USB Device Application Kit for Digi Embedded Linux

User's Manual

Digi document reference number: 90000925_A

© Digi International Inc. 2008. All Rights Reserved.

The Digi logo is a registered trademark of Digi International, Inc.

All other trademarks mentioned in this document are the property of their respective owners.

Information in this document is subject to change without notice and does not represent a commitment on the part of Digi International.

Digi provides this document “as is,” without warranty of any kind, expressed or implied, including, but not limited to, the implied warranties of fitness or merchantability for a particular purpose. Digi may make improvements and/or changes in this manual or in the product(s) and/or the program(s) described in this manual at any time.

This product could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes may be incorporated in new editions of the publication.

Digi International Inc.

11001 Bren Road East

Minnetonka, MN 55343 (USA)

* +1 877 912-3444 or +1 952 912-3444

http://www.digi.com

Contents

1. Introduction 6
1.1. Features 6
2. Requirements 7
2.1. Host system requirements 7
3. Installation 8
3.1. Linux kernel configuration 8
4. Using the USB Device functionality 10
4.1. Serial gadget 10
4.2. Ethernet gadget 11
4.3. File-backed storage gadget 15

Conventions used in this manual

Here is a list of the typographical conventions used in this manual:

Style	New terms and variables in commands, code, and other input.
`Style`	In examples, to show the contents of files, the output from commands. In, text the C code.
	Variables to be replaced with actual values are shown in italics.
Style	For menu items, dialogs, tabs, buttons, and other controls.
	In examples, to show the text that you enter
Menu name > option	A menu followed by one or more options; for example, File > New.

This manual also uses these frames and symbols:

This is a warning. It helps solve or avoid common mistakes or problems.

This is a hint. It contains useful information about a topic.

```
> This is a host computer session
> Bold text indicates what must be entered.
```

```
> This is a target session
> Bold text indicates what must be entered.
```

Abbreviations

ASCII	American Standard Code for Information Interchange
CPU	Central Processing Unit
ESD	Electrostatic Discharge
GPIO	General Purpose Input/Output
OS	Operating System
PC	Personal Computer
RAM	Random Access Memory
ROOTFS	Root File System
TFTP	Trivial File Transfer Protocol
TTY	Teletypewriter
USB	Universal Serial Bus

1. Introduction

The USB Device Application Kit is a product that brings the USB Device physical interface to your Digi Development board or JumpStart board.

The USB Device Application Kit hardware board connects to the Development or JumpStart board via the USB Application Header and the provided 16-pin flat cable. It basically mounts a USB PHY and USB B female connector.

The USB Device Application Kit software contains the driver needed to support the USB Device functionality in Digi Embedded Linux.

1.1. Features

The following are the main features of the USB Device Application Kit:

- USB 2.0 Full Speed (Low speed not supported)
- USB Device board is powered from Development or JumpStart board
- Operation modes: serial, mass storage, and Ethernet
- ±15kV ESD protection

2. Requirements

This User's Manual assumes that the reader is able to create, compile and install a kernel + kernel modules + rootfs project using Digi Embedded Linux environment.

2.1. Host system requirements

To create a DigiEL project with support for the USB Device Application kit, the Host PC should meet the following requirements:

- Linux OS with support in kernel for USB host (serial, mass storage and Ethernet)
- Digi Embedded Linux 4.0 (or higher)

To test the USB Device functionality by connecting the target to a Windows host, a PC with the following requirements is needed:

- Windows XP
- USB standard drivers for serial, Ethernet, and mass storage

3. Installation

To install the software place the installer file **usbdev_appkit_installer.run** into any folder of a Linux host, and then execute it.

```
> ./usbdev_appkit_installer.run

This will install USB Device Application Kit software package

Do you want to proceed? [y]:
```

After you have accepted the license agreement you will be prompted for the installation folder of Digi Embedded Linux, and whether you want the installer to generate backups for the files it will replace.

The script installs the USB Device AppKit driver into your Digi Embedded Linux distribution.

The installer will also create a folder called *windows_inf_files*, with *.inf files for using the Serial and Ethernet gadgets on Windows hosts.

3.1. Linux kernel configuration

The USB Device Application Kit driver supports three standard gadget profiles: Serial, Ethernet, and Storage, allowing the user to use the target as a serial port, an Ethernet card or a mass storage device.

For working with the USB Device Application Kit a **kernel + kernel modules + rootfs** project is needed.

Once the software has been installed, the project kernel needs to be configured for supporting USB Device functionality. The following items must be included in the kernel configuration if not already set:

Under Device Drivers > USB Support > USB Gadget Support:

- Support for USB Gadgets (must be selected as built-in feature)
- USB Peripheral Controller
  - NS9XXX
    - Full Speed
- NS9XXX USB Device PHY
  - External PHY MAX3454E
- USB Gadget Drivers (must be selected as loadable module)
  - Ethernet Gadget (with CDC Ethernet support)
    - RNDIS support (EXPERIMENTAL)
  - File-backed Storage Gadget (must be selected as module)
  - Serial Gadget (with CDC ACM support) (must be selected as module)

3.1.1. Build the project

Now the kernel and the rootfs can be compiled and installed (with **make** and **make install**).

The new kernel contains the USB Device driver compiled as *built-in*, while the new rootfs contains the gadget drivers compiled as loadable modules.

4. Using the USB Device functionality

The USB Device driver provides the functionality upon which the USB gadget drivers work. The USB gadget drivers implement one or more "functions", each providing a different capability to the USB host. In this document we'll see three of these gadget drivers:

- Serial gadget
- Ethernet gadget
- File-backed storage gadget

Only one gadget function can be used at a time. Before using a gadget driver you must first make sure that no other gadget driver is loaded.

4.1. Serial gadget

The serial gadget exposes a TTY style serial line interface, usable with *Minicom* and similar tools. Most Linux hosts can talk to this using the generic usb-serial driver. The latest versions of this driver implement the CDC ACM class. This driver works with the MS-Windows **usbser.sys** driver, the Linux cdc-acm driver, and many other USB Host systems.

The 2.6 kernel has a detailed text file at **Documentation/usb/gadget_serial.txt** with information on how to set up this driver with both Windows and Linux systems. Follow the instructions in this file for exposing your target as a serial port to the eyes of the USB host.

You can find this document under your DigiEL installation folder. For example at /usr/local/DigiEL-4.0/kernel/linux/Documentation/usb/gadget_serial.txt

4.1.1. Testing the serial gadget

If you've followed the instructions in **Documentation/usb/gadget_serial.txt**, after connecting the target to the host via the USB cable you will have a device node called **/dev/ttygserial** on the target side and a device node called **/dev/ttyUSB0** on the host side. The USB connection between target and host works now as any serial line.

Now a quick test of the two new ports can be executed by sending ASCII-data using the standard shell commands `cat' and `echo'. But before this, a configuration of the corresponding TTY-layers is required. This can be done with the help of the **stty** tool on both the host and the target:

```
> stty -F /dev/ttyUSB0 -hupcl -icrnl -ixon opost onlcr -isig -icanon -echo -
echoe -echok -echoctl -echoke time 20
```

On the target we need a similar configuration:

```
> stty -F /dev/ttygserial -hupcl icrnl -ixon -opost -onlcr -isig -icanon -
echo -echoe -echok -echoctl -echoke time 20
```

Now, to do a quick test do the following: On the host, read the contents arriving at the new serial port with:

```
> cat /dev/ttyUSB0
```

On the target, write a text line to the serial port with:

```
> echo "Hello from target side" > /dev/ttygserial
```

You will see the text line arriving at the host computer.

4.1.2. Create the device node at boot time

In the instructions for loading the serial gadget you need to manually create the device node with the **mknod** command. To automate this task you can add the following line to the file **/etc/makedevs.conf** of your target's root file system:

```
dev/ttygserial c 644 0 0 127 0
```

This way, the device node for the serial gadget will be created during boot time, and you won't need to create it by hand.

4.1.3. Opening a console shell in the serial port

You might want to have control of the target by means of a console shell via the USB serial port. This is how you would do it:

On the target side, execute a console on the serial port with the **getty** command:

```
> getty -L -l /bin/sh -n 38400 /dev/ttygserial
```

Now on the host side, open a minicom session on the USB serial port /dev/ttyUSB0 at the same baudrate (in the example 38400).

4.2. Ethernet gadget

By loading the Ethernet gadget the target enumerates to the host as an Ethernet device, using the "usbnet" driver with Linux hosts or Microsoft's RNDIS driver with Windows hosts.

To load the Ethernet gadget do this on the target:

```
> modprobe g_ether
usb_gadget_register_driver() 'ether'
ether gadget: using random self ethernet address
ether gadget: using random host ethernet address
usb0: Ethernet Gadget, version: May Day 2005
usb0: using ns9xxx_udc, OUT ep2 IN ep1 STATUS ep3
usb0: MAC 66:e1:08:78:43:eb
usb0: HOST MAC 0e:5f:52:8a:b7:e7
usb0: RNDIS ready
```

```
usb0: full speed config #1: 100 mA, Ethernet Gadget, using CDC Ethernet
```

This command will create an Ethernet interface in the target called **usb0** and will assign random MAC addresses to the target and the host.

The last line appears when you plug the USB cable between the target and the host. Plug it now if you haven't done it already.

4.2.1. On Linux hosts

Load the usbnet driver on the host:

```
> modprobe usbnet
```

This will create an Ethernet interface on the host side, also called **usb0**. Now we need to give these new Ethernet interfaces IP addresses.

For example, on the host side:

```
> ifconfig usb0 192.168.44.1 netmask 255.255.255.0
```

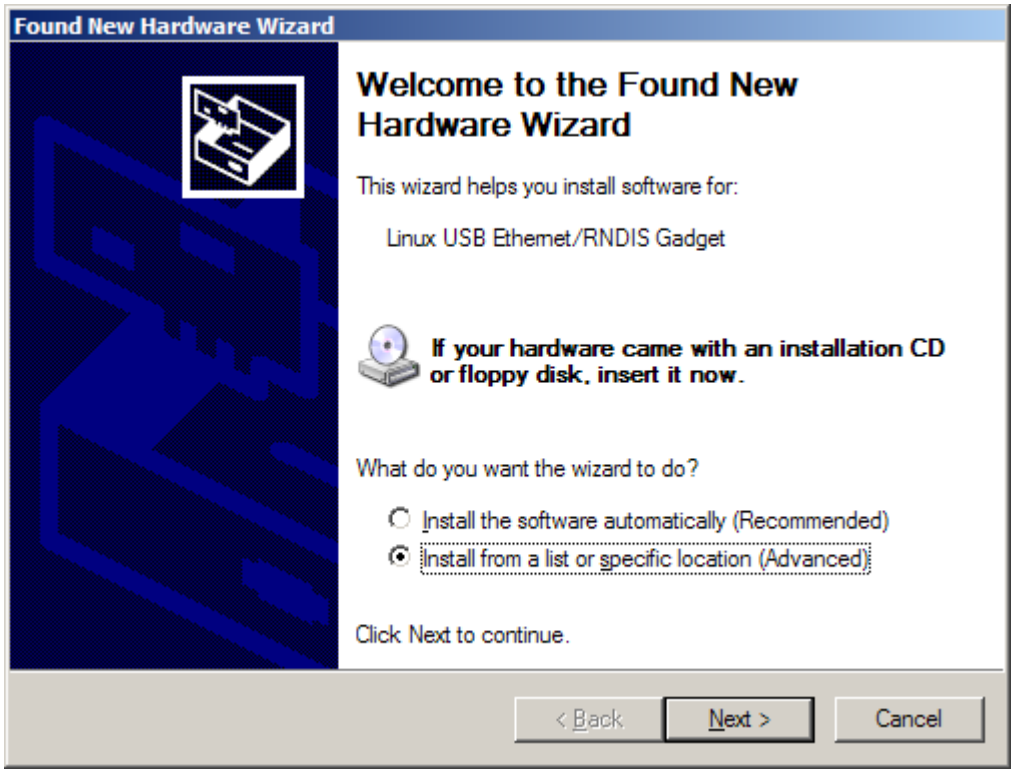
And on the target side:

```
> ifconfig usb0 192.168.44.30 netmask 255.255.255.0
```

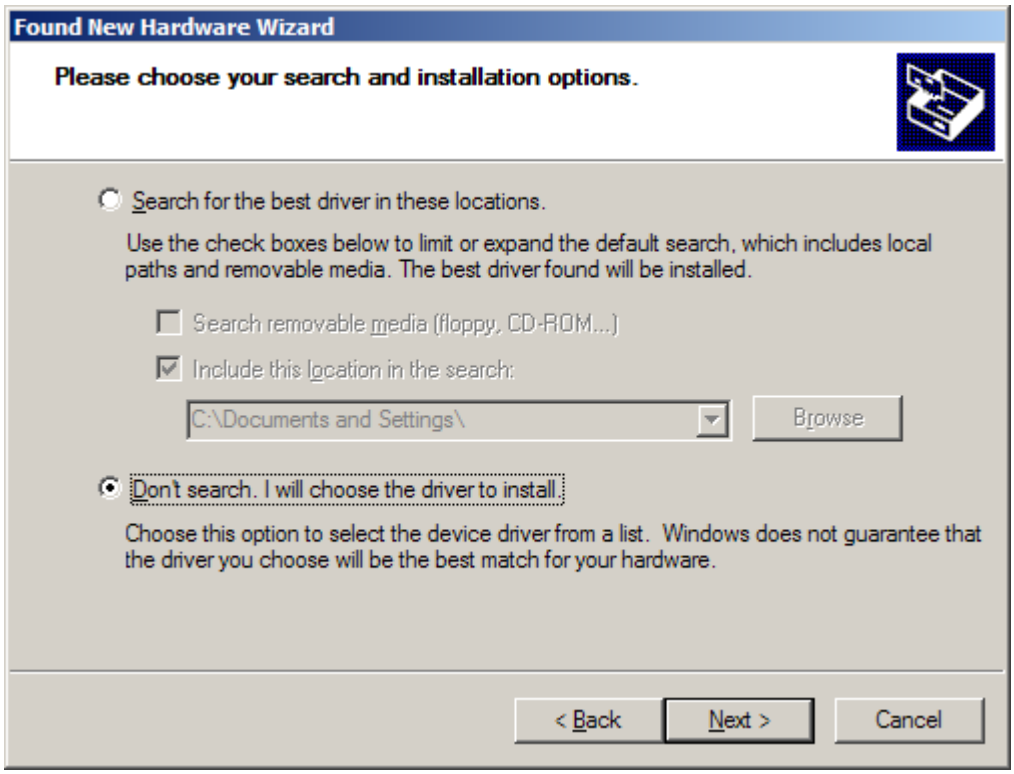
Now the target can be accessed via the USB cable as if it were an Ethernet port. You can do a ping or open a telnet session from the host to the target or vice versa.

4.2.2. On Windows hosts

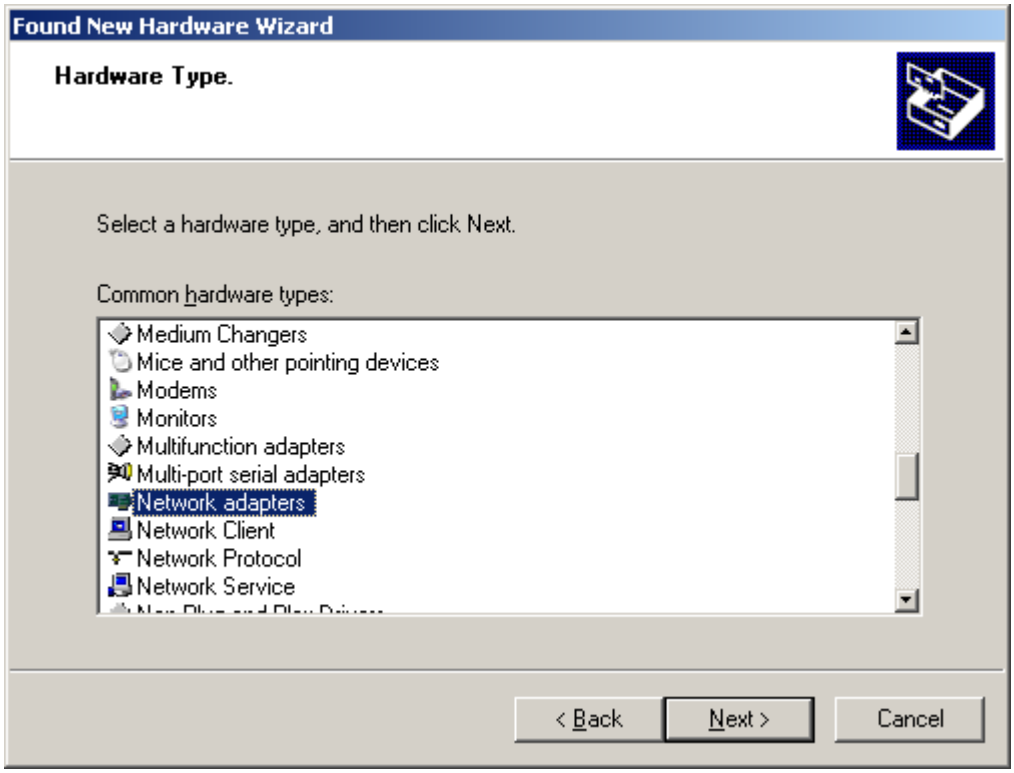
When the USB cable is plugged on the Windows host, the PC detects a new USB RNDIS/Ethernet Gadget and brings up the "Found New Hardware" dialog. Select **Install from a list or specific location (advanced)** and click Next.



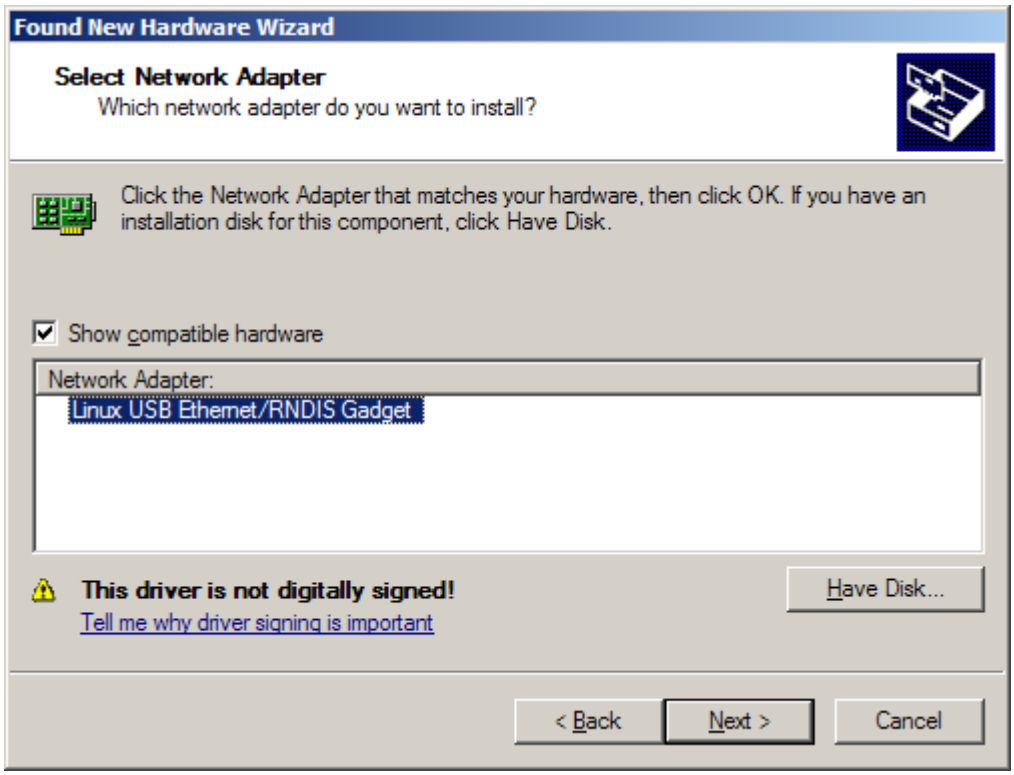
Next, select **Don't search. I will chose the driver to install** and click Next.



If this is your first install it will bring up a 'Hardware Type' dialog and ask you what kind of device you want to install. Scroll down and select **Network adapters**.



The next window will ask you to find the appropriate driver. We will use the **linux.inf** file provided in the **windows_inf_files** folder, that was created during the installation. Save this file onto your computer and then click **Have Disk** and point the dialog to the directory where you saved the file. Select the driver and click Next.



If you receive a prompt about unsigned drivers select **Continue Anyway**. After that, the driver should install successfully and you will get a new Network adapter on your windows machine.

The next thing to be done is to configure the IP address of this Network adapter by means of the Control Panel. Set for example an IP address of 192.168.44.1, subnetmask 255.255.255.0.

On the target side an Ethernet interface called **usb0** has been created. You must configure an IP address for this one too.

```
> ifconfig usb0 192.168.44.30 netmask 255.255.255.0
```

Now the target can be accessed via the USB cable as if it were an Ethernet port. You can do a ping or open a telnet session from the host to the target or vice versa.

4.3. File-backed storage gadget

This gadget implements the USB Mass Storage class, appearing to the host as a SCSI disk drive. This uses a file or block device as a backing store for the drive.

4.3.1. Creating a backing storage file

Backing storage needs some preparation before the gadget can use it. The backing storage is a regular file that must be created beforehand with its full desired size. In the following example we show how to create a backing file of 2MB in the root directory of the target:

```
> dd bs=1M count=2 if=/dev/zero of=/backing_file
2+0 records in
2+0 records out
```

This needs to be done before loading the gadget driver, but it only has to be done once.

4.3.2. Partitioning the backing storage

Creating the backing storage file is not enough to work with it. It's like having a raw disk, we still need to create partitions and install a filesystem before we can use it.

We will use the **fdisk** program to partition the backing file and to specify the values of Cylinders, Heads and Sectors of the backing file. These values are somehow arbitrary, but since the storage gadget driver uses a sector size of 512 bytes, the formula is:

512 * Cylinders * Heads * Sectors = Total bytes of backing file storage

For our example of 2MB storage file, we can use values of

Cylinders=32

Heads=16

Sectors=8

So that 512*32*16*8 = 2MB

On the target we execute the **fdisk** program with the CHS values:

```
> fdisk –C 32 –H 16 –S 8 /backing_file
```

Now we need to create a primary partition. Press **n** at the prompt, then **p** for primary and **1** for partition number 1. Then accept with intro the rest of default values (this will create a partition that occupies the full size of the backing file):

```
Command (m for help): n
Command action
   e   extended
   p   primary partition (1-4)
p
Partition number (1-4): 1
First cylinder (1-32, default 1):
Using default value 1
Last cylinder or +size or +sizeM or +sizeK (1-32, default 32):
Using default value 32
```

The new partition is created by default as a Linux partition. Let's change the partition type (with the "t" command) to FAT32 (code "b") so that we can put a FAT filesystem in it:

```
Command (m for help): t
Selected partition 1
Hex code (type L to list codes): b
Changed system type of partition 1 to b (Win95 FAT32)
```

Now print the partition table that you have created to verify that everything is alright. The partition table can be printed with the **p** command:

```
Command (m for help): p
Disk /backing_file: 0 MB, 0 bytes
16 heads, 8 sectors/track, 32 cylinders
Units = cylinders of 128 * 512 = 65536 bytes

            Device Boot    Start       End    Blocks   Id  System
/backing_file1               1          32        2044   b Win95 FAT32
```

Finally write out ("w") the partition table to the backing storage:

```
Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.
Re-read table failed with error 25: Inappropriate ioctl for device.
Reboot your system to ensure the partition table is updated.

WARNING: If you have created or modified any DOS 6.x
partitions, please see the fdisk manual page for additional
information.
Syncing disks.
```

At this point, the partition has been created but it doesn't hold a filesystem yet. The easiest way to do this is to load the driver, connect the USB cable to the host PC and create the filesystem on the recently created partition from the host.

4.3.3. Adding a file system in Linux hosts

On the target, load the driver with:

```
> modprobe g_file_storage file=/backing_file
usb_gadget_register_driver() 'g_file_storage'
g_file_storage gadget: File-backed Storage Gadget, version: 28 November 2005
g_file_storage gadget: Number of LUNs=1
```

```
g_file_storage gadget-lun0: ro=0, file: /backing_file

g_file_storage gadget: full speed config #1
```

The last line appears when you plug the USB cable between the target and the Linux host. On the host computer, the target will appear as a new SCSI device and will have been assigned a device node in the form **sd**x, and the partition will appear as **sd**x**1**. To see exactly which device the host assigned, print out the system log messages with:

```
> tail /var/log/messages
usb 1-1: new full speed USB device using ohci_hcd and address 30
scsi6 : SCSI emulation for USB Mass Storage devices
   Vendor: Linux     Model: File-Stor Gadget  Rev: 0318
   Type:   Direct-Access                      ANSI SCSI revision: 02
 SCSI device sde: 4096 512-byte hdwr sectors (2 MB)
 SCSI device sde: 4096 512-byte hdwr sectors (2 MB)
  sde: sde1
 sd 6:0:0:0: Attached scsi disk sde
 sd 6:0:0:0: Attached scsi generic sg4 type 0
```

In the example, the host assigned **sde1** as the device node for the partition of our backing file.

Now from the host we create an MSDOS file system into that partition (you might need root permissions):

```
> mkdosfs /dev/sde1
```

4.3.3.1. Mounting the file system

Now the device has to be mounted in the host to have access to read/write files and folders in it. If your host PC is prepared to automatically mount devices, you can simply disconnect the USB cable and connect it again. After some seconds, the PC will automatically detect and mount the USB device and it will appear as a USB storage drive, with the size of the partition we created (in our example 2MB). Initially, the disk is empty.

If the PC doesn't auto mount the device you need to do it by hand (you might need root permissions):

```
> mount /dev/sde1 /media/usbdisk –t vfat
```

In this example we've mounted the device into the folder /media/usbdisk. Now we can create files and folders within this folder. Before removing the USB cable we need to **umount** this folder.

The placeholder for the files and folders that we create there, is the 2MB file /backing_file that we created in the first steps.

4.3.4. Adding a file system in Windows hosts

On the target, load the driver with:

```
> modprobe g_file_storage file=/backing_file
usb_gadget_register_driver() 'g_file_storage'
g_file_storage gadget: File-backed Storage Gadget, version: 28 November 2005
g_file_storage gadget: Number of LUNs=1
```

```
g_file_storage gadget-lun0: ro=0, file: /backing_file

g_file_storage gadget: full speed config #1
```

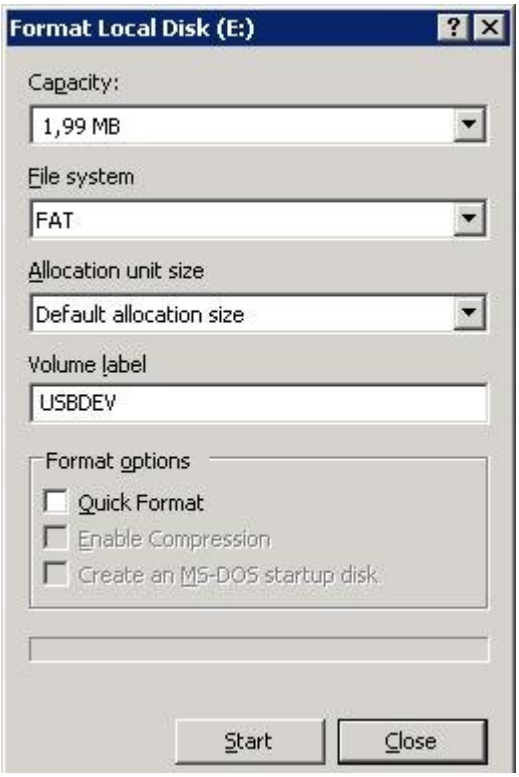
The last line appears when you plug the USB cable between the target and the Windows host. The PC will detect that a new USB mass storage device was connected. If you haven't connected the USB cable to the Windows host, do it now.

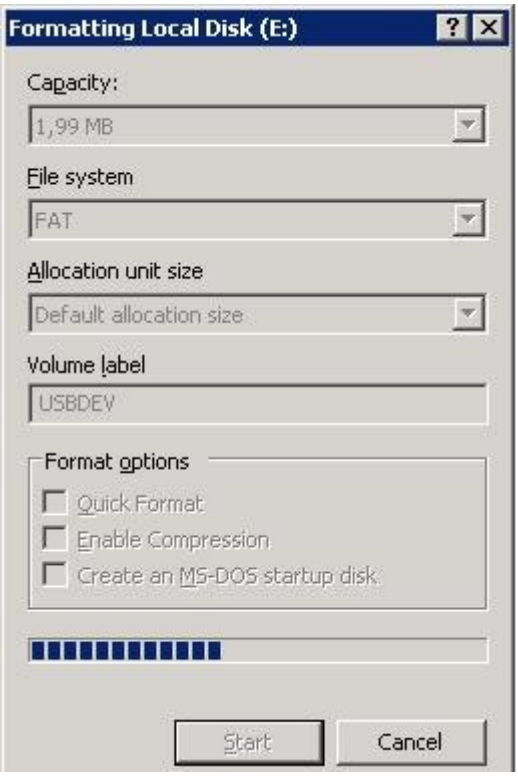
A small icon will appear in the taskbar showing that a USB mass storage device was connected:

Double click this icon to see the unit that Windows has assigned to your USB mass storage device (in the example screenshot, drive letter E was assigned):

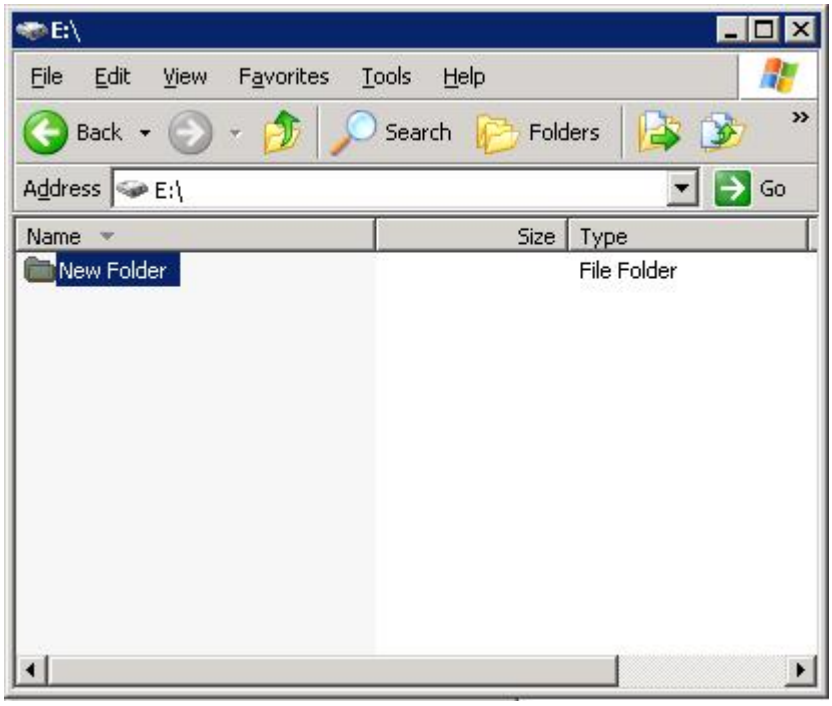
Close this dialog and try to access the new drive from the Windows Explorer. A warning message will appear to inform that the disk is not formatted yet. Click **Yes** to format it.

Select FAT file system and an optional volume label. (under 11 characters). Then click **Start**.





Once the volume has been formatted with FAT file system you can use it as any other removable media and create folders, copy files, etc.



If you use the "Safely Remove Hardware" manager remember to also disconnect the USB cable from the host computer. Otherwise the device may block.