



Mobile applications testing using Python

An experimental technique

Eing Ong, Intuit Inc.

Session outline



- ⦿ Introduction
- ⦿ Simulator basics
- ⦿ **M**obile **e**nd-to-end **t**esting (Moet)
- ⦿ Building your mobile tests
- ⦿ Demo
- ⦿ Advantages and limitations
- ⦿ Q & A

What are we solving for



- ⦿ Diverse mobile platforms
- ⦿ Low cost solution
- ⦿ End-to-end mobile tests
- ⦿ Leverage black box testers

Simulator Basics



● BlackBerry™

- Starting simulator

fledge.exe

/app=jvm.dll

/session=<model>

/app-param=

JvmAlxConfigFile:<model>.xml

/handheld=<model>

- Communicating with simulator

fledgecontroller.exe /session=<model>



Simulator commands



Actions	Steps
Start 9630 Tour simulator	<pre>fledge.exe /app=jvm.dll /session=9630 /handheld=9630 /app- param=JvmAlxConfigFile:9630.xml</pre>
Install application	<ol style="list-style-type: none">1. Copy app.jar, app.jad, app.cod to Javaloader directory2. JavaLoader.exe -u load app.jad3. Delete app.jar, app.jad, app.cod
Save screenshot as test.png in \$TEST_OUTPUT	<ol style="list-style-type: none">1. JavaLoader.exe -u screenshot test.png2. mv test.png \$TEST_OUTPUT

bblib.py



Actions	Steps	bblib.py
Start 9630 Tour simulator	fledge.exe /app=jvm.dll /session=9630 /handheld=9630 /app-param=JvmAlxConfigFile:9630.xml	fledgeStart()
Install application	1. Copy app.jar, app.jad, app.cod to Javaloader directory 2. JavaLoader.exe -u load app.jad 3. Delete app.jar, app.jad, app.cod	install()
Save screenshot as test.png in \$TEST_OUTPUT	1. JavaLoader.exe -u screenshot test.png 2. mv test.png \$TEST_OUTPUT	screenshot('test')

Simulator commands



Action	Steps
Enter 'Hello World'	StringInjection(Hello) KeyPress(SPACE) KeyRelease(SPACE) StringInjection(World)
Touch screen at (10, 100)	TouchScreenPress(10, 100, 0) TouchScreenClick() TouchScreenUnclick() TouchScreenUnpress(0)
Thumbwheel up twice	ThumbWheelRoll(-1) ThumbWheelRoll(-1)

bblib.py



Action	Steps	bblib.py
Enter 'Hello World'	StringInjection(Hello) KeyPress(SPACE) KeyRelease(SPACE) StringInjection(World)	enter('Hello World')
Touch screen at (10, 100)	TouchScreenPress(10, 100, 0) TouchScreenClick() TouchScreenUnclick() TouchScreenUnpress(0)	touch(10, 100)
Thumbwheel up twice	ThumbWheelRoll(-1) ThumbWheelRoll(-1)	thumbwheel ('up', 2)

Simulator Basics



◎ Android™

- Create AVD

`$ANDROID_HOME/tools/android`

- Starting emulator

`emulator -avd <avd>`

- Communicating with emulator

`adb shell`



Simulator command



Action	Steps
Enter 'Hello World'	<pre>adb shell "sendevent /dev/input/event0 1 42 1; sendevent /dev/input/event0 1 42 0; sendevent /dev/input/event0 1 35 1; sendevent /dev/input/event0 1 35 0; sendevent /dev/input/event0 1 18 1; sendevent /dev/input/event0 1 18 0; sendevent /dev/input/event0 1 38 1; sendevent /dev/input/event0 1 38 0; sendevent /dev/input/event0 1 38 1; sendevent /dev/input/event0 1 38 0; sendevent /dev/input/event0 1 24 1; sendevent /dev/input/event0 1 24 0; ... "</pre>

androidlib.py



Action	Steps	androidlib.py
Enter 'Hello World'	<pre>adb shell "sendevent /dev/input/event0 1 42 1; sendevent /dev/input/event0 1 42 0; sendevent /dev/input/event0 1 35 1; sendevent /dev/input/event0 1 35 0; sendevent /dev/input/event0 1 18 1; sendevent /dev/input/event0 1 18 0; sendevent /dev/input/event0 1 38 1; sendevent /dev/input/event0 1 38 0; sendevent /dev/input/event0 1 38 1; sendevent /dev/input/event0 1 38 0; sendevent /dev/input/event0 1 24 1; sendevent /dev/input/event0 1 24 0; ... "</pre>	<pre>enter('Hello World')</pre>

Moet



© Mobile end-to-End testing

- Open sourced on github
- Simulator libraries

androidlib.py

bblib.py

- Image processing library

imagelib.py

- Testing utilities library

testlib.py

logger.py

Moet Framework



Mobile Application Interface



Device Independent Tests



Runtime binding



Simulator libraries

Android app library

androidlib.py

BlackBerry app library

bblib.py

Test Automation Overview



1. Define application interface

This interface is device-agnostic.

2. Implement the interface

Implement the interface in your supported devices e.g. Android.

Utilize python mobile libraries e.g. androidlib.py.

3. Write your tests

Tests are device independent and reusable on all supported devices.

4. Run

Step 1 : Define app interface



class AppInterface:

""" Application interface for all devices to implement """

def add(self, contact):

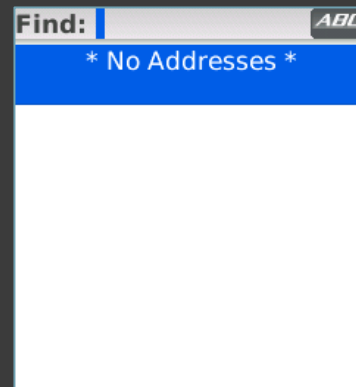
"""Add contact """

def find(self, contact):

""" Find contact """

def delete(self, contact):

"""Delete contact """



Test Automation Overview



1. Define application interface

This interface is device-agnostic.

2. Implement the interface

Implement the interface in your supported devices.

Utilize most libraries.

3. Write your tests

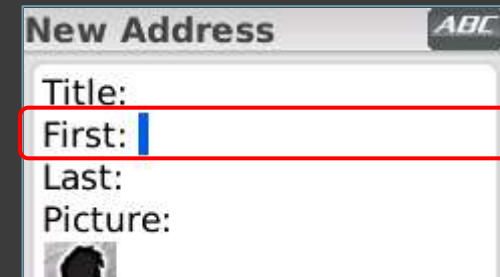
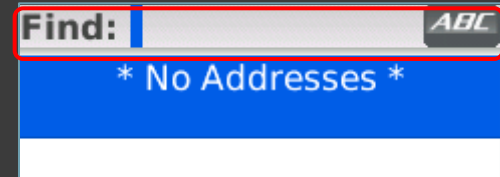
Tests are device independent and reusable on all supported devices.

4. Run

Step 2 (Pearl) : Implement the interface



```
def add(self, contact):  
    """ Add contact """  
  
    # click add contact  
    enter()  
  
    # enter name  
    enter(contact.getFirstname())  
    thumbwheel('down', 1)  
    ...  
    # save  
    menu()  
    enter()
```



Step 2 (Android) : Implement the interface

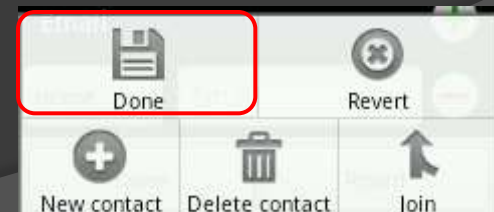
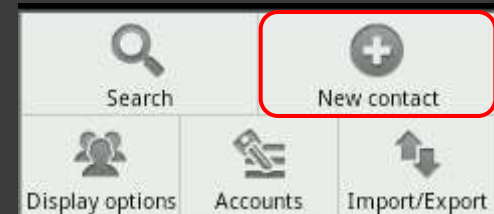


```
def add(self, contact):  
    """ Add contact """
```

```
    # click add contact  
    menu()  
    scroll('up')  
    scroll('right')  
    enter()
```

```
    # enter name  
    enter(contact.getFirstname())  
    scroll('down')
```

```
    ...  
    # save  
    menu()  
    scroll('down')  
    enter()
```



Step 2 (recap) : Implement the interface



```
def PearlImpl(appbase.AppInterface):  
    def add(self, contact):  
        """ Add contact """  
        enter()  
        enter(contact.getFirstname())  
        thumbwheel('down', 1)  
        ...  
        menu()  
        enter()
```

```
def AndroidImpl(appbase.AppInterface):  
    def add(self, contact):  
        """ Add contact """  
        menu()  
        scroll('up')  
        scroll('right')  
        enter()  
        enter(contact.getFirstname())  
        scroll('down')  
        ...  
        menu()  
        scroll('down')  
        enter()
```

Test Automation Overview



1. Define application interface

This interface is device-agnostic.

2. Implement the interface

Implement the interface in your supported devices e.g. Android.

Utilize python mobile libraries e.g. androidlib.py.

3. Write your tests

Tests are device independent and reusable on all supported devices.

4. Run

Step 3 : Writing tests



```
class AddContactTest(unittest.TestCase):
```

```
    device = testenv.getDeviceClass()
```

```
    def addContactWithOnlyFirstnameTest(self):
```

```
        self.contact.setFirstname(firstname)
```

```
        self.device.add(self.contact)
```

```
    def addContactWithOnlyLastnameTest(self):
```

```
        self.contact.setLastname(lastname)
```

```
        self.device.add(self.contact)
```

Step 3 : Runtime binding



```
def getDeviceClass(self):
```

```
    """ Returns the device to test """
```

```
    mobileDevice = self.getMobileDevice()
```

```
    if mobileDevice == 'pearl':
```

```
        import pearl
```

```
        deviceClass = pearl.PearlImpl()
```

```
    elif mobileDevice == 'android':
```

```
        import android
```

```
        deviceClass = android.AndroidImpl()
```

```
    return deviceClass
```

More device-independent tests



Additional tests are easy to write

```
def addContactWithEmailTest(self):  
def addContactWithAddressesTest(self):  
def addContactWithAllDetailsTest(self):  
def addContactWithLongDetailsTest(self):  
def addContactAddressWithStateZip(self):  
def addContactAddressWithCityStateZip(self):  
def addContactAddressWithNoDataNegativeTest(self):
```

Step 4 : Run



- ◎ Basic run command
 - `python <test.py>`

- ◎ Python test frameworks
 - unittest
 - PyUnit
 - python-nose

Test Verification



⦿ Server hosted apps

- API assertions
- Database assertions

⦿ Image assertions

```
self.assertTrue(
    imagelib.compare(
        self.device, testname, '100%x90%', tolerance))
    # Crop settings examples
    # 100%x80%+10%+20% (crop size + offset)
    # 320x90+0+0
    # +0+90
```



Test Logging



● Logs

AddressTest.log :

2010-06-10 15:19:46,773 - **testCreateAddressMethod** - INFO -

[Address] 200 Villa St Mountain View CA 94040 BUSINESS ADDRESS

● Initialization

```
self.log = self.device.initLogger(self._testMethodName,  
                                   self.__class__.__name__)
```

● Usage

```
self.log.info('Starting test: ' + self._testMethodName)  
self.log.debug(self.contact)  
self.log.error('Missing image to compare')
```

Demo



- ⦿ Simulators
 - Android
 - BlackBerry Pearl
- ⦿ Moet
- ⦿ Test automation
 - Address book app
 - Add contact
 - Find contact
 - Delete contact



Advantages



- ⦿ Low cost and ease of use
- ⦿ Reusable end-to-end tests
- ⦿ No device sharing/scheduling
- ⦿ Bigger device pool
- ⦿ Reduce manual testing time
- ⦿ Run on developer machines
- ⦿ Debugging capabilities

Limitations



- ⦿ Requires ethernet or internet connectivity
- ⦿ Does not simulate network performance
- ⦿ Does not support hardware controls testing
- ⦿ Dependent on simulator reliability
- ⦿ Limited peer-to-peer applications testing

Resources



Moet <http://github.com/moet/moet/>

Android®

Emulator <http://developer.android.com/guide/developing/tools/emulator.html>

ADB <http://android-dls.com/wiki/index.php?title=ADB>

Forum <http://developer.android.com/resources/community-groups.html>

BlackBerry®

Downloads <http://na.blackberry.com/eng/developers/javaappdev/javadevenv.jsp>

Fledge Controller

http://docs.blackberry.com/en/developers/deliverables/6338/Testing_apps_using_the_BB_Smrtphn_Smltr_607559_11.jsp

Forum <http://supportforums.blackberry.com/>

Q & A





Thanks and enjoy the rest of PyCon 2011

For more details on the presentation, contact
devcon@intuit.com
twitter @eingong