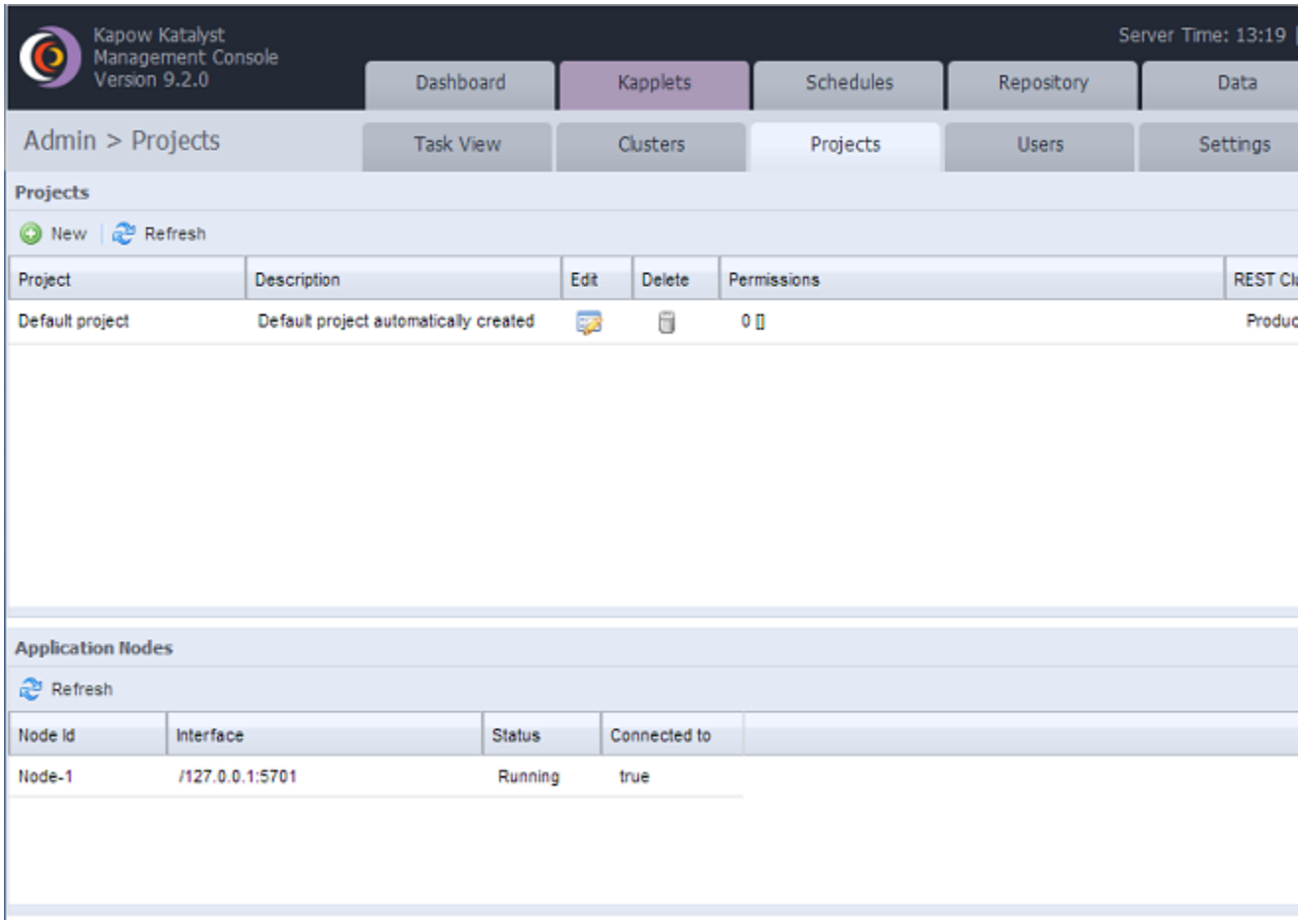
If you try to login with the dev user (password dev), you should see the following page





User not mapped
For non-administrator users, permissions is based on the user's group membership. The dev user is a member of the group DEVELOPER, and since we have not given users in the DEVELOPER group access to any projects, they are not able to log in.

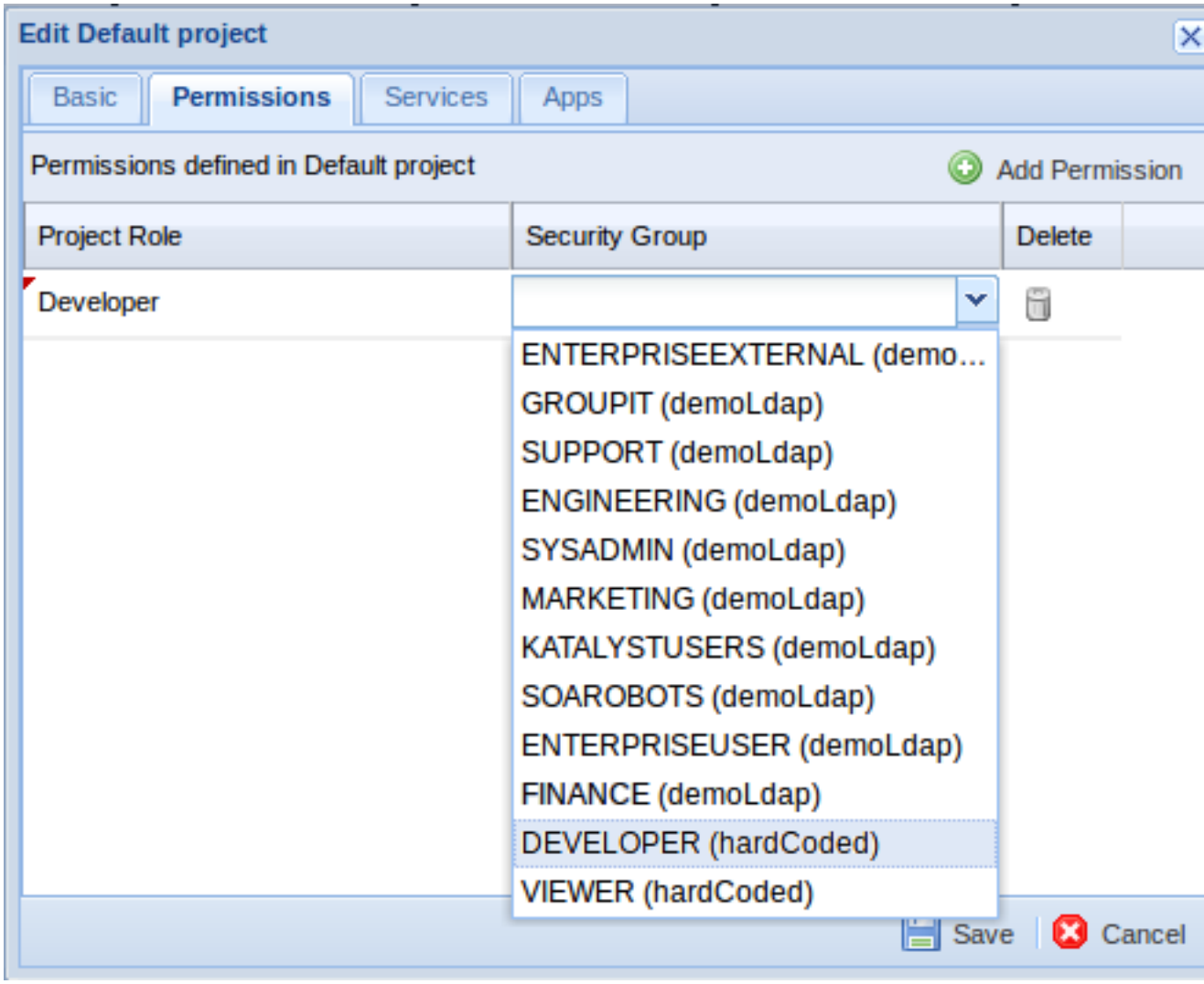
To allow the dev user to log in, we must log in as administrator and go to the Projects tab which is a sub tab of the Admin tab. It looks like this:



Initial project permissions
As we can see in the Permission column, there are 0 permission mappings.

We now click edit to open the Edit Project dialog and go to the permissions tab. Here we see a grid with columns Project Role, and Security Group. A project role determines a set of actions that may be performed inside the Management Console, such as uploading robots, creating schedules, viewing logs etc. (more details on this later). Within a project you assign a project role to a security group. That way, all users of the selected security group will be able to perform the actions allowed by the assigned project role.

Click the "Add Permission" button to add permissions in this project. This adds a new line to the grid, and inserts a dropdown allowing us to select a project role, select the project role *developer*. Now double click in the Security Group column, get the the dropdown allowing you to select the DEVELOPER security group (of which our dev user is a member). It should look like this:



Adding project permissions
Now click Save.

All members of the DEVELOPER group can now perform the actions allowed by the role developer. All roles are defined in the `WEB-INF/roles.xml` file. The roles defined in roles.xml defines low level permission for each tab. Here is an example Developer role:

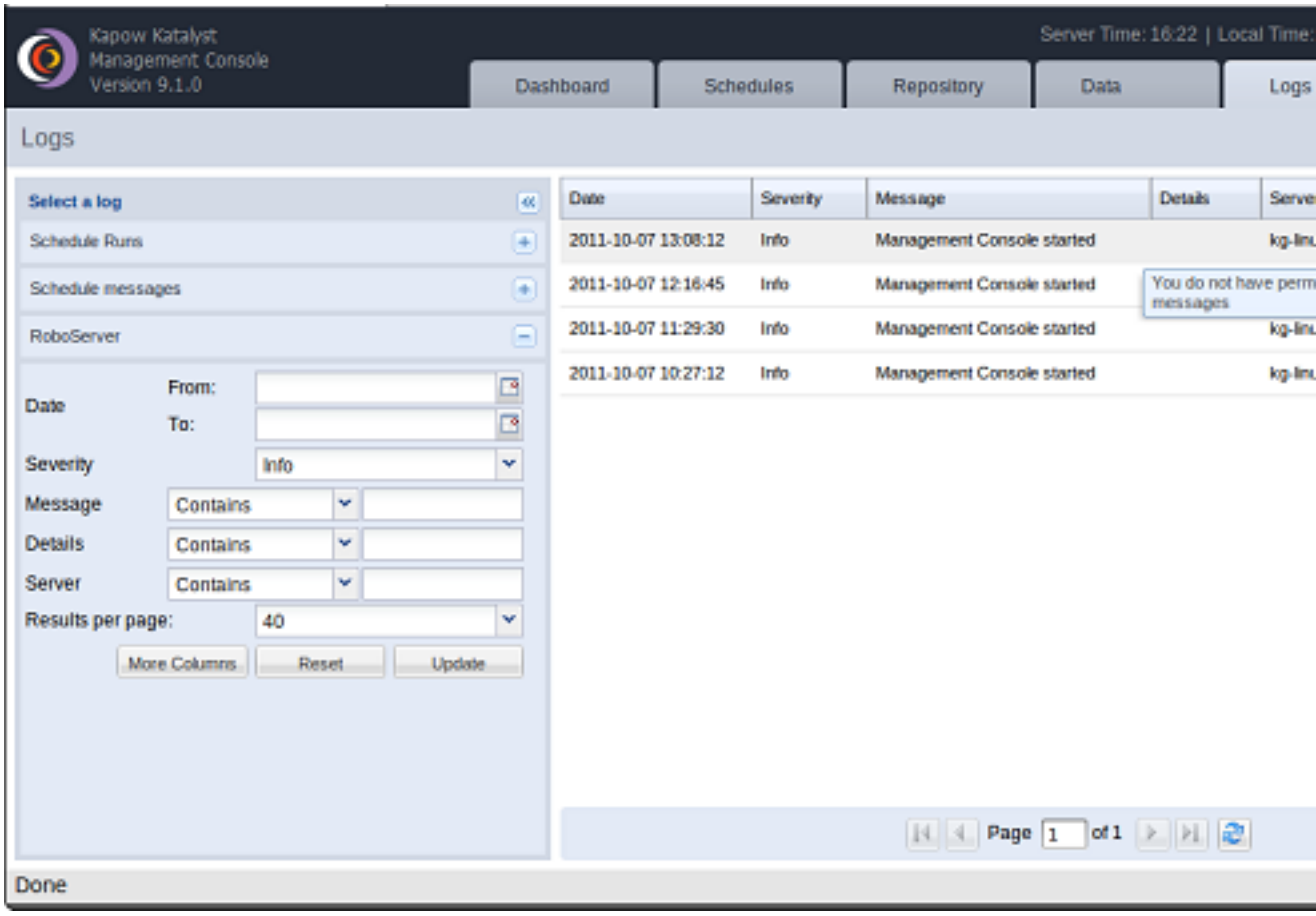
```
<bean class="com.kapowtech.scheduler.client.auth.Role">
    <property name="roleName" value="Developer"/>
    <property name="description" value="A user than can edit/delete schedules, rob
    <property name="permissions">
        <bean class="com.kapowtech.scheduler.client.auth.Permissions">
            <property name="dashBoardPermissions">
                <bean class="com.kapowtech.scheduler.client.auth.DashBoardPermissi
                    <property name="viewDashboard" value="true"/>
                </bean>
            </property>
            <property name="schedulesTabPermissions">
                <bean class="com.kapowtech.scheduler.client.auth.SchedulesTabPermi
                    <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.RobotsTabPermissi
            <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.TypesTabPermissio
            <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.SnippetsTabPermis
            <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.ResourcesTabPermi
            <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.OAuthTabPermissio
            <property name="OAuthTab" value="true"/>
        </bean>
    </property>
    <property name="dataViewPermissions">
        <bean class="com.kapowtech.scheduler.client.auth.DataViewPermissio
            <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
                                    the project name is one that doesn't exists in
                        <property name="viewOrphanProjects" value="false"/>
                    </bean>
                </property>
                <property name="taskViewTabPermissions">
                    <bean class="com.kapowtech.scheduler.client.auth.TaskViewTabPermis
                        <property name="viewTasks" value="true"/>
                        <property name="stopTask" value="false"/>
                    </bean>
                </property>
                <property name="clustersTabPermissions">
                    <bean class="com.kapowtech.scheduler.client.auth.ClustersTabPermis
                        <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.ProjectsTabPermis
                        <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
                        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>
```

All permissions are defined as simple true/false. If the property value is true, a user in that role may perform the action indicated by the property name. If we look at the `logTabPermissions` we can see that a developer is allowed to view data in all logs, but not delete any log messages.

Lets log in as the dev user and see how the permissions are reflected in the Management Console You log out by clicking the menu button in the upper right corner, then log in as the dev user. Now go to the Logs tab, select the RoboServer log in the ribbon tabs to the left. Notice how the delete button is disabled, and hovering gives a tooltip message.



View logs but not delete

You can assign multiple roles to the same security group, and you can assign the same role to multiple security groups. If a user holds multiple roles, he can do anything that at least one of the roles allow. With multiple projects in Management Console, users of different projects can be completely separated by assigning their groups to project-specific roles.

The predefined roles are suggestions, but you can add any number of additional roles, or change the existing roles to fit your needs.

Actions that can be performed on the Settings, Backup and License tabs (sub tabs to Admin) are only available to users that are members of an `adminGroup`.

Security

In the `Configuration.xml` file, it is possible to configure some additional security settings for the Management Console.

The security configuration section of the file looks like this:

```
<bean id="securityConfiguration" class="com.kapowtech.mc.config.Securi
    <property name="allowScriptExecution" value="false"/>
    <property name="jdbcDriverUpload" value="LOCALHOST"/>
</bean>
```

It contains two security settings which are described below.

As part of a schedule, a user may configure a script that is used for Pre/Post processing, e.g. before/after any robots in the schedule are executed. These scripts must be located on the Management Console computer's file system. For security reasons, the running of external scripts is disabled by default. To allow external scripts to execute, you should change the value of the **allowScriptExecution** property to true.

If you use a pre/post-processing script in a schedule, and scripts are disabled by the administrator, the schedule will fail immediately with the following message *Scripts have been disabled by the administrator*. without executing any robots. Users can save schedules with pre/post-processing scripts even if the feature is disabled. NB: Even if external scripts are disabled, you can still use the pseudo script command "runrobot: test.robot" as part of pre/post-processing.

By default, only the admin user when accessing the Management Console from the localhost is allowed to upload JDBC drivers. To change this behavior, modify the **jdbcDriverUpload** property. The following are the possible values, the property can have:

- NONE: upload of JDBC drivers is not allowed for any users
- LOCALHOST: the default value, the admin user is allowed to upload drivers if accessing the Management Console from the localhost
- ANY_HOST: the admin user is allowed to upload drivers from any host

SiteMinder Integration

The Management Console supports pre-authentication using SiteMinder SSO. With SiteMinder the identity of the user is established before accessing the Management Console, and the user's identity is communicated through a HTTP header. The identity of the user has to be in the form of an LDAP distinguished Name, so the Management Console can resolve the user's LDAP group memberships.

SiteMinder integration is disabled by default, and can be enabled in `/WEB-INF/Configuration.xml`. The following section defined the SiteMinder configuration bean:

```
<bean id="siteMinderConfiguration" class="com.kapowtech.mc.config.SiteMinderCon
     <property name="enabled" value="false"/>
     <property name="headerName" value="sm_userdn"/>
     <property name="accountAttribute" value="sAMAccountName"/>
     <property name="accountAttributePattern" value="(.*)"/>
 </bean>
```

You must first set the `enabled` property to `true`, then specify the name of the HTTP header which contains the user's distinguished name. The `accountAttribute` property identify which of the user's LDAP attributes will be used as his/hers account name (The default uses the sAMAccountName, which is the user's Windows login name). The `accountAttributePattern` property specifies how the account name is parsed from the attribute value. `accountAttributePattern` must be a regular

expression (with a single group of parentheses identifying the account name) , in the example `(.*)` means everything in the attribute. If you wanted to extract the account name form the users email address the configuration, could look like this:

```
 <bean id="siteMinderConfiguration" class="com.kapowtech.mc.config.SiteMinderC
    <property name="enabled" value="true"/>
    <property name="headerName" value="sm_userdn"/>
    <property name="accountAttribute" value="userPrincipalName"/>
    <property name="accountAttributePattern" value="([^@]*)@.*"/>
</bean>
```

here `([^@]*)@.*` will match an email address and extract everything before the @ as the account name.

Since SiteMinder uses part of the LDAP login configuration, this must be enabled and configured in `/WEB-INF/login.xml` see. Since you can't log in using the normal login form, you will have to add a user group to the `adminGroups` bean in `/WEB-INF/login.xml`, before you can start configuring the Management Console

It is not possible to logout of the system, since the presence of the SiteMinder header would mean that you is always authenticated. To "logout" you have to close your browser.

Limitations

SiteMinder integration only works when the Management Console is accessed through the browser. However Management Console can also be accessed by application which are not browsers, and for this to work you need to bypass the SiteMinder authentication mechanism, for some URLs. These clients include

Design Studio
: Robot developers will need to access Management Console to obtain a developer seat license, to upload robots, and get database configurations and other settings stored in Management Console.

  This requires access to the following URLs (relative to the context path where Management Console is deployed).

  - /License/*
  - /secure/Repository/*
  - /IDESettings/*

  Access to these URLs are protected by basic authentication

REST services
: Robots can be exposed as REST services, which are protected by basic authentication by default, but can be exposed without authentication. These are typically invoked by external applications. REST uses the following URL:

  - /rest/*

Deployment Checklist

This table contains a deployment checklist. You can print it by clicking the print icon on the toolbar to the left.

Items marked ***** are optional.

Items marked ****** may be optional.

Table 154. Deployment Checklist

Check	Item	Description / Notes
	Create new Database	A separate Database (Schema) for hosting the Management Console tables.
	Install JDBC Drivers	Put the JDBC driver for the hosting database into the web container's `lib` folder.
	**Run SQL scripts	Only necessary if the credentials used to access the database doesn't have the CREATE and ALTER TABLE privileges.
	Configure Data source	Configure JNDI data source for the hosting database in `jdbc/Kapow/platform`.
	Enable Login Provider	Configure a login provider in `login.xml`
	*Replace Default Certificate	We recommend that you replace the build in certificate used to encrypt/decrypt passwords See Password Encryption

Advanced Configuration

This section contains a number of advanced configuration options which may be useful.

LDAP Integration

The installation guide shows you how to authenticate users against a list of hard-coded credentials, this section will describe how to authenticate users against LDAP.

In `login.xml` you will find the following definition

```
<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=kapowd
    <property name="userSearchFilter" value="(userPrincipalName={0}@kapowdemo.
    <property name="userSearchSubtree" value="true"/>
    <property name="groupSearchBase" value="OU=Security Groups,OU=TheEnterpris
    <property name="groupSearchFilter" value="(member={0})"/>
    <property name="groupRoleAttribute" value="cn"/>
    <property name="groupSearchSubtree" value="true"/>
    <property name="convertToUpperCase" value="true"/>
</bean>
```

This defines an `LdapLogin` bean named `ldap`. The bean defines a number of properties that controls the LDAP integration. If you are familiar with the way Tomcat integrates to LDAP this should be quite familiar.

Table 155. LDAP properties

Property	Description
`ldapServerURL`	The URL to the LDAP server. This uses the `ldap://` protocol.
`userDn`	The DN (distinguished name) used to log in to LDAP to authenticate other users.
`password`	The password for the `userDN` account. As the password will be stored in clear text in this file you should use an account that only has 'read' access.
`userSearchBase`	The base directory in the LDAP tree where users can be found.
`userSearchFilter`	The filter that is applied to find the username.
`userSearchSubtree`	Set to true if users may be located in the sub-directory of the `userSearchBase`.
`groupSearchBase`	The base directory in the LDAP tree where groups can be found.
`groupSearchFilter`	The filter that is applied to identify the users in this group.
`groupRoleAttribute`	The attribute that holds the group name.
`groupSearchSubtree`	Set to true if groups may be located in the sub-directory of the `groupSearchBase`.
`convertToUpperCase`	Should the group names be converted to upper case, true by default.
`allGroupsFilter`	Optional. Controls which groups are displayed when creating project permissions, see below.

If you want to use an LDAP account to administer the Management Console, you must add one of the groups that you are are a member of to the `adminGroups` bean in `login.xml`, as described in Project Permissions. Be advised that anyone that is a member of a group listed in `adminGroups` will be a Management Console administrator, so you may want to create a new LDAP group for this purpose. Take care to use the upper case group name if `convertToUpperCase` is true.

When you select a project permission you will see that all the group names have been pulled from LDAP to populate the drop down. The groups are located by using the `groupRoleAttribute` to construct a filter to fetch all groups. Sometimes you don't want all LDAP groups displayed here, in which case override this behavior by providing your own filter, this is done by adding an additional property to the `LdapLogin`.

```
<property name="allGroupsFilter" value="(cn=*)"/>
```

will find all group names, if the group name is in the `cn` attribute (this is the default). If you only want groups starting with the letter 'e' you could use

```
<property name="allGroupsFilter" value="(cn=E*)"/>
```

The filter uses basic LDAP queries, so you can find documentation elsewhere for more complex queries.

High Availability

If high availability (failover) is required you can configure multiple Management Console instances to work together as a cluster. Four components must be clustered to achieve full failover.

Table 156. Deployment Checklist

Component	Description
Load balancer	An HTTP load balancer is required to distribute requests between multiple Tomcat servers.
Clustered platform database	The Management Console stores schedules, robots etc. in the platform database. In a failover scenario the platform database should run on a clustered DBMS to avoid a single point of failure.
Tomcat session replication	Although the Management Console doesn't store any data directly in the user's session (except during Import/Export), the session holds the user's authentication information. If session replication is not enabled, the user will have to login again if the Tomcat he is currently connected to crashes.
Hazelcast	Hazelcast (www.hazelcast.com) is used to cluster data structures over multiple JVMs. Inside the Management Console this is used to provide clustering of vital data structures, and to provide intercommunication between application instances. Here is a example: When you run a robot on RoboServer, a thread is required to process the status messages returned by RoboServer. This thread will be running inside a concrete Tomcat instance. In a clustered environment, a user trying to stop the robot may in fact be generating the stop request on another Tomcat instance than the instance running the robot. In that case the stop request is broadcast through Hazelcast to all instances and the instance running the robot will receive it and act to stop the robot.

Deploying multiple instances of Management Console

You should have 2 or more identical Tomcat installations, and deploy the same version of `ManagementConsole.war` on them all. Make sure the `web.xml Configuration.xml`, `login.xml` and `roles.xml` files are the same across all the instances.

Load balancer configuration

We will not describe specifics of load balancer configuration. What may be useful to know is how to determine if the application started correctly.

If the `ManagementConsole.xml` (context configuration) or `web.xml` files are invalid, the application will not be deployed on Tomcat, and requests will normally return 404 (since you will hit the Tomcat's ROOT application which doesn't have anything deployed on `/ManagementConsole/`).

Any other errors encountered during application startup are shown to the user when the application loads. This way you don't always have to check the log to figure out why the application didn't load correctly.

This is, however, a bit impractical as the application will return `200 OK` even if there were errors during startup. Also, if authentication is enabled, you will have to login before you can see the error messages.

To make it easier for load balancers to see if the application started correctly, you can make a request to the URL `/ManagementConsole/Ping`. This will either return HTTP status code `200` if the application loaded correctly, or `500` with a stack trace of the error.

Tomcat Session Replication

Session replication is configured in `/conf/server.xml`. Here is an example that uses multicast for instance discovery (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;.*\

        <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.ClusterSes
    </Cluster>
```

You also have to set the `jvmRoute` attribute on the `<Engine>` element in server.xml, like this

```
<Engine jvmRoute="tomcat2" name="Catalina" defaultHost="MyHost">
```

. *Note:* if you are using `mod_jk` as a poor man's load balancer, the value of the `jvmRoute` has to match the name listed in the `workers.properties` file references by the `mod_jk` configuration.

See your Tomcat documentation [http://tomcat.apache.org/tomcat-5.5-doc/cluster-howto.html] for details.

Hazelcast configuration

The most basic Hazelcast settings can be edited in `Configuration.xml`, while more advanced settings such as SSL encryption must be configured in `/WEB-INF/Hazelcast.xml`

When Management Console starts, it creates a Hazelcast node on port 5701 (or the next available port that is available). By default this Hazelcast node will bind to IP address `127.0.0.1`. You will have to change the bind address to a public IP/host name before it can participate in a cluster. This is done by modifying the `interface` property of the `cluster` bean in `Configuration.xml`. It might look like this:

```
<bean id="cluster" class="com.kapowtech.mc.config.ClusterConfig" >
    <property name="port" value="26000"/>
    <property name="interface" value="10.0.0.25"/>
 .......
</bean>
```

The * is used as a wildcard, in this case the application will try bind to the 'first' interface that has an IP address starting with `10.0.0`. It is possible, but not recommended to use `*.*.*.*` as you may end up binding to `127.0.0.1`, or another virtual interface.

When you start additional instances of Management Console, their Hazelcast instances will try to find any existing Hazelcast node and join the cluster. This discovery can be done through multicast or through TCP/IP.

To use multicast discovery you must modify the `cluster` bean in `Configuration.xml`. This is done my un-commenting the following line

```
<property name="joinConfig" ref="multicastCluster"/>
```

`multicastCluster` is a reference to the `multicastCluster` bean, which defines the multicast group and port. You may change it to fit your network topology.

If your network doesn't allow multicast you will have to use the `tcpCluster`. That is done by un-commenting this line instead:

```
<property name="joinConfig" ref="tcpCluster"/>
```

The `tcpCluster` bean contains a list of `TcpPeer`, one for each other Hazelcast node. If you use the same TCP port for all Hazelcast nodes you don't need to specify a port number (each node will assume that its peers are running on the same port as itself). If you have two nodes configured in a TCP cluster it could look like this:

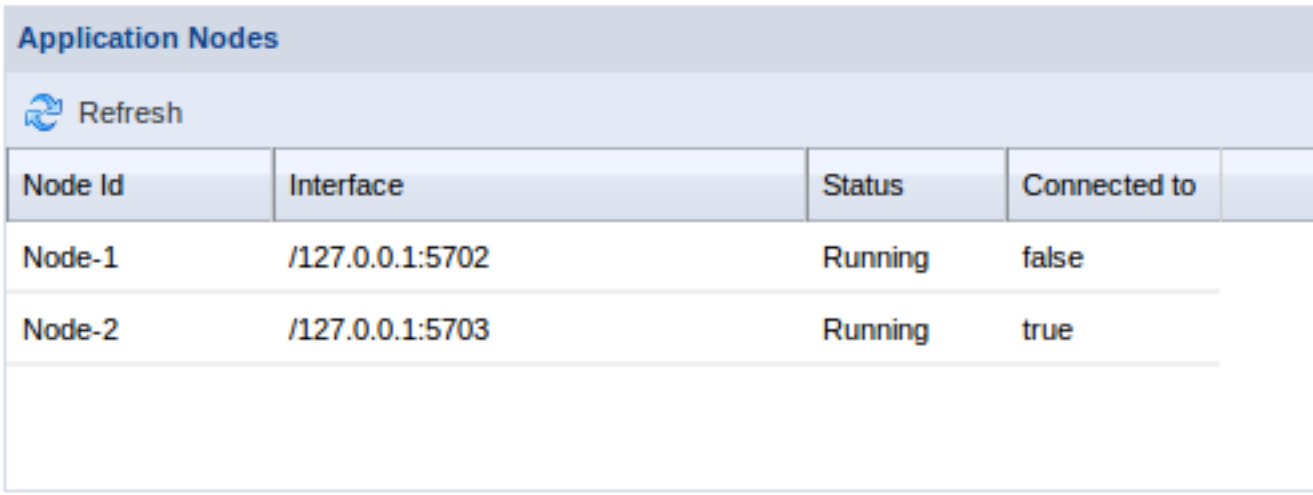
```
<bean id="tcpCluster" class="com.kapowtech.mc.config.TcpJoinConfig">
    <property name="peers">
        <list>
            <bean class="com.kapowtech.mc.config.TcpPeer">
                <property name="host" value="10.0.0.25"/>
            </bean>
            <bean class="com.kapowtech.mc.config.TcpPeer">
                <property name="host" value="10.0.0.26"/>
            </bean>
```

```
            </list>
        </property>
    </bean>
```

Notice that both nodes are in the list. This means that regardless which node starts first it will be able to find its peer. It also allows you to use identical `Configuration.xml` files in both applications. Also, TCP ports numbers are not defined, so each peer will try to connect to the other one on the same port as it is listening on itself.

Application Nodes

You can verify that the application is properly clustered by going to the Projects tab, and look at the Application Nodes section in the bottom of the page. Here you should see something like this



This means that the cluster currently consists of two nodes. The interface column will list the IP/host and port that Hazelcast is using for inter-cluster communication. The Connected to column informs you which of the two nodes you are connected to at the moment. If you shut down the server you are currently connected to, you will automatically be re-routed to another live instance by the load balancer.

URI Encoding

If you plan to upload robots with names that contain non-ASCII characters, like Danish ÆØÅ or German ß to the repository, you have to configure the URI Encoding on your web container to UTF-8.

On Tomcat this is done on the `<connector>` definition found in `server.xml` file inside the `/conf` folder. Here you add the attribute `URIEncoding="UTF-8"` like this:

```
<Connector port="8080" URIEncoding="UTF-8"...../>
```

Password Encryption

As of version 8.2 the Management Console uses certificate based (public-, private-key) encryption when storing passwords. When you import from a previous version password will automatically be re-encrypted using the new certificate based algorithm.

The certificate and the matching private key is stored in a Java keystore, Management Console ships with a keystore that contains a default certificate and private key. Since all customers get the same keystore we recommend that you create your own keystore, otherwise anyone will be able to load your exports and potentially get your passwords.

Create your own keystore

If you have already started the Management Console you will need to upgrade the certificate.

The keystore must be in pkcs12 format, and can be created using the `keytool` application that comes with the Java SDK (which can be downloaded from Oracle.com, currently available here [http://www.oracle.com/technetwork/java/javase/downloads/index.html]). The following command creates a new pkcs12 keystore with a certificate that is valid for 365 days.

```
keytool -genkey -alias mc -keyalg RSA -validity 3650 -keystore mc.p12 -store
```

You will be prompted for password, and the information that will be stored in the X.509 private key. The command will create a file `mc.p12` (the value from the -keystore argument) in the current directory. -validity 3650 means the certificate will be valid for 10 years.

> *We don't recommend that you use a certificate issued by a certificate authority (CA) since pkcs12 holds both the private key and the public certificate, and the password to the private key will be written in clear text as part of the application configuration.*

To instruct Management Console to use the new certificate, change the `Configuration.xml` file. The file is located inside the `ManagementConsole.war` web archive, which must be unpacked, see Deploying into Tomcat for details. Inside `Configuration.xml` you will find the following entry:

```
<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>
```

Here you must specify the location, password and alias of the keystore. If you copy the keystore into `ManagementConsole.war` the location must be relative to the root of the application. If you want to refer to a keystore stored in the file system, the location must start with `file://`, and must be an absolute reference to the keystore location.

Upgrading the keystore

The first time Management Console starts, it creates a checksum using the private key from the keystore, this allows it to detect when the keystore has been replaced, and verify that passwords can in fact be decrypted with the provided certificate. If you have already started Management Console before installing your own keystore, you have will have to configure Management Console to perform a password conversion.

First copy the current keystore file into a new location, like your users home folder, then modify `Configuration.xml` to create a password converter with reference to the old keystore, like this:

```
 <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.servi
    <constructor-arg ref="oldKeyStore"/>
</bean>
```

This configures a password converter to use the previous certificate to decrypt any existing passwords and checksum (you will have to provide correct location, alias and password for the old keystore), and use the new private key (as configured above) to re-encrypt passwords and create a new checksum. The conversion will occur the next time the Management Console is started, the conversion occurs while the application is starting and may take some time if there are many schedules. You don't have to remove the `oldKeyStore` and `passwordConverter` beans from `Configuration.xml`, as the password conversion is only triggered when the checksum and keystore is out-of-sync, and after the conversion the checksum will match the new keystore).

Getting Support

The following resources will be useful in case of problems.

Customer Support

If you are having any kind of problems using Kapow Katalyst, do not hesitate to ask for help. For information on how to do this, please email Kapow Software Customer Support at `support@kapowsoftware.com`. Telephone support is also available, please reference the contact phone numbers provided to you with your Kapow license keys.

Customers who are active on Premier Support also have the ability to call the toll-free support number for Priority 1 issues, 24 hours a day, 7 days a week. If you are a customer on Premier Support and have lost this number, please contact us at `support@kapowsoftware.com` and we will be happy to provide it to you.

In many of the Kapow Katalyst applications, you can also send a bug report from within the application. To do this, select Report Bug in the Help menu. Please provide as much information as possible about the bug and what you did just before the bug occurred.

Self-Service Portal and Knowledge Base

Kapow customers who are active on maintenance also are entitled to obtain access to our self-service online support portal [https://na3.salesforce.com/sserv/login.jsp?orgId=00D500000006c7a], which includes solutions to commonly found problems, as well as a Knowledge Base containing implementation tips and tricks, self-help videos, and more.

If you do not have a user ID and would like to log into the portal, please contact us at `support@kapowsoftware.com` and we will get you set up.

Support Policy

With release 9.2 we announce the following support policies:

- Kapow Katalyst 9.1 will be supported through September 1, 2014.
- Kapow Katalyst 9.0 will be supported through June 1, 2014.
- Kapow Katalyst 8.3 will be supported through May 1, 2014.
- Kapow Katalyst 8.2 will be supported through November 1, 2013.
- Kapow Katalyst 8.1 will be supported through May 1, 2013.
- Kapow Katalyst 8.0 will be supported through November 1, 2012.
- Kapow Web Data Server 7.2 will be supported through May 1, 2012.

License

License Information

See below for the terms and conditions of Kapow Katalyst version 9.2.0:

- Kapow Katalyst End User License Agreement
- Third Party and Open Source Code – License Terms
- Product Privacy Policy

End User License Agreement

Last Updated: March 29, 2011

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.

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.

Third Party and Open Source Code – License Terms

Kapow Software

Third Party and Open Source Code – License Terms

Last Updated: June 3rd, 2013

1) Apache Commons Configuration, Apache Commons Digester, Apache Commons BeanUtils, Apache Velocity, Apache Velocity Tools, Apache Commons Lang, Apache Commons IO, Apache Commons Collections, Apache Commons FileUpload, Apache Commons HttpClient, Apache Commons Pool, Apache Commons DBCP, Apache Commons CLI, Apache Commons Codec, Apache Commons Logging, Apache Derby, Apache Zookeeper, Apache Hadoop, Apache HBase, Guava, java-diff-utils, JSON.simple Not-Yet-Commons-SSL, joda-time, Batik, ROME, Gwt, WinFoldersJava, Tomcat, Spring, Quartz, Hazelcast, Jakarta POI, Jackson, opencsv, and Jetty are licensed under Apache License, Version 2.0 (the “Apache License”), available at http://www.apache.org/licenses/LICENSE-2.0; you may not use these file except in compliance with the Apache License.

Unless required by applicable law or agreed to in writing, software distributed under the Apache License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the Apache License for the specific language governing permissions and limitations under the Apache License.

2) JSON is licensed under the following license terms (also available at http://www.json.org/license.html):

Copyright (c) 2002 JSON.org

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

The Software shall be used for Good, not Evil.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

3) Eclipse and EclipseLink are licensed under the Eclipse Public License Version 1.0, available at http://www.eclipse.org/legal/epl-v10.html.

In addition, JUnit is licensed under IBM Common Public License Version 0.5, which has been superseded by Eclipse Public License Version 1.0 available at http://www.eclipse.org/legal/epl-v10.html.

Source code for these programs is available from Kapow by contacting Kapow at support@kapowsoftware.com.

THE PROGRAMS ARE PROVIDED ON AN "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, EITHER EXPRESS OR IMPLIED INCLUDING, WITHOUT LIMITATION, ANY WARRANTIES OR CONDITIONS OF TITLE, NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. NEITHER RECIPIENT NOR ANY CONTRIBUTORS SHALL HAVE ANY LIABILITY FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING WITHOUT LIMITATION LOST PROFITS), HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OR DISTRIBUTION OF THE PROGRAMS OR THE EXERCISE OF ANY RIGHTS GRANTED HEREUNDER, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

4) Icons are licensed from famfamfam.com under Creative Commons Attribution 2.5 License available at http://creativecommons.org/licenses/by/2.5/.

5) Rhino is licensed from the Mozilla Foundation, and Kapow's Modifications to Rhino (the documentation for which is in the source code for the Covered Code) are licensed, under Netscape Public License Version 1.1 available at http://www.mozilla.org/MPL/NPL-1.1.html [http://www.mozilla.org/MPL/NPL-1.1.html].

Source code for the Covered Code is available from Kapow by contacting Kapow at support@kapowsoftware.com.

KAPOW DISCLAIMS (FOR ITSELF, THE INITIAL DEVELOPER AND ALL CONTRIBUTORS) ALL SUPPORT, WARRANTY OR INDEMNITY OBLIGATIONS, OR OTHER LIABILITY, RELATING TO THE COVERED CODE TO THE FULLEST EXTENT PERMITTED UNDER APPLICABLE LAW.

6) Gecko and Java port of Mozilla charset detector are licensed under Mozilla Public License Version 1.1 available at http://www.mozilla.org/MPL/MPL-1.1.html [http://www.mozilla.org/MPL/MPL-1.1.html].

Source code for the Covered Code is available from Kapow by contacting Kapow at support@kapowsoftware.com.

KAPOW DISCLAIMS ANY AND ALL SUPPORT, WARRANTY OR INDEMNITY OBLIGATIONS, OR OTHER LIABILITY, RELATING TO THE COVERED CODE TO THE FULLEST EXTENT PERMITTED UNDER APPLICABLE LAW.

7) This product includes software developed by the Apache Software Foundation (http://www.apache.org/). The following are licensed under the Apache Software License, Version 1.1 available at http://apache.org/licenses/LICENSE-1.1.

- Xalan (Copyright (C) 1999-2000 The Apache Software Foundation. All rights reserved)
- Xerces (Copyright (C) 1999-2000 The Apache Software Foundation. All rights reserved)

8) This product includes software developed by the Apache Software Foundation (http://www.apache.org/). The following are licensed under the Apache Software License, Version 2.0 available at http://www.apache.org/licenses/LICENSE-2.0.

- Log4net (Copyright (C) 2007 The Apache Software Foundation. All rights reserved)
- Log4j (Copyright (C) 2007 The Apache Software Foundation. All rights reserved.)

9) Dom4j version 1.6.1 is licensed under the BSD-style license below: http://dom4j.sourceforge.net/dom4j-1.6.1/license.html.

Redistribution and use of this software and associated documentation ("Software"), with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain copyright statements and notices. Redistributions must also contain a copy of this document.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. The name "DOM4J" must not be used to endorse or promote products derived from this Software without prior written permission of MetaStuff, Ltd. For written permission, please contact dom4j-info@metastuff.com.

4. Products derived from this Software may not be called "DOM4J" nor may "DOM4J" appear in their names without prior written permission of MetaStuff, Ltd. DOM4J is a registered trademark of MetaStuff, Ltd.

5. Due credit should be given to the DOM4J Project - http://dom4j.sourceforge.net

THIS SOFTWARE IS PROVIDED BY METASTUFF, LTD. AND CONTRIBUTORS "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL METASTUFF, LTD. OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Copyright 2001-2005 (C) MetaStuff, Ltd. All Rights Reserved.

10) Xstream version 1.3.1 is licensed under the BSD license below available at http://xstream.codehaus.org/license.html.

Copyright (c) 2003-2006, Joe Walnes

Copyright (c) 2006-2007, XStream Committers

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the name of XStream nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

11) jta.jar version 1.0.1B is licensed under the License Agreement for Java (TM) Transaction API (JTA) Specification Interface classes 1.0.1B Maintenance Release below.

SUN MICROSYSTEMS, INC. BINARY CODE LICENSE AGREEMENT

READ THE TERMS OF THIS AGREEMENT AND ANY PROVIDED SUPPLEMENTAL LICENSE TERMS (COLLECTIVELY "AGREEMENT") CAREFULLY BEFORE OPENING THE SOFTWARE MEDIA PACKAGE. BY OPENING THE SOFTWARE MEDIA PACKAGE, YOU AGREE TO THE TERMS OF THIS AGREEMENT. IF YOU ARE ACCESSING THE SOFTWARE ELECTRONICALLY, INDICATE YOUR ACCEPTANCE OF THESE TERMS BY SELECTING THE "ACCEPT" BUTTON AT THE END OF THIS AGREEMENT. IF YOU DO NOT AGREE TO ALL THESE TERMS, PROMPTLY RETURN THE UNUSED SOFTWARE TO YOUR PLACE OF PURCHASE FOR A REFUND OR, IF THE SOFTWARE IS ACCESSED ELECTRONICALLY, SELECT THE "DECLINE" BUTTON AT THE END OF THIS AGREEMENT.

1. LICENSE TO USE. Sun grants you a non-exclusive and non-transferable license for the internal use only of the accompanying software and documentation and any error corrections provided by Sun (collectively "Software"), by the number of users and the class of computer hardware for which the corresponding fee has been paid.

2. RESTRICTIONS. Software is confidential and copyrighted. Title to Software and all associated intellectual property rights is retained by Sun and/or its licensors. Except as specifically authorized in any Supplemental License Terms, you may not make copies of Software, other than a single copy of Software for archival purposes. Unless enforcement is prohibited by applicable law, you may not modify, decompile, or reverse engineer Software. Licensee acknowledges that Licensed Software is not designed or intended for use in the design, construction, operation or maintenance of any nuclear facility. Sun Microsystems, Inc. disclaims any express or implied warranty of fitness for such uses. No right, title or interest in or to any trademark, service mark, logo or trade name of Sun or its licensors is granted under this Agreement.

3. LIMITED WARRANTY. Sun warrants to you that for a period of ninety (90) days from the date of purchase, as evidenced by a copy of the receipt, the media on which Software is furnished (if any) will be free of defects in materials and workmanship under normal use. Except for the foregoing, Software is provided "AS IS". Your exclusive remedy and Sun's entire liability under this limited warranty will be at Sun's option to replace Software media or refund the fee paid for Software.

4. DISCLAIMER OF WARRANTY. UNLESS SPECIFIED IN THIS AGREEMENT, ALL EXPRESS OR IMPLIED CONDITIONS, REPRESENTATIONS AND WARRANTIES, INCLUDING ANY IMPLIED WARRANTY OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT ARE DISCLAIMED, EXCEPT TO THE EXTENT THAT THESE DISCLAIMERS ARE HELD TO BE LEGALLY INVALID.

5. LIMITATION OF LIABILITY. TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL SUN OR ITS LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA,

OR FOR SPECIAL, INDIRECT, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF OR RELATED TO THE USE OF OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. In no event will Sun's liability to you, whether in contract, tort (including negligence), or otherwise, exceed the amount paid by you for Software under this Agreement. The foregoing limitations will apply even if the above stated warranty fails of its essential purpose.

6. Termination. This Agreement is effective until terminated. You may terminate this Agreement at any time by destroying all copies of Software. This Agreement will terminate immediately without notice from Sun if you fail to comply with any provision of this Agreement. Upon Termination, you must destroy all copies of Software.

7. Export Regulations. All Software and technical data delivered under this Agreement are subject to US export control laws and may be subject to export or import regulations in other countries. You agree to comply strictly with all such laws and regulations and acknowledge that you have the responsibility to obtain such licenses to export, re-export, or import as may be required after delivery to you.

8. U.S. Government Restricted Rights. If Software is being acquired by or on behalf of the U.S. Government or by a U.S. Government prime contractor or subcontractor (at any tier), then the Government's rights in Software and accompanying documentation will be only as set forth in this Agreement; this is in accordance with 48 CFR 227.7201 through 227.7202-4 (for Department of Defense (DOD) acquisitions) and with 48 CFR 2.101 and 12.212 (for non-DOD acquisitions).

9. Governing Law. Any action related to this Agreement will be governed by California law and controlling U.S. federal law. No choice of law rules of any jurisdiction will apply.

10. Severability. If any provision of this Agreement is held to be unenforceable, this Agreement will remain in effect with the provision omitted, unless omission would frustrate the intent of the parties, in which case this Agreement will immediately terminate.

11. Integration. This Agreement is the entire agreement between you and Sun relating to its subject matter. It supersedes all prior or contemporaneous oral or written communications, proposals, representations and warranties and prevails over any conflicting or additional terms of any quote, order, acknowledgment, or other communication between the parties relating to its subject matter during the term of this Agreement. No modification of this Agreement will be binding, unless in writing and signed by an authorized representative of each party.

JAVA(TM) INTERFACE CLASSES JAVA TRANSACTION API (JTA), VERSION 1.0.1B, MAINTENANCE RELEASE - SUPPLEMENTAL LICENSE TERMS

These supplemental license terms ("Supplemental Terms") add to or modify the terms of the Binary Code License Agreement (collectively, the "Agreement"). Capitalized terms not defined in these Supplemental Terms shall have the same meanings ascribed to them in the Agreement. These Supplemental Terms shall supersede any inconsistent or conflicting terms in the Agreement, or in any license contained within the Software.

1. Software Internal Use and Development License Grant. Subject to the terms and conditions of this Agreement, including, but not limited to Section 3 (Java Technology Restrictions) of these Supplemental Terms, Sun grants you a non-exclusive, non-transferable, limited license to reproduce internally and use internally the binary form of the Software, complete and unmodified, for the sole purpose of designing, developing and testing your Java applets and applications ("Programs").

2. License to Distribute Software. In addition to the license granted in Section 1 (Software Internal Use and Development License Grant) of these Supplemental Terms, subject to the terms and conditions of

this Agreement, including but not limited to Section 3 (Java Technology Restrictions), Sun grants you a non-exclusive, non-transferable, limited license to reproduce and distribute the Software in binary form only, provided that you (i) distribute the Software complete and unmodified and only bundled as part of your Programs, (ii) do not distribute additional software intended to replace any component(s) of the Software, (iii) do not remove or alter any proprietary legends or notices contained in the Software, (iv) only distribute the Software subject to a license agreement that protects Sun's interests consistent with the terms contained in this Agreement, and (v) agree to defend and indemnify Sun and its licensors from and against any damages, costs, liabilities, settlement amounts and/or expenses (including attorneys' fees) incurred in connection with any claim, lawsuit or action by any third party that arises or results from the use or distribution of any and all Programs and/or Software.

3. Java Technology Restrictions. You may not modify the Java Platform Interface ("JPI", identified as classes contained within the "java" package or any subpackages of the "java" package), by creating additional classes within the JPI or otherwise causing the addition to or modification of the classes in the JPI. In the event that you create an additional class and associated API(s) which (i) extends the functionality of the Java Platform, and (ii) is exposed to third party software developers for the purpose of developing additional software which invokes such additional API, you must promptly publish broadly an accurate specification for such API for free use by all developers. You may not create, or authorize your licensees to create additional classes, interfaces, or subpackages that are in any way identified as "java", "javax", "sun" or similar convention as specified by Sun in any naming convention designation.

4. Trademarks and Logos. You acknowledge and agree as between you and Sun that Sun owns the SUN, SOLARIS, JAVA, JINI, FORTE, and iPLANET trademarks and all SUN, SOLARIS, JAVA, JINI, FORTE, and iPLANET-related trademarks, service marks, logos and other brand designations ("Sun Marks"), and you agree to comply with the Sun Trademark and Logo Usage Requirements currently located at http://www.sun.com/policies/trademarks. Any use you make of the Sun Marks inures to Sun's benefit.

5. Source Code. Software may contain source code that is provided solely for reference purposes pursuant to the terms of this Agreement. Source code may not be redistributed unless expressly provided for in this Agreement.

6. Termination for Infringement. Either party may terminate this Agreement immediately should any Software become, or in either party's opinion be likely to become, the subject of a claim of infringement of any intellectual property right.

For inquiries please contact: Sun Microsystems, Inc. 4150 Network Circle, Santa Clara, California 95054. (LFI#121049/Form ID#011801)

12) Flot is licensed under the MIT License below.

Copyright (C) 2007-2009 IOLA and Ole Laursen

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,

ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

13) GFlot Version 1.0.0 is licensed under the MIT License below.

Copyright (C) 2008 Nanometrics Inc.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

14) CUP Parser Generator is licensed under the following terms:

Copyright 1996-1999 by Scott Hudson, Frank Flannery, C. Scott Ananian

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both the copyright notice and this permission notice and warranty disclaimer appear in supporting documentation, and that the names of the authors or their employers not be used in advertising or publicity pertaining to distribution of the software without specific, written prior permission.

The authors and their employers disclaim all warranties with regard to this software, including all implied warranties of merchantability and fitness. In no event shall the authors or their employers be liable for any special, indirect or consequential damages or any damages whatsoever resulting from loss of use, data or profits, whether in an action of contract, negligence or other tortious action, arising out of or in connection with the use or performance of this software.

15) PDFBox is licensed under the BSD License below.

Copyright (c) 2003, www.pdfbox.org

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. Neither the name of pdfbox; nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

16) The Legion of The Bouncy Castle is licensed under a license adapted from the MIT License.

Copyright (c) 2000 – 2006, The Legion Of The Bouncy Castle (http://www.bouncycastle.org)

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

17) Fontbox is licensed under the BSD License below.

Copyright (C) 2003-2005, www.fontbox.org [http://www.fontbox.org]

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted, provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. Neither the name of fontbox nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS

BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

18) Cryptix is licensed under the license below.

Cryptix General License

Copyright (c) 1995-2005 The Cryptix Foundation Limited.

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

THIS SOFTWARE IS PROVIDED BY THE CRYPTIX FOUNDATION LIMITED AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE CRYPTIX FOUNDATION LIMITED OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

19) This product includes software developed for the Protomatter Software Project (http://protomatter.sourceforge.net/). The Protomatter software is licensed under the Protomatter Software License, Version 1.0 below.

The Protomatter Software License, Version 1.0 derived from The Apache Software License, Version 1.1

Copyright (c) 1998-2002 Nate Sammons. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

3. The end-user documentation included with the redistribution, if any, must include the following acknowledgment: "This product includes software developed for the Protomatter Software Project (http://

protomatter.sourceforge.net/)." Alternately, this acknowledgment may appear in the software itself, if and wherever such third-party acknowledgments normally appear.

4. The names "Protomatter" and "Protomatter Software Project" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact support@protomatter.com.

5. Products derived from this software may not be called "Protomatter", nor may "Protomatter" appear in their name, without prior written permission of the Protomatter Software Project (support@protomatter.com).

THIS SOFTWARE IS PROVIDED "AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE PROTOMATTER SOFTWARE PROJECT OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

20) Jaxen is licensed under the BSD license below:

Copyright 2003-2006 The Werken Company. All Rights Reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

* Neither the name of the Jaxen Project nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

21) Driver for IBM's DB2 is subject to the license you acquired from IBM for the database.

22) SLF4J is licensed under the MIT License below.

Copyright (c) 2004-2011 QOS.ch

23) dnsjava is licensed under the BSD license below:

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

* Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

* Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

* Neither the name of the Jaxen Project nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

24) JavaMail API is licensed under the CDDL license.

25) jQuery is licensed under the MIT License below.

limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

26) Scribe is licensed under the MIT License below.

Copyright (c) 2010 Pablo Fernandez

All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

27) The jsoup code-base (include source and compiled packages) are distributed under the open source MIT license as described below.

Copyright (c) 2009 - 2012 Jonathan Hedley (jonathan@hedley.net)

All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,

FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

28) The ICU library is distributed under the ICU license as described below.

Copyright (c) 1995-2013 International Business Machines Corporation and others

All rights reserved.

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, provided that the above copyright notice(s) and this permission notice appear in all copies of the Software and that both the above copyright notice(s) and this permission notice appear in supporting documentation.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE BE LIABLE FOR ANY CLAIM, OR ANY SPECIAL INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization of the copyright holder.

29) The Unicode Data Files and Software are distributed under the license described below.

Unicode Data Files include all data files under the directories http://www.unicode.org/Public/, http://www.unicode.org/reports/, and http://www.unicode.org/cldr/data/. Unicode Data Files do not include PDF online code charts under the directoryhttp://www.unicode.org/Public/. Software includes any source code published in the Unicode Standard or under the directories http://www.unicode.org/Public/, http://www.unicode.org/reports/, and http://www.unicode.org/cldr/data/.

NOTICE TO USER: Carefully read the following legal agreement. BY DOWNLOADING, INSTALLING, COPYING OR OTHERWISE USING UNICODE INC.'S DATA FILES ("DATA FILES"), AND/OR SOFTWARE ("SOFTWARE"), YOU UNEQUIVOCALLY ACCEPT, AND AGREE TO BE BOUND BY, ALL OF THE TERMS AND CONDITIONS OF THIS AGREEMENT. IF YOU DO NOT AGREE, DO NOT DOWNLOAD, INSTALL, COPY, DISTRIBUTE OR USE THE DATA FILES OR SOFTWARE.

COPYRIGHT AND PERMISSION NOTICE

Copyright (c) 1991-2013 Unicode, Inc. All rights reserved. Distributed under the Terms of Use in http://www.unicode.org/copyright.html.

Permission is hereby granted, free of charge, to any person obtaining a copy of the Unicode data files and any associated documentation (the "Data Files") or Unicode software and any associated documentation (the "Software") to deal in the Data Files or Software without restriction, including without limitation the

rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Data Files or Software, and to permit persons to whom the Data Files or Software are furnished to do so, provided that (a) the above copyright notice(s) and this permission notice appear with all copies of the Data Files or Software, (b) both the above copyright notice(s) and this permission notice appear in associated documentation, and (c) there is clear notice in each modified Data File or in the Software as well as in the documentation associated with the Data File(s) or Software that the data or software has been modified.

THE DATA FILES AND SOFTWARE ARE PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE BE LIABLE FOR ANY CLAIM, OR ANY SPECIAL INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THE DATA FILES OR SOFTWARE.

Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in these Data Files or Software without prior written authorization of the copyright holder.

Unicode and the Unicode logo are trademarks of Unicode, Inc. in the United States and other countries. All third party trademarks referenced herein are the property of their respective owners.

30) The Chinese/Japanese Word Break Dictionary Data (cjdict.txt) is distributed under the license below.

The Google Chrome software developed by Google is licensed under the BSD license. Other software included in this distribution is provided under other licenses, as set forth below.

The BSD License

http://opensource.org/licenses/bsd-license.php

Copyright (C) 2006-2008, Google Inc.

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

Neither the name of Google Inc. nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;

OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

The word list in cjdict.txt are generated by combining three word lists listed below with further processing for compound word breaking. The frequency is generated with an iterative training against Google web corpora.

- Libtabe (Chinese)
  - https://sourceforge.net/project/?group_id=1519
  - Its license terms and conditions are shown below.
- IPADIC (Japanese)
  - http://chasen.aist-nara.ac.jp/chasen/distribution.html
  - Its license terms and conditions are shown below.

---------COPYING.libtabe ---- BEGIN--------------------

Copyrighy (c) 1999 TaBE Project.

Copyright (c) 1999 Pai-Hsiang Hsiao.

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

. Neither the name of the TaBE Project nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Copyright (c) 1999 Computer Systems and Communication Lab, Institute of Information Science, Academia Sinica.

All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.

. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.

. Neither the name of the Computer Systems and Communication Lab nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE REGENTS OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Copyright 1996 Chih-Hao Tsai @ Beckman Institute, University of Illinois c-tsai4@uiuc.edu http://casper.beckman.uiuc.edu/~c-tsai4

----------COPYING.libtabe-----END----------

----------COPYING.ipadic-----BEGIN----------

Copyright 2000, 2001, 2002, 2003 Nara Institute of Science and Technology. All Rights Reserved.

Use, reproduction, and distribution of this software is permitted. Any copy of this software, whether in its original form or modified, must include both the above copyright notice and the following paragraphs.

Nara Institute of Science and Technology (NAIST), the copyright holders, disclaims all warranties with regard to this software, including all implied warranties of merchantability and fitness, in no event shall NAIST be liable for any special, indirect or consequential damages or any damages whatsoever resulting from loss of use, data or profits, whether in an action of contract, negligence or other tortuous action, arising out of or in connection with the use or performance of this software.

A large portion of the dictionary entries originate from ICOT Free Software. The following conditions for ICOT Free Software applies to the current dictionary as well.

Each User may also freely distribute the Program, whether in its original form or modified, to any third party or parties, PROVIDED that the provisions of Section 3 ("NO WARRANTY") will ALWAYS appear on, or be attached to, the Program, which is distributed substantially in the same form as set out herein and that such intended distribution, if actually made, will neither violate or otherwise contravene any of the laws and regulations of the countries having jurisdiction over the User or the intended distribution itself.

NO WARRANTY

The program was produced on an experimental basis in the course of the research and development conducted during the project and is provided to users as so produced on an experimental basis. Accordingly,

the program is provided without any warranty whatsoever, whether express, implied, statutory or otherwise. The term "warranty" used herein includes, but is not limited to, any warranty of the quality, performance, merchantability and fitness for a particular purpose of the program and the nonexistence of any infringement or violation of any right of any third party.

Each user of the program will agree and understand, and be deemed to have agreed and understood, that there is no warranty whatsoever for the program and, accordingly, the entire risk arising from or otherwise connected with the program is assumed by the user.

Therefore, neither ICOT, the copyright holder, or any other organization that participated in or was otherwise related to the development of the program and their respective officials, directors, officers and other employees shall be held liable for any and all damages, including, without limitation, general, special, incidental and consequential damages, arising out of or otherwise in connection with the use or inability to use the program or any product, material or result produced or otherwise obtained by using the program, regardless of whether they have been advised of, or otherwise had knowledge of, the possibility of such damages at any time during the project or thereafter. Each user will be deemed to have agreed to the foregoing by his or her commencement of use of the program. The term "use" as used herein includes, but is not limited to, the use, modification, copying and distribution of the program and the production of secondary products from the program.

In the case where the program, whether in its original form or modified, was distributed or delivered to or received by a user from any person, organization or entity other than ICOT, unless it makes or grants independently of ICOT any specific warranty to the user in writing, such person, organization or entity, will also be exempted from and not be held liable to the user for any such damages as noted above as far as the program is concerned.

----------COPYING.ipadic-----END----------

Product Privacy Policy

LEGAL DISCLOSURE

In Brief

Our goal is to adhere to the highest standards of ethics and professionalism in all we do; a key component of this goal is to display respect for our customers' privacy. This statement details our policies and practices relating to the use of any personally identifying information we collect from our customers. We collect only the personally identifiable information from our customers appropriate for a professional business relationship and to enable us effectively manage software usage and customer relationships. We do not collect personally identifiable information without our customers' knowledge, and we do not sell, rent, or in any way redistribute personal information for purposes of solicitation to anyone outside of Kapow's organization or our direct business partners.

In Detail

What information we collect

Using a "call home" feature built into RoboServer and Management Console, we collect and track the following information upon software startup and on a daily basis from our customers: company name, email address, application name, its version and edition numbers, IP address, license key, number of steps executed, number of HTTP Request, number of bytes loaded, number of robots executed, number of returned values, number of stored values, number of HBase puts, number of domains accessed (but not their names), usage of legacy environments, number of schedules, Kapplets, robots, projects, clusters,

RoboServers, OAuth Users, OAuth Applications in the Management Console, number of schedule runs and Kapplet run in Management Console, number of CPUs that RoboServer is running on and the operating system for the server. This is information our customers voluntarily provide to us upon execution of a sales order and/or software license agreement, and that is required for us to manage their software usage and maintain an enterprise solutions business relationship with their organizations.

In addition we collect a number software usage parameters and statistics to get information helping us to improve and develop the software. These consist counters and indicators of usage of certain features of the product. This helps us to determine the importance of such features and, in case of legacy features, if these can be discontinued and removed from the product.

What we do with that information

We use the information to build statistics and help us monitor/manage our customers' usage of their software and compliance with their contractual obligations to Kapow. The collected statistics will also be used as a basis to understand customer use patterns and to continuously improve our products.

Sharing of your personal information

We respect our customers' privacy and therefore we do not sell, rent or in any other way share your personal or company-specific information with unaffiliated outside parties (publications, list brokers, etc.). The information provided to us may be shared with various departments across our organization as required for communication or compliance management purposes; with our closely affiliated business partners (i.e. third parties authorized to sell and support our software) for similar use; or with designated persons as required to comply with a court order. We may from time to time share aggregated demographic data (for example, "20% of our customers are manufacturers of machinery or capital equipment") about our customer base with outside parties, and we may share personally identifiable information with certain outside parties such as industry publications or analysts for specific purposes and with your permission (for example, arranging for an editor to contact you about writing an article on your company's use of our enterprise solutions). We may occasionally identify customer company names and use selected quotes, with attribution, in materials such as company presentations.

Protection from intrusion and theft

We necessarily collect some personal and company-specific information from our customers, and other interested parties who voluntarily provide us with such information. We appreciate the trust that our customers place in us to safeguard this data. While no one can absolutely guarantee that the information that they collect, online or offline, will not be compromised due to electronic intrusion or theft, our company works to protect customer data by:

- following security best practices;
- utilizing intrusion detection and prevention tools (such as firewalls);
- monitoring industry security alerts; and
- applying vendor security patches to our operating systems and applications.

Restricting access to information

You are not restricted from implementing a firewall to block Kapow's retrieval of information. If the information we attempt to collect is blocked by a firewall and, thus, not received by Kapow’s tracker.kapowtech.com system, there will be no negative impact or degradation of performance of that customer’s RoboServer software.

Changes to this policy

We may from time to time amend this Privacy Policy, without notice, to you. Therefore, you should review this policy periodically to assure that you are aware of what personal information we collect and how we use it.

How to contact us

Please address any questions or comments on this Privacy Policy or related matters to Kapow Technologies, Inc., 260 Sheridan Avenue, Suite 420, Palo Alto, CA 94306.