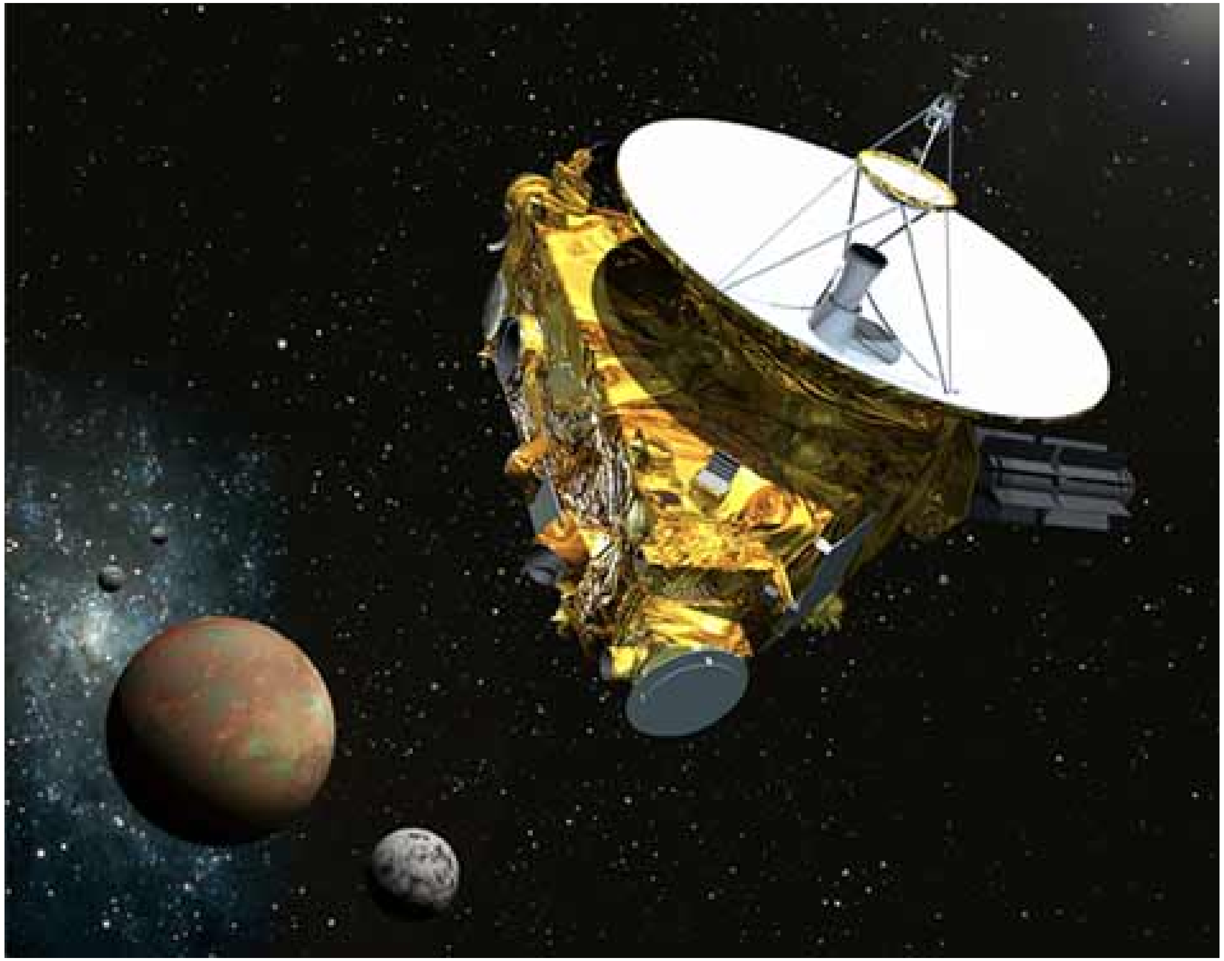
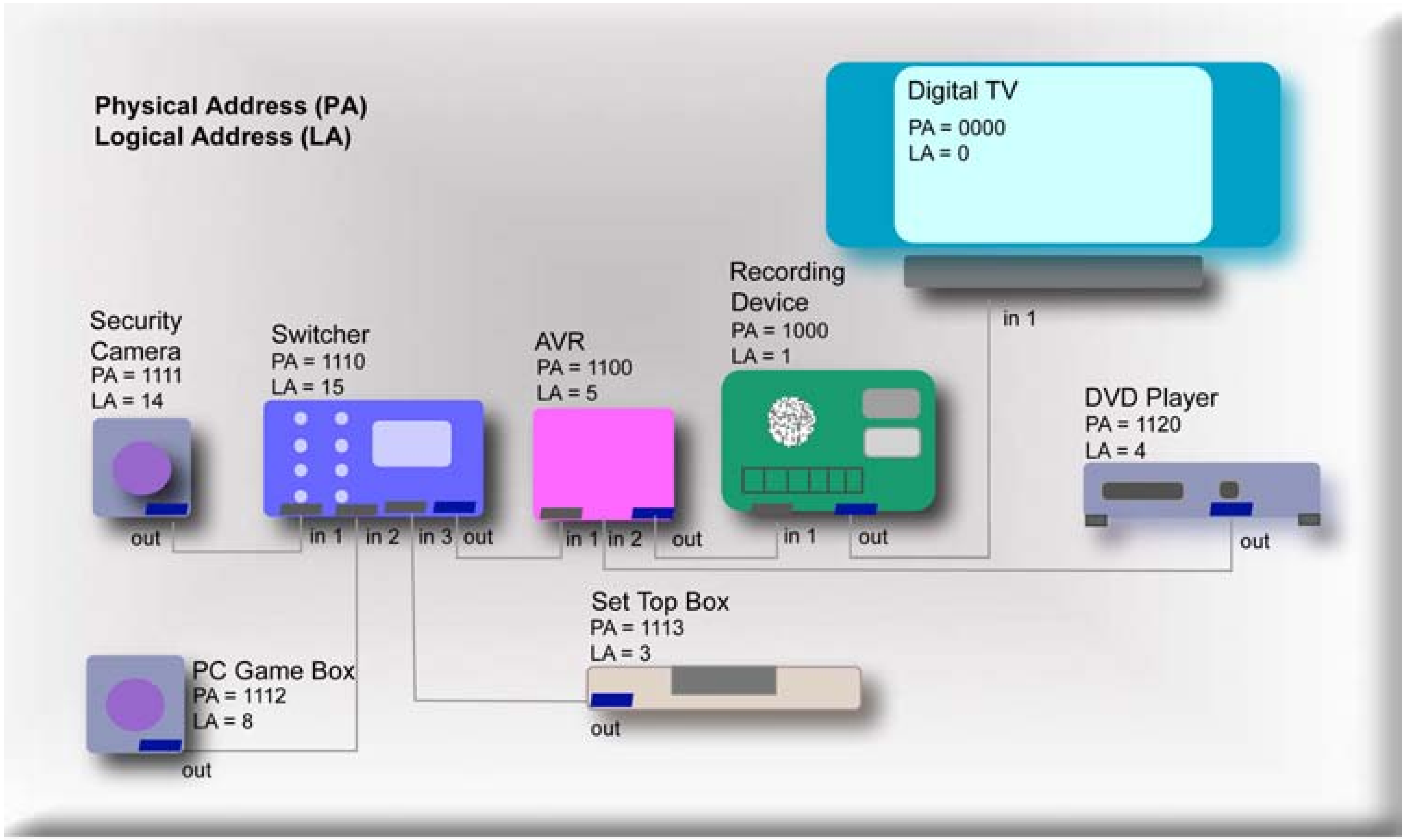
HDMI CEC (Consumer Electronics Control):

What? Why? How?

Hans Verkuil
Cisco Systems Norway



New Horizons Probe @ 4.5 billion km: 1000-2000 bits/s



CEC @ 1 meter: 400 bits/s

What Is CEC?

Consumer Electronics Control

- An optional supplement to HDMI using pin 13 of the HDMI connector.
- Provides high-level control functions between the various audiovisual products in a user's environment.
- Based on the old AV.link scart standard (EN 50157-2-[123]).
- Implemented in HDMI receivers/transmitters and USB HDMI-passthrough devices.
- Data packets: 1 header byte + 0 to 15 data bytes.
- Very, very slow data rate ~400 bits/s.

Physical Address

Physical address: Hierarchy placement

Range:	0.0.0.0 - F.F.F.F	
Root device:	0.0.0.0:	HDMI Sink, typically a TV
First device:	1.0.0.0:	Device connected to HDMI port 1 on root device
Second device:	2.0.0.0:	Device connected to HDMI port 2 on root device
Third device:	1.1.0.0:	Device connected to HDMI port 1 on first device
Invalid:	f.f.f.f:	There is no physical address. CEC is typically disabled in this case

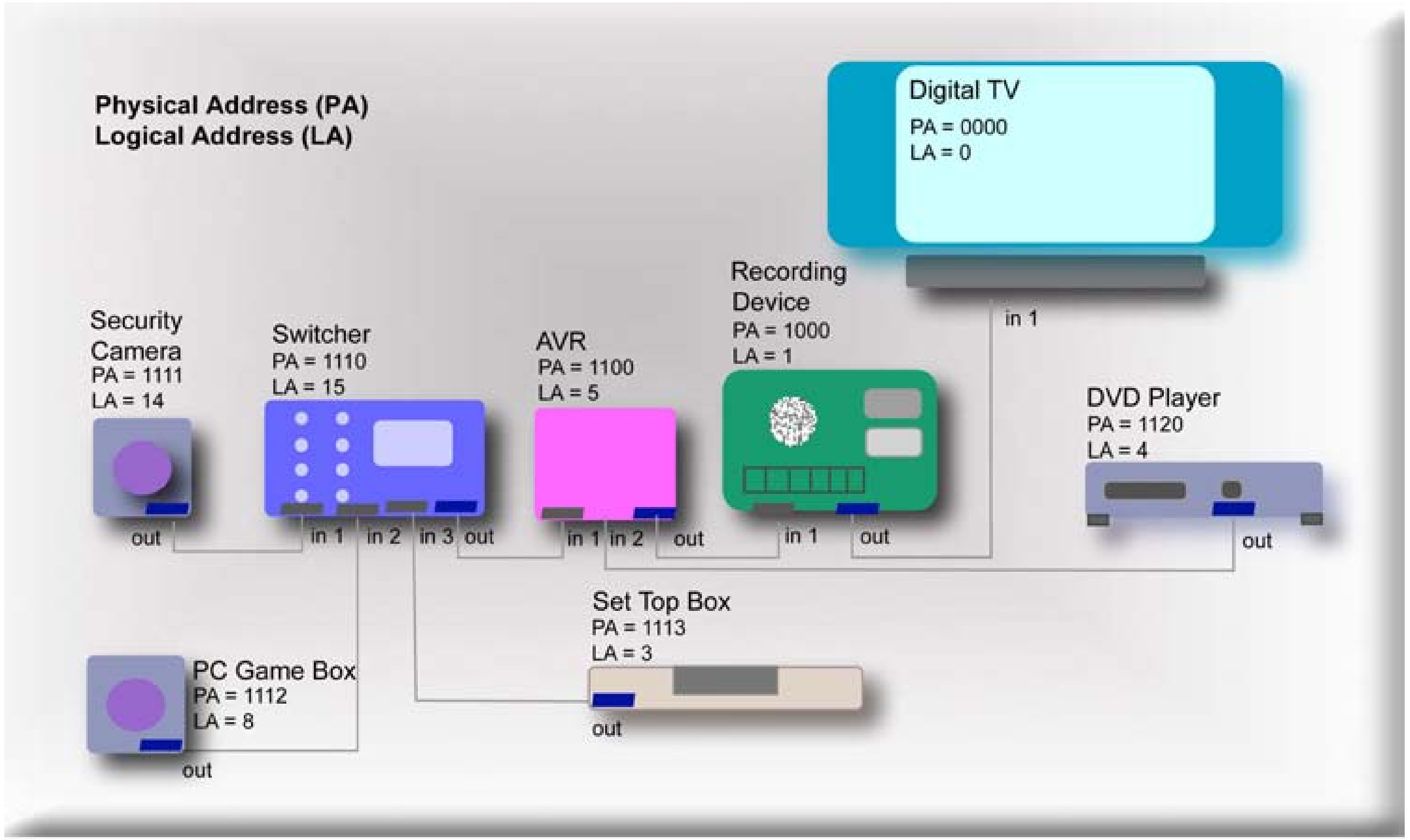
Sources get their Physical address from the EDID obtained from the sink.

Logical Address

Logical address 0-15: product type dependent.
Not really an address, more like a nickname.

0: TV (root device)	8: Playback device 2
1: Recording device 1	9: Recording device 3
2: Recording device 2	10: Tuner 4
3: Tuner 1	11: Playback device 3
4: Playback device 1	12: Backup 1
5: Audio system	13: Backup 2
6: Tuner 2	14: Specific use
7: Tuner 3	15: Unregistered (as Initiator address) Broadcast (as Destination address)

Topology Example



Trade Names

- Anynet+ (SAMSUNG)
- BRAVIA Link (Sony)
- Kuro Link (Pioneer)
- EasyLink (Philips)
- SimpLink (LG)
- VIERA link (Panasonic)

Why Implement CEC?

End-User Features

- One Touch Play: allows a device to be played and become the active source with a single button press.
- System Standby: enables the user to switch all devices to the Standby state with one button press.
- One Touch Record: offers a What You See Is What You Record (WYSIWYR) facility, meaning that whatever is shown on the TV screen is recorded on a selected Recording Device.
- Timer Programming: allows the user to program the timers in a Recording Device from an EPG running on a TV or STB.
- Deck Control: enables a device to control (e.g. play, fast forward etc.) and interrogate a Playback Device (a deck).

End-User Features

- Tuner Control: allows a device to control the tuner of another device.
- Device Menu Control: enables a device to control the menu of another device by passing through user interface commands.
- Remote Control Pass Through: enables remote control commands to be passed through to other devices within the system.
- System Audio Control: allows an Audio Amplifier / Receiver to be used with the TV. The volume can be controlled using any the remote controls of any suitably-equipped devices in the system.

Supporting Features

- Device OSD Name Transfer: enables devices to upload their preferred OSD name to the TV. The TV can then use this name in any menus associated with that device.
- Device Power Status: allows the current power status of a device to be discovered.
- OSD Display: enables a device to use the on-screen display of the TV to display text strings.
- Routing Control: allows the control of CEC Switches for streaming of a new source device.
- System Information: queries the system to determine device addresses and language.
- Dynamic Audio Lipsync: used by sinks to announce their audio latency.

Supporting Features

- Vendor Specific Commands: allows a set of vendor-defined commands to be used between devices of that vendor.
- Audio Rate Control: allows an Amplifier to fractionally increase or decrease the playback rate of an audio source.
- Audio Return Channel Control: controls the Audio Return Channel (ARC) part of the HDMI Ethernet and Audio Return Channel (HEAC).
- Capability Discovery and Control: controls HDMI Ethernet Channel (HEC) part of HEAC.

Capability Discovery and Control (CDC) for HEAC (HDMI Ethernet and Audio Return Channel)

- HEAC provides a full duplex connection between HDMI devices which conforms to 100Base-TX IEEE 802.3 Standard. This is defined as HDMI Ethernet Channel (HEC): very rarely implemented since wifi is used instead.
- HEAC provides audio data streaming which conforms to IEC 60958-1 standard from an HDMI Sink to an HDMI Source or repeater. This is defined as Audio Return Channel (ARC): much more common, used to e.g. transfer TV audio to an AV Receiver.
- For ethernet both the Utility (14) and Hot Plug Detect (19) lines are used. ARC can run in single mode (only pin 14 is used) or common mode (both pins are used).
- HEAC uses CDC for Capability Discovery and Control.
- If the Hot Plug Detect pin is used, then special CDC messages replace the hotplug signal.

Problems

- CEC is optional, so no guarantees.
- CEC version 1.4 leaves too much to the implementor, leading to inconsistent implementations. Version 2.0 is more strict.
- Ad-Hoc protocol: a clear example of a protocol where vendors hack something which then becomes part of the standard.
- Painfully slow and small payload.
- The protocol defines many special cases that are difficult to code. A kernel framework would provide a single place where such constraints can be coded.

How Is CEC Implemented?

Implementation Requirements

- CEC is highly asynchronous, blocking waits must be avoided.
- Replies to messages are out-of-order, not something userspace should have to deal with.
- It depends on the use-case which messages have to be handled by userspace or kernelspace, so both need to be supported.
- Creating and parsing messages should be standardized in a header that can be used both by userspace and kernelspace.
- Needs to support HDMI receivers (V4L2), transmitters (DRM, V4L2) and USB CEC dongles.
- Should handle disconnect/connect scenarios autonomously. That is, userspace shouldn’t have to deal with reclaiming logical addresses after a reconnect.

CEC Framework

- Creates a /dev/cecX device node.
- The driver determines the level of control userspace is allowed.
- Drivers implement the low-level CEC adapter operations.
- Drivers also implement high-level CEC functionality such as ARC and hotplug detect messages (not in the first release).
- The framework deals with the details of the protocol and the asynchronous aspects.
- The framework processes the core CEC messages automatically (unless userspace enables passthrough mode).
- The framework allows monitoring the CEC line.

CEC Framework

- cec-funcs.h contains static inline functions that fill and parse cec_msg structs.
- When transmitting a message you can asynchronously wait for a reply.

```
struct cec_msg {
        __u64 ts;
        __u32 len;
        __u32 timeout;
        __u32 sequence;
        __u8 rx_status;
        __u8 tx_status;
        __u8 msg[16];
        __u8 reply;
        __u8 tx_arb_lost_cnt;
        __u8 tx_nack_cnt;
        __u8 tx_low_drive_cnt;
        __u8 tx_error_cnt;
        __u8 reserved[33];
};

struct cec_msg msg;
cec_msg_init(&msg, 4, 0);
cec_msg_set_osd_string(&msg, CEC_OP_DISP_CTL_DEFAULT,
                       "Hello World");
```

CEC Framework

```
static inline void cec_msg_set_osd_string(struct cec_msg *msg,
                                          __u8 disp_ctl,
                                          const char *osd)
{
        unsigned len = strlen(osd);

        if (len > 13)
                len = 13;
        msg->len = 3 + len;
        msg->msg[1] = CEC_MSG_SET_OSD_STRING;
        msg->msg[2] = disp_ctl;
        memcpy(msg->msg + 3, osd, len);
}

static inline void cec_ops_set_osd_string(const struct cec_msg *msg,
                                          __u8 *disp_ctl,
                                          char *osd)
{
        unsigned len = msg->len - 3;

        *disp_ctl = msg->msg[2];
        if (len > 13)
                len = 13;
        memcpy(osd, msg->msg + 3, len);
        osd[len] = '\0';
}
```

CEC Framework

```
static inline void cec_msg_cec_version(struct cec_msg *msg,
        __u8 cec_version)
{
        msg->len = 3;
        msg->msg[1] = CEC_MSG_CEC_VERSION;
        msg->msg[2] = cec_version;
}

static inline void cec_ops_cec_version(const struct cec_msg *msg,
                                       __u8 *cec_version)
{
        *cec_version = msg->msg[2];
}

static inline void cec_msg_get_cec_version(struct cec_msg *msg,
                                           bool reply)
{
        msg->len = 2;
        msg->msg[1] = CEC_MSG_GET_CEC_VERSION;
        msg->reply = reply ? CEC_MSG_CEC_VERSION : 0;
}
```

CEC Adapter Driver

```
struct cec_adap_ops {
        /* Low-level callbacks */
        int (*adap_enable)(struct cec_adapter *adap, bool enable);
        int (*adap_monitor_all_enable)(struct cec_adapter *adap, bool enable);
        int (*adap_log_addr)(struct cec_adapter *adap, u8 logical_addr);
        int (*adap_transmit)(struct cec_adapter *adap, u8 attempts,
                             u32 signal_free_time, struct cec_msg *msg);
        void (*adap_log_status)(struct cec_adapter *adap);

        /* High-level CEC message callback */
        int (*received)(struct cec_adapter *adap, struct cec_msg *msg);
};

void cec_transmit_done(struct cec_adapter *adap, u8 status, u8 arb_lost_cnt,
                       u8 nack_cnt, u8 low_drive_cnt, u8 error_cnt);
void cec_received_msg(struct cec_adapter *adap, struct cec_msg *msg);
```

CEC API

- CEC_ADAP_G_CAPS: returns the capabilities of the CEC adapter. Most important is CEC_CAP_PHYS_ADDR which if set means that userspace has to set the physical address, normally done by reading the EDID.
- CEC_ADAP_LOG_STATUS: log the current CEC adapter status.
- CEC_ADAP_G/S_PHYS_ADDR: get/set the physical address for the CEC adapter.

CEC API

- CEC_ADAP_G/S_LOG_ADDRS: get/set the logical addresses and other fixed information for the CEC adapter.

```
struct cec_log_addrs {
        __u8 log_addr[CEC_MAX_LOG_ADDRS];
        __u16 log_addr_mask;
        __u8 cec_version;
        __u8 num_log_addrs;
        __u32 vendor_id;
        char osd_name[15];
        __u8 primary_device_type[CEC_MAX_LOG_ADDRS];
        __u8 log_addr_type[CEC_MAX_LOG_ADDRS];
        /* CEC 2.0 */
        __u8 all_device_types[CEC_MAX_LOG_ADDRS];
        __u8 features[CEC_MAX_LOG_ADDRS][12];
        __u8 reserved[65];
};
```

CEC API

- CEC_G/S_MODE: get/set the mode of the filehandle. Filehandles can be in initiator mode and/or in follower mode. In initiator mode they can transmit CEC messages and receive directed replies. In follower mode they can receive CEC messages and are expected to handle those messages if needed.
- Initiator modes:
  - `CEC_MODE_NO_INITIATOR`: cannot initiate CEC messages.
  - `CEC_MODE_INITIATOR`: can initiate CEC messages, unless there is an exclusive initiator filehandle.
  - `CEC_MODE_EXCL_INITIATOR`: the only filehandle that can initiate CEC messages.

CEC API

- Follower modes:
  - `CEC_MODE_NO_FOLLOWER`: will not follow received CEC messages.
  - `CEC_MODE_FOLLOWER`: will follow CEC messages, unless there is an exclusive follower filehandle.
  - `CEC_MODE_EXCL_FOLLOWER`: the only filehandle that can follow CEC messages.
  - `CEC_MODE_EXCL_FOLLOWER_PASSTHRU`: the only filehandle that can follow CEC messages. It also puts the CEC framework into passthrough mode where it will pass on all message and no longer process core CEC messages.
  - `CEC_MODE_MONITOR`: put the filehandle in monitor mode. This filehandle cannot initiate messages, nor is it a follower.
  - `CEC_MODE_MONITOR_ALL`: put the filehandle in monitor all mode. This filehandle cannot initiate messages, nor is it a follower.

CEC API

- `CEC_RECEIVE/TRANSMIT`: receive and transmit CEC messages. Optionally CEC_TRANSMIT can wait for a directed reply.
- `CEC_DQEVENT`: event handling: when the physical or logical addresses change userspace will be informed. Important to detect disconnect/connect changes.

CEC Utilities

- cec-ctl supports all CEC messages (autogenerated code) and is a quick way of interactively configuring a CEC adapter and sending, receiving and monitoring CEC messages.
- cec-compliance will eventually do a proper CEC compliance test to see if the CEC implementation is correct (i.e. all messages that should be implemented are implemented).

CEC To Do

- Improve cec-compliance to increase the test coverage of the CEC API.
- Finish support for the Pulse Eight USB dongle and omap4 CEC support.
- Simplify the adv7604/7842/7511 CEC implementation.
- Merge the framework!
- Future work:
  - Add ARC/CDC Hotplug support
  - Incorporate more high-level CEC requirements in the framework. For example:

    “A device shall not send a <Polling Message> which is not a re-transmission attempt to the same address more frequently than once per Minimum Polling Period, which is defined as 14 seconds. Note that longer periods between <Polling Messages> are preferred.”

    or:

    “If System Audio Mode is ON, the TV may poll the Audio System no more than once every 5 seconds, in order to ensure that the Audio System is still connected.”

Resources

Resources

- Git repository for the CEC framework:
  http://git.linuxtv.org/hverkuil/media_tree.git/log/?h=cec15
  Details on using the cec framework in the kernel are in Documentation/cec.txt
- Git repository for the cec-ctl/cec-compliance utilities:
  http://git.linuxtv.org/cgit.cgi/hverkuil/v4l-utils.git/log/?h=cec
- Current CEC API documentation:
  https://hverkuil.home.xs4all.nl/cec.html#cec
- My email: hverkuil@xs4all.nl

Questions?