monkeyrunner

The monkeyrunner tool provides an API for writing programs that control an Android device or emulator from outside of Android code. With monkeyrunner, you can write a Python program that installs an Android application or test package, runs it, sends keystrokes to it, takes screenshots of its user interface, and stores screenshots on the workstation. The monkeyrunner tool is primarily designed to test applications and devices at the functional/framework level and for running unit test suites, but you are free to use it for other purposes.

The monkeyrunner tool is not related to the UI/Application Exerciser Monkey (/tools/help/monkey.html), also known as the `monkey` tool. The `monkey` tool runs in an `adb` (/tools/help/adb.html) shell directly on the device or emulator and generates pseudo-random streams of user and system events. In comparison, the monkeyrunner tool controls devices and emulators from a workstation by sending specific commands and events from an API.

The monkeyrunner tool provides these unique features for Android testing:

- Multiple device control: The monkeyrunner API can apply one or more test suites across multiple devices or emulators. You can physically attach all the devices or start up all the emulators (or both) at once, connect to each one in turn programmatically, and then run one or more tests. You can also start up an emulator configuration programmatically, run one or more tests, and then shut down the emulator.
- Functional testing: monkeyrunner can run an automated start-to-finish test of an Android application. You provide input values with keystrokes or touch events, and view the results as screenshots.
- Regression testing - monkeyrunner can test application stability by running an application and comparing its output screenshots to a set of screenshots that are known to be correct.
- Extensible automation - Since monkeyrunner is an API toolkit, you can develop an entire system of Python-based modules and programs for controlling Android devices. Besides using the monkeyrunner API itself, you can use the standard Python `os` and `subprocess` modules to call Android tools such as Android Debug Bridge.

You can also add your own classes to the monkeyrunner API. This is described in more detail in the section Extending monkeyrunner with plugins (#Plugins).

The monkeyrunner tool uses Jython (http://www.jython.org/), a implementation of Python that uses the Java programming language. Jython allows the monkeyrunner API to interact easily with the Android framework. With Jython you can use Python syntax to access the constants, classes, and methods of the API.

IN THIS DOCUMENT

SEE ALSO

Testing Fundamentals

A Simple monkeyrunner Program

Here is a simple monkeyrunner program that connects to a device, creating a `MonkeyDevice` (/tools/help/MonkeyDevice.html) object. Using the `MonkeyDevice` object, the program installs an Android application package, runs one of its activities, and sends key events to the activity. The program then takes a screenshot of the result, creating a `MonkeyImage` (/tools/help/MonkeyImage.html) object. From this object, the program writes out a `.png` file containing the screenshot.

```
# Imports the monkeyrunner modules used by this program
from com.android.monkeyrunner import MonkeyRunner, MonkeyDevice

# Connects to the current device, returning a MonkeyDevice object
device = MonkeyRunner.waitForConnection()

# Installs the Android package. Notice that this method returns a boolean, so you can test
# to see if the installation worked.
device.installPackage('myproject/bin/MyApplication.apk')

# sets a variable with the package's internal name
package = 'com.example.android.myapplication'

# sets a variable with the name of an Activity in the package
activity = 'com.example.android.myapplication.MainActivity'

# sets the name of the component to start
runComponent = package + '/' + activity
```

```
# Runs the component
device.startActivity(component=runComponent)

# Presses the Menu button
device.press('KEYCODE_MENU', MonkeyDevice.DOWN_AND_UP)

# Takes a screenshot
result = device.takeSnapshot()

# Writes the screenshot to a file
result.writeToFile('myproject/shot1.png','png')
```

The monkeyrunner API

The monkeyrunner API is contained in three modules in the package `com.android.monkeyrunner`:

- `MonkeyRunner`: A class of utility methods for monkeyrunner programs. This class provides a method for connecting monkeyrunner to a device or emulator. It also provides methods for creating UIs for a monkeyrunner program and for displaying the built-in help.
- `MonkeyDevice`: Represents a device or emulator. This class provides methods for installing and uninstalling packages, starting an Activity, and sending keyboard or touch events to an application. You also use this class to run test packages.
- `MonkeyImage`: Represents a screen capture image. This class provides methods for capturing screens, converting bitmap images to various formats, comparing two MonkeyImage objects, and writing an image to a file.

In a Python program, you access each class as a Python module. The monkeyrunner tool does not import these modules automatically. To import a module, use the Python `from` statement:

```
from com.android.monkeyrunner import <module>
```

where `<module>` is the class name you want to import. You can import more than one module in the same `from` statement by separating the module names with commas.

Running monkeyrunner

You can either run monkeyrunner programs from a file, or enter monkeyrunner statements in an interactive session. You do both by invoking the `monkeyrunner` command which is found in the `tools/` subdirectory of your SDK directory. If you provide a filename as an argument, the `monkeyrunner` command runs the file's contents as a Python program; otherwise, it starts an interactive session.

The syntax of the `monkeyrunner` command is

```
monkeyrunner -plugin <plugin_jar> <program_filename> <program_options>
```

Table 1 explains the flags and arguments.

Table 1. `monkeyrunner` flags and arguments.

Argument	Description
`-plugin <plugin_jar>`	(Optional) Specifies a `.jar` file containing a plugin for monkeyrunner. To learn more about monkeyrunner plugins, see Extending monkeyrunner with plugins. To specify more than one file, include the argument multiple times.
`<program_filename>`	If you provide this argument, the `monkeyrunner` command runs the contents of the file as a Python program. If the argument is not provided, the command starts an interactive session.
`<program_options>`	(Optional) Flags and arguments for the program in <program_file>.

monkeyrunner Built-in Help

You can generate an API reference for monkeyrunner by running:

```
monkeyrunner help.py <format> <outfile>
```

The arguments are:

- `<format>` is either `text` for plain text output or `html` for HTML output.
- `<outfile>` is a path-qualified name for the output file.

Extending monkeyrunner with Plugins

You can extend the monkeyrunner API with classes you write in the Java programming language and build into one or more `.jar` files. You can use this feature to extend the monkeyrunner API with your own classes or to extend the existing classes. You can also use this feature to initialize the monkeyrunner environment.

To provide a plugin to monkeyrunner, invoke the `monkeyrunner` command with the `–plugin <plugin_jar>` argument described in table 1 (#table1).

In your plugin code, you can import and extend the the main monkeyrunner classes `MonkeyDevice`, `MonkeyImage`, and `MonkeyRunner` in `com.android.monkeyrunner` (see The monkeyrunner API (#APIClasses)).

Note that plugins do not give you access to the Android SDK. You can't import packages such as `com.android.app`. This is because monkeyrunner interacts with the device or emulator below the level of the framework APIs.

The plugin startup class

The `.jar` file for a plugin can specify a class that is instantiated before script processing starts. To specify this class, add the key `MonkeyRunnerStartupRunner` to the `.jar` file's manifest. The value should be the name of the class to run at startup. The following snippet shows how you would do this within an `ant` build script:

```
<jar jarfile="myplugin" basedir="${build.dir}">
<manifest>
<attribute name="MonkeyRunnerStartupRunner" value="com.myapp.myplugin"/>
</manifest>
</jar>
```

To get access to monkeyrunner's runtime environment, the startup class can implement `com.google.common.base.Predicate<PythonInterpreter>`. For example, this class sets up some variables in the default namespace:

```
package com.android.example;

import com.google.common.base.Predicate;
import org.python.util.PythonInterpreter;

public class Main implements Predicate<PythonInterpreter> {
    @Override
    public boolean apply(PythonInterpreter anInterpreter) {

        /*
        * Examples of creating and initializing variables in the monkeyrunner environment's
        * namespace. During execution, the monkeyrunner program can refer to the variables "n
        * and "use_emulator"
        *
        */
        anInterpreter.set("newtest", "enabled");
        anInterpreter.set("use_emulator", 1);

        return true;
    }
}
```