Simple Procedure for Adding Smart BASIC to Xcode

[Current as of December 02, 2016]

Smart BASIC allows you to create genuine iOS applications which can be valid for App Store submission or Ad-Hoc distribution.

Although all iOS applications should be compiled with Xcode, you will not need to write any single line in Objective C language - you will simply compile a template Xcode project together with your smart BASIC program code and any support files such as images or sound files.

Of course you will need an Apple Developer License and knowledge of how to use Xcode for compiling and code signing your applications. But these topics are out of scope of smart BASIC support and should be referred to respective Apple help resources.

This tutorial explains step by step how to create an iOS application from a short example smart BASIC program called Turtle.

1. Copy the following text of the BASIC program below and create a text file named "turtle.txt" from it or open the attached file in Adobe.

```
' Turtle graphics 20141001
' Coded for Smart Basic by Henko
' Slow Turtle modification by Mr.K
' Dutchman added settings, slider and auto-scaling
' Shadow added by Henko (on special request by Mr.K)
'===== settings ======
delay=0.0
shadows=0 ' 1 or 0
direction=1 ' 1=ClockWise, -1=counterCW
bcolor(1,1,0) ' background r,g,b
dcolor(1,0,0) ' drawing color
border=4 ' size of edge
'---------------------
sw=screen_width()
sh=screen_height()
size=min(sw,sh)/2.5
dy=size/20 'offset for slider
t_init(shadows)
count=countvalue
begin:
angle=-direction*360/count
t.s=3*size/count
for i=1 to count
  for j=1 to count ! t_step(angle) ! pause delay ! next j
  t_turn(angle,0)
  next i
```

```
' wait for slider change
wait:
if button_pressed("stop") then end
if slider_changed("count") then
  do ! until slider_changed("count")<>1
  newcount=countvalue
else ! goto wait
endif
if newcount=count then wait
count=newcount
clearcanvas(border)
goto begin
end
def countvalue
countvalue=3+int(18*slider_value("count"))
end def
def t
x=0 ! y=0 ! s=0 ! a=0
end def
def t_move(continue)
dis=continue*t.s ! xdis=dis*cos(t.a) ! ydis=dis*sin(t.a)
t.x+=xdis ! t.y-=ydis
draw line to t.x,t.y
end def
def t_step(angle)
t.a+=angle ! t_move(1)
end def
def t_turn(angle,continue)
t.a+=angle ! t_move(continue)
end def
def bcolor(r,g,b)
.rb=r ! .gb=g ! .bb=b
end def
def dcolor(r,g,b)
.rt=r ! .gt=g ! .bt=b
end def
def clearcanvas(edge)
graphics clear 0,0,0
fill color .rb,.gb,.bb
fill rect edge,edge to 2*.size-edge,2*.size-edge
end def
def t_init(withshadow)
graphics ! graphics clear ! option angle degrees
.xc=.size ! .yc=.size
t.x=.xc ! t.y=.yc ! t.a=0
sprite "turtle" begin 2*.size,2*.size
clearcanvas(.border)
sprite end
option sprite pos central
sprite "turtle" at .sw/2,.sh/2-.dy
sprite "turtle" show
sprite "turtle" begin
iy=.sh/2+.size ! bwidth=.size/4 ! bx=.sw/2+.size-bwidth
slider "count" value 0.5 at .sw/2-.size,iy hsize 2*.size-bwidth
set buttons custom ! draw color 1,1,0 ! fill color 1,0,0
button "stop" title "STOP" at bx,iy-dy size bwidth,dy
draw color .rt,.gt,.bt ! draw size 2
if withshadow then shadow on
draw to t.x,t.y
end def
```

(Original code available for download at http://kibernetik.pro/forum/viewtopic.php?p=3493#p3493

2. Download smart BASIC SDK for Xcode from the following location:

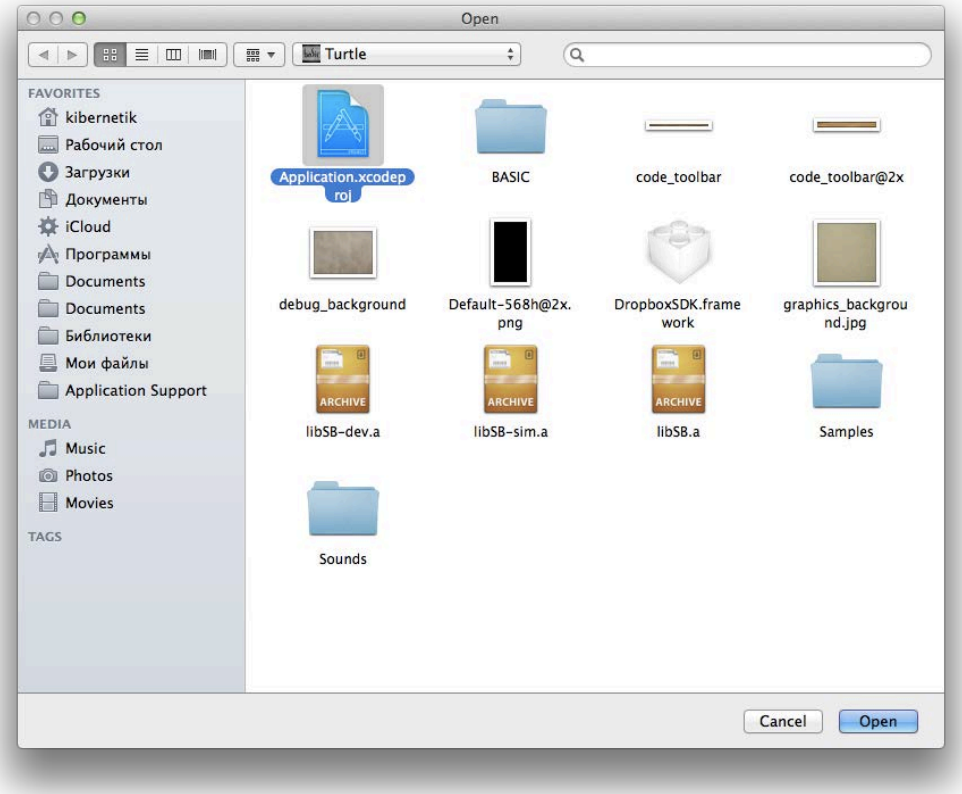
   http://kibernetik.pro/BASIC%20SDK.zip

3. Decompress the contents of the zip file. There will be a single folder named “BASIC SDK” containing several files and folders. This folder is your template folder for all future program projects.

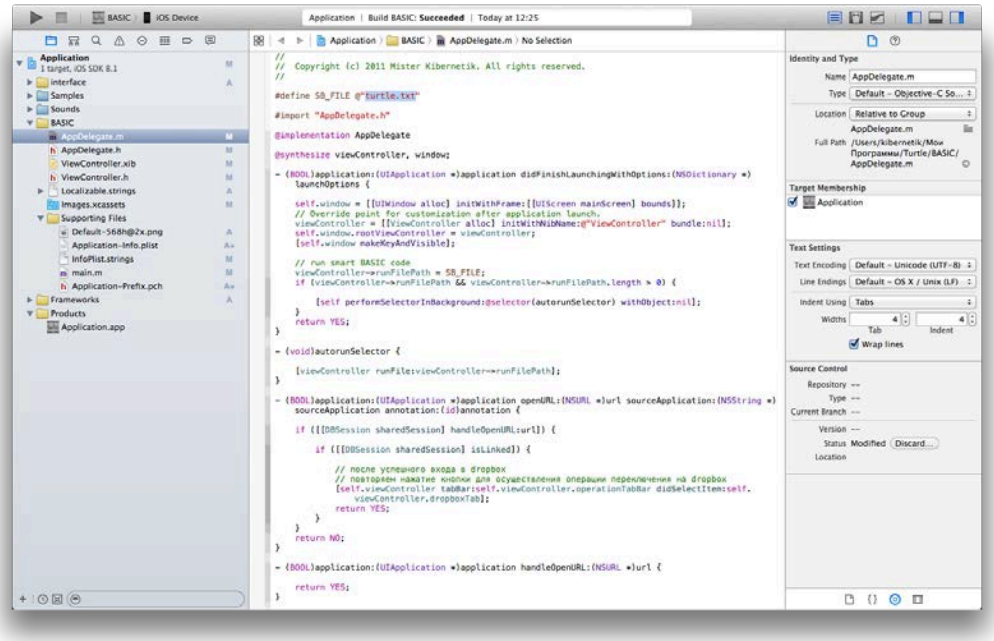
4. Rename the template folder name from "BASIC SDK" to "Turtle" for this example.

5. Copy the text file "turtle.txt" that you created in step #1 to the template folder "Turtle/Samples" folder.

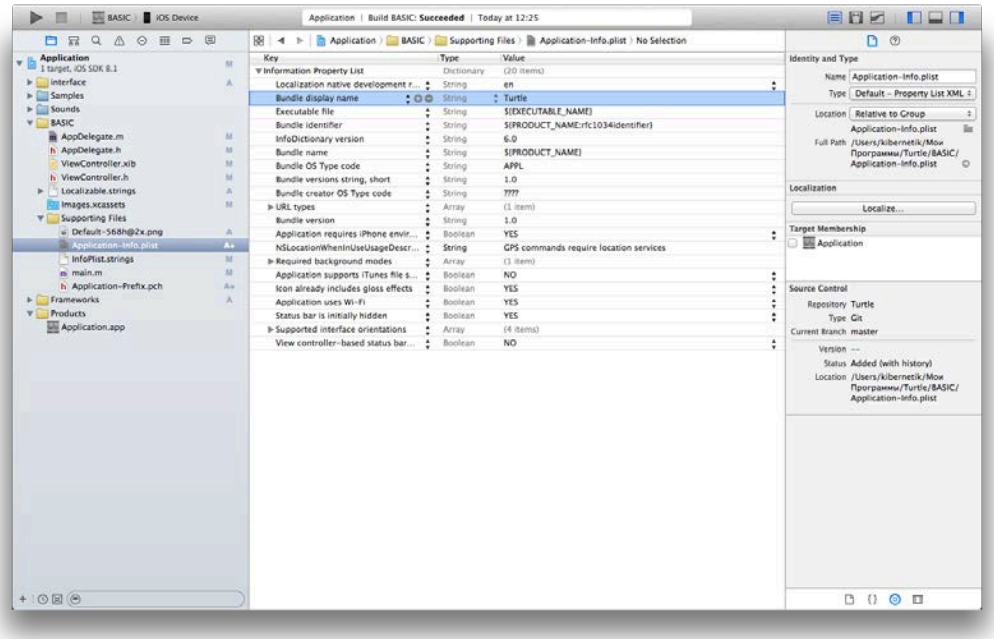
6. In Finder, open the template project in Xcode by double-clicking the file named “Application.xcodeproj” in the newly renamed Turtle template folder.



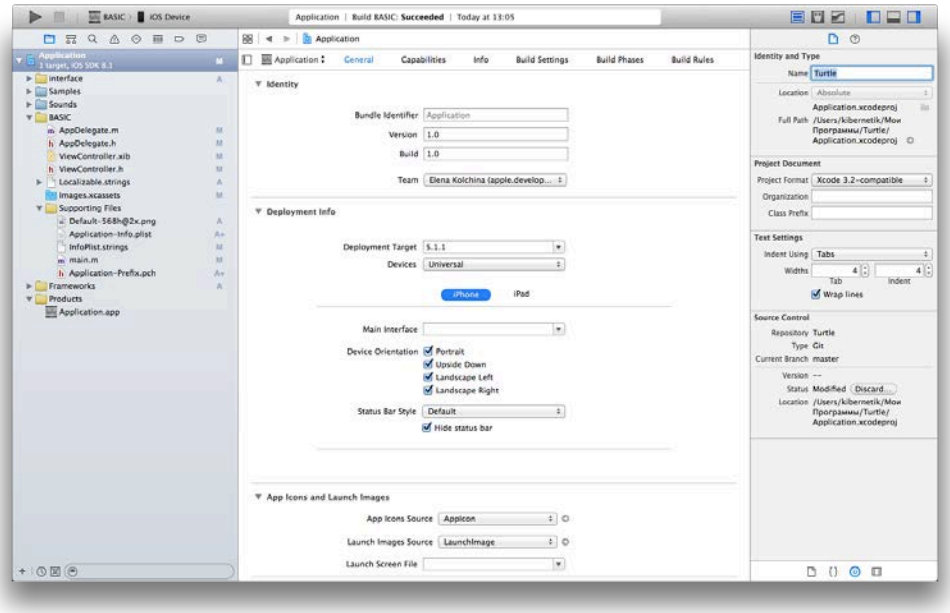
7. In Xcode, select the file "BASIC/AppDelegate.m" and enter the filename “turtle.txt” inside the empty quotes for the line “define SB_FILE @...”



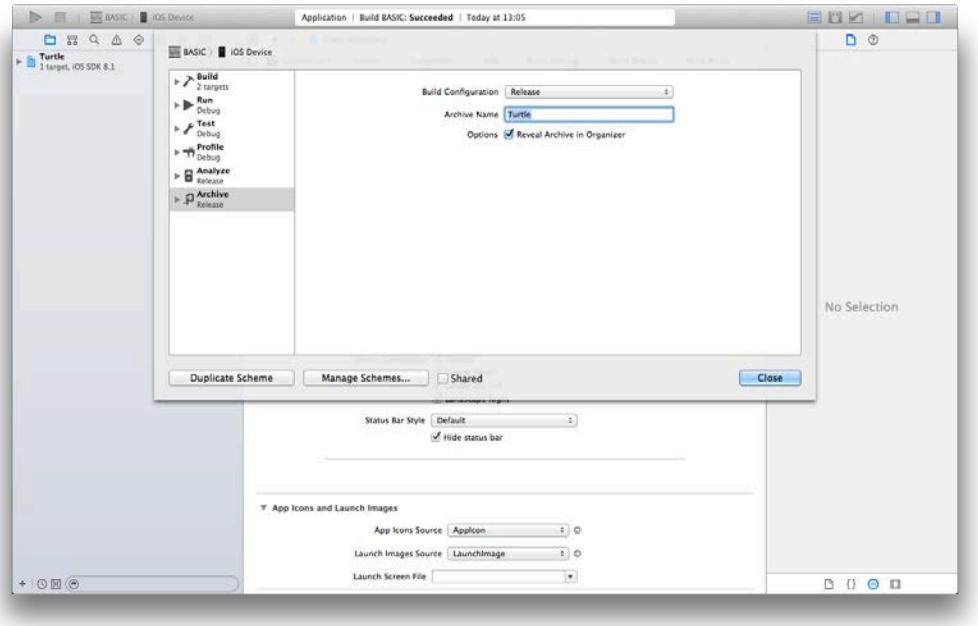
8. In the file "BASIC/Supporting Files/Application-Info.plist", change the value for "Bundle display name" from the default "Application" to "Turtle".



9. Update the project Name field from the default "Application" to "Turtle".



10. Select the Xcode menu item **Product > Scheme > Edit Scheme...** and in the Archive section, update the default name "Application" to "Turtle".



11. Finally, compile the project by selecting from the Xcode menu **Product > Run**. This will save all of your settings, compile the code and run it in the selected simulator.

App Distribution Options

You can now do anything you want with your application; compile it for App Store submission, for Ad-Hoc distribution or test it in the iOS simulator.

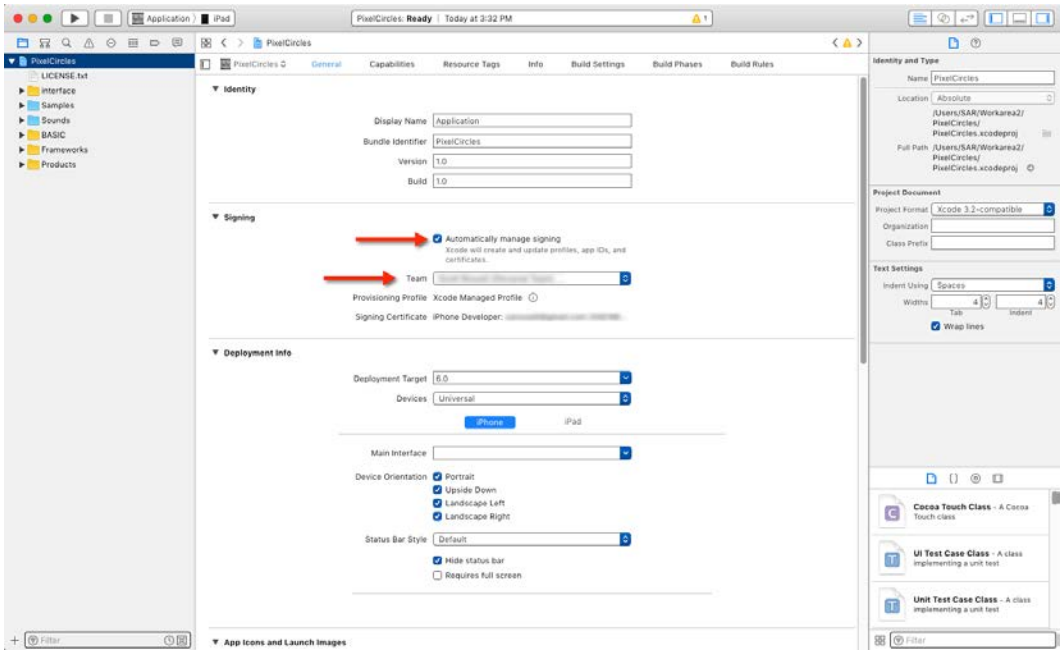
Compiling for distribution through the Apple App Store is beyond the scope of this document and requires the purchase of an Apple Developer's License.

Ad-Hoc distribution is for private testing of your app on your own devices and can be accomplished very easily. You can also run your app in an Xcode iOS simulator. Depending on the power of you computer and the simulator version, the simulators can be either slower or faster than the actual device.

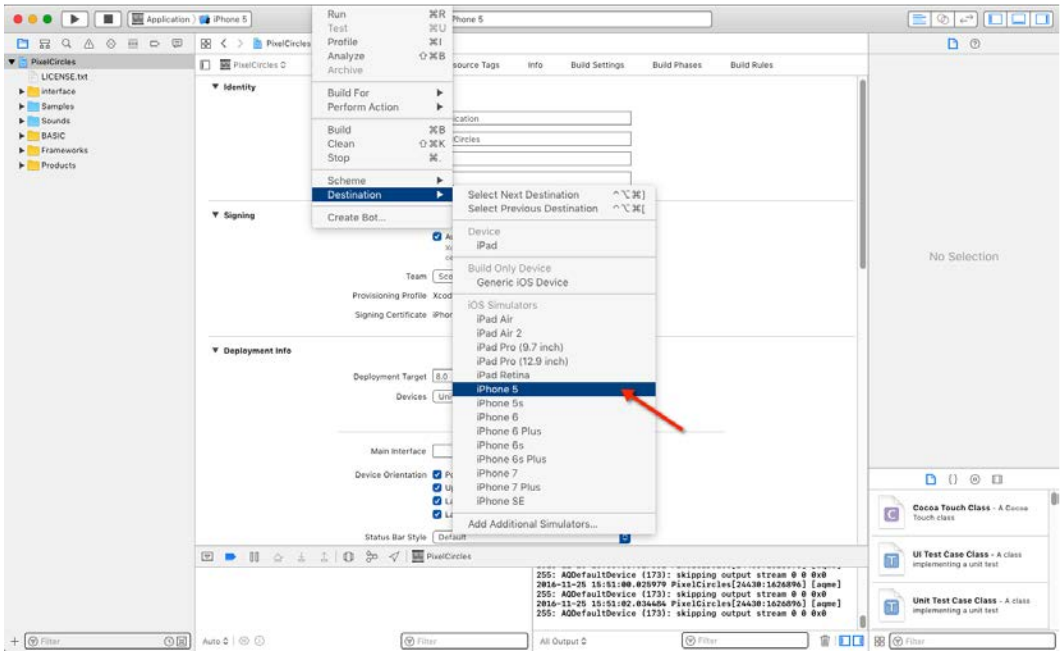
The procedures for Ad-Hoc and Simulator testing are similar with just a couple of differences.

Simulator Testing

1. In the Project Editor under **General: Signing**, make sure the option "**Automatically manage signing**" is selected, then for the **Team** field, use the pull down to select your ID. (If you haven't already entered your Apple ID into Xcode, select the button to do so, then choose your ID from the drop down menu for Team. You can also enter it from the main menu in **Xcode > Preferences: Accounts**.)



2. Select the desired iOS simulator from the menu item **Product > Destination**.



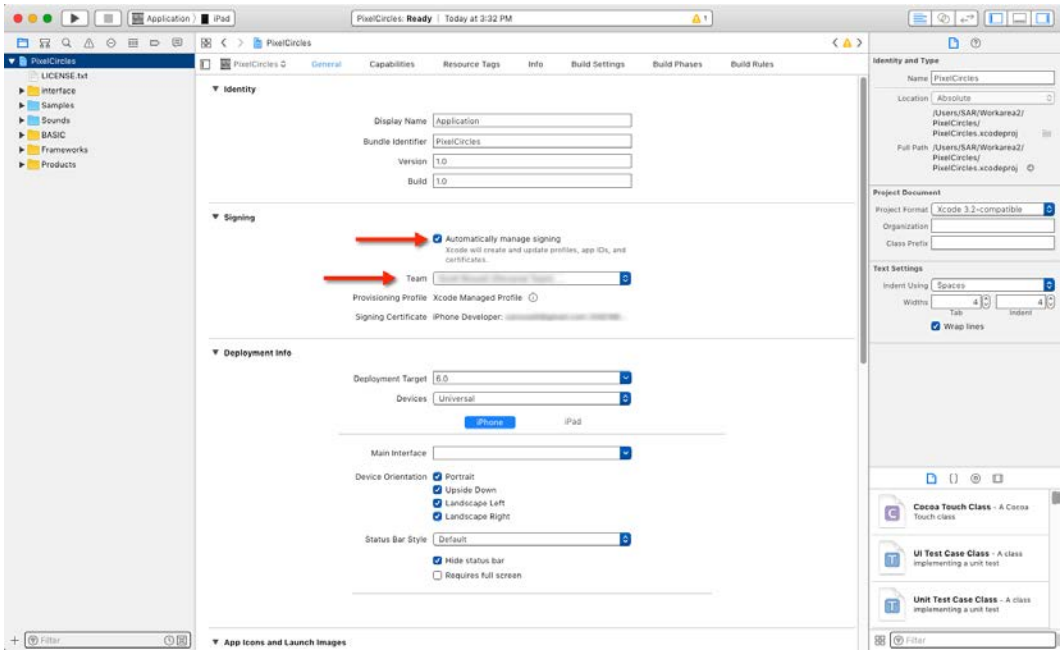
3. From the main menu, select **Product > Run**. The app will compile then start the iOS simulator and automatically run your app.

Ad-Hoc iDevice Testing

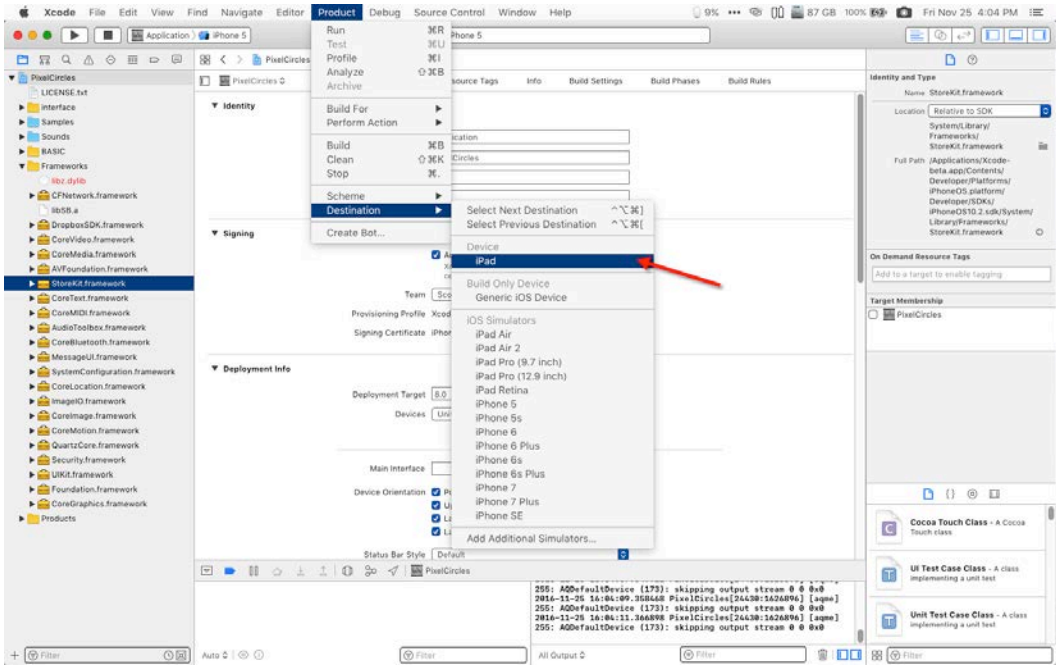
There are just two minor differences when compiling for an actual iDevice. The first difference is the library file choice in the BASIC SDK template folder. The template folder contains three library files; “libSB.a”, “libSB-dev.a”(device), and “libSB-sim.a”(simulator). By default the “libSB.a” file is an exact duplicate of the “libSB-sim.a” file. This is because Xcode uses the “libSB.a” file for the build, but you have to choose which library to use depending on the destination. For a simulator build, you don’t have to do anything. The “libSB.a” file is already a renamed copy of the “libSB-sim.a” file. But for ad-hoc distribution to a device, you must rename the “libSB-dev.a” file to “libSB.a” (replacing any existing copy) before building the project.

The second difference is you have to plug your iDevice into the computer for Xcode to recognize it as a build destination.

1. In the Project Editor under **General: Signing**, make sure the option "**Automatically manage signing**" is selected, then for the **Team** field, use the pull down to select your ID. (If you haven't already entered your Apple ID into Xcode, select the button to do so, then choose your ID from the drop down menu for Team. You can also enter it from the main menu in **Xcode > Preferences: Accounts**.)



2. Plug in your iDevice (iPhone, iPad, etc) using the USB cable.
3. Select your iDevice from the menu item **Product > Destination**.



4. In Finder, rename the “libSB-dev.a” file to “libSB.a”.

5. From the main menu, select **Product > Run**. The app will compile and start the app on your iDevice.

Optimizing App Size

There is one file in the SDK BASIC template that stands out as particularly large: default_bank.sf2 in the Sounds directory. If your application will not be using the MIDI NOTES commands to play sounds, you can replace this file with a specially created empty version of the file of the same name. A copy of this file is included in this PDF in the Attachments on the left.

Simply replace the existing “default_bank.sf2” file with the special empty file named “empty_default_bank.sf2” and then rename it to “default_bank.sf2”.

Your resulting compiled apps will no longer be able to produce realistic sampled MIDI sounds using the NOTES command, but you can still play individual audio files using the MUSIC command and your app will be substantially smaller in size.

NOTES:

a) These procedures was tested using the latest Xcode version 8.1 (8B62) and 8.2 beta running on macOS Sierra.

b) Xcode 8.x does not support compiling code for iOS versions older than 8.0. To do so, you must install a separate previous version of Xcode. You can also run multiple copies of Xcode on the same computer.

The bulk of this tutorial was copied from the forum entry entitled “Adding smart BASIC to Xcode Tutorial” on the Mr. Kibernetik Software forum at http://kibernetik.pro/forum/viewtopic.php?f=34&t=726 dated Oct 27, 2014. Subsequent edits, updates and PDF conversion by Scott A. Rossell.