Native Data Handling

What is native code for Android devices, and what is the NDK?

Android apps run within the Dalvik virtual machine, which interprets device-agnostic, cross-platform commands into instructions for the specific device that it is running on.The speed and memory overhead is a worthwhile tradeoff. In some cases, developers need the absolute fastest performance possible. The NDK allows embedding C and C++ components within Android apps, allowing the most performance-intensive pieces to be as close to the hardware as possible. This comes at a cost, though — using native code complicates development. There are more tools to use and infrastructure to set up. Also, some details that were handled by the Dalvik virtual machine must now be handled by the developer. For these reasons, native code should be used only when necessary.

When native code is needed

There are times that using native code can be advantageous, such as processing data or computing physics and graphics for games. Access to existing native libraries, as well as high-performance code, can also be good reasons.

Uses for the native code

Game engine developers often dive right in to native code. The limited speed and memory of mobile devices means native code may be necessary to squeeze every bit of potential out for them

The native-activity sample resides under the NDK installation root, in samples/native-activity. It is a very simple example of a purely native application, with no Java source code. In the absence of any Java source, the Java compiler still creates an executable stub for the virtual machine to run.

```
#include <EGL/egl.h>
#include <GLES/gl.h>
```

#include <android/sensor.h>
#include <android/log.h>
#include <android_native_app_glue>

Create a new Native Activity project

In this tutorial, you'll first create a new Android Native Activity project and then build and run the default app in the Visual Studio Emulator for Android.

To create a new project

1. Open Visual Studio. On the menu bar, choose **File**, **New**, **Project**.
2. In the **New Project** dialog box, under **Templates**, choose **Visual C++**, **Cross Platform**, and then choose the **Native-Activity Application (Android)** template.
3. Give the app a name like MyAndroidApp, and then choose **OK**.

   Visual Studio creates the new solution and opens Solution Explorer.

The new Android Native Activity app solution includes two projects:

- **MyAndroidApp.NativeActivity** contains the references and glue code for your app to run as a Native Activity on Android. The implementation of the entry points from the glue code are in main.cpp. Precompiled headers are in pch.h. This Native Activity app project is compiled into a shared library .so file which is picked up by the Packaging project.
- **MyAndroidApp.Packaging** creates the .apk file for deployment on an Android device or emulator. This contains the resources and AndroidManifest.xml file where you set manifest properties. It also contains the build.xml file that controls the Ant build process. It's set as the startup project by default, so that it can be deployed and run directly from Visual Studio.

Build and run the default Android Native Activity app

Build and run the app generated by the template to verify your installation and setup. For this initial test, run the app on one of the device profiles installed by the Visual Studio Emulator for Android. If you prefer to test your app on another target, you can load the target emulator or connect the device to your computer.

To build and run the default Native Activity app

1. If it is not already selected, choose **x86** from the **Solution Platforms** dropdown list.

   If the **Solution Platforms** list isn't showing, choose **Solution Platforms** from the **Add/Remove Buttons** list, and then choose your platform.

2. On the menu bar, choose **Build**, **Build Solution**.

The Output window displays the output of the build process for the two projects in the solution.

3. Choose one of the VS Emulator Android Phone (x86) profiles as your deployment target.

   If you have installed other emulators or connected an Android device, you can choose them in the deployment target dropdown list.

4. Press F5 to start debugging, or Shift+F5 to start without debugging.

   Visual Studio starts the emulator, which takes a few seconds to load and deploy your code. Once your app has started, you can set breakpoints and use the debugger to step through code, examine locals, and watch values.

5. Press Shift + F5 to stop debugging.

   The emulator is a separate process that continues to run. You can edit, compile, and deploy your code multiple times to the same emulator.