The NVIDIA Tegra 250 Devkit Windows CE 6.0 Platform Support Pack

Version 5265393

Contents

Introduction

The Windows CE 6.0 Tegra OS Support Pack for the NVIDIA Tegra 250 devkit is designed to provide all Tegra-specific components required to support development and use of a Tegra 250 devkit with Windows CE 6.0. It contains the software required to:

- Flash and boot the devkit into Windows CE 6.0
- Compile and link Khronos media API-based applications for a CE 6.0-based Tegra 250 devkit
- Pre-compile shaders for the Tegra platform using a Windows host PC
- Change the display resolution and depth of the devkit's video-out

Note that significant non-NVIDIA software (MS Visual Studio) is required to develop for a CE 6.0-based Tegra 250 devkit. These requirements are detailed in the following sections.

This Support Pack is not a full SDK. It does not contain samples or support libraries; these are supplied separately. It contains only the NVIDIA-specific tools, headers and libraries required for compiling and linking applications for the devkit running CE 6.0. It is a prerequisite for CE 6.0-compatible NVIDIA samples and for application porting to CE 6.0-based Tegra devices.

System Requirements

This pack requires additional hardware and software above and beyond the pack itself and the devkit as shipped. Additional accessories required beyond the ones listed in the next section are described in the NVIDIA Tegra 250 HW Setup Guide.

Hardware

A Windows "host" PC for flashing and development.

NVIDIA recommends:

- At least a ~2.0GHz CPU
- 1-2GB RAM

Software

- Microsoft Windows XP or newer operating system installed on the host PC
  - XP has been extensively tested
  - Vista has been given basic testing
  - Windows 7 has been given basic testing
  - ***Note:*** 64-bit OSes are not currently supported
- Microsoft Visual Studio:
  - Microsoft Visual Studio 2005 *with Smart Device Programmability support installed and Service Pack 1 installed*

  OR
  - Microsoft Visual Studio 2008 *with Smart Device Programmability support installed*
- Microsoft Standard SDK for Windows CE 5.0
- Connectivity
  - Microsoft ActiveSync 4.5 (for Windows XP)

  OR
  - Windows Mobile Device Center 6.1 (for Windows Vista and Windows 7)

Microsoft Visual Studio

Microsoft Visual Studio 2005 or 2008 is required in order to develop for the devkit when it is running CE 6.0. The project files are shipped as Visual Studio 2005 files, but have been tested to auto-convert correctly in Visual Studio 2008.

If you are using Visual Studio 2005, you must install Service Pack 1 for Visual Studio 2005 prior to using the SDK.

Both versions of Visual Studio must have Smart Device Programmability installed in order to develop for Windows CE. If you do not have it installed, you must add the feature from your original install CDs, or reinstall again with Smart Device Programmability checked to install.

Microsoft Standard SDK for Windows CE 5.0

Microsoft's Standard SDK for Windows CE 5.0 (the CE5.0 SDK is used regardless of the exact OS installed on the device – CE5.0, WM6.1 or CE6.0) must also be downloaded, installed, and available in your version of Visual Studio. Visual Studio 2005 does not include that SDK at all. There is the possibility that some versions of Visual Studio 2008 may include the SDK in their Smart Device support, but under our testing the Standard SDK needed to be downloaded and installed separately. **Please make sure that you install Visual Studio before installing Windows CE 5.0 SDK if it needs to be done separately.**

The last-known location to download the Standard SDK for Windows CE 5.0 is: http://www.microsoft.com/downloads/details.aspx?familyid=fa1a3d66-3f61-4ddc-9510-ae450e2318c3&displaylang=en)

If for some reason the Standard SDK is not available at the above link, a web search for "standard sdk for windows ce 5.0" should have as its first or second result a link to the proper page in Microsoft's download center with the most recent version of that SDK.

ActiveSync / Windows Mobile Device Center

In order to connect the host PC and devkit devicevia USB (to copy data between the device and PC or to debug applications running on the devkit from the host PC), Microsoft device connectivity software must be installed and configured.

- On Windows XP, the package is ActiveSync. For more information, see: http://www.microsoft.com/windowsmobile/activesync/activesync45.mspx
- On Windows Vista and Windows 7, the package is Windows Mobile Device Center (WMDC). For more information, see: http://www.microsoft.com/windowsmobile/en-us/downloads/microsoft/device-center-download.mspx

Developers should download and install one of these device connectivity packages on the host PC before continuing. Later sections of this document will lead you through the initial connection of the PC and the devkit over ActiveSync/WMDC.

If you are using Visual Studio 2005, you **must** install Visual Studio 2005 Service Pack 1 to enable debugging via USB. Otherwise, Visual Studio **will be able to deploy** the application to the

devkit, but it **will not be able start the debugging connection**. In this case, it will display a dialog box indicating: "Unable to start debugging". Installing the service pack should fix this issue.

Installing the Support Pack

Uninstalling Previous Tegra SDKs

If you have NVIDIA's NVAP SDK 0.1.x through 0.4.x installed on your host PC, you must first take the following actions to uninstall it and avoid conflicts:

1) Remove any runtime DLL directories from previous NVAP SDKs from the system's `PATH` environment, such as the `platformlibs` paths added when installing a previous NVAP SDK.
2) Remove the old NVAPSDK environment variable as well.

Note that multiple CE6 Platform Support Packs of different versions can be installed to a host PC at once. This allows the developer to switch between multiple OS images during development and testing as needed. See the later section "***Installing Multiple Tegra CE6 Support Packs at Once***"

Installing the Support Pack to the Host PC

To install the Support Pack, simply double click the installation file

```
ce6_tegra_250_<version>.msi
```

Prerequisites Verification

The installer will detect and display the list of installed software required and/or recommended for development on CE 6.0-based devkits. Install any missing software before proceeding with installation. Please refer to the Prerequisites documentation for details.

If you have installed Microsoft Standard SDK for Windows CE 5.0 before installing Microsoft Visual Studio then it might not detect Standard SDK for Windows CE 5.0 since the registry for Visual Studio will not have its entry. Proceed without reinstalling the software. You should not have any issues but if you see any issues while developing (for example not being able to build for target platform) then reinstall the needed software in the right order.

End-User License

Next you will be prompted with End-User License Agreement. Read the agreement and to accept it check the box "I accept the terms in the License Agreement". Click next to proceed.

Installation Type

There are 3 ways to install the Support Pack. Read the instructions on the screen and choose one of the three options.

"Typical" and "Complete" setup options will install the Support Pack in

```
C:\Program Files\NVIDIA Corporation\ ce6_tegra_250_<version>\
```

location by default. If you want to install the Support Pack to a different location then choose the "Custom" setup option, change the installation location and click Next.

Install the new Platform Support Pack to a ***different*** directory tree than any other platform support packs. Overlapping platform packs in a single tree will lead to problems.

Do not move the Support Pack manually after install – the installer sets environment variables that point to the installed location of the Support Pack. If the pack must be moved, uninstall it via the "Add/Remove Programs" control panel and reinstall to the new location from the .msi file.

Following the selection of the installation type, file copying will begin. It can take several minutes to complete the installation. Once installation is complete, click Finish. Since the installer adds and modifies environment variables including the PATH, you will be prompted to acknowledge that restarting the host PC is recommended.

Environment Variables

The installer adds or updates several environment variables to locate the Support Pack:

`NV_WINCE_T2_PLAT`: This environment variable is set to the path of the root of the Support Pack, and can be referenced in VCPROJ files to locate Khronos headers and libraries.

`PATH`: This standard environment variable is modified to append the path to the `host_bin` subdirectory of the Support Pack to the end of the search path. This makes it easier to run the host-based shader-compiler tools.

Layout of the Platform Support Pack

The platform support pack has the following hierarchy on the host PC once installed:

`$(NV_WINCE_T2_PLAT)\`

`bin\`

Contains the subtree NvDispRes, a tool that can be copied onto the devkit and used on the devkit to change the width, height and bits-per-pixel of the video-out. The later section "The NvDispRes Tool" describes how to install and use this tool on the devkit.

`host_bin\`

Contains the `cgc.exe` and `shaderfix.exe` tools that can be used to pre-compile shaders for the Tegra. Note that current Windows CE 6.0 OS images support source shaders directly. While shaders can be pre-compiled, they do not have to be (as was the case in previous developer OS images). This full path is added to the Windows host PC's `$PATH` variable by the installer.

`include\`

Contains standard, Khronos-mandated subdirectories for each Khronos API's headers (e.g. KD for OpenKODE Core). These headers match the ones used to build the drivers included in the Platform Support pack's OS image.

`lib\`

`release\`

Contains CE 6.0-compatible link libraries for the Khronos APIs. These libraries match the ones used to build the drivers included in the Platform Support pack's OS image.

`os\`

Contains the tools and data required to use the Windows host PC to flash the supplied Windows CE 6.0 OS image to the devkit.

Installing Windows CE 6.0 to the Devkit

Once the Platform Support pack is installed to the host PC, it is possible to install ("flash") the included OS image to the devkit. The required scripts and data are contained in the os subdirectory of the Platform Support pack.

Prerequisites

Refer to the diagrams of the connections and the main board in the NVIDIA Tegra 250 HW Setup guide for details required by these sections.

Selecting and Connecting the Desired Display

The windows CE 6.0 OS image can support booting to VGA (15-pin D-Sub) or HDMI (HDMI also supports DVI-D via HDMI-to-DVI-D connectors). The selection of display device is currently an OS flash-time decision. Select your desired video-out option and connect the display to the corresponding jack.

For maximum compatibility, please ensure that your boot display is plugged in before powering on the devkit, so that the OS can detect the display properly during boot. Additionally, if you choose to use a Keyboard-Video-Mouse (KVM) switch to share display and input devices, you should have the devkit's input on the KVM switch selected and active prior to boot to ensure it can properly read the capabilities of the boot display.

Determining the Devkit Hardware Version

There are two major versions of the Tegra 250 devkit: 1GB RAM and 512MB RAM. Most devkits will be 1GB RAM, but in order to ensure that the devkit functions correctly, the exact version must be verified by checking the version ID on a sticker on the main board. The sticker is highlighted in the following image:

The serial number can be decoded as follows:

Printed Number	Hardware Version
600-81162-**5641**-XXX	1GB RAM
600-81162-**5541**-XXX	512MB RAM

Note the variant of your devkit hardware before continuing.

Placing the Devkit into Recovery ("Flashing") Mode

The devkit must be turned on, connected to the host PC and placed into recovery mode. Refer to the NVIDIA Tegra 250 HW Setup guide for details on how to do this.

The Tegra 250 CE 6.0 Platform Support pack includes Windows recovery mode USB drivers in

```
$(NV_WINCE_T2_PLAT)\os\usbpcdriver
```

Flashing the OS Image

Once the display type is selected, the devkit HW version is known and the devkit is in recovery mode, the OS image can be flashed. The Platform Support pack includes four batch files that can be used to flash the OS:

Display Type	Devkit HW Version	Flash Batch Script
CRT	512MB	**nvflash_crt_512mb.bat**
CRT	1GB	**nvflash_crt_1gb.bat**
HDMI	512MB	**Nvflash_hdmi_512mb.bat**
HDMI	1GB	**nvflash_hdmi_1gb.bat**

The flashing process will begin immediately. At the end of a successful flashing, the device will reboot to the desired video out mode with the Windows CE 6.0 desktop. If run from a command prompt (rather than double-clicking the batch file), the resulting output should be similar to the following:

```
>nvflash_crt_1gb.bat
>nvflash.exe -bct devkit_333Mhz_1GB.bct --setbct --bl EBOOT.nb0 --configfile
wince600_nand.cfg --odmdata 0x300011 --create --go
Nvflash started
rcm version 0X20001
System Information:
   chip name: t20
   chip id: 0x20 major: 1 minor: 2
   chip sku: 0x8
   chip uid: 0x097c81c642a11097
   macrovision: disabled
   hdcp: enabled
   sbk burned: false
   dk burned: false
   boot device: nand
   operating mode: 3
   device config strap: 0
   device config fuse: 0
   sdram config strap: 0

sending file: devkit_333Mhz_1GB.bct
- 4080/4080 bytes sent
devkit_333Mhz_1GB.bct sent successfully
odm data: 0x300011
downloading bootloader -- load address: 0x8300000 entry point: 0x8300000
sending file: EBOOT.nb0
/ 2048000/2048000 bytes sent
EBOOT.nb0 sent successfully
waiting for bootloader to initialize
bootloader downloaded successfully
setting device: 1 0
creating partition: BCT
creating partition: PT
```

```
creating partition: EBT
creating partition: BMP
creating partition: CE6
creating partition: ARG
creating partition: DRM
creating partition: UIP
creating partition: USP
creating partition: USR
Formatting partition 2 BCT please wait.. done!
Formatting partition 3 PT please wait.. done!
Formatting partition 4 EBT please wait.. done!
Formatting partition 5 BMP please wait.. done!
Formatting partition 7 CE6 please wait.. done!
Formatting partition 8 ARG please wait.. done!
Formatting partition 9 DRM please wait.. done!
Formatting partition 11 UIP please wait.. done!
Formatting partition 12 USP please wait.. done!
Formatting partition 13 USR please wait.. done!
done!
sending file: EBOOT.NB0
/ 2048000/2048000 bytes sent
EBOOT.NB0 sent successfully
sending file: nvlogo_64x39_rgb565_wheader.raw
- 5008/5008 bytes sent
nvlogo_64x39_rgb565_wheader.raw sent successfully
sending file: NK.NB0
/ 41377132/41377132 bytes sent
NK.NB0 sent successfully
sending file: nvstorage.dat
- 4650/4650 bytes sent
nvstorage.dat sent successfully
```

The Khronos APIs

The Support pack ships with the following Khronos libraries:

OpenKODE Core 1.0: `KD/kd.h`, `libKD.lib`: POSIX-like functionality for files, I/O, etc, along with basic window-system and input-handling functionality. See the Khronos documentation for details.

http://www.khronos.org/registry/kode/specs/openkode.1.0.pdf

OpenGL ES 2.0: `GLES2/gl2.h`, `GLES2/gl2ext.h`, `libGLES20.lib`: Shader-based 3D rendering. See the OpenGL ES 2.0 and GLSL-E 2.0 Shading Language documentation for details.

http://www.khronos.org/files/opengles_spec_2_0.pdf,

http://www.khronos.org/files/opengles_shading_language.pdf

OpenGL ES 1.1: `GLES/gl.h`, `GLES/glext.h`: Fixed-function-based 3D rendering. See the OpenGL ES 1.1 documentation for details.

http://www.khronos.org/registry/gles/specs/1.1/es_cm_spec_1.1.10.pdf

OpenMAX IL 1.1: `openmax/il/*`: Support for hardware-accelerated video, audio and imaging. See the Khronos documentation for details.

http://www.khronos.org/files/openmax_il_spec_1_1_1.pdf

OpenVG 1.0: `VG/*`: Support for vector graphics. See the Khronos documentation for details.

http://www.khronos.org/files/openvg_1_0_1.pdf

EGL 1.3: `EGL/egl.h`: Buffer and Context management, linking OpenKODE Core and OpenGL ES 2.0. See the Khronos documentation for details.

http://www.khronos.org/registry/egl/specs/eglspec.1.3.pdf

Because developing Khronos applications for Tegra is supported across multiple platforms, specific Khronos media features and issues are not listed in this documentation. The cross-platform Tegra-specific Khronos features and issues may be found in Khronos-specific programming documentation on the Tegra developers site.

Building Code for Tegra

In order to build an application for the devkit running Windows CE 6.0, either a new project must be created, or a new platform configuration added to an existing project.

Creating a new Tegra-compatible MSVC++ Project (Preferred)

This is the preferred method. Within MSVC++:

1) Open the "File" menu's "New" and then "Project…" submenus
2) Under "Visual C++", select "Smart Device"
3) Select the desired project type, location and name
4) Click "OK"
5) In the wizard, click "Next"
6) Remove all SDKs from the right column and add only "STANDARDSDK_500"
7) Click "Next"
8) Select desired application/library options
9) Select "Finish"
10) To compile for Tegra, select the configuration "STANDARDSDK_500 (ARMV4I)" (if desired, the other CPU configurations can be removed using the "Configuration Manager…" in the "Build" menu later.)

Adding a Tegra-compatible Target to an Existing MSVC++ Project

Note that this method is not preferred, as it does not add common project settings to the Tegra configuration (e.g. defining the identifier "ARM"). However, to add a Tegra-compatible target to an existing (e.g. WinXP) application project within MSVC++:

1) Open the "Build" menu
2) Select "Configuration Manager…"
3) Under "Active Solution Platform:", select "<New>"
4) In the dialog, select the platform "STANDARDSDK_500 (ARMV4I)"
5) Click the "OK" button to create the new build target

Note that when compiling this project, you may receive the error:

```
"error PRJ0004 : Could not generate command line for the 'VCCLCompilerTool' tool."
```

If so, it is likely caused by the debugging information setting being "edit and continue", which is not supported on this platform. To fix this:

1) Open the "Project" menu
2) Select the "Properties…" option
3) Select the "C/C++: General" tab
4) Change "Debug Information Format" from "4" to a supported drop-down, such as "Program Database"

Adding Khronos API Support to a Tegra-compatible Project

While the previous sections will allow base Windows CE applications to compile, link and run on the devkit, the projects will not yet compile and link with Khronos media APIs. In order to compile and link with Tegra-compatible Khronos headers and libraries, a few paths must be added to the project.

1) Add the following paths to the "C/C++:General:Additional Include Directories" in all of the Tegra configurations in the project:
   `$(NV_WINCE_T2_PLAT)/include`
   (for most Khronos APIs)
   `$(NV_WINCE_T2_PLAT)/include/OpenMax/il`
   (for OpenMAX IL)
2) Add the following paths to the "Linker:General:Additional Library Directories" in all of the Tegra configurations in the project:
   `$(NV_WINCE_T2_PLAT)/lib/release`

The Platform Support pack installer will have already set the variable `$(NV_WINCE_T2_PLAT)` as required. However, after installing the Platform Support pack, exit and restart any open MSVC++ instances, so that they can re-read the environment variables.

Only release Khronos libraries are shipped with the platform support pack, so both release and debug configurations of applications should link with the release Khronos libraries.

Link Libraries for Khronos Applications on Tegra

In order to successfully link an OpenKODE Core app for Tegra, both `libKD.lib` and `libnvkdmain.lib` must be added to the "additional libraries" line. Failure to include latter library will result in "undefined main" link errors.

VS2008 Compile Options

FPU Hardware

The Visual Studio 2008 compiler can generate hardware floating-point code for Tegra if the following steps are taken in each project:

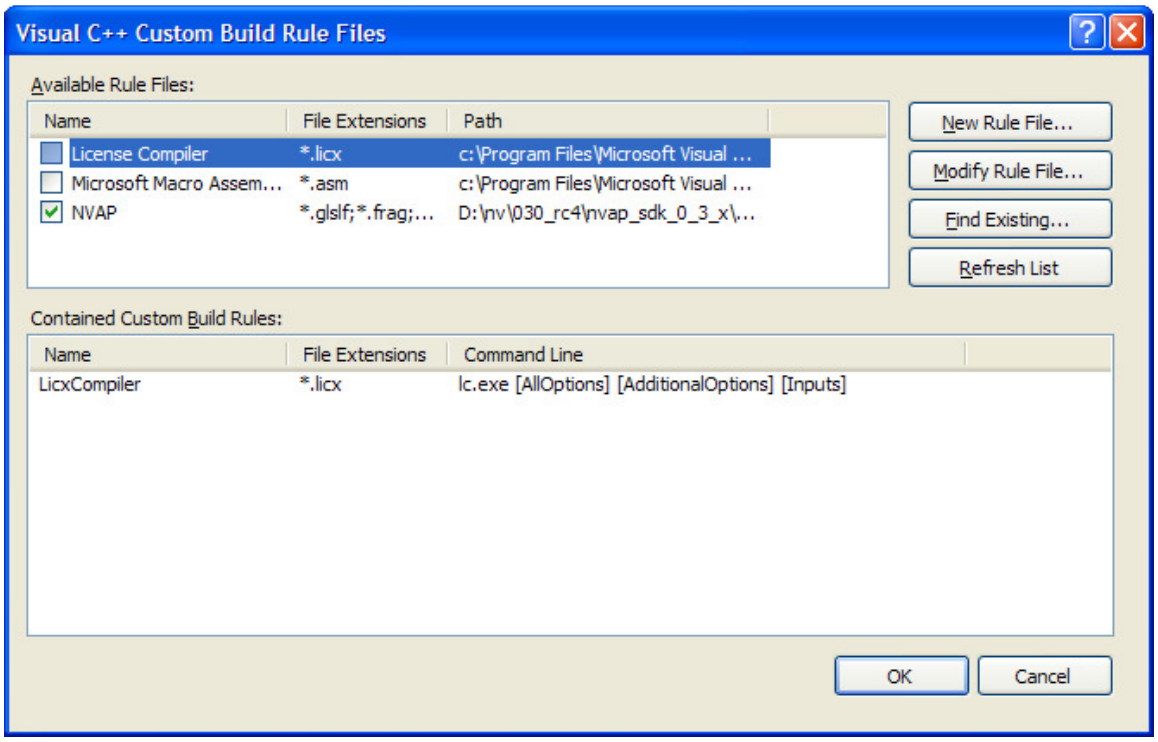
1) Open the "Project" menu and select "Properties"
2) In the resulting dialog, select the page "Configuration Properties : C/C++ : Advanced"
3) Change "Enable floating point emulation" to "No"
4) Change "Compile for architecture" to "ARM5t /QRarch5t"
5) Select the page "Configuration Properties : C/C++ : Command Line : Additional Options"
6) Add "/QRfpe-" to the set of options
7) Close the dialog with "OK"

Compiling GLSL Shaders for Tegra

While the current OS images do support compiling shaders at runtime using `glShaderSource`, it can often be more efficient to pre-compile shaders. For important details regarding pre-compiled shaders, see the Tegra OpenGL ES 2.0 programming documentation. Shaders can actually be compiled from within a Visual Studio project using the following method. It is possible to add shaders to a project in Visual Studio, which will then compile the shaders from source into Tegra binaries. The steps required to do this are as follows:

1) Open the desired application project in Visual Studio. The project must be writeable.
2) Right click on the project (not the solution) in the solution explorer, and select the menu item "Custom Build Rules…"

3) The resulting dialog will look something like the following image



Note the "NVAP" rule in the upper box. If this does not show up, then you will need to install the rule file. You will only need to do this install once, and then the rule will appear (disabled) in later projects.

4) To install the rule file, click the "Find Existing…" button and browse to `$(NV_WINCE_T2_PLAT)\host_bin`, selecting the file `shaders.rules`.

The NVAP rule will now appear in the upper box

5) Once the rule is available (either per-existing or just added), set the checkbox next to the NVAP rule. This will enable shader compilation in the current project
6) Add the shader source files to the set of project source files in the solution explorer. The shaders will be rebuilt as needed, based on the modification times of the binary and source shaders.

Alternatively, two batch files are provided in the `host_bin` directory (and are thus installed in the host PC's PATH): `glslv.bat` and `glslf.bat`. These batch files take as their only argument the path to a GLSL vertex or fragment shader file (respectively) and compile the source shader into a binary shader with extension `.nvbv` or `.nvbf` (respectively).

Connecting to the Tegra Devkit

Connecting the devkit to the Host PC with ActiveSync (Windows XP)

Once the devkit is booted to Windows CE, the last important setup step is to ensure that the device can connect to the host PC over USB via ActiveSync. This section assumes that the host PC has ActiveSync 4.5 installed.

Boot the devkit normally, and connect the host PC to the devkit with the mini-USB-to-USB cable, just as was done when flashing the OS. The host PC should automatically detect the devkit and establish an ActiveSync connection (This might take up to a minute the first time).

Windows CE6

While establishing the connection you will be prompted with a dialog for New Partnership. Click No and then Next – we do not need to set up a data-sync partnership, as we only need the basic connection.

Connecting the devkit to the Host PC with Windows Mobile Device Center (Windows Vista / Windows 7)

Instructions for configuring WMDC on Windows Vista and Windows 7 may be found at http://technet.microsoft.com/en-us/library/cc722321(WS.10).aspx

Running/Debugging

It is also possible to deploy the application and remotely debug it using Visual Studio on the host PC over USB. **The following instructions assume that ActiveSync has been installed and that the host PC and devkit are connected using ActiveSync or WMDC**.

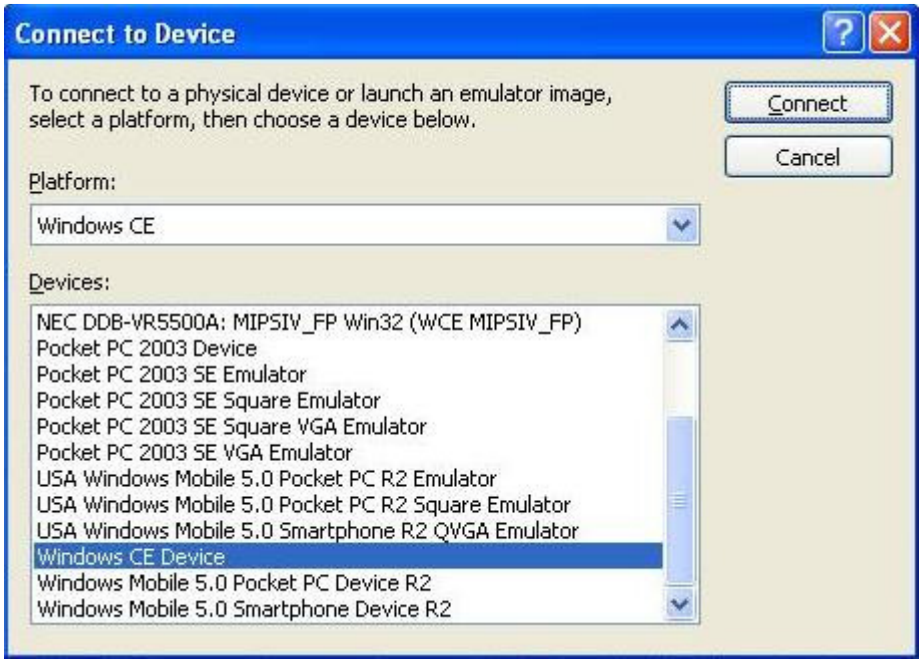
Connecting Visual Studio to the Devkit

Debugging is made very simple using ActiveSync/WMDC and Visual Studio.

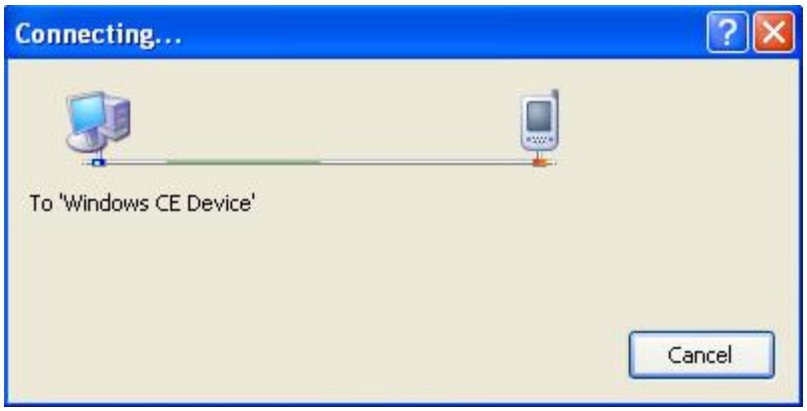
Open the desired development solution file, and select an application that has been built for the devkit.

Select the "STANDARDSDK_500 (ARMV4I)" platform and "Debug/Release" configuration from dropdown menu.

From the "Tools" menu select "Connect to Device…". You will be prompted with a "Connect to Device" dialog. Choose "Windows CE" platform and "Windows CE Device" device.



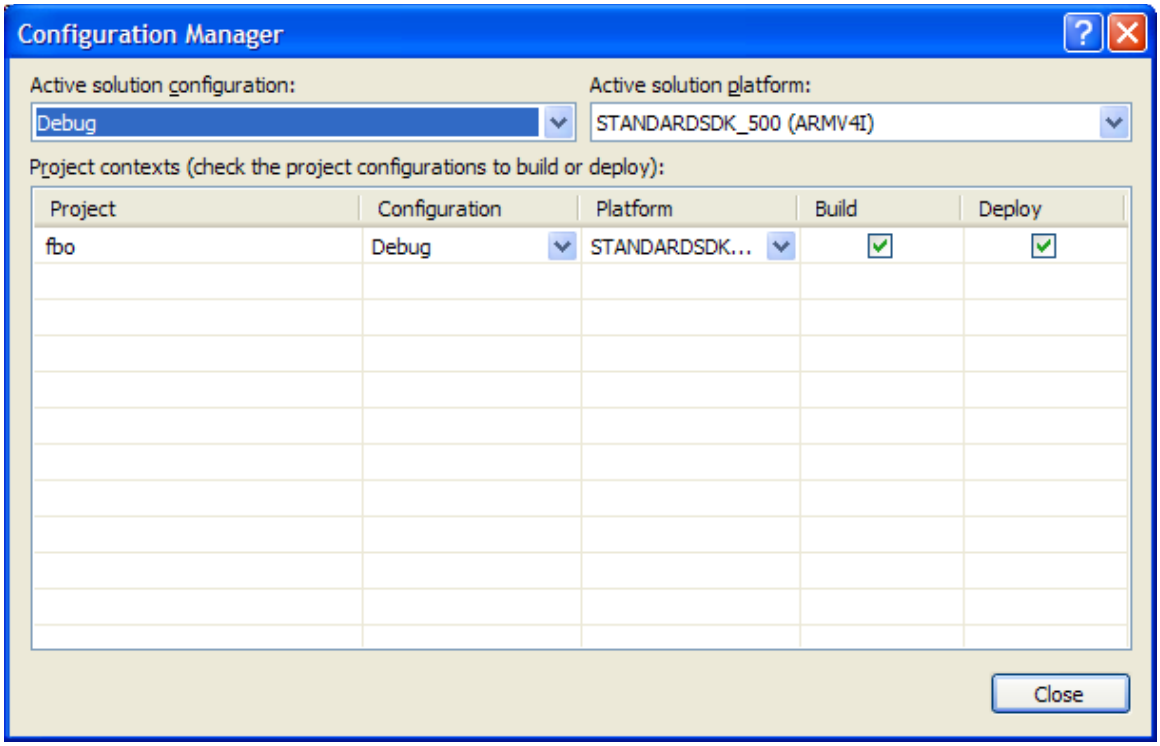
Click "Connect". Another "Connecting..." dialog will open while establishing the connection.



"Close" the dialog when connection is successfully established.

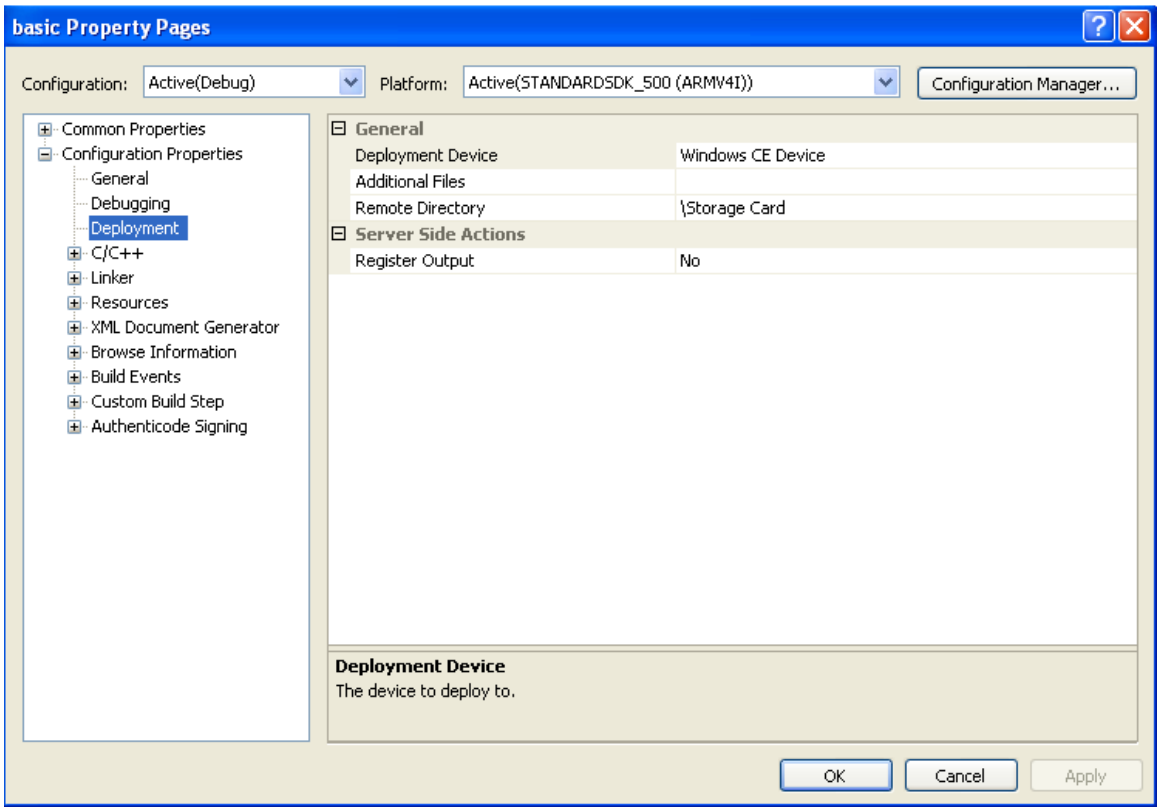
Enabling Application Deployment

Automatic copying of the application executable to the target device (deployment) is optional in Visual Studio. To ensure that it is enabled, Right click on the **solution** in the solution explorer and select "Configuration Manager…". Ensure that the "Deploy" option is enabled for the desired projects, platforms (STANDARDSDK_500) and configurations:



Setting the Remote Executable Location

Visual Studio normally deploys the executable to the device if the copy on the device does not match the most recently-built version. Before starting the application, the executable file is automatically transferred to the devkit's file system. The target location of the transfer may be set using "Project Properties" > "Configuration Properties" > "Deployment" > "Remote Directory" option.



If you want the exe to be copied in SD card the Remote Directory should be **"\Storage Card"**.

If the demo is using any data files then they need to be transferred before running the demo to appropriate location.

Note that the Tegra CE6 OpenKODE implementation defines the root of the application's file system as the application executable location. Thus, if you create an SD card image as directed in the SDK documentation, you must be sure to deploy your application to `\Storage Card`, not the default `\Program Files`, or else the application will not be able to find its data files.

Debugging an Application

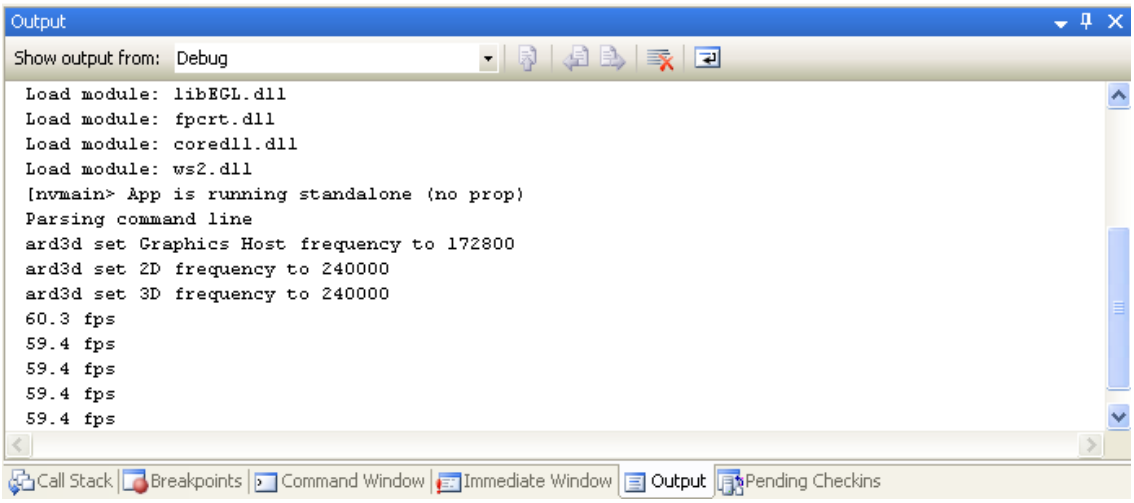
If you want to add a breakpoint in the demo, open the source file and add the breakpoint like any other Visual Studio project.

```
static GLuint prog = 0;
KDint nv_init(void)
{
    NvFSAppendSearchPath("/res/shared");

    nv_set_shader_path("/res/basic/shaders/");
    prog = nv_load_program("basic");
```

Start debugging the demo project (Right click on the demo project and choose "Debug" > "Start new instance". If the project is set as Startup project then you can also use F5). Debugging works in the same way as with any other Visual Studio project (use F5, F10 and F11 etc to start debugging, step into and step over). However, note that the remote nature of the application being debugged can mean that stepping line to line can be slower than local PC debugging of WinXP applications.

The text output of the demo will be printed in the **Output** window of Visual Studio.



You can also use the **Call Stack** window, **Breakpoints** window, **Auto**; **Local** and **Watch variables** window and other debugging features. Stop debugging (Shift + F5) to end the demo.

Please note that terminating the demo from the Visual Studio debugger will end the demo abruptly and may leak resources or leave the device in an odd state.

Issues with Remote Debugging on the Devkit

The connection to the device over USB can be lost during debugging of an application. If you find such an issue, check if your mini-USB cable is well connected. In some cases, you may be able to fix the problem by restarting the application again. It is also possible that the issue is in the underlying OS and needs to be reported to your NVIDIA contact.

Conditional breakpoints or data breakpoints can sometimes bring down the performance of the application.

Viewing/Transferring Files onto the Devkit

Viewing and transferring files on the devkit is made very simple using ActiveSync/WMDC. Once a connection is established between the PC and the devkit over USB, open the "My Computer" icon on the PC and click on "Mobile Devices". You can use the Windows file copy-paste method to transfer files.

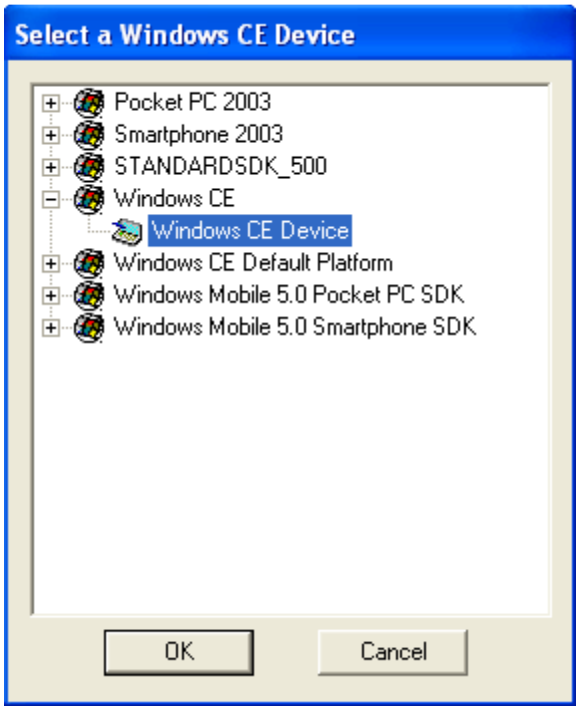
To transfer data onto an SD card you can either use above method or a PC-based SD card reader.

The devkit also has a limited amount of internal, persistent (cross-reboot) storage in the root of the filesystem (My Device). Transferring data into this persistent storage can only be done via a USB connection as shown above. Note that while this storage is persistent across reboots and power-cycles, flashing a new OS image or re-flashing the current OS image will erase this persistent storage.

Viewing/Editing the Registry on the Devkit

On your PC click Start > All Programs > Microsoft Visual Studio 2005/2008 > Visual Studio Remote Tools > Remote Registry Editor.

It will open "Select a Windows CE Device" dialog.



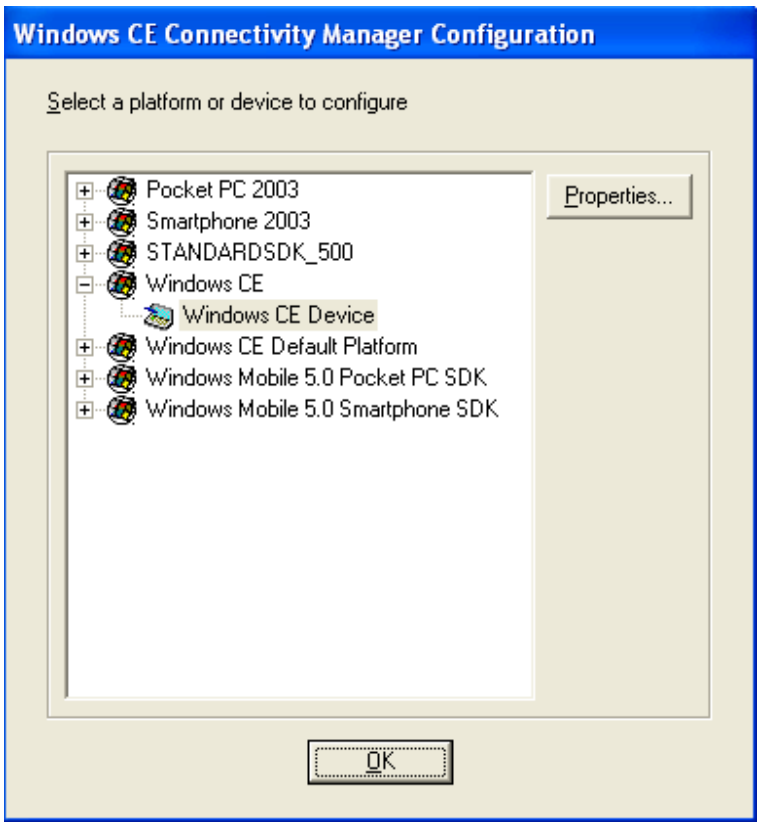
Choose "Windows CE" > "Windows CE Device" and click OK. This will start establishing connection to the devkit.

After the connection is established you will be able to see and edit the registry on the devkit.

In case the connection is not being established:

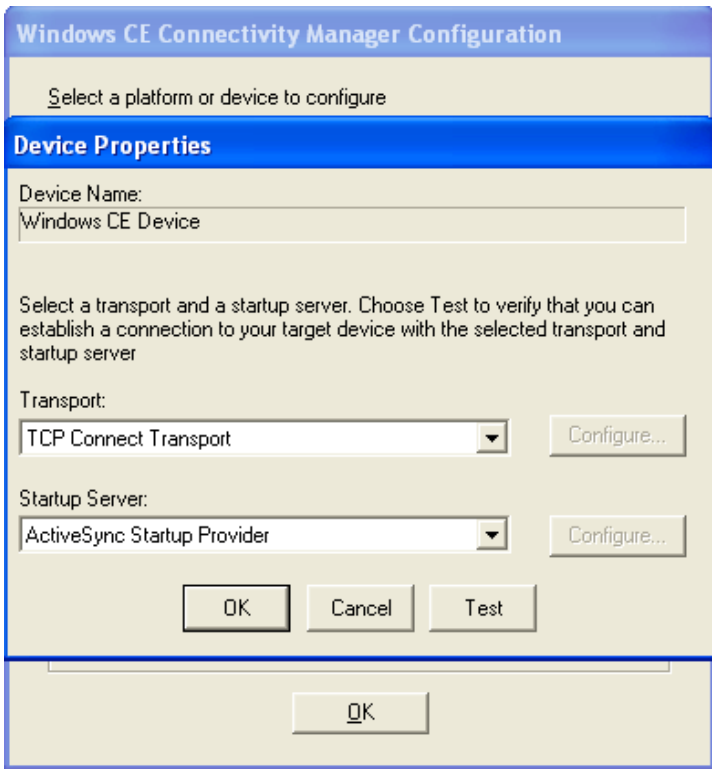
Go to “Target” menu > “Connectivity Options…”

You will be prompted with a “Windows CE Connectivity Manager Configuration” dialog.

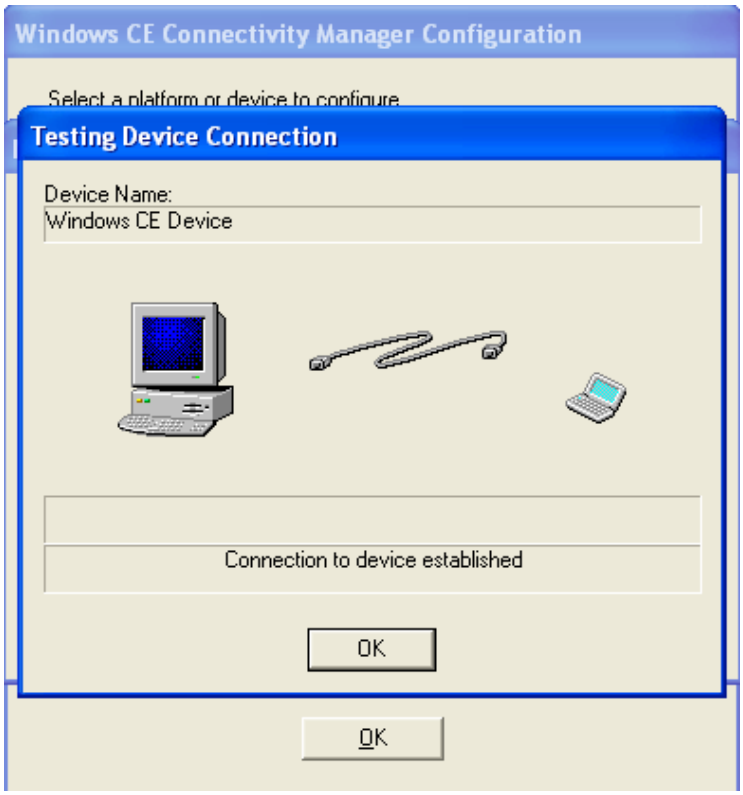


Choose the platform “Windows CE” > “Windows CE Device” and click on “Properties” button.

You will get another dialog “Device Properties”. Check if the Device Name is “Windows CE Device”. Select “TCP Connect Transport” from Transport dropdown list. Select “ActiveSync Startup Provider” from Startup Server dropdown list.



Click "Test" to check if the connection can now be established.



If a connection is established click OK and go back to the registry editor. You can now view and edit the registry.

The NvDispRes Tool

The platform support pack ships with an NVIDIA-supported tool that allows the width, height and color depth of the devkit's display-out to be changed.

Installing the Tool

To install the tool, simply copy the tree:

```
$(NV_WINCE_T2_PLAT)\bin\nvdispres
```

To the devkit's filesystem, either by copying the tree from the host PC to an SD card or USB drive, or else over ActiveSync/WMDC to the root filesystem. The entire tree should be copied with the hierarchy intact.

Using the Tool

Launch the tool on the devkit as you would any other app: by double-clicking the `nvdispres.exe` icon in File Explorer. A fullscreen application with the following layout should appear on the display out:

If the current display resolution matches any of the options, then that option will be highlighted. In addition, the current display depth will be highlighted. Simply select the desired 4:3 or widescreen resolution and display depth using the pointer. Once selected, click either the "Apply" button to lock in the settings, or "Cancel" to leave them as they were prior to running the tool.

Once the tool has closed, the devkit must be restarted for the changes to take effect.

Custom Resolutions

If the desired resolution is not listed in the set of tool buttons, the configuration file:

```
nvdispres\resources\nvdispres\config.txt
```

can be edited to add additional settings. Simply add a line as follows inside of either of the two brace-delimited blocks (`Resolutions4:3` or `ResolutionsWide`):

```
Resolution 640 480
```

Replacing 640 and 480 with the desired width and height in pixels (lines are case-sensitive). This will add the desired button to the selected sub-menu. Re-run the tool to see (and be able to select) the new resolutions. Existing resolutions can also be deleted to make space, if desired.

Ensure that the supplied resolution is supported by the external display device (absolute max: 1920x1080 HDMI, 1600x1200 CRT).

Known Issues

If the display resolution selected is not supported by Tegra or by the display device, then the device may not reboot to a usable state. The easiest method of recovering is to reflash the device. However, if a reflash is not desired, it may be possible to make the device display again by following these instructions:

1) Boot the device (you will not see any output – wait ~15 secs after power-on)
2) Connect USB and ensure that the device is connected via ActiveSync/WMDC. If this cannot connect, you will not be able to complete this method. If you cannot connect, reflash the device.
3) Run the remote registry-editing tool and browse to the key:
   `HKLM\Software\NVIDIA Corporation\NVDDI`
4) Edit the following values to a safe resolution (e.g. 1024x768):
   `DesktopWidth`
   `DesktopHeight`

5) Exit the remote registry editor
6) Wait 15 seconds for the registry to be written to persistent storage
7) Reboot the device

Programming Notes

OpenKODE Core File System

The root of the OpenKODE Core file system on the Tegra under CE6 is based off of the location of the application's executable. Thus, given a path to the executable `$(APP_PATH)`, the OpenKODE directories map as follows:

OpenKODE Core Directory	**Windows Path**
`/res`	`$(APP_PATH)\resources`
`/data`	`$(APP_PATH)\data`

Installing Multiple Tegra CE6 Support Packs at Once

In some cases, it may be useful for developers to be able to switch between different OS images during development. The installers for the Tegra CE6 platform packs allow multiple packs with different version numbers to be installed at once. To switch to a previously installed pack "0000001" that has since been replaced subsequently installing pack "0000002", simply do the following:

1) Right-click on "My Computer" on the desktop or in a Windows Explorer window
2) Select "Properties" from the pop-up menu
3) Select the "Advanced" tab
4) Click "Environment Variables"
5) Find "NV_WINCE_T2_PLAT" under "System Variables" and double-click it to edit
6) Change the value to point to the root of the "0000001" pack instead of the "0000002" pack
7) Click "OK" as needed to close the dialogs
8) Close and reopen any DOS prompts and MS Visual Studio instances

9) Flash the OS in the “0000001” pack to the devkit
10) Rebuild all of the applications of interest (to be safe)

Known Bugs

CE 6.0 OS Image Version 5265393

OpenMax IL

- Running on the devkit on CRT or HDMI/DVI, after playing a movie using the overlay renderer component, a graph component fails to transition from Idle to Loaded state during the normal 'cleanup' process. As a result, the devkit hangs and the display output turns off. This issue affects the `Videoplayer` demo (`nv_mediaplayer.lib` when used in overlay mode) and the `Simpleplayback` sample shipped with the Tegra Khronos Apps SDK.
- Running on the devkit on CRT or HDMI/DVI, trying to play a movie with the hdmi-overlay renderer component, the device hangs and screen turns off during initial transition from Idle to Executing. This issue affects the `Videoplayer` demo (`nv_mediaplayer.lib`) shipped with the Tegra Khronos Apps SDK.
- Running on the devkit on CRT or HDMI/DVI, trying to play a movie with the EGLSurface rendering mode of `Videoplayer` (or `nv_mediaplayer` library) shipped with the Tegra Khronos Apps SDK, audio for the movie plays properly, but screen is blank. Movie ends and OMX cleans up properly, without issues, returning to normal application UI.

As a result of these issues, applications wishing to use OMX to play video should render the video using the EGLImage rendering mode, not overlays or EGLSurface.

OpenKODE Core

- OpenKODE Core fullscreen windows currently leave the Windows taskbar visible. An application-level workaround will be posted to the developers website as soon as possible.

Windows CE 6.0 Limitations

The following limitations are inherent in the design of Windows CE 6.0 and will not be fixed in an OS image update:

- Windows CE6.0 has a hard limit of 512MB of physical memory. Thus, when used with CE6, a 1GB Tegra 250 developer board will still have <512MB of available memory at boot time.
- Windows CE6.0 is limited to a single processor, and will not take advantage of the second core on the device at this time.

NVIDIA Corporation
2701 San Tomas Expressway
Santa Clara, CA 95050
www.nvidia.com