Fear and Loathing in the Media Transfer Protocol

Linus Walleij, Embedded Linux Conference, San Jose

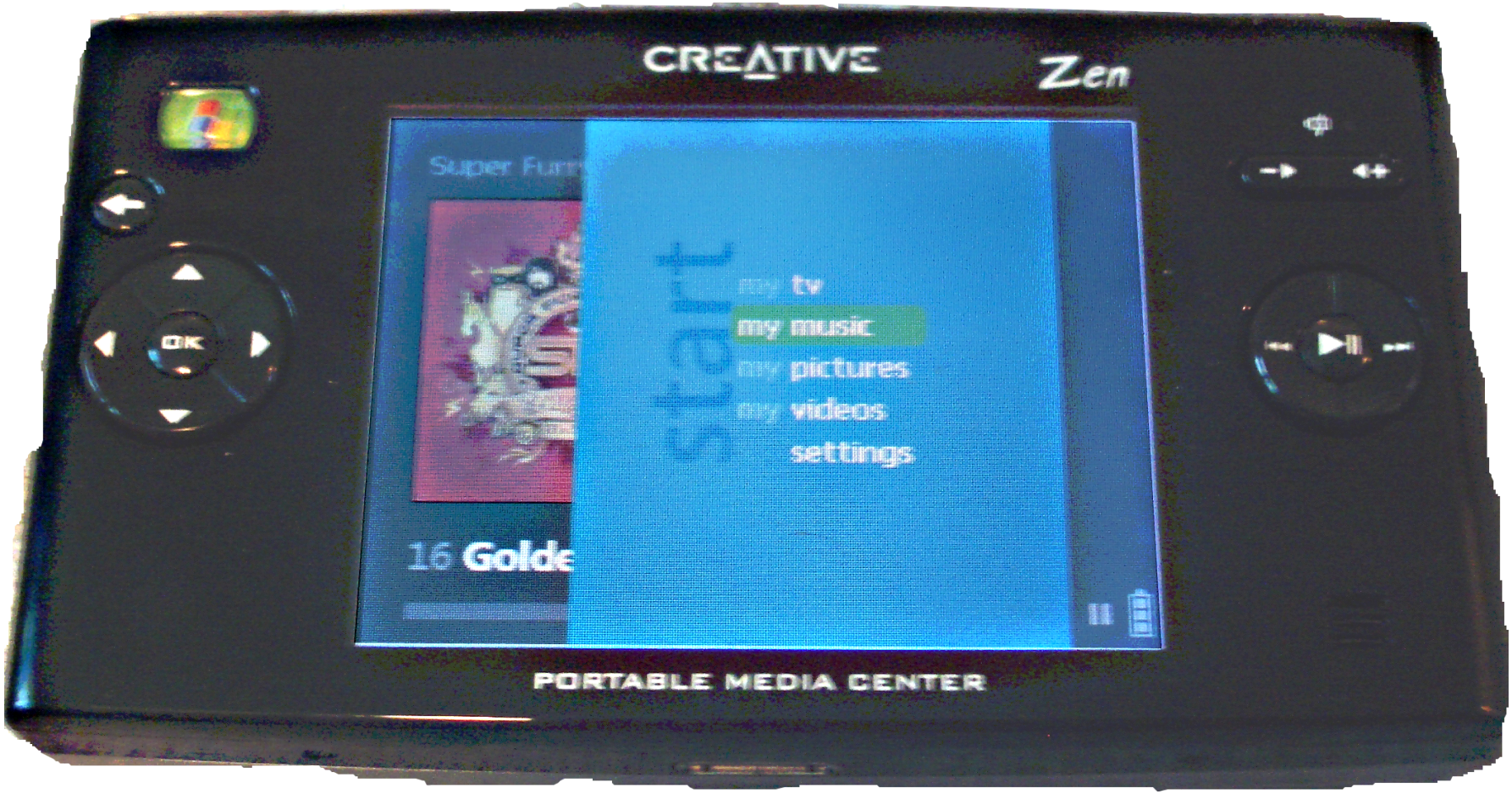
What is the Media Transfer Protocol?

- A superset of **Picture Transfer Protocol**, ISO 15740
- A transactional file system that reminds you of FTP - the device always “owns” the underlying file system, no random access
- Dropped transfers (read: unplugged USB cable) does not corrupt the device file system
- Completely binary
- Determined size of transactions- does not allow streaming to file of unknown final size
- Fine-grained semantics defined by behaviour of Windows MTP daemon akin to how SAMBA is developed to mimic Windows SMB daemon

What is libmtp?

- Userspace library in turn based on libusb (and udev) - today I would probably write parts of it in the kernel
- The host side “initiator” implementation of MTP used in all Linux and MacOS programs talking to MTP devices, notably GNOME VFS MTP backend and KDE kio-mtp backend for desktops
- Based on the libgphoto2 generic camera library ptp2, part of gPhoto, and maintained as a “synchronized fork” in cooperation with libgphoto2 maintainer Markus Meissner
- Began in january 2006 as Creative Labs moved from custom protocol (libnjb) to MTP as part of the Portable Media Center push from Microsoft in 2004

MTP Was Introduced with the PMCs



Apple competitor product family marketed with “PlaysForSure” WMDRM (compare FairPlay) ideas in 2004.
Source: Wikimedia Commons

Low Level USB Interface

```
lsusb -v
Bus 001 Device 006: ID 0b05:4cd0 ASUSTek Computer, Inc.
(...)
Device Descriptor:
(...)
  bDeviceClass            0 (Defined at Interface level)
(...)
  Configuration Descriptor:
(...)
   Interface Descriptor:
(...)
   bNumEndpoints           3
   bInterfaceClass       255 Vendor Specific Class
   bInterfaceSubClass    255 Vendor Specific Subclass
   bInterfaceProtocol      0
   iInterface              4 MTP
```

```
Endpoint Descriptor:
        bLength                 7
        bDescriptorType         5
        bEndpointAddress        0x81  EP 1 IN
        bmAttributes            2
        Transfer Type           Bulk
        Synch Type              None
        Usage Type              Data
        wMaxPacketSize          0x0200  1x 512 bytes
        bInterval               0
Endpoint Descriptor:
        bLength                 7
        bDescriptorType         5
        bEndpointAddress        0x02  EP 2 OUT
        bmAttributes            2
        Transfer Type           Bulk
        Synch Type              None
        Usage Type              Data
        wMaxPacketSize          0x0200  1x 512 bytes
        bInterval               0
Endpoint Descriptor:
        bLength                 7
        bDescriptorType         5
        bEndpointAddress        0x82  EP 2 IN
        bmAttributes            3
        Transfer Type           Interrupt
        Synch Type              None
        Usage Type              Data
        wMaxPacketSize          0x001c  1x 28 bytes
        bInterval               6
```

OS Descriptor

- According to the spec, MTP should be use the PTP device class (0x06) with Interface subclass 0x01
- Yet nobody does, let me guess: Windows does not support this
- Instead you must:

(1) attempt to get device descriptor 0xEE, if this contains the magic letters “MSFT” you send

(2) a special control message which then returns another set of magic bytes containing the string “MTP” and then

(3) a second control message which usually return the same thing again

- If any of these mismatch, the device is not MTP

OS Descriptor

```
Microsoft device descriptor 0xee:
    0000: 1203 4d00 5300 4600 5400 3100 3000 3000   ..M.S.F.T.1.0.0.
    0010: 3000                                     0.
Microsoft device response to control message 1, CMD 0x30:
    0000: 2800 0000 0001 0400 0100 0000 0000 0000   (...............
    0010: 0001 4d54 5000 0000 0000 0000 0000 0000   ..MTP...........
    0020: 0000 0000 0000 0000                       ........
Microsoft device response to control message 2, CMD 0x30:
    0000: 2800 0000 0001 0400 0100 0000 0000 0000   (...............
    0010: 0001 4d54 5000 0000 0000 0000 0000 0000   ..MTP...........
    0020: 0000 0000 0000 0000                       ........
```

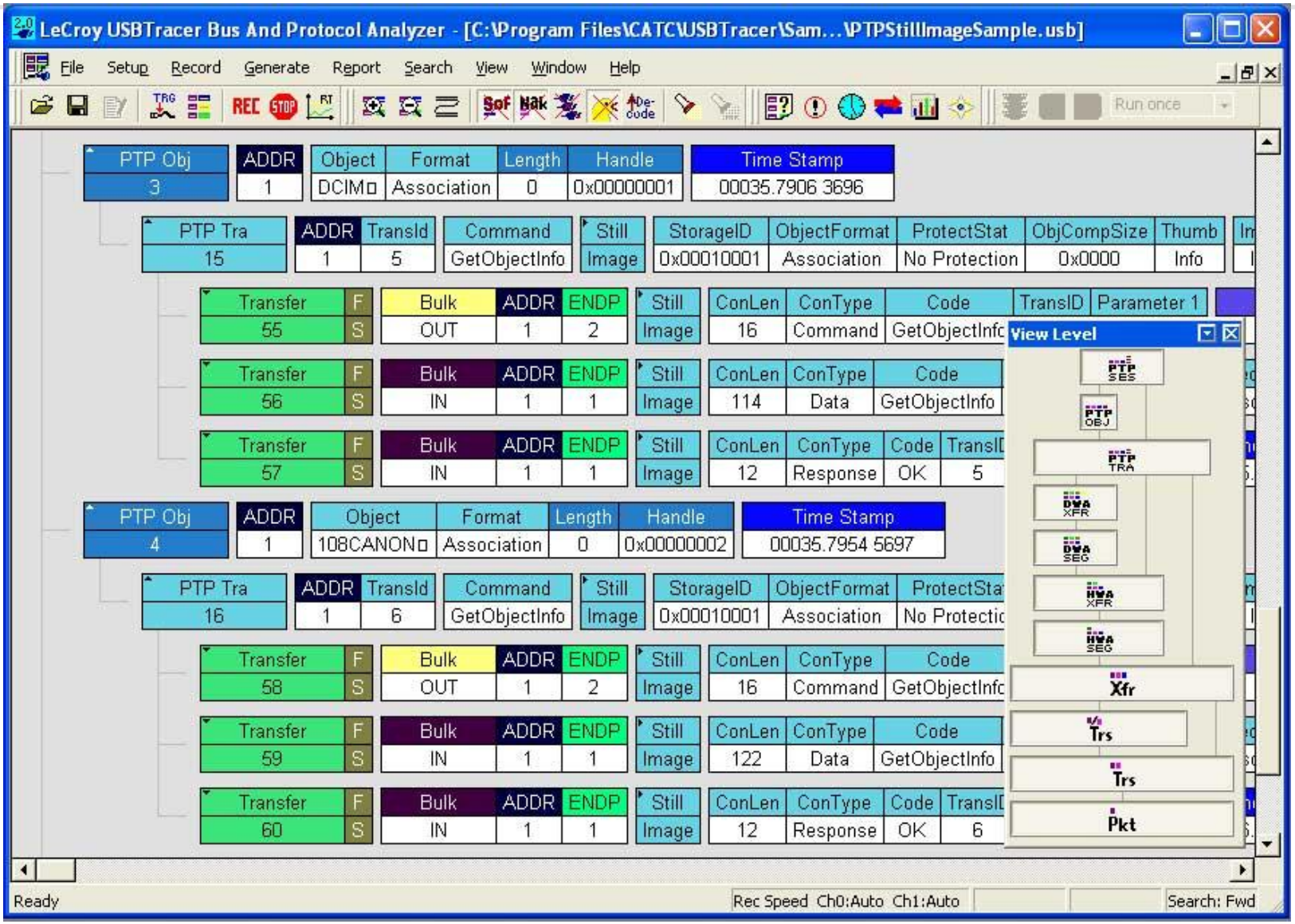
PTP/MTP basics

- PTP/MTP can open a **session** with the device
- During the session data is sent and retrieved
- Basic **device info** can be retrieved, like serial number or battery level
- A number of **operations** can be performed on the device, like enumerating storages, most importantly getting and sending *objects*
- It is possible to ask the device what objects it supports and what kind of metadata can be tagged onto objects
- **Events** can be triggered from the device, such as for a new object being added on the device when it is plugged into a host

Objects, 1st generation devices

- PTP and MTP deal with **objects**, not files. These have a unique 32bit unsigned ID
- Each object has a unique type **object format code**, a 16bit unsigned number, could be MP3 (0x3009), MPEG (0x300b) etc akin to MIME types
- Each object then has a set of associated metadata called **object properties**
- Metadata is the usual stuff like modification date and name, metadata exist for things like artist and album name, composer, genre, duration, … can be strings, u64, u32, u16, u8 or arrays of unsigned - and numbers can be restricted to enumerators

Message Format



What is then MTP-specific actually?

- You can actually ask the device what properties are supported for a certain object type (operation 0x9801, GetObjectPropsSupported)
- Accelerated operations to send or retrieve object properties as a big binary list instead of one-by-one (GetObjPropList, SetObjPropList, SendObjPropList)
- Operations to get and set object references for playlists and albums (or any other abstract list)
- Then a lot of weird WMDRM commands that nobody uses and even weirder commands for the Zune and early Windows Phone DRM (not really part of the spec)

Example of what the devices say

```
Device info:
   Manufacturer: Samsung Electronics Co., Ltd.
   Model: Samsung Wave(GT-S8500)
   Device version: S8500XXJF1
   Serial number: 35922303026479
   Vendor extension ID: 0x00000006
   Vendor extension description: microsoft.com: 1.0;
  microsoft.com/WMPPD: 11.0; microsoft.com/WMDRMPD: 10.1;
  Microsoft.com/DeviceServices:1.0;
   Detected object size: 64 bits
Supported operations:
   1001: get device info
   1002: Open session
   1003: Close session
   1004: Get storage IDs
   1005: Get storage info
   1006: Get number of objects
   1007: Get object handles
(...)
Events supported:
   0x4001
   0x4004
(...)
Device Properties Supported:
   0x5001: Battery Level
   0xd401: Synchronization Partner
   0xd402: Friendly Device Name
(...)
```

```
Playable File (Object) Types and Object Properties Supported:
(...)
   3009: MP3
   dc01: Storage ID UINT32 data type ANY 32BIT VALUE form READ
   ONLY
   dc02: Object Format UINT16 data type ANY 16BIT VALUE form
   READ ONLY
   dc03: Protection Status UINT16 data type enumeration: 0, 1,
   32770, 32771,  READ ONLY
   dc04: Object Size UINT64 data type READ ONLY
   dc07: Object File Name STRING data type REGULAR EXPRESSION
   FORM GET/SET
   dc0b: Parent Object UINT32 data type ANY 32BIT VALUE form
   READ ONLY
   dc41: Persistant Unique Object Identifier UINT128 data type
   READ ONLY
(...)
   dc46: Artist STRING data type GET/SET
   dc89: Duration UINT32 data type range: MIN 0, MAX -1, STEP 1
   GET/SET
   dc8b: Track UINT16 data type ANY 16BIT VALUE form GET/SET
   dc8c: Genre STRING data type GET/SET
   dc97: Effective Rating UINT16 data type range: MIN 0, MAX
   100, STEP 1 GET/SET
   dc99: Original Release Date STRING data type DATETIME FORM
   GET/SET
```

Objects, tagged with metadata

- The file representation was initially flat, without any folder hierarchy. Early MP3 players just have a crude filesystem for everything, and an object and metadata database (file system + BerkeleyDB on an RTOS typically)
- Folders and similar are then implemented on top of the object storage through abstract entities: association objects which may be a “generic folder”, albums and playlists
- All of these things are just lists of 32bit unsigned words indicating associated object IDs

First generation object access API

- LIBMTP_Get_Filelisting() - returns a list of everything in the database on all storages, then you have to build a view of it in memory
- Accessors to send and retrieve files, also send tracks (audio or other multimedia) and tag them with metadata
- LIBMTP_Get_Folder_List() - a separate API for dealing with folders as abstract objects - so that clients didn't necessary had to implement hierarchy awareness
- LIBMTP_Get_Playlist_List() and LIBMTP_Get_Album_List() to globally handle abstract objects of these types - makes a lot of sense

How Hard Can it Be?

- It's a straight forward protocol specification, maintained by the USB Implementers Forum, so just implement it and be happy right?
- We actually started libmtp before the spec was even conceived, and it was a Microsoft proprietary protocol.
- It was based on USB low-level protocol sniffing, as we did not agree to the license MS used for the specification (which was however available)
- When the open spec came out we already had all of it implemented
- But there is no conformance test suite … all devices are tested against Windows and Windows only

Dealing with Device Quirks

- We register USB VID+PID tuples, and assign quirk flags
- For device side implementations that we can identify or detect (such as Android), quirks can be added automatically and VID+PID registry is not necessary
- Yet we try to stash our database with every new device that comes out
- The database is also used to generate the udev script that tells userspace that an MTP device was plugged in
- And we have a real nasty binary to autoprobe devices with certain characteristics that indicate they may be MTP devices called “mtp-probe”

Enter Android: No Flatfile Database

- With Android, devices started to use a new access pattern exercising the object associations to mimic a hierarchical file system
- The device was queried in the style of asking for all associations of type folder in the root, then advancing through the hierarchy by asking for folders below another folder etc, which is perfect for GUI VFS representation like GNOME VFS, KDE KIO or Windows File Explorer
- This is divergent to how elder devices are ideally accessed
- Now a proper filesystem is always the backing storage

Second generation object access API

- Implemented by Google for the MacOS Android file transfer client
- LIBMTP_Get_Files_And_Folders(storage, parent) - access a certain storage and get one level of object information below a parent folder (or root) on that storage
- Old API used for creating folders and sending files and tracks below that hierarchy since these APIs already support specifying storage and parents
- The playlist and album management APIs are often unused, devices will often just ignore attempts to create abstract lists and use custom files for this

Device Sins 1

- Devices bug out or ignore the request to close a session due to only being tested with Windows - windows hog the device and wait for you to unplug the USB cable. This can be avoided by testing with libmtp!
- Assuming the host will hog the device immediately make some devices go numb unless accessed in MTP mode within 7 seconds from plug-in
- Devices announce capabilities thay crash if you try to use them, such as getting the device certificate on a lot of Android phones that does not really support Windows DRM (Janus)

Device Sins 2

- Newer devices really does not support the old access pattern of dumping out the whole database from the storage, this is part of the spec but seldom tested
- Using the same VID+PID for several interface modes: some which aren't MTP. HTC Zopo, HD2, Bird (0xbb4/0x0c02) - then we just can't detect the applicable protocol from VID+PID but have to inspect interfaces and endpoints
- Samsung's MTP stack accepts and discards all metadata sent and instead parses ID3 tags and other embedded metadata from the files and is thus only dealing with files

Device Sins 3

- Aricent MTP stack used in earlier SonyEricsson Android devices would generate broken headers (operation code and Transaction ID came back damaged) something that is a bug but was tolerated by the Windows MTP stack and thus not detected
- ...and some other minor bug flags, like the device needing to be reset during initialization to properly connect

Android's MTP Stack

- Android's MTP stack was included in Ice Cream Sanwich 4.0.x
- It announces that it does support MTP accelerated commands Get/Set/Send ObjectPropList but fail so to handle them so we have to detect Android and fall back to generic PTP handling (presumably because this is what Windows does with devices it does not know about, and they have a whitelist of devices to use it on)
- At the same time Android implements two **new** commands for random access in files/objects: GetPartialObject, SendPartialObject, BeginEditObject, EndEditObject, and TruncateObject. GVFS will exploit these commands for random access

Esoteric Features

- MTP contains playback commands for remote control of devices, we have only ever seen two devices that actually say they implement that: Trekstor Vibez and Nokia 808
- Windows WMDRM (Janus) was used by early devices, certificates, secure time etc are compulsary to claim to support but Android devices just disregard it
- ZUNE came and went, the authentication scheme was reverse engineered by a libmtp developer but depended on external keys to perform, and those need be provided by Microsoft
- Microsoft silently dropped the ZUNE authentication features from the Windows Phone product line

Next Steps

- Get libmtp 1.1.7 out the door
- Clean out the bug database
- Apply patches
- Deprecate the old API for 1.2.0 and remove in 1.3.0?

Help us writing libmtp!

- I have been the main maintainer of libmtp even though it is used by many, many frontends
- I need to go over many many bug and device reports and get libmtp 1.1.7 out the door
- We need more people in this project, familiar with MTP devices, libmtp and git
- I spend most of my time in the Linux kernel these days
- Questions?

More about Linaro Connect: http://connect.linaro.org
More about Linaro: http://www.linaro.org/about/
More about Linaro engineering: http://www.linaro.org/engineering/
Linaro members: www.linaro.org/members