



This article aims to give developers ideas about how to better find the sources of errors in their PyS60 applications, as default error reports can be hard to understand and standalone applications don't even have error reports.

Contents

- [1 Method 1: Confirmation](#)
- [2 Method 2: Exception Harness](#)
- [3 Method 3: Log File](#)
- [4 Method 4: Exceptions from Callbacks](#)
- [5 Method 5: Trace](#)
- [6 Method 6: Trace with Exceptions from Callbacks](#)

Method 1: Confirmation

A simple way of finding out what is wrong in a sequence of code is to show a confirmation message after an operation is successfully performed. The message can either be printed on the screen or spoken by the phone.

For example, let's say we want to alphabetize a list and append an element to it:

```
import appuifw, e32, audio

l=['alpha', 'beta', 'lambda', 'gamma']
n=len(l)
i=0
while(i<n-1):
    j=i+1
    while(j<n):
        if(l[i]>l[j]):
            a=l[i]
            l[i]=l[j]
            l[j]=a
        j+=1
    i+=1

#If all goes well to this point, we can print a message...
print "The list has been alphabetized"

#... or have the phone tell us
audio.say("The list has been alphabetized")

l.append('omega')

print "The element 'omega' has been added to the list"

applock=e32.Ao_lock()
applock.wait()          #Tell the application not to terminate immediately
```

Method 2: Exception Harness

Known as the "exception harness", this method is more useful for standalone applications. It mimics a stack trace of the Python Script Shell.

```
try:
    # Actual program is here.
    1 / 0
except:
    import sys
    import traceback
    import e32
    import appuifw
    appuifw.app.screen="normal"           # Restore screen to normal size.
    appuifw.app.focus=None               # Disable focus callback.
    body=appuifw.Text()
    appuifw.app.body=body                 # Create and use a text control.
    applock=e32.Ao_lock()
    def quit():applock.signal()
    appuifw.app.exit_key_handler=quit     # Override softkey handler.
    appuifw.app.menu=[(u"Exit", quit)]    # Override application menu.
    body.set(unicode("\n".join(traceback.format_exception(*sys.exc_info()))))
    applock.wait()                       # Wait for exit key to be pressed.
    appuifw.app.set_exit()
```

Method 3: Log File

Here we record the error to a text log file.

```
def main():
    try:
        Your main code goes here.
    except:
        import sys
        import traceback
        import appuifw
        cla, exc, trbk = sys.exc_info()
        excName = cla.__name__
        try:
            excArgs = exc.__dict__["args"]
        except KeyError:
            excArgs = "<no args>"
        excTb = traceback.format_tb(trbk, 5)
        errorString = repr(excName) + '-' + repr(excArgs) + '-' + repr(excTb) + '\n'
        print errorString
        appuifw.note(u'Application errors, see log file for more information', "error")
        file = open(u'C:\\Log.txt', 'a')
        file.write(errorString)
        file.close()
        raise

if __name__ == "__main__":
    main()
```

Here the errors will be recorded in the Log file at C:\\Log.txt

Method 4: Exceptions from Callbacks

Exceptions from callbacks are not propagated to the "main thread" (in quotes because no real threading is going on in the background, only Symbian active objects).

You can wrap each callback in a function that collects the exception in a global list and signals the lock you're waiting. You can then handle exceptions collected in the list sequentially when the lock is signalled.

```
#
# developed by jethro.fn.
#

import traceback

# Exceptions from callbacks are collected here.
exceptions = []

# Decorator for callback functions
def collect_exception(func):
    # Using Python (>v2.2) nested scopes.
    def callit(*args, **kwds):
        global script_lock
        try:
            return func(*args, **kwds)
        except:
            excstr = "\n".join(traceback.format_exception(*sys.exc_info()))
            exceptions.append(excstr) # Store exception string in a list.
            script_lock.signal()     # Notify main function about exception.
            raise                   # Re-raise exception, just in case.
    return callit

...

def main():
    global exceptions, script_lock

    script_lock = e32.Ao_lock()

    ...

    appuifw.app.menu = [(u'Raise NameError',
                        collect_exception(raise_nameerror)),
                       (u'Exit',
                        script_lock.signal)]

    ...

    script_lock.wait()

    # Report exceptions. Could be more than one, in theory.
    for excstr in exceptions:
        appuifw.note(unicode(excstr), 'error')
    exceptions = [] # All exceptions accounted for, clear list.
```

Method 5: Trace

```

#
# GPL license : traceS60.py by cyke64
#

import sys
import linecache
from e32 import ao_sleep
refresh=lambda:ao_sleep(0)

class trace:
    def __init__(self,runfile,f_all=u'e:\\traceit.txt',f_main=u'e:\\traceitmain.txt'):
        self.out_all=open(f_all,'w')
        self.out_main=open(f_main,'w')
        self.runfile=runfile

    def go(self):
        sys.settrace(self.traceit)

    def stop(self):
        sys.settrace(None)
        self.out_all.close()
        self.out_main.close()

    def traceit(self,frame, event, arg):
        lineno = frame.f_lineno
        name = frame.f_globals["__name__"]
        if "__file__" in frame.f_globals:
            file_trace=frame.f_globals["__file__"]
            line=linecache.getline(file_trace,lineno)
        else:
            file_trace=self.runfile
            line=linecache.getline(file_trace,lineno)
        self.out_main.write("%s*s*\n*s*\n" %(event,lineno,line.rstrip()))
        self.out_all.write("%s*s*of %s(%s)\n*s*\n" %(event,lineno,name,file_trace,line.rstrip()))
        refresh()
        return self.traceit

```

method use example :

```

import random
import traceS60

def main():
    print "In main"
    for i in range(5):
        print i, random.randrange(0, 10)
    print "Done."

# very important give in trace object the REAL path of your script !

tr=traceS60.trace('e:\\system\\apps\\python\\my\\silly_trace.py')
tr.go()
main()
tr.stop()

```

Method 6: Trace with Exceptions from Callbacks

Mixing method 5 and 6 for a complete PyS60 debugging tool !

```
#
# Code mixed by Hiisi
#
import sys
import e32
import appuifw
import linecache

# Debugging Level (0: Release, 1: Show Traceback, 2: Trace All)
debuglevel = 1

# Exceptions from callbacks are stored here.
exceptions = []

class Trace:
    """Class for tracing script

    This class is developed by cyke64.
    Lisenced under GNU GPL.
    """
    def __init__(self, runfile, f_all=u'E:\\Python\\traceit.txt',
                 f_main=u'E:\\Python\\traceitmain.txt'):
        self.out_all=open(f_all, 'w')
        self.out_main=open(f_main, 'w')
        self.runfile = runfile

    def go(self):
        sys.settrace(self.traceit)

    def stop(self):
        sys.settrace(None)
        self.out_all.close()
        self.out_main.close()

    def traceit(self, frame, event, arg):
        lineno = frame.f_lineno
        name = frame.f_globals['__name__']
        if '__file__' in frame.f_globals:
            file_trace=frame.f_globals['__file__']
            line = linecache.getline(file_trace, lineno)
        else:
            file_trace = self.runfile
            line = linecache.getline(file_trace, lineno)
        self.out_main.write('%s*s*\n*s*\n' \
                           % (event, lineno, line.rstrip()))
        self.out_all.write('%s*s*of %s(%)\n*s*\n' \
                           % (event, lineno, name, file_trace, line.rstrip()))
        e32.ao_sleep(0)
        return self.traceit

    def call_callback(func):
        """Catch exception

        This function is developed by jethro.fn.
        """
        def call_func(*args, **kwds):
            import traceback
```

Python_debugging_techniques

```
global exceptions
global script_lock
try:
    return func(*args, **kwds)
except:
    # Collect Exceptions
    exception = ''.join(traceback.format_exception(*sys.exc_info()))
    exceptions.append(exception)

    # Signal lock in main thread (immediate termination)
    if debuglevel == 2: script_lock.signal()
return call_func

def raise_nameerror():
    """Raise NameError"""
    # foo = u'bar'
    foo

def main():
    """Main thread"""
    global script_lock
    global exceptions

    # main UI
    script_lock = e32.Ao_lock()
    appuifw.app.title = u'RaiseErrorTest'
    appuifw.app.body = appuifw.Text()
    appuifw.app.menu = [(u'Raise NameError',
                        call_callback(raise_nameerror)),
                       (u'Exit', script_lock.signal)]
    appuifw.app.exit_key_handler = script_lock.signal
    script_lock.wait()

    # Handling exception
    if debuglevel and exceptions:
        show_traceback(exceptions)
        exceptions = []

def show_traceback(exceptions):
    """Show traceback"""
    appuifw.app.title = u'Traceback'
    traceback = '%d error(s) occurred.\n\n' % len(exceptions)
    for exception in exceptions:
        traceback += '%s\n' % exception
    body = appuifw.Text(unicode(traceback))
    body.set_pos(0)
    appuifw.app.body = body
    appuifw.app.menu = [(u'Exit', script_lock.signal)]
    script_lock.wait()

if __name__ == '__main__':
    try:
        if debuglevel == 2:
            trace = Trace('E:\\Python\\RaiseErrorTest.py')
            trace.go()
            main()
            trace.stop()
        else:
            main()
    except:
        raise
```

External links :

- [traceS60](#)
- [exception callback capture](#)