iPhone port of GHC v1.3 – Installation and user's guide

Stephen Blackheath 30 Apr 2010

Revision History

1.0 - 18 Jun 2009 – Initial release

1.1 – 8 Jul 2009 – Fixed a problem where iphone*-cabal would install libraries under ~/.cabal into a directory that clashed with stock cabal.

1.2 – 13 Oct 2009 – Fixed a problem where some packages wouldn't build due to an error finding -ldl

1.3 – 30 Apr 2010 – 'unlit' executable was missing which meant some packages whose source code was in literate Haskell wouldn't build.

Introduction

This document describes installation and usage for the port of the GHC (Glasgow Haskell Compiler) version 6.10.2 to the iPhone platform provided by Blackheath's Virtual Dog Grooming Co. Ltd.

Installation

You can install from a binary image or build the compiler yourself from source. Please choose which one, then follow the instructions below.

Binary installation

1. Install standard GHC from http://haskell.org/ghc/.

2. As root, expand ghc-iphone.tar.bz2 into the root directory /. It creates a directory called /opt/iphone

3. Add the bin directory to your PATH, like this:

   ```
   export PATH="/opt/iphone/bin:$PATH"
   ```

4. Now you can use these commands. Mostly you won't be using these directly, because of the Xcode integration, described later.

   iphone-cabal
   iphone-ghc
   iphone-ghc-pkg
   iphone-simulator-cabal
   iphone-simulator-ghc
   iphone-simulator-ghc-pkg

Note: There is no iphone version of hsc2hs – iphone-cabal will use the standard GHC's version, and will compile for iPhone using the iPhone simulator compiler. So far this works (surprisingly) well.

Source installation

1. Install standard GHC from http://haskell.org/ghc/. The installation has been tested with version 6.10.2.

2. Install the packages **alex** and **happy** from hackage. The best thing to do is to install **cabal-install** first, then type

   ```
   cabal install alex
   cabal install happy
   ```

3. Obtain these files from haskell.org:

   ```
   HTTP-4000.0.5.tar.gz
   cabal-install-0.6.2.tar.gz
   ghc-6.10.2-src-extralibs.tar.bz2
   ghc-6.10.2-src.tar.bz2
   zlib-0.5.0.0.tar.gz
   ```

4. Remove extraneous stuff from your PATH. I don't know exactly what the problematic program is, but having MacPorts stuff in your PATH makes 'configure' break. You need $HOME/.cabal/bin, though, because that's where **alex** and **happy** live.

5. Create yourself a directory, then type this (substituting the directory where your files are):

   ```
   tar xfj ~/packages/iphone/ghc-6.10.2-src.tar.bz2
   tar xfj ~/packages/iphone/ghc-6.10.2-src-extralibs.tar.bz2
   cp ~/packages/iphone/*.gz ghc-6.10.2/
   cat ~/packages/iphone/ghc-iphone.patch | patch -p0
   cd ghc-6.10.2
   sh ./config-iphone.sh
   make
   sudo make install
   ```

6. Now repeat the exact process above in a new directory, except with `./config-iphone-simulator.sh`. Since it's a different directory, you can run both builds in parallel.

To create a binary distribution: sudo make install DESTDIR=<absolute_path>

Sample projects

The distribution includes two sample projects:

- HaskellDraw
- GetInBehind

These can be loaded straight into Xcode and built. They will run unmodified on the iPhone simulator, but to run on the iPhone device, you will need to configure code signing on the project according to Apple's requirements (see below).

Setting up a new Xcode project – General

In the instructions below, <project> needs to be substituted with the name of your project.

On the device – disable thumb mode

You must disable thumb mode, otherwise you'll get "thumb bl/blx out of range (4302672 max is +/-4M)"

- Right click on top level of project tree
  - Select Get Info
  - Select Build tab

- Search for 'thumb' and uncheck 'Compile for Thumb'

Setting up code signing on the iPhone

In order for your application to run on the iPhone device, Apple requires this to be done correctly. This isn't required for the simulator.

- Go to http://developer.apple.com/iphone/
- Log in
- Navigate to "iPhone Developer Program Portal" (top right)
- Select "Provisioning" at left
- Select "How To" at top
- Scroll down top and open "Building and Installing your Development Application"
- Follow the instructions

Setting up a new Xcode project - GHC/Xcode integration

Add Haskell target

Make the Xcode project automatically build the Haskell project. First, you need a skeleton Haskell source and .cabal file.

- Select Project > New Target
- Select "Other" on the left
- On the top right, select "External Target"
- Click Next
- For "Target Name" enter the name "Haskell" here (or whatever you prefer)
- Click Finish
- Double-click on Targets / Haskell in the project tree.
- Change "Build Tool" to /opt/iphone/bin/build-iphone-haskell.sh
- "Arguments" is ignored, but you can leave it set to "$(ACTION)"
- Make "Haskell" a dependency of your main target. To do this, double-click on your main target, Targets / <project> in the project tree.
- In the General tab, drag your Haskell target into the 'Direct Dependencies' list.

Build the Haskell target

We need to do this before we can do the next step of adding the Haskell binary.

- Click "Build and go >" to build the Haskell
- Double click on "Errors and Warnings" (in the project tree) and see if you got any Haskell errors. Click on the "text page" icon in this window and you'll see the text output of the Haskll build. Once the Haskell has built with no errors, continue to the next section.
- The link will most likely fail at this stage, but that doesn't matter - we just need the Haskell compile to complete at this point.

Add the Haskell binary to the project tree

This allows Xcode's linker to find the Haskell code.

- In the project tree, right click on Targets / <project> / Link Binary With Libraries
- Select Add > Existing Files
- Select the file called build/<platform>/dist/build/<project>/lib<project>.a (where <platform> is something like "Debug-iphoneos" and <project> is the name of your

project, e.g. "HaskellDraw")
- Leave everything in the dialog box as default and click Add.
- Right click the newly added item in the tree (under Link binary With Libraries) and select Get Info
- In the General tab, select: Path Type: Relative to Build Product
    - This will make it always choose the correct path if you change your platform or switch between Debug/Release.
- Close the dialog box

Fix “Library search paths”

The process above where we add the Haskell code leaves some paths configured that will prevent seamless switching between iPhone and iPhone simulator platforms.

Here's how to fix it:

- In project tree, right mouse on Targets / <project> and select Get Info
- In Build tab, scroll down to "Library Search Paths" section.
- Double click on it, and ensure that it doesn't contain any paths specific to a target platform, such as "$(SRCROOT)/build/Debug-iphoneos/dist/build/HaskellDraw" (specific to iphoneos). Remove these.
- Add $(BUILT_PRODUCTS_DIR)/dist/build/<project>
- The only other entry that should be there is "$(inherited)", unless you want to add ones of your own.

The project should now automatically run the correct version of cabal for your target (simulator or real iphone), and it should also clean the Haskell 'dist' directory when you clean the project in Xcode. If this doesn't work, then re-check the instructions above.

iPhone Debugger's “spurious SIGTRAP” issue

For some reason, when running your application in the debugger on the iPhone device, the Haskell application receives spurious SIGTRAP signals, which causes the debugger to stop without any message. This generally happens during Haskell initialization, so the application doesn't get very far at all.

This can be fixed in the following way:

Create a file called ~/.gdbinit (that is, .gdbinit in your home directory), containing this line:

```
handle SIGTRAP nostop noprint
```

Now the application will run properly if you click "Build and go" and you will see any stdout/stderr output of the program in Xcode's Console window. This probably breaks the Xcode debugger's ability to set breakpoints.

Note that when running in the debugger like this on the iPhone device, the application is about half as fast as when it runs by launching it from an icon.

Some general notes on iPhone/Xcode

Hopefully these may save you some trouble

- If you double click on "Errors and Warnings", you can see the build happening in the window that comes up.

- To see if Xcode is talking to your iPhone, go to menu item Window / Organizer It should

have a green bullet next to the device. If it isn't working, you might need to restart (that is, completely power down and re-start) the iPhone, which involves holding down the small button on the top left of the iPhone for a few seconds.

- If you've set up the authentication, but you still get ApplicationVerificationFailed, try this:
  - Menu item: Build > Clean All Targets
  - See http://www.lensenergy.com/archives/2009/01/iphone-dev-tip1---applicationv.html

- If you get "ApplicationAlreadyInstalled" then uninstall the application on the iPhone:
  - Hold your finger on the app icon for a few seconds until everything goes wobbly
  - Click the (X) at the top left and confirm that you want to uninstall it
  - Push the big button to get rid of wobbliness.

- By default Xcode will insert tab characters into your Haskell source, which will make things not work. To fix this:
  - Select Preferences from the Xcode menu
  - Select the Indentation tab
  - Uncheck "Tab key inserts tab, not spaces"

Pool sizes for foreign function "wrapper" callbacks

A declaration for an "wrapper" callback looks like this:

```
foreign import ccall safe "wrapper"
        mkDelegate :: IO () -> IO (FunPtr (IO ()))
```

To implement these, GHC normally generates a small piece of executable code at runtime, called an "adjustor". The purpose of these "wrapper" declarations is to generate a C-callable function pointer that executes a Haskell closure.

Apple has a code-signing policy, that says that all code that runs on the iPhone must be signed by Apple. GHC wants to generate code at runtime, and this can't be signed by Apple. This is enforced by the iPhone's kernel, so GHC's standard implementation is impossible.

This iPhone port solves the problem by pre-compiling a pool of functions, and allocating from it. Because the pool size for each wrapper is fixed, this creates the problem that the pool can run out. If this happens, the application will die with this message:

```
HaskellDraw: internal error: iPhoneCreateAdjustor - adjustor pool
'Main_d1tU' is empty (capacity 32)
    (GHC version 6.10.2-iphone for arm_apple_darwin)
```

(The name of the module where the "wrapper" was defined appears before the underscore character.)

Each "wrapper" declaration has its own pool, whose size defaults to 32. This means that at any one time, there can exist no more than 32 adjustors created by the defined function (in this example, mkDelegate). The standard library function `freeHaskellFunPtr` frees an adjustor.

If the pool is too small for a given application, you can increase it by using a `{-# POOLSIZE x #-}` pragma, which must appear after the "wrapper" token. e.g.

```
foreign import ccall safe "wrapper" {-# POOLSIZE 100 #-}
        mkDelegate :: IO () -> IO (FunPtr (IO ()))
```

Because pool sizes are limited, it should be considered unsafe to call `freeHaskellFunPtr` in a finalizer, because garbage collection is not predictable. If this functionality is desired, it may

be possible to modify GHC-iPhone to force a garbage collection and try again. However, the current implementation does not do this.

The `{-# P00LSIZE x #-}` pragma is NOT standard GHC – it is specific to this iPhone GHC port. If you use it in standard GHC, it will report a warning "`test.hs:5:36 Unrecognised pragma`", but will otherwise work normally. This keeps code written for iPhone fully portable to the standard GHC.