

FreescalE MQX RTOS Example Guide

mfs_ftp example

This document explains the mfs_ftp example, what to expect from the example and a brief introduction to the API.

The example

The mfs_ftp example code is a simple FTP server application. This example uses the USB Mass Storage class to attach a USB flash driver to the board. The file system accessible through FTP is the file system in the USB memory key attached to the board.

Running the example

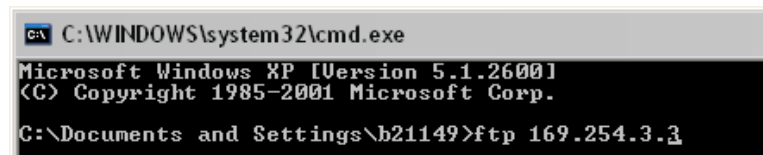
Connect an Ethernet cable from the RJ45 (Ethernet) connector from the board to the RJ45 connector in the PC. Connect a USB memory key in the board. For some board a USB converted from USB to mini-USB is needed.

The default IP address of the board is 169.254.3.3. Typically, when you connect your computer directly to the board, the computer will default to an auto IP address on the same subnet as the board (169.254.x.x), therefore requiring no setup.

Note: The PC (windows system) may take a few minutes to default to the auto IP address and make the connection. However, if you have trouble connecting, you may configure the IP address of the computer manually. Select Start > Settings > Network Connections > Local Area Connection. Note your original TCP/IP settings, and then set your IP address to 169.254.3.4 and your subnet mask to 255.255.0.0.

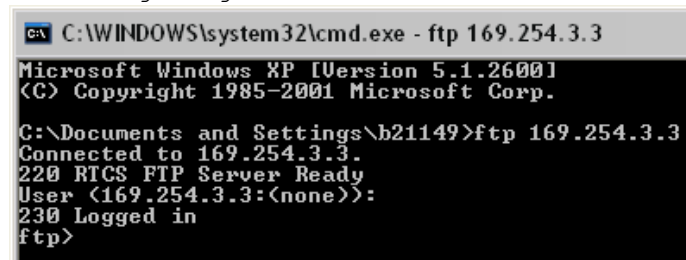
At this point the FTP server should be running in the board. Now it is time to start an FTP client session from the windows system.

Select Start > Run. Type the text "cmd" and press ok. Type the text "ftp 169.254.3.3", this will start an FTP session to the board that has the IP address 169.254.3.3



```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.
C:\Documents and Settings\b21149>ftp 169.254.3.3
```

At the user prompt just press the enter key. This will tell the FTP server to log as an anonymous user and will not ask for a password. After this the session has been started and should look as the following image.

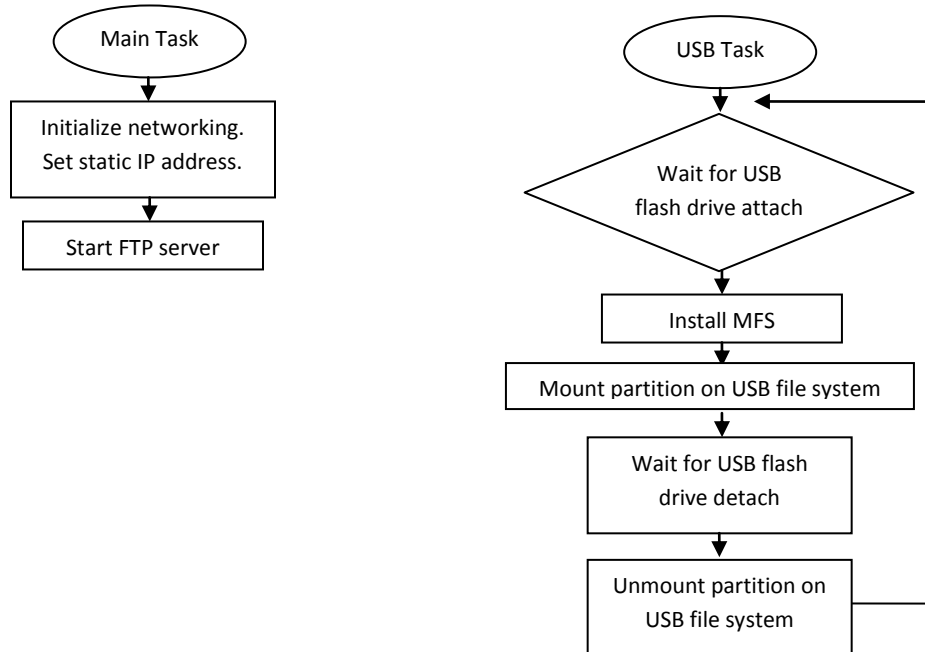


```
C:\WINDOWS\system32\cmd.exe - ftp 169.254.3.3
Microsoft Windows XP [Version 5.1.2600]
(C) Copyright 1985-2001 Microsoft Corp.
C:\Documents and Settings\b21149>ftp 169.254.3.3
Connected to 169.254.3.3.
220 RICS FTP Server Ready
User (169.254.3.3:(none)):
230 Logged in
ftp>
```

Now it is possible to issue ftp commands (i.e. dir, put, get, etc.).

Explaining the example

The application demo creates two tasks. The flow of each task is described in the next figures.



The main task starts with the networking initialization and start an ftp server in the port 21. The ftp server is started with the following line:

```
FTPd_init("FTP_server", 7, 5000 );
```

The commands supported by the FTP Server are configurable. The application must initialize a NULL terminated global variables `FTPd_COMMAND_STRUCT FTPd_commands[]` with the supported commands and `char FTPd_rootdir[]` with the default FTP root directory path. This root directory is installed by the `USB_task`. The example defines it as follows:

```
char FTPd_rootdir[] = {"c:\\"};
```

The USB Task creates a semaphore and an event related to the USB resource. The Event indicates a USB event to the rest of the application code. For the case of the demo events are attach and detach of a USB memory stick. The semaphore notifies the availability of a valid USB stick connected to the Demo. The semaphore is enabled after the USB stick is detected as a valid USB device and after the file system installs correctly.

The `usb_host_mass_device_event();` function executes when a device is attached or detached. This function tests the `switch_code` number that caused the callback. In the case of an attach event the structure device is filled with the `USB_DEVICE_ATTACHED` code and the `USB_Event` event is set. The `USB_Event` is created in the `USB_Task();` function. Detach events are similar to attach

events. In the case of a detach event the device structure is filled with the USB_DEVICE_DETACHED code and the USB_event is set.

The USB_task function creates a light weight semaphore named USB_Stick();. This semaphore is set by the task when a USB Stick is connected and it is available for read write operations. Light weight semaphores are created with the _lwsem_create(); function. The first parameter of the function receives the address of a semaphore. The second parameter receives the initial semaphore counter.

An event is created to poll the USB device status within the USB_Task(); function. The event is created with the _lwevent_create(); function. The first parameter is the address of the event to be created. The second parameter receives flags to set event options.

The first step required to act as a host is to initialize the stack in host mode. This allows the stack to install a host interrupt handler and initialize the necessary memory required to run the stack.

The host is now initialized and the driver is installed. The next step is to register driver information so that the specific host is configured with the information in the ClassDriverInfoTable array. The _usb_host_driver_info_register links the classes specified in the array with the callback function that each class executes on events.

The USB_unlock(); macro enables interrupts.

Once initialization and configuration finishes the task loop executes. The _lwevent_wait_ticks(); function waits forever until the USB_Event is set. The event is only set when attach or detach occurs. When an event occurs the device.STATE condition variable is tested.

On the detection of an event the device variable information is used to select the USB interface. The usb_hostdev_select_interface(); function causes the stack to allocate memory and prepare to start communicating with this device.

If the USB device enumerated correctly, the task can now install the file system for the USB Stick. File system installation is explained in the next section.

After correct file system installation the USB_Stick semaphore is posted to indicate the other tasks that there is a USB Mass storage device available as a resource. For the case of a detach event the file system is uninstalled and the _lwsem_wait(); function disables the semaphore.

The partition manager device driver is designed to be installed under the MFS device driver. It lets MFS work independently of the multiple partitions on a disk. It also enforces mutually exclusive access to the disk, which means that two concurrent write operations from two different MFS devices cannot conflict. The partition manager device driver can remove partitions, as well as create new ones. The partition manager device driver creates multiple primary partitions. It does not support extended partitions.

The _io_part_mgr_install function initializes MFS and allocates memory for all of the internal MFS data structures. It also reads some required drive

information from the disk on which it is installed. MFS supports FAT12, FAT16, and FAT32 file systems. If the disk has a different file system or if it is unformatted, you can use MFS to format it to one of the supported file systems.

The `usb_filesystem_install()`; function receives the USB handler, the block device name, the partition manager name, and the file system name. Some local variables are used to execute each step of the file system installation process.

The `usb_fs_ptr` value is returned after the execution of the file system install process. The first step of the process allocates zeroed memory with the required size of a USB file system structure.

The USB device is installed with the `_io_usb_mfs_install()`; function with the device name and the USB handle as parameters. After installation the `DEV_NAME` of the `usb_fs_ptr` variable is set to "USB:".

A 500 milliseconds delay is generated using the `_time_delay()`; function. Next, the USB device is open as a mass storage device. Function `fopen()`; opens the USB device and the resulting handle is assigned to the `DEV_FD_PTR` element of the `usb_fs_ptr` structure. If the `fopen` operation failed an error message is displayed.

The `_io_ioctl()`; function accesses the mass storage device and set it to Block Mode. When access to the device is available the vendor information, the product ID, and the Product Revision are read and printed in the console.

The partition manager device driver is installed and opened like other devices. It must also be closed and uninstalled when an application no longer needs it.

Partition Manager is installed with the `_io_part_mgr_install()`; function. The device number and partition manager name are passed as parameters to the function. If an error results of the partition manager installation a message is displayed in the console. On successful partition manager installation the `PM_NAME` element of the `usb_fs_ptr` structure is set to "PM_C1:".

`Fopen` opens the partition manager and the resulting file pointer is assigned to the `PM_FD_PTR` element of the `usb_fs_ptr` structure. In the case of an error a message is displayed in the console and the file system is uninstalled.

A partition is opened with the `fopen()`; function using the handle for the partition manager.

The `partition_number` parameter of the `_io_ioctl()`; function is passed by reference. This value is modified inside the function. If an error code is returned by `_io_ioctl()`; the partition manager handler is closed with the `fclose()`; function. The partition manager uninstalls with the `_io_part_mgr_uninstall()`; function.

In this case the MFS is installed without partition with the `_io_mfs_install()`; function.

MFS is installed with the device handler pointer, a file system name and a default partition value of 0.

If the partition number is valid the MFS installs with the same handler and file system name but using the partition number as a parameter.

After file system installation the status of the MFS is tested. The FS_NAME element of the usb_fs_ptr structure is set to "c:".

The fopen(); function takes the file system name as parameter. If no error occurs the file system is ready to be used by the application and the usb_fs_ptr structure is returned.