VNC® Mobile Solution
For Automotive

iPod Out Guide For Windows CE

Version 2.4

1 Introduction

This document describes the *iPod Out Extension To VNC Mobile Solution*, as applied to the Windows CE platform.

iPod Out is a device-operating mode that permits remote control of certain iPod devices, such as the iPhone and iPad. When in iPod Out mode, an iPod device generates a simplified user interface over an analogue video conection. The interface lets you browse the music collection on the device, and play individual items.

The architecture of the *iPod Out Extension* is fully described in the document *iPod Out Guide*, provided with this distribution. That document also explains how to set up an iPod Out development prototype, with either an Embedded Linux or an Android development board being used to host the *VNC Automotive Viewer*.

This *current* document functions as an addendum to the *iPod Out Guide*, providing information specific to the use of a Windows CE development board to host the *VNC Automotive Viewer*.

1.1 How this document is organized

Section 1 consists of the present introduction.

Section 2, *Requirements*, lists all of the hardware, software, and documentation required for constructing the *iPod Out Extension* for Windows CE.

Section 3, *Architecture and implementation*, explains the architecture of the *iPod Out Extension* for Windows CE, and provides details of its implementation in this prototype.

Section 4, *Preparing the Windows CE image*, lists the downloads and build procedures required for preparing a Windows CE image suitable for use with the *iPod Out Extension*.

Section 5, *Preparing the Embedded Linux image*, lists the patch, configuration, and build activities required for the preparation of an Embedded Linux image to host the *video sample application* required for this prototype.

Section 6, *Preparing hardware components,* describes how to set up the current prototype, by transferring images to SD cards, interconnecting the required development boards, and following boot procedures.

Section 7, *Running the prototype*, describes how to use the prototype to perform remote control of a mobile device.

2 Requirements

This section addresses the requirements for setting up and using this prototype of the *iPod Out Extension To VNC Mobile Solution* with the Windows CE platform.

2.1 Accessing essential documents

Before attempting to follow the instructions in the current document, you should be familiar with the following key elements of the main documentation set provided with *VNC Mobile Solution For Automotive*:

- The documents *Creating A Windows CE Image* and *Creating An Embedded Linux Image.* These respectively describe the requirements and necessary procedural steps for creating Windows CE and Embedded Linux images that can run on Freescale i.MX51 development boards. Images must be prepared in accordance with the extensive instructions provided in these documents.
- *Sample Applications Guide.* This provides an overview of the viewer and servers provided as ready-to-go binaries by *VNC Mobile Solution.*

- *Rebuilding the Automotive Viewer Sample Application*. This provides extensive information on rebuilding and deploying the Automotive viewer on various platforms, including Windows CE.
- *Setting Up USB Connections.* This explains how to make USB connections between viewer-platforms (including those incorporating development boards) and devices.

Also essential is a full understanding of the *iPod Out Guide*, provided with the current distribution: this explains the overall architecture of the *iPod Out Extension,* and shows how a prototype can be assembled for use with an Embedded Linux or Android development board.

2.2 Hardware, cabling, and software

In order to prepare this prototype of the *iPod Out Extension* for Windows CE you require:

- All of the hardware, cabling, and software listed in the document *Creating A Windows CE Image*, including a complete software environment that has already been used to produce a Windows CE image, and a Freescale i.MX51 development board, prepared as described in that document.

  Note that the board will continue to be used with its LCD screen attached: the screen will be used to display output from connected mobile devices.
- A software environment that has been used to produce an Embedded Linux image, as described in the document *Creating An Embedded Linux Image*, and a Freescale i.MX51 development board, prepared as described in that document.

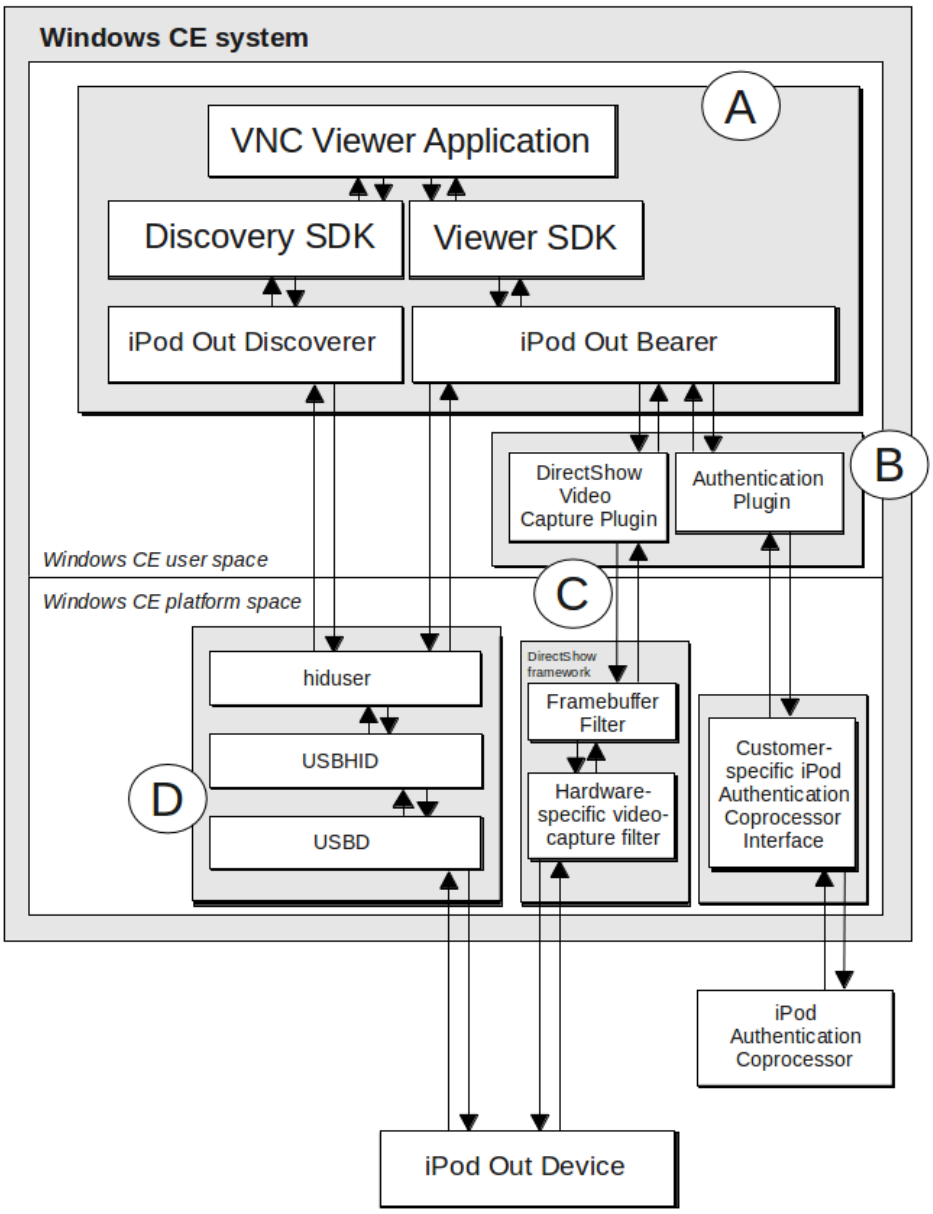
  Note that this second board will be run without an LCD screen attached.
- A Windows and a Linux computer. Administrative permission is expected to be required on both.
- All licenses, and all cabling and connectivity-support listed in the document *iPod Out Guide*; plus an additional USB cable, with which the two development boards can be interconnected.

Various additional software download, are required specifically for preparation of the *iPod Out Extension* for Windows CE. These are described in detail below.

3 Architecture and implementation

The basic architecture of the *iPod Out Extension,* along with a detailed description of the sequence whereby a connection is achieved and a remote-control session conducted, is described in detail in the document *iPod Out Guide*.

The remote-control of an iPod device from the Windows CE platform requires several components in addition to or as replacements for those detailed in the *iPod Out Guide*. This incremented architecture is represented by the following diagram:



The annotations to this diagram are as follows:

A. The architecture of the *VNC Automotive Viewer* is unchanged. The viewer application relies on the *Viewer SDK* for maintaining remote control sessions, including framebuffer update procedures: the *Viewer SDK* communicates with each specific device over the appropriate protocol by means of a purposed *bearer*, or communication library. The *iPod Out Bearer* is here provided in a version specific to the Windows CE platform. The *Discovery SDK* and its associated *discoverer* alert the viewer application whenever a new device is physically connected to the Windows CE system, and allow initiation of a remote control session.

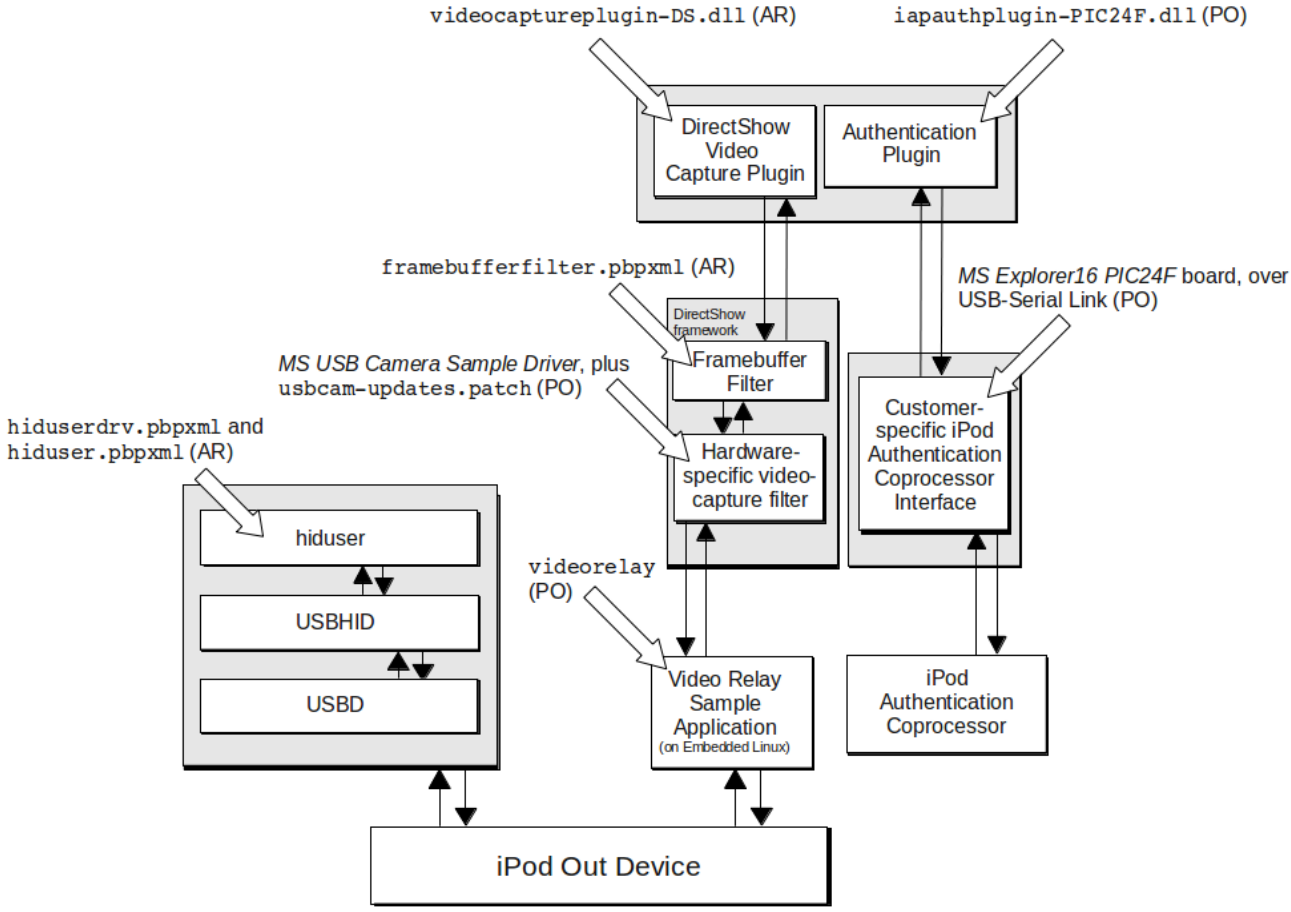
B. The *Authentication Plugin* functions exactly as described in the *iPod Out Guide*, interfacing with the hardware of the *iPod Authentication Coprocessor* via the interfaces specific to the customer's platform. For more information, see the *iPod Out Guide*.

C. A *Video Capture Plugin* must be used to pass digital video frames to the bearer. For Windows CE, RealVNC provides a plugin that works with the Microsoft *DirectShow* framework. Incoming video must be passed to the plugin by means of a hardware-specific video-capture filter, and a special *Framebuffer Filter* provided by RealVNC.

D. The iPod Out device communicates via USB by means of the *iPod Accessory Protocol.* Within the Windows CE system itself, communication between the viewer application and the iPod device is handled via USBD and USBHID. Since this mechanism is not open to user-space applications by default, RealVNC provides the *hiduser* driver to open the communication path. This must be added as a subproject to the Windows CE source, as described below in section 4.4, *Adding subprojects.*

3.1 Implementation details of the current prototype

The diagram below indicates how the current prototype implements key aspects of the architecture, by naming each component and specifying its place in the system. Note that each component followed by the letters *AR* is a standing *Architectural Requirement;* each followed by *PO* is an implementation instance, valid for this *Prototype Only*.



For further information on these components, see the section following.

3.2 The current distribution

The current distribution from RealVNC, `VNCMobileSolution-iPodOut-2.4`, contains two branches:

- *Evaluation*. This branch provides ready-to-run binaries, and so facilitates quick evaluation of *iPod Out Extension* technologies. Sub-branches are provided for *Android OS, Linux*, and *Windows CE*. The *Windows CE* sub-branchch contains an Automotive viewer, plus supportive libraries.

  Note that not every component required for evaluation activities is provided as a ready-to-run binary. Some build activities of source provided in the *Development* branch of the distribution are still required.

- *Development*. This branch provides components required for development purposes. It contains three sub-branches — for Android, Linux, and Windows CE. The last of these contains the following sub-directories:

  - `native` — this contains supportive libraries, header files, and documentation. Each library is provided as a fully built dynamically linkable library. These are:

    - `vncbearer-IPODOUT.dll` — this is the bearer used by *VNC Mobile Solution* to achieve and maintain a remote control session via the *iPod Out Extension*. No source is provided.

    - `videocaptureplugin-DS.dll` — this provides Microsoft *DirectShow* video capture support. No source is provided.

    - `iapauthplugin-PIC24F.dll` — this provides support for the iPod Out bearer's authentication activities, by interfacing with the *Microchip Explorer16 PIC24F* development board.

      This library is provided in both binary and source form, and is only useful for the purposes of prototyping: it must be entirely replaced by a production-specific version before full deployment on an automotive head unit can be attempted.

    - `vncdiscoverer-ipod.dll` — this provides support for the spontaneous discovery of mobile devices, when they are connected to the viewer platform by means of USB. Note that this is provided as source, as well as in binary form.

  - `platform` — this contains special projects, provided by RealVNC for support of Windows CE as a platform for *VNC Mobile Viewer*. These include the *framebufferfilter* and *hiduser* projects, and the *usbcam* patch. Details of how to apply these components are provided below.

  - `samples` — this contains source code in five sub-directories, which are:

    - `authplugin-PIC24F` — this is source code that can be used to build the `iapauthplugin-PIC24F.dll` library, which can be used for authentication during prototyping and development only, as described above.

    - `configuration` — this contains a configuration file used by the bearer.

    - `discoverer` — this contains source code for the iPod Out discoverer, named `vncdiscoverer-ipod.dll`.

    - `microchip-pic24f-explorer16-mfi`. This contains the buildable source for firmware, used by the *Microchip Explorer16 PIC24F* development board assembly. Note that an updated version (v1.02) of this firmware is provided with the *iPod Out Extension To VNC Mobile Solution*, Version 2.4, and this *must* be used. If you attempt to use the firmware provided with Version 2.2, failure will occur.

      Note: All evaluators must build from this source, according to the instructions in this document, and then flash the resulting firmware to the PIC24F board.

- `videorelay` — this contains a sample application that supports video display on the Windows CE development board, but runs on the Embedded Linux board (with the Embedded Linux board being connected both to the iPod Out device and the Windows CE development board).

3.2.1 Deploying the video relay

Preparation of the *iPod Out Extension* for Windows CE requires significant preparations and build activities on a Windows PC. However, you must also use a Linux desktop machine to build both an Embedded Linux image, and the *video relay* sample application that will run on the Linux development board. Therefore, the entire *VNCMobileSolution-iPodOut-2.4-package* must be opened on your Windows desktop, and at least the <*VNCMobileSolution-iPodOut-2.4-package*>`/Development/WindowsCE/samples/videorelay` folder on your Linux system.

3.3 Licensing

To run the *iPod Out Extension* prototype, you require two licenses: one for *VNC Automotive Viewer*, the other for the *iPod Out Bearer*. These must be obtained from RealVNC.

4 Preparing the Windows CE image

This section explains how to prepare a Windows CE image suitable for hosting the *VNC Automotive Viewer* in order to remote-control an Apple mobile device via the iPod Out mechanism.

4.1 Initial preparations of image and application infrastructure

Follow all of the instructions in the document *Creating A Windows CE Image*. Only when you have created the image as explained there should you continue with the instructions in this section.

Information on deploying *VNC Automotive Viewer* on Windows CE is provided in the document *Sample Applications Guide*. Information on rebuilding the viewer, and on building infrastructural elements supportive of connections to a variety of devices, is provided in the document *Rebuilding The Automotive Viewer Sample Application*. You may wish to familiarize yourself with these documents before continuing.

4.2 Adding the Microsoft USB Camera Sample Driver

Once you have completed your addition of the October 2012 QFE update, add the Microsoft USB Camera Sample Driver as follows:

1. Close Visual Studio.
2. Download the USB Camera Sample Driver.

   This can be found at http://www.microsoft.com/en-gb/download/details.aspx?id=19512. Click on the *Download* button and run the MSI file. The Camera Sample Driver will be placed in `C:\WINCE600\PUBLIC\ThirdParty\Catalog\USBCam`.
3. Download the GNU *patch* program.

   This can be found at: http://gnuwin32.sourceforge.net/packages/patch.htm. Click on the Download link adjacent to the Description *Complete package, except sources*. Then perform the instructions specified by the *Patch Setup Wizard.* This installs `Patch-2.5.9`.

4. Add the following to your PATH: `C:\Program Files\GnuWin32\bin`. (Instructions on how to add to your Windows PATH variable can be found in the document *Rebuilding the Automotive Viewer Sample Application.*)

   To test that the patch program has been installed, bring up a Command Prompt and type `patch -v`. This should print a copyright notice to the console.

5. Make changes to the USB Camera Sample Driver that you installed, by using the *patch* program to apply a patch provided with the current distribution. Enter the following commands:

   ```
   cd C:\WINCE600\PUBLIC\ThirdParty\Catalog\USBCam
   patch -p1 -i C:\<path-to-VNCMobile-Solution-iPodOut-2.4-package>\Development\WindowsCE
                                                    \platform\usbcam\usbcam-updates.patch
   ```

   The following output should be observed:

   ```
   patching file 'Driver/sensorformats.h'
   patching file 'Driver/usbcam.reg'
   patching file 'Driver/usbpdd.cpp'
   ```

This concludes the procedure for adding and patching the USB Camera Sample Driver.

4.3 Adding subprojects

The current distribution provides several subprojects that must now be added to the Windows CE source code. Proceed as follows:

1. Re-open Visual Studio, and ensure that the `iMX51-EVK-Mobility` project is established in it.

2. In the **Solution Explorer** pane, expose the `iMX51-EVK-Mobility > Subprojects` node, right-click on it, and select **Add Existing Subproject** from the pull-down menu.

   In the dialog window that appears, navigate to C:\<*path-to-VNCMobile-Solution-iPodOut-2.4-package*>`\Development\WindowsCE\platform\framebufferfilter`, and open the file `framebufferfilter.pbpxml`.

   This causes the `framebufferfilter` subproject to be added to the project, and so appear listed under the `Subprojects` node.

3. Repeat the procedure described in step 2 above, in order to access C:\<*path-to-VNCMobile-Solution-iPodOut-2.4-package*>`\Development\WindowsCE\platform\hiduser\drv`, and open the file `hiduserdrv.pbpxml`.

   This adds the *driver* subproject of the HidUser component, which duly appears as `hiduserdrv`.

4. Repeat the same procedure again, in order to access C:\<*path-to-VNCMobile-Solution-iPodOut-2.4-package*>`\Development\WindowsCE\platform\hiduser\lib`, and open the file `hiduser.pbpxml`.

   This adds the *library* subproject of the HidUser component, which appears as `hiduser`.

5. Set the build order for the subprojects, by right-clicking on the `Subprojects` node, and selecting **Set Subproject Build Order**. Use the dialog that appears to place the `hiduserdrv` project in a position higher in the list than `hiduser`. Then click **OK** to close the window.

This concludes the procedure for adding subprojects.

4.4 Adding the FTDI USB-Serial driver

The *iPod Out Extension* requires that the i.MX51 development board hosting the Automotive Viewer be connected to a PIC board: this set-up is described in the document *iPod Out Guide*. In the case where Windows CE constitutes the viewer-platform, a special driver must be added, to facilitate the connection via a USB-Serial link, since the available Serial port is already used for debug and bootloader control. (In the event of experimentation with a board featuring a free serial port, addition of this USB-Serial driver might be unnecessary.)

Proceed as follows:

1. Download the FTDI Virtual COM Port (VCP) driver for Windows CE 6.0 (ARM), from the following URL: http://www.ftdichip.com/Drivers/VCP.htm.
2. Open the zip file, and copy the file `ARMv4VCPDriver60\ftdi_ser.dll` to the following location: `C:\WINCE600\PLATFORM\iMX51-EVK\FILES`.
3. In the **Solution Explorer** pane of Visual Studio, open the file `iMX51-EVK-Mobility > Parameter Files > Freescale i.MX51 EVK: ARMV4I (Active) > project.bib`.
4. Add the following line to the `MODULES` section:

   ```
   ftdi_ser.dll    $(_FLATRELEASEDIR)\ftdi_ser.dll             NK     SHK
   ```

5. Return to the contents of the zip file, and in a Windows Desktop text editor, open the file `ARMv4VCPDriver60\INF FILES\registry settings.txt`.
6. Copy the following lines from `registry settings.txt` onto the end of the `project.reg` file:

   ```
   ; FTDI USB-serial driver
   [HKEY_LOCAL_MACHINE\Drivers\USB\ClientDrivers\FTDI_DEVICE]
   "Prefix"="COM"
   "Dll"="ftdi_ser.dll"
   "ConfigData"=hex:\
   01,00,3f,3f,10,27,88,13,c4,09,e2,04,71,02,38,41,9c,80,4e,c0,34,00,1a,00,0d,\
          00,06,40,03,80,00,00,d0,80
   "InitialIndex"=dword:00000000
   "DeviceArrayIndex"=dword:00000000
   "LatencyTimer"=dword:00000010

   [HKEY_LOCAL_MACHINE\Drivers\USB\LoadClients\1027_24577\Default\Default\FTDI_DEVIC
   E]
   "DLL"="ftdi_ser.dll"

   [HKEY_LOCAL_MACHINE\Drivers\USB\LoadClients\Default\Default\255\FTDI_DEVICE]
   "DLL"="ftdi_ser.dll"
   ```

   Save the file and exit.

4.5 Selecting required OS components

In Visual Studio, access **View > Other Windows > Catalog Items View** from the pull-down menu at the top menu bar. Ensure that the following components all have a green tick next to them:

- `Core OS > CEBASE > Applications and Services Development > Component Services (COM and DOM) > Component Object Model > COM`

- `Core OS > CEBASE > Core OS Services > USB Host Support > USB Human Input Device (HID) Class Driver`
- `Core OS > CEBASE > Graphics and Multimedia Technologies > Media > DirectShow > DirectShow Core`
- `Core OS > CEBASE > Graphics and Multimedia Technologies > Media > Video Codecs and Renderers > DirectShow Video Renderer`
- `Third Party > USB Camera Driver`
- `Device Drivers > Camera > NULL Camera Driver`

Note: The last of these generates a dummy video capture filter, which outputs a moving black square across a white background, which can be used for testing purposes.

4.6 Build the image

To build the image, select **Build > Rebuild Solution** from the top menu bar.

5 Preparing the Embedded Linux image

This section explains how to prepare an Embedded Linux image suitable for hosting the *video relay* service. In the context of the current prototype, this is required to support USBCam. Note that when an alternative hardware-specific video-capture filter is used (as required for customers' production purposes), neither the Embedded Linux image nor the *video relay* service is required.

5.1 Initial image-preparation

Follow the instructions in the document *Creating An Embedded Linux Image*. Only when you have created the image as explained there, and have the corresponding source tree established on your Linux desktop computer, should you continue with the instructions in this section.

5.2 Downloading additional patches

Download all of the following RealVNC patches:

- https://github.com/RealVNC/MobileSolution-KernelPatches/blob/master/iMX51-Linux/src/0020-usb-gadget-uvc-Remove-non-required-locking-from-uvc_.patch
- https://github.com/RealVNC/MobileSolution-KernelPatches/blob/master/iMX51-Linux/src/0021-Prevent-USB-gadget-drivers-from-enabling-port-too-so.patch
- https://github.com/RealVNC/MobileSolution-KernelPatches/blob/master/iMX51-Linux/src/0022-Customise-UVC-webcam-gadget-driver-for-use-by-VideoR.patch
- https://github.com/RealVNC/MobileSolution-KernelPatches/blob/master/iMX51-Linux/src/0023-usb-gadget-Don-t-attempt-to-dequeue-requests-for-a-d.patch

5.3 Adding the downloaded patches

Add the patches by entering the following commands:

```
cd /opt/freescale/ltib/usr/src/linux-2.6.35.3
```



```
patch -p1 -i <path-to-saved-patches>/0020-usb-gadget-uvc-Remove-non-required-locking-from-uvc_.patch
patch -p1 -i <path-to-saved-patches>/0021-Prevent-USB-gadget-drivers-from-enabling-port-too-so.patch
patch -p1 -i <path-to-saved-patches>/0022-Customise-UVC-webcam-gadget-driver-for-use-by-VideoR.patch
patch -p1 -i <path-to-saved-patches>/0023-usb-gadget-Don-t-attempt-to-dequeue-requests-for-a-d.patch
```

The following output should be observed:

```
patching file drivers/usb/gadget/uvc_queue.c
patching file drivers/usb/gadget/arcotg_udc.c
patching file drivers/usb/gadget/uvc_v4l2.c
patching file drivers/usb/gadget/uvc_video.c
patching file drivers/usb/gadget/webcam.c
patching file drivers/usb/gadget/arcotg_udc.c
```

5.4 Reconfiguring the kernel

Proceed as follows:

1. Navigate to the `ltib` directory, and start a kernel reconfiguration, as follows:

   ```
   cd /opt/freescale/ltib
   ./ltib --config
   ```

   This brings up the `Freescale iMX5x` user interface.

2. On the initial user interface screen, labelled `Freescale iMX5x Based Boards`, use the arrow key to navigate to the `Configure the kernel` option, and use the spacebar to select it. Then exit to the Linux prompt, electing to save your configuration.

   This brings up the `Linux Kernel Configuration` user interface.

3. Enable options as follows:

   ```
   Device Drivers ->
         [*] USB support ->
               <*> USB Gadget Support --->
                     <M> USB Gadget Drivers
                      (M) USB Webcam Gadget
   ```

4. Exit to the Linux prompt, saving the configuration. This causes ltib to build the kernel and root filesystem.

5. Make sure the toolchain for the Freescale environment is on your path:

   ```
   export PATH=/opt/freescale/usr/local/gcc-4.4.4-glibc-2.11.1-multilib-1.0/arm-fsl-linux-gnueabi/bin:$PATH
   ```

5.5 Preparing the video sample application

You must now configure and build the video relay sample application, and then include it with your Embedded Linux image. This section assumes that the *<VNCMobileSolution-iPodOut-2.4-package>*`/Development/WindowsCE/samples/videorelay` directory has been deployed to your Linux

desktop machine. It also assumes that you have the Embedded Linux kernel source in the directory `/opt/freescale/ltib/usr/src/linux-2.6.35.3`.

5.5.1 Build and make the sample application

Enter the following commands:

```
cd <VNCMobileSolution-iPodOut-2.4-package>/Development/WindowsCE/samples/videorelay
./configure --build=x86_64-none-linux-gnueabi --host=arm-none-linux-gnueabi \
                  --with-rootfs=/opt/freescale/ltib/rootfs \
                  --with-kerneltree=/opt/freescale/ltib/usr/src/linux-2.6.35.3
make
```

Optionally, check the results of the build by means of the `file` command, as follows:

```
file ./videorelay
```

The output should be as follows:

```
./videorelay: ELF 32-bit LSB executable, ARM, version 1 (SYSV), dynamically linked
(uses shared libs), for GNU/Linux 2.6.26, not stripped
```

5.5.2 Deploying the sample application into the root filesystem

To do this, use the `deployVideoRelay.sh` script, provided within the `videorelay` directory, specifying the flag `-freescale` and the target location. Note that this script must be run with administrator privileges. You have three options:

- Deploy the application into the root file system source tree:

  ```
  ./deployVideoRelay.sh -freescale /opt/freescale/ltib/rootfs
  ```

  Note that at this point, you will need to rebuild, to ensure that the application becomes contained in the filesystem image. Enter the following commands:

  ```
  cd /opt/freescale/ltib
  ./ltib --deploy
  ```

- Deploy directly to an SD card:

  `./deployVideoRelay.sh -freescale /media/<sdcard-mount-point>`

  Note that the entire Embedded Linux image with also need to be transferred to an SD card, as described in the next section. You may wish to transfer the Embedded Linux image first.

- Deploy onto an NFS-mounted filesystem. See the document *Creating An Embedded Linux Image*, specifically the section entitled *Deploying the root filesystem for network access.*

This concludes preparation of the Embedded Linux image.

6 Preparing hardware components

This section explains how to set up the hardware for the running of the current prototype. Note that a production implementation of the *iPod Out Extension* for Windows CE might be considerably different, and indeed feature fewer components.

6.1 Transferring images to SD cards

The Windows CE and Embedded Linux images you have built must be transferred to SD cards, which will be introduced into their respective development boards prior to booting.

6.1.1 Transferring the Windows CE image to an SD card

Information on transferring the Windows CE image is provided in the document *Creating a Windows CE Image*, in the section entitled *Copying images to the SD card.* All the modifications you have made as a result of instructions in this document will be included in the transferred image files.

6.1.2 Transferring the Automotive sample application to the SD card

The *iPod Out Extension* requires that the *VNC Automotive Viewer* sample application for Windows CE, along with all necessary supportive infrastructural components, be transferred to the same SD card as the Windows CE OS image files. Therefore, once the image files have been transferred, proceed as follows:

1. With the SD card still in the reader, and the reader still attached to your Windows desktop system, locate the corresponding device in the **My Computer** window. Double click, to enter the device.
2. Bring up a separate **Windows Explorer** window, and navigate to the folder containing *VNC Automotive Viewer* and associated infrastructure. If you wish to deploy the files that are provided ready-to-run for evaluation purposes, access the archive file `<VNCMobileSolution-iPodOut-2.4-package>\Evaluation\WindowsCE\WinCE6-ARM\RealVNC-iPodOut-Bearer-<version-number>-WinCE-automotiveViewer-ARMV4I.zip`.

   Alternatively, if you have performed rebuilding activities as described in the document *Rebuilding the Automotive Viewer Sample Application*, you must co-locate the equivalent files yourself.Note that in both cases, you require one license for *VNC Automotive Viewer*, and another for the *iPod Out Bearer*. These must be obtained directly from RealVNC. The simplest way of deploying the licenses is to include them both in the same folder as the rest of the viewer-related infrastructure.
3. Transfer all of the viewer-related files into the SD card window. This places the files on the SD card.

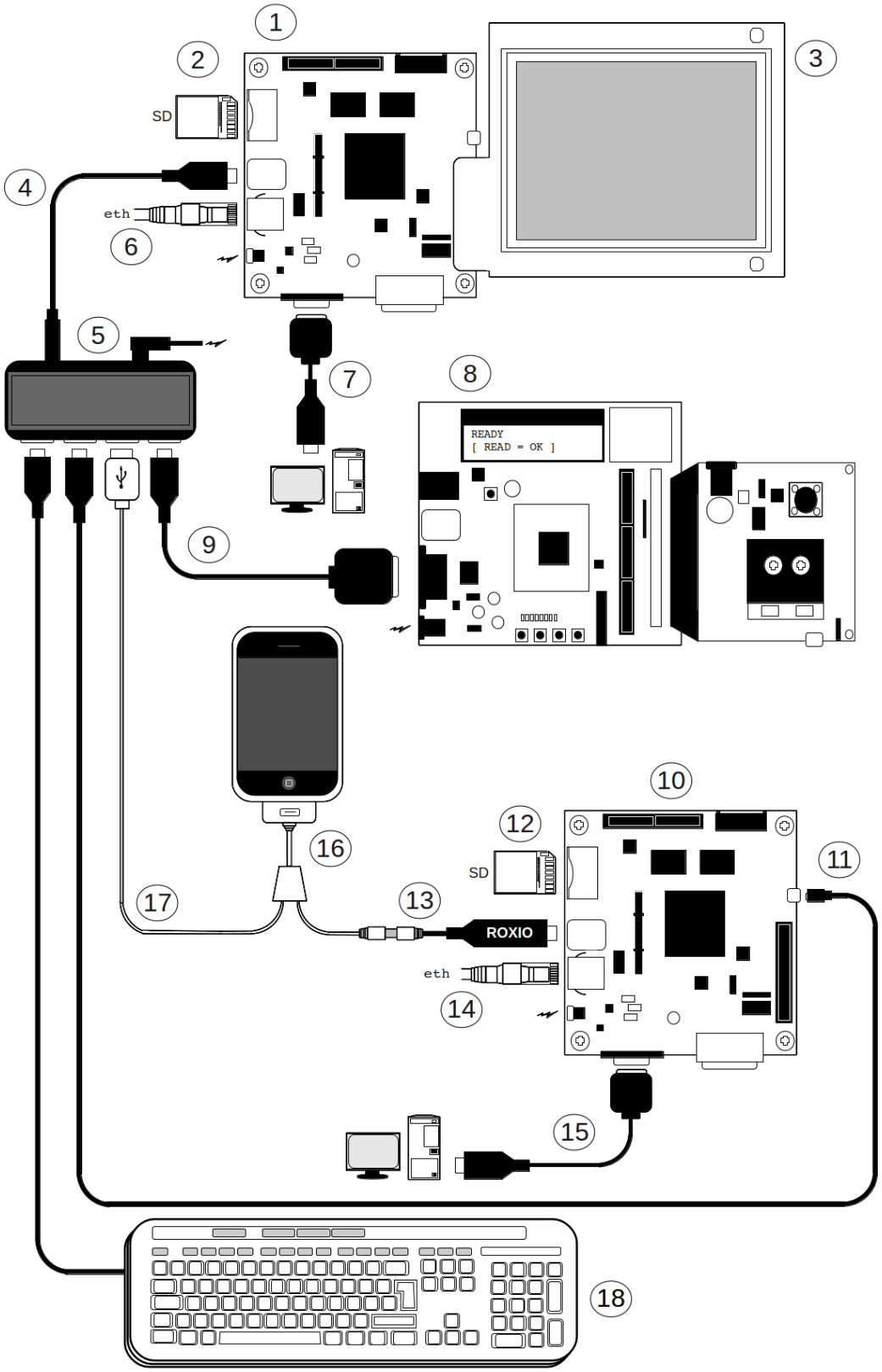
At this point, you can withdraw the card from the reader, since it contains all the Windows CE infrastructure that you require. Note that once the Windows CE development board has been successfully booted, the viewer-related files appear in the **My Device > SDMemory** folder, and the viewer can be run directly from there.

6.1.3 Transferring the Embedded Linux image to an SD card

Procedures for co-locating Embedded Linux image files and transferring them to an SD card are provided in the document *Creating An Embedded Linux Image*.

6.2 Setting up hardware for the prototype

The hardware setup for this prototype is represented by the following diagram:



The annotations to this diagram are as follows:

1. A Freescale i.MX51 development board is required to run the Windows CE image that you have prepared.
2. The image is introduced into the board via an SD card, which is inserted in an inverted position into the slot on the underside of the board.
3. The LCD screen of the board should be attached, since you will use this to interface with Windows CE and the viewer sample application.
4. The board should be connected to the other components in the prototype via USB.
5. A powered hub is required to interconnect all of the components in the prototype.
6. The Windows CE development board can *optionally* be connected to the local network via an ethernet cable. However, this is not required for iPod Out connectivity.
7. *Optionally*, the board can be connected to a local PC running Windows, and observed and controlled via a terminal console program (such as TeraTerm). This is not a requirement for running the prototype, but is useful in terms of ensuring that the board goes through its initial boot phase correctly.
8. The PIC board, used to provide iPod Authentication, must be set up in the way explained in the *iPod Out Guide*.
9. The PIC board must be connected to the other components via the powered hub, by means of a Serial-to-USB interface.
10. A second i.MX51 development board is required, to run the Embedded Linux image that you have prepared.
11. This board must be interconnected with the other prototype components via the powered hub. The Micro-B USB interface should be used. Note that no LCD screen need be attached to this board.
12. As with the Windows CE development board, the image should be introduced via an SD card, inserted in an inverted position into the slot on the underside of the board. The Embedded Linux image on the SD card must contain the *Video Relay* sample application.
13. A *Roxio Video Capture USB* dongle is used to transfer the analogue video produced by the iPod device to digital video frames, and hand them off to the Embedded Linux board, to which it must be connected via its USB interface.
14. The Embedded Linux development board can *optionally* be connected to the local network via an ethernet cable. However, this is not required for iPod Out connectivity.
15. *Optionally*, the board can be connected to a local PC running Linux, and observed and controlled via a terminal console program (such as Minicom). As with the Windows CE board, this is not necessary for running the prototype, but probably useful in support of setup and debugging.
16. The iPod device should be connected to the Roxio dongle via the 30-pin connector.
17. The iPod device should be interconnected with the Windows CE board via a USB interface to the powered hub.
18. A keyboard should be connected to the powered hub via a USB interface, and used for interactions with the Windows CE board.

Note: The Embedded Linux board may need to be fully powered up and running *before* the Windows CE board is switched on.

7 Running the prototype

After both boards have been successfully booted, and the PIC board and iPod Out device are also both hooked up, the Automotive viewer can be run from the directory containing all viewer files, using the following command:

```
automotiveviewer -bearercfg IPODOUT-ipodout.ini -disc
```

Once the device has been discovered, the remote control user interface appears in the viewer window on the Windows CE board.