Ookala Modular Calibration Framework
Technical Reference Overview

Hewlett-Packard Development Company, L.P.

4 November 2008

Revision History
- Revision 0.8 - 04 Nov 08 - Beta evaluation updates
- Revision 0.7 - 29 Sep 08 - Development updates
- Revision 0.6 - 17 Sep 08 - Initial

Table of Contents

Introduction

The Ookala Modular Calibration Framework (OMCF) is an open-source project designed to support extensible, standardized color calibration of color-critical display environments. The modular architecture allows for easy customization of a variety of display output and color sensor scenarios.

The initial software release demonstrates a sample utility suite based upon the HP DreamColor LP2480zx Professional Display and the HP DreamColor colorimeter based upon X-Rite, Incorporated's hardware and software technologies.

The OMCF is based upon open standards such as wxWidgets (wxwidgets.org) for GUI support, allowing cross-platform solutions to expand as community development continues.

Software Overview

Architecture

The Ookala Modular Calibration Framework (OMCF) is a simple, dynamically defined environment for executing plugin modules as runtime chains to perform display and sensor communications, color engine calculations, and limited state archiving. Rapid development and execution can occur from within the full calibration application (***ucal***) or from standalone utility examples (***hpdc_util***, ***read_sensor***) provided. Both tie to the centralized library manager (***libookala***) that manages a plugin registry and data storage structure.

Color sensor interfaces are facilitated through the use of OEM software development kits (SDK) specific to those instruments and operating system requirements. The Linux-derived examples rely upon *libusb* interfaces for convenient user-space implementations (rather than more rigid kernel-module implementations).

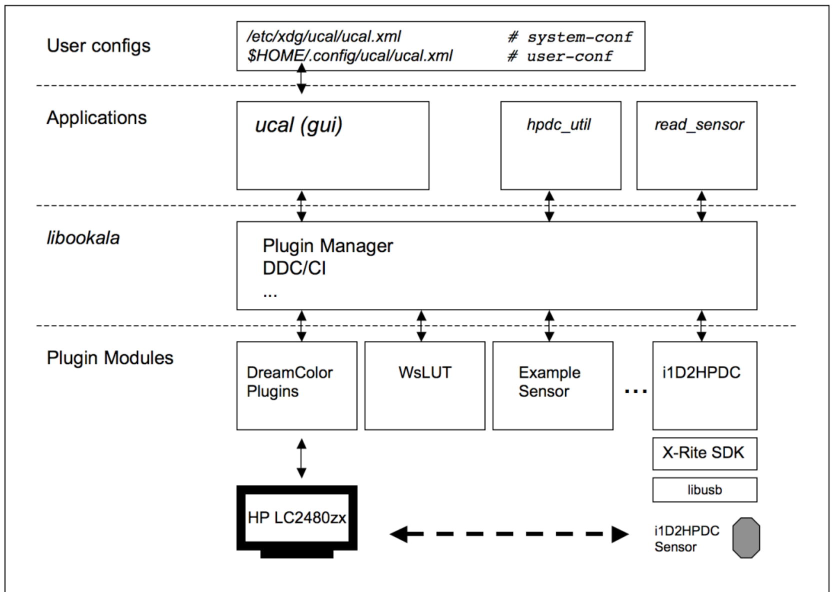

Figure 1: Ookala Modular Calibration Framework

Component Applications

Two sets of development applications are described below. Refer to the reference appendix at the end of this overview for a more comprehensive review of each program.

ucal: GUI-driven framework

The *ucal* application framework provides the primary color calibration interface for driving user-defined plugin modules as execution chains for monitor and color sensor combinations. Upon execution, the application will:

- read and parse a system-defined (`/etc/xdg/ucal/ucal.xml`) or user-defined XML *ucal* configuration file (`$HOME/.config/ucal/ucal.xml`) defining plugin modules, data dictionary parameters, and runtime chain options
- build and initialize GUI in desktop taskbar
- setup internal management requirements
- process chain requests through main event loop until exit
- cleanup resources as required

A commented sample XML *ucal* configuration file for the HP DreamColor LP2480zx is provided that demonstrates a variety of interactive plugin chain utilities to calibrate the target monitor (the list below are separate menu options to execute a particular chain action):

- *sweep gamma* - interactive gamma-lut animator
- *randomize* - randomizes current lut contents
- *reset lut* - resets a lut linear ramp
- *high-gamma lut* - display a fixed gamma (4.0)
- *low-gamma lut* - display a fixed gamma (0.5)
- *calibrate* - perform a calibration pass with defined sensor and display (D65/REC709)
- *switch calibration targets* - set to target 0 (preset 6)
- *switch calibration targets* - set to target 1 (preset 5)
- *check calibration status* - reload current saved state from file (/tmp/records.xml)
- *calibrate other color spaces* - calibrate against HP DreamColor factory presets

hpdc_util: HP DreamColor applet for interactive plugin development
read_sensor: sensor-specific utility for interactive plugin development

Two complementary example utilities are provided for quick development and interactive test of color sensor plugin modules.

The *hpdc_util* provides an elementary command-line interface to the HP DreamColor LP2480zx Professional Display to switch color spaces, query/set matrices, snapshot pre- and post-lut settings, control color processing stages, enable the pattern generator with an RGB color, control the OSD state, and provide a few other useful development controls.

The *read_sensor* applet interfaces directly with the ookala library and designated color sensor plugin without requiring the entire *ucal* environment or plugin chain definition. The example provided shows developers how to directly call their color sensor plugin (and its data dictionary) to read and report measurements of scenarios driven by the *hpdc_util*.

Library *libookala*

The core library, *libookala*, features include:

- plugin registry management
- dictionary items management (data store for plugin controls and parameters)
- general Display Data Chanel (DDC/CI) interfaces
- basic color calculation facilities such as interpolation methods
- data archiving interfaces (DataSavior),
- class definitions for Plugin, Sensor, PluginChains, etc.

Plugins

Sample plugin modules are provided to interface with the HP DreamColor LP2480zw Professional Display. These include:

- *DreamColor* - interface module for DreamColor calibration, control, and color space management
- *WsLut* - host-side (not display-side) interface for handling window-system lookup tables including reset and randomize
- *DevI2c* - basic device I2c interface controls
- *ExampleSensor* - generic example of color sensor interface class with dummy sensor
- *WxBasicGui, WxDreamColorCalib, WxLutSweep, WxSensorPrep* - wxWidgets interface examples for GUI, DreamColor calibration, lookup table sweep, and color sensor preparation

---

Examples

The following examples provide a quick, layered introduction to the OMCF installation as detailed in the *Reference Appendices* that follow.

1. **Build Installation Sanity Check**

Following a successful build and installation on an HP Linux Workstation of the OMCF and wxWidgets library interface, the *read_sensor* applet (if configured to use the dummy ExampleSensor plugin described in the appendix) will interface with the libookala library, dynamically load the ExampleSensor plugin, define a data dictionary entry, and call the plugin with its measureYxy() interface, reporting the following:

```
# read_sensor -n 1 /usr/local/ookala/lib
Loaded Ookala::ExampleSensor from /usr/local/ookala/lib/libExampleSensor.so
 actionTaken: detect and open sensor device...
 actionTaken: confirm state...
 DictOptions: setup base defaults...
 DictOptions: setup LCD parameters...
 DictOptions: Calibration Index set1
```

```
measureXY: selecting calibration matrix...
measureXY: setup LCD or CRT...
measureXY: integration time applied...
measureXY: reading measurement Yxy (candela)...
measureXY: read (dummy) values Yxy (1.0, 0.31, 0.32) ...
1.000000 0.312779 0.329183 (dummy sensor)
disposing of open sensor
```

This sample assumes that the *read_sensor* applet was compiled with `main.cpp.ExampleSensor` as its target source file and the ExampleSensor plugin module was configured, built, and installed.

It will check not only proper build and installation of libraries and library assets, but it will also confirm that the dynamic plugin module interface is working on the target Linux distribution.

Note: the plugin module pathname in the command line is used to point at development directories for a given module already hard-coded into the source code for convenience... this allowed for flexible development and test... the requirement for this option can be changed to suit a developer's workflow.

2. **HP DreamColor Monitor Query**

The *hpdc_util* command-line application can provide a quick, useful interface to addressing the state of the monitor through the *libookala* interface. Executing an inquiry will determine if proper *i2c* communications channels are functioning for your Linux workstation.

The following command should result in a dump of the current HP LP2480zx state:

```
hpdc_util -q /usr/local/ookala/lib
```

The full functionality described by the help menu of *hpdc_util* is listed in its Reference Appendix: Advanced Topic entry at the end of this document.

Similarly, the *read_sensor* command-line application can be modified to setup and call a color sensor plugin that interfaces with a 3rd-party instrument library (SDK) and *libusb*. The modifications include:

- properly defining the in-line plugin definition for target color sensor, e.g., "libi1D2HPDC.so"
- modifying the data dictionary (Dict Options) section for any setup parameters for the plugin
- executing a `SENSOR_ACTION_DETECT` callback to properly discover and open the device
- optionally executing a `SENSOR_ACTION_DARK_READING` if required for your device (LCD prep)
- calling the sensor measureYxy() for a reading

This simple demonstration is useful for not only directly debugging a new plugin module (for example, a new color sensor interface), but it can complete a simple workflow measurement when paired with *hpdc_util* to drive (and verify) the HP LP2480zx and its DreamColor color engine.

3. ***ucal*** **Exercise User Environment**

When the libookala / OMCF installation is built and verified (and a proper color sensor is engaged through its plugin module - confirmed in step 2 above), then the *ucal* application can be started to test and confirm the final user environment.

a. configure the desired runtime scenario (e.g., color sensor and display) using the instructions described in the appendix entry for ***runtime configuration***
b. confirm that the target color sensor is available, calibrated, and functioning by running a read_sensor pass on it (from section 2 above)
c. launch the *ucal* application if all is well - it will likely reside at `/usr/local/ookala/bin/ucal`

d. when the *ucal* application is running, it displays itself in the taskbar of the current desktop - a mouse click should present the list of chain actions made available from the default *ucal.xml* sample that was configured for this purpose

e. using the described action chains, a user should be able to perform a number of default color calibration tasks including a full custom calibration and calibration against factory preset defaults - see the list described in the framework summary covered during the Software Overview section that started this document

Note: the sample ucal.xml configuration file will save monitor state to the file */tmp/records.xml* unless redefined otherwise.

Reference Appendices

Reference: Software and Use License

Reference: System Requirements

The Ookala Modular Calibration Framework (OMCF) was developed and tested on Red Hat Enterprise Linux (RHEL) systems, running workstation client versions 4.6 (i386, x86_64) and 5.2 (i386, x86_64). Known issues are recorded in a separate section of this document.

This software was developed for the following monitor types:

- HP DreamColor LP2480zw Professional Display

Additional software component requirements for compile and use of framework include:

- *xwWidgets* (wxWidgets.org) - cross-platform toolkit for GUI (version 2.8.8 or later)

For Linux development, the following components were used for compile and test:

- *GNU g++, automake* - development environment native to RHEL distributions tested
- *Qt qmake* - some makefiles in related support packages use this technology (not required for OMCF)
- *X-Rite* (xrite.com) - for utilization of these color sensors, licensed X-Rite Linux-compatible software development toolkits (SDK) are required for sensor plugin development (*this includes the X-Rite Eye-One C SDK for HP Linux Workstations - i1CSDK-3.4.1.132 or later*)

- *libusb* - sensor libraries such as those provided by X-Rite require this support
- *libusb-devel* - some environments to not load these packages automatically (files such as `/usr/include/usb.h` required for support plugins)
- *libxml2* - XML parser required for libookala configuration files
- *i2c-dev* - at runtime, this kernel module must be available to facilitate proper DDC/IC communication with the monitor (an init script, `omcf-init`, is included to help facilitate this at startup of some distributions)

Reference: Build Example

A sample *rpm*-compatible spec file is included with this distribution. If a customized installation is necessary, the following sample can be adapted for your environment. The following example assumes a RHEL 4.6 or equivalent Linux system. Paths are relative for a i586 (32-bit) system. A sample build script (`ookala_manual_build.sh`) for this procedure is also available in the scripts directory.

1. Create target directories and populate with required components, e.g., wxWidgets, 3rd party sensor headers/libraries, etc:

    Ookala target directories:
    - `/usr/local/ookala/bin`
    - `/usr/local/ookala/docs`
    - `/usr/local/ookala/icons`
    - `/usr/local/ookala/include`
    - `/usr/local/ookala/lib[64]`
    - `/usr/local/ookala/scripts`

    wxWidgets, 3rd-party sensor headers / libraries:
    - `/usr/local/bin`
    - `/usr/local/include`
    - `/usr/local/lib[64]`

    wxWidgets note: we configured and built with:
    - `configure --libdir=/usr/local/lib[64] --enable-monolithic --with-opengl \`
      `--enable-shared --with-gtk`
    - if built and installed manually, update dynamic linker cache with `ldconfig /usr/local/lib[64]`

2. Update local copy of OMCF configure script to most current status:
    - `aclocal; autoconf; automake`

3. Configure OMCF for build. Unique flags of interest include:

    ```
    configure [OPTION]... [VAR=VALUE]...
    ```

    - `--x-includes`   # X11 includes location (if undefined $x_includes)
    - `--x-libraries`   # X11 library location (if undefined $x_libraries)
    - `--with-examplesensor`   # ExampleSensor (default dummy sensor)
    - `--with-chroma5`   # X-Rite Chroma5 (requires X-Rite licensed SDK)
    - `--with-i1d2hpdc`   # HP i1D2 X-Rite sensor (requires X-Rite licensed SDK)
    - `--with-wx-gcc4`   # GNU g++ compiler (v4.x or later) needs help with wxWidgets

    Note: Each plugin module option also includes a --with-XXX-headers=PATH and --with-XXX-libs=PATH option that is recommended for use when configured. See `configure` help menu after properly created by step 2 above.

4. Build source via make command:

    ```
    make
    ```

5. Install successful build via make command:

```
        make install
```

6. Refresh dynamic library cache:

```
        ldconfig /usr/local/lib
        ldconfig /usr/local/ookala/lib[64]
```

7. Copy icons to target location from icons directory (not currently configured with installer):

```
        cp icons/* /usr/local/ookala/icons
```

Documentation and additional scripts can also be copied to target directories as needed.

Note: Most build scenarios will complain or fail due to insufficient work for step 1 (building, installing, and defining the correct locations for headers and libraries).

Reference: Runtime Configuration

Basic runtime configuration for Linux workstation consists of:

1. Updating execution search paths:

```
        # export PATH=$PATH:/usr/local/ookala/bin
```

2. Properly configuring the ucal configuration file:

   a. for system-wide defaults, copy a version of *ucal.xml.sample* to `/etc/xdg/ucal/ucal.xml`
   b. for user personalization, copy a version of *ucal.xml.sample* to `$HOME/.config/ucal/ucal.xml`
   c. edit ucal.xml to reflect your color sensor scenario, e.g., replace all instances of *MySENSOR* with *ExampleSensor* (the sample dummy sensor) for example
   d. if you locate assets such as *icons* or *plugin module* locations, update those, too, as required (details are provided as inline documentation)
   e. search priorities for this file are determined by the following environment variables and paths in the following order:
      - $XDG_CONFIG_HOME = `$HOME/.config/ucal/ucal.xml`
      - $XDG_CONFIG_DIRS = `/etc/xdg/ucal/ucal.xml`

3. Specifying Ookala modules and library paths
   - *optional* - only required for *hpdc_util* and *read_sensor* examples because they access these modules directly from the specified path - (see their command line parameters)

4. Xorg.conf will require the *extmod* (extension module) to support the *WsLut* example module used in the sample ucal.xml configuration file (*WsLut* requires `XF86VidModeQueryExtension`)

5. On Linux systems, DDC/IC is accomplished via *i2c* device accesses. For RHEL 5.2 systems, the i2c-dev is not automatically loaded (used primarily with `lm_sensors` package). An example /etc/init.d script is provided (`scripts/omcf-init`) or an equivalent call to modprobe is in order if i2c_dev is not available in the user environment:

```
        /sbin/modprobe i2c-dev > /dev/null 2>&1
```

6. Pluggable Authentication Modules (PAM) protected systems may require non-administrator access configuration for USB color sensors. Based upon the Linux distribution, the following may help general users connect to USB color sensors:

```
/etc/security/console.perms.d/60-libusb.perms

     <usbdevices>=/dev/bus/usb/*/*
     <console> 0600 <usbdevices> 0600 root
```

udev-based systems may require something like:

```
/etc/udev/rules.d/40-permissions.rules

     ...
     # USB serial converters
     SUBSYSTEM=="usb_device", GOTO="usb_serial_start"
     SUBSYSTEM=="usb", ENV{DEVTYPE}=="usb_device", \
                          GOTO="usb_serial_start"
     GOTO="usb_serial_end"
     LABEL="usb_serial_start"

     # X-Rite Gretag-Macbeth AG
     ATTRS{idVendor}=="0971", ATTRS{idProduct}=="2003", \
                          GROUP="xrplugdev"
     LABEL="usb_serial_end"
     ...
```

Reference: Known Issues

This is an ongoing development list:

1. g++ (ver 4.x or later) with RHEL5 complains about wxWidgets type-punned pointers (workaround: added `--with-wx-gcc4` flag to configuration script to enable -fno-strict-aliasing during compile)
2. ongoing code commenting and cleanup continues - in particular, debugging statements will be addressed in a future release
3. building 3rd party sensor libraries requires care on 64-bit systems (-m64 and other flags may be required) to operate correctly - highly recommend redefining `QMAKE_CFLAGS` with contents of `QMAKE_CFLAGS_RELEASE` to ensure most accurate system build with 3rd party sensor library plugins (see `/usr/lib64/qt-X.X/mkspecs/default/qmake.conf` for details)
4. build configuration may break with undefined `AM_OPTIONS_WXCONFIG` messages - this is because wxwin.m4 is not found (workaround: `ln -s /usr/local/share/aclocal/wxwin.m4 /usr/share/aclocal/wxwin.m4`)
5. *ucal* may fail to load improperly defined or configured plugin modules - either from ucal.xml or the plugins themselves - currently this can only be discovered by reading debugging output from command line
6. when first run, if the *ucal* menu is missing a particular action choice or choices, check to ensure that the absolute path to the sensor library plugins is correct (load failures manifest this behavior)
7. the current implementation only supports addressing *one* color sensor and *one* target monitor at this time - data structures anticipate broader expansion, but this is not implemented at this release
8. sample *rpm* spec file is still rough and in development - the same can be said for the sample build scripts that were provided with beta evaluation code
   - recommended evaluation tweaks to *rpm* spec file example include:
     - uncomment `AutoReqProv: no` (remove hash #)
     - at `./configure` line, add in desired pre-cooked sensor macros above `%{wc_gcc4_flag}` for previously build libraries, e.g., `%{chroma5_flags}` and/or `%{i1d2hpdc_flags}` (do not forget the backslash!)
     - at the OMCF Files section, uncomment the appropriate sensor library by removing the hash # (or add your own) as needed, e.g., `%{_ookaladir}/%{_lib}/libChroma5.*`
9. the *read_sensor* applet is configured to use the *ExampleSensor* module by default - if another one is desired, reapply the symbolic link to main.cpp in the apps source directory for read_sensor and copy the appropriate makefile.am for your desired build (refer to local README for more information).

References: Advance Topics

apps: ucal

The ucal is the primary application framework for color calibration using plugin modules and defined execution chains (parsed from a system location `/etc/xdg/ucal/ucal.xml` or user-defined `$HOME/.config/ucal/ucal.xml`). It is built upon *libookala*, the plugin registry and DDC/CI interface for all OMCF applications.

ucal.xml.sample

The demonstration configuration sample is designed for use with the HP DreamColor LP2480zx Professional Display. Besides the features mentioned in the Software Overview section, study of this commented file provides insights into the data-driven *DreamColor plugin* and tested parameters for calibrating the LP2480zx monitor, e.g., channel tolerances, monitor preset management, color space definitions, etc.

Basic layout of this file is:

```
<ucaldata>
...
  <loadplugin>..</loadplugin>        # define all plugins required for name resolution
  <loadplugin>..</loadplugin>
...
  <!-- define default present -->
...
<chain name="..">
  <dict name="..">                   # define dictionary item data for plugin
    <dictitem type=".." name="..">
    ...
  </dict>
  ...
  <plugin>..</plugin>                # define plugins to call for this chain when
requested
  <plugin>..</plugin>
  ...
</chain>

<chain name="..">
...
</chain>

</ucaldata>
```

When this configuration file is input and parsed, the *ucal* application loads all plugin modules, resolves references, builds data dictionary storage for plugin module execution, defines plugin chains, and presents the available actions in a GUI pull-down menu on the taskbar of a given desktop environment.

HP DreamColor Monitor Calibration Design

The *ucal* application in conjunction with the *plugins*, *ucal.xml* configuration setup, and user *color sensor package* currently works on the HP DreamColor LP2480zx Professional Display in the following manner:

1. select the correct monitor color-preset and turn on the internal monitor pattern generator [10-bit]
2. measure red, green, blue (without color processing) to establish the native display (raw) gamut
3. from the target, chromaticity, and luminance, attempt to determine the white point for the backlight
4. re-measure red, green, blue, plus white point to calculate the appropriate 3x3 matrix (uploaded to monitor later) targeting the preferred color space
5. measure the tone-curve of the monitor (used later for post-LUT transforms)
6. examine and program the backlight registers, converging their values with desired white point
7. upload the pre-LUT (user color space) and post-LUT (measured post response) to monitor
8. sample select color patches to determine if tolerances were achieved

apps: hpdc_util

The *hpdc_util* application provides an elementary command line interface to developing with and controlling the HP DreamColor LP2480zx monitor. Current functionality includes:

```
USAGE: hpdc_util <options> <plugin_path>
       -c N           Switch to color space N
       -q             Query current color space data
       -m             Query the current matrix
       -M "m00 ..."   Set the current matrix
       -p filename    Download pre-Lut to filename
       -P filename    Download post-Lut to filename
       -e             Enable color processing
       -E             Disable color processing
       -g "R G B"     Enable pattern generator with a color
       -G             Disable pattern generator
       -o             Unlock the OSD
       -O             Lock the OSD
       -b brightness  Set the backlight brightness register (8 bits)
       -d N           Copy the current color space to preset N
       -t             Test torino state
```

To test the correct functionality of this command, the following command queries the attached monitor and responds with a detailed dump of the current color space data active in the display:

```
hpdc_util -q /usr/local/ookala/lib
```

apps: read_sensor

The *read_sensor* sample demonstrates a simple, monolithic interface to libookala for loading plugin modules, defining necessary data dictionaries, and calling a requested function (such as requesting a color sensor sample). This works around the need to invoke the *ucal* framework and provides additional flexibility for development and debugging of plugin modules with the *libookala* library interface.

The current *read_sensor* example demonstrates loading the ExampleSensor plugin, defining its data dictionary as if controlling an actual color sensor device, and then requesting a measurement (which returns dummy values at the moment). The ExampleSensor also prints to *stdout* internal actions taken by the plugin and dictionary managers to further aid in learning the interface and its functions.

```
# read_sensor -n 1 /usr/local/ookala/lib
Loaded Ookala::ExampleSensor from /usr/local/ookala/lib/libExampleSensor.so
 actionTaken: detect and open sensor device...
 actionTaken: confirm state...
 DictOptions: setup base defaults...
 DictOptions: setup LCD parameters...
 DictOptions: Calibration Index set1
 measureXY: selecting calibration matrix...
 measureXY: setup LCD or CRT...
 measureXY: integration time applied...
 measureXY: reading measurement Yxy (candela)...
 measureXY: read (dummy) values Yxy (1.0, 0.31, 0.32) ...
 1.000000 0.312779 0.329183 (dummy sensor)
 disposing of open sensor
```

libookala

The libookala interface defines and manages plugins, plugin registries, chains (actions), a basic DDC/CI interface, library data types, and base classes for `Plugin`, `PluginRegistry`, `PluginChain`, `Dict`, `DictHash`, `CalibRecordDictItem`, `Sensor`, and `UI` plugins.

The `Sensor` base class, for example, defines an essential plugin interface for which developers must link their color sensor libraries and instruments to dynamically.

```
class EXIMPORT Sensor: public Plugin
{
  public:
    Sensor();
    Sensor(const Sensor &src);
    virtual ~Sensor();
    Sensor & operator=(const Sensor &src);
    virtual std::vector<uint32_t> sensors();
    virtual bool measureYxy(Yxy &value, PluginChain *chain, uint32_t sensorId);
    virtual std::vector<uint32_t> actionNeeded(PluginChain *chain, uint32_t sensorId);
    virtual bool actionTaken(uint32_t actionKey, PluginChain *chain, uint32_t sensorId);
  protected:
  private:
};
```

Additional documentation for these data structures is in development.

plugins: DreamColor

The primary plugin interface to the HP DreamColor LP2480zx including control, color space management, and calibration handling.

plugins: WsLut

A host-side (not display-side) plugin interface for handling window-system lookup tables (LUT) including reset and randomize.

plugins: DevI2c

A device I2c interface plugin for communication requirements. Requires the *i2c-dev* kernel module for proper device discovery and communication.

plugins: ExampleSensor

The *ExampleSensor* demonstrates a basic color sensor interface implementation. Methods implemented beyond the basics include *enumeration*, *measureYxy* (read and return sensor value), *actionNeeded* (queued event request such as sensor detection or dark reading calibration for LCD monitors), *actionTaken* (callback for requested action event), and a dictionary data structure get/set mechanism (depending upon plugin's data and state requirements).

plugins: WxBasicGui

plugins: WxDreamColorCalib

plugins: WxLutSweep

plugins: WxSensorPrep