PressCal Customization

Before you delve into the details below, there are some alternatives. First, PressCal has a command-line interface. This opens up possibilities for automation, and/or incorporation into existing software. Second, it is possible to make custom grading sets, either with a setting, or added to the grading database. Third, there is a statistical output capability for data analysis with R or Excel. Finally, the authors are available for hire, to perform customization work. If you have an idea that would be of general use, we'd like to hear about it. We intend to improve the software, and are open to suggestions.

General Description

PressCal is a Perl script and collection of supporting modules, supplied as a TextMate bundle. Perl is an interpreted language, which makes PressCal easy to customize, with some basic programming knowledge. The script is excessively commented. You are welcome to use it freely, under terms of the GPLv3 license.

Bundle Structure

The PressCal bundle is located in the user's **Application Support** folder. Install PressCal, if you haven't already. Using the Terminal app, open this folder with the following command (followed by **return**),

```
open ~/Library/Application\ Support/TextMate/Pristine\ Copy/Bundles/
PressCal.tmbundle/Support/lib/perl5
```

This folder contains the Perl script, **PressCal.plx**, and folders containing the supporting Perl modules. These are the components you're most likely to change. The modules we developed are in the `ICC` folder.

There are additional Perl modules located in the `darwin-thread-multi-2level` folder. These so-called **XS** modules are written in **C** language, compiled, and linked to the main Perl program. The XS modules we developed are located in the `ICC/Support` folder. The **lapack.pm** module provides an interface to the **BLAS** and **LAPACK** libraries. The **levmar.pm** module is an interface to the **levmar** library. The **Rmath.pm** module is an interface to the **Rmath** library. These are required modules, but it's unlikely you'll need or want to alter them.

Documentation

There is no separate documentation for the **PressCal.plx** script. Of course, the user manual explains the functioning of PressCal, and its settings. The script itself is excessively commented (much of the code is self-explanatory, but is commented anyway). Several of the ICC modules are properly documented.

Development using TextMate Editor

To open PressCal in TextMate from the Terminal,

```
open -a TextMate ~/Library/Application\ Support/TextMate/Pristine\ Copy/Bundles/
PressCal.tmbundle/Support/lib/perl5/PressCal.plx
```

If you plan to edit PressCal, save it to a convenient location (not in the PressCal bundle). This will ensure you don't accidentally overwrite your edited version by updating or reinstalling the TextMate bundle. It also ensures that PressCal will continue to function normally, when run from a settings (YAML) file.

Next, open TextMate's preferences and click on the "Bundles" icon. Scroll down to the Perl bundle, and check that box. TextMate will download and install the Perl bundle. This adds Perl-specific text coloring and commands, including the ability to run the script using the ⌘**R** key combination.

Test this by entering ⌘**R**. This should result in an immediate error, beginning with "`Can't locate ICC/Profile.pm in @INC`". The script was saved to a new location, and Perl cannot find the modules it needs.

This is easily fixed by adding this (single) line to the saved PressCal version,

```
use lib "$ENV{'HOME'}/Library/Application Support/TextMate/Pristine Copy/
Bundles/PressCal.tmbundle/Support/lib/perl5";
```

This line should be located between the existing program lines `=cut` and `use ICC::Profile;`.

```
see the user manual for a complete explanation of the OPTIMAL method, and the software settings.

=cut

use lib "$ENV{'HOME'}/Library/Application Support/TextMate/Pristine
Copy/Bundles/PressCal.tmbundle/Support/lib/perl5";

use ICC::Profile;  # color toolkit modules
use ICC::Support::Rmath;  # R statistical library module
use Data::Alias;  # Alias module
```

Version 16.3U of the **PressCal.plx** script contains 107 functions in 10,905 lines of code. The TextMate editor has a "folding" capability that makes it feasible to edit such a large document. When you open **PressCal.plx** for the first time, the code should be completely folded. If line numbers are enabled (opt-⌘**L** key), you should see something like this as you scroll down,

```
# main program
# return: (status)
sub main {...}

# optimize ink balance
# parameters: (solid_sample_slice, reference_abs_Lab_values, reference_rel_Lab_values,
  press_abs_Lab_values, ref_to_chart_object, A2B1_object, B2A1_object, PCS_object, ref_to_settings_hash)
# return: (status)
sub ink_balance {...}

# grade press measurements
# parameters: (solid_sample_slice, reference_abs_Lab_values, reference_rel_Lab_values,
  press_abs_Lab_values, ref_to_chart_object, A2B1_object, B2A1_object, PCS_object, ref_to_settings_hash)
# returns: (status)
sub grade {...}
```

The little triangles in the left-hand margin control folding. If you click on a triangle, that region will unfold (open up). If you click the triangle again, it will fold (close). You can fold or unfold all contained triangles with opt-click. You can fold or unfold all triangles in the document with the opt-⌘**O** key. Because the script is so large, it is best to keep everything folded, except for the function(s) you're working with.

Note that, when editing in TextMate, PressCal is using the built-in settings, which are displayed in green, near the start of the script. Add the following lines to the built-in settings,

```
# set command (0 – curves, 1 – ink balance, 2 – grade)
command: 0
# set debug mode (0 – normal, 1 – debug)
debug: 1
```

The **command:** setting selects the PressCal tool, and the **debug:** setting enables full display of errors and warnings. Test this with the ⌘**R** key combination. Now, you're ready to program.

Program Overview

When you run PressCal, the main function is executed. The **main** function is the curve building tool. The ink balance and grade tools were added later. The **ink_balance** and **grade** functions were identical to the **main** function, up to a certain point. So, we used Perl's **goto** function to factor the common code. Near line 360, the program may branch to the **ink_balance** or **grade** functions, depending on the **command:** setting value.

Settings

PressCal has over 40 defined settings, which are usually supplied as a YAML file. The **load_settings** function is called near the start of the **main** function. This converts the YAML file to a Perl hash. Comments are removed, and the remaining text is matched with a regular expression, to extract key and value, line by line. The hash values are further processed using Perl's **eval** function. You may add your own settings, providing you don't use existing keys.

A setting is accessed by its key. For example,

```
$path = $set->{'profile_path'};
```

You may add an element to the hash,

```
$set->{'my_key'} = $my_value;
```

The value may be a scalar, an array reference, a hash reference, or a function reference.

Reference Profile

The reference profile provides the color target. The profile is opened as an **ICC::Profile** object. It is normally a printer profile. We use the A2B1 tag (device to PCS) for optimization, and the B2A1 tag (PCS to device) to select samples. These tags are also opened as objects, and have a **transform** method,

```
$Lab = $A2B1->transform($device);

$dev = $B2A1->transform($Lab);
```

If the reference profile is an RGB working space (display profile), equivalent A2B1 and B2A1 objects are created.

The A2B1 and B2A1 tags contain media-relative data. The media white point is stored in a separate profile tag, and is copied to the $wtpt variable.

Press Measurements

Press measurements are saved from other software, such as X-Rite's i1Profiler. There are two standard file formats used for measurements – ASCII and CxF3 (XML). PressCal will open either format into an **ICC::Support::Chart** object. Color measurements may be spectral, L*a*b*, or XYZ. Spectral measurements are converted to colorimetric using the **`color:`** setting. The default colorimetry is D50/2 degrees. Measurements are accessed using the **device**, **spectral**, **lab**, and **xyz** methods,

```
$Lab = $press->lab($samples);

$dev = $press->device($samples);
```

The argument for these methods is a list of samples ($samples). The list is an array reference,

```
$samples = [1, 2, 3, 4, 5]; # samples 1, 2, 3, 4, and 5

$samples = [1 .. 5]; # samples 1 thru 5
```

Note the sample index is base 1 – there is no sample 0.

There is an optional setting for plate curves, to be used when the test plates aren't linear. If this setting is defined, plate curves are loaded, and applied to the device data used at various points.

Ink Map

The ink map is an array with an element for each ink channel. The ink map describes how the tone curve for each ink channel will be generated. If the element is an integer, that channel will be used in the optimiza-

tion. In addition, the element maps the ink channel to the device channel of the reference profile,

```
$ink_map = [0, 1, 2, 3]; # normal CMYK mapping

$ink_map = [3, 0, 1, 2]; # press data is KCMY, K maps to device channel 3, C
maps to device channel 0, etc.
```

If the element is not an integer, that ink channel will not be used in the optimization. Instead, it's tone curve will be computed with an alternate method, A, B, C, D, E, or F – TVI, S – SCTV, N – G7 Black, L - Linear,

```
$ink_map = [A, A, A, B]; # CMY computed using TVI curve A, K computed using TVI
curve B, as defined in ISO 12647-2

$ink_map = [0, 1, 2, 3, S, S]; # normal CMYK mapping, 5th and 6th ink channels
computed using SCTV

$ink_map = [0, 1, 2, N]; # CMY computed using optimization, K computed using G7
black NPDC
```

Sample Selection

The **`select:`** setting is a string of tokens used to select samples for optimization. This string is supplied to the **select_token** method of the press object. Samples containing non-optimized inks are filtered out.

The **`ced_select:`** setting is a string of tokens used to select samples for the CED graphs. By default, this is the same as the optimization selection. The selection is influenced by the ink map, so the **`ced_ink_map:`** setting is also provided.

Sample sets for the alternate curve methods, A, B, C, D, E, or F – TVI, S – SCTV, N – G7 Black, are located by the **ink_ramp** function.

The sample selection token **rt**() selects samples based on a round-trip through the reference profile,

```
$round_trip = $B2A1->transform($A2B1->transform($device));
```

If the round-trip black %-dot value changes less than the **rt**() parameter, that sample is included. These samples are realistic, in that they're close to colors appearing in images made with the reference profile. This selection may be viewed as *filtering out* samples that are unlikely to appear in images, e.g. darker gray CMY only samples.

Optimization

The Levenberg-Marquardt optimization technique is used to compute curves. PressCal uses open-source **levmar** library. Levmar is written in **C**, and is accessed using the **ICC::Support::Levmar** XS module, which loads and interfaces a compiled version of levmar.

Levmar provides a variety of functions, e.g. single or double precision, analytical or numerical Jacobian, constrained or unconstrained. We use the double precision versions, since Perl's internal representation of floating point numbers is double precision. We use the numerical Jacobian versions, because an analytic Jacobian is not feasible with $\mathbf{\Delta E_{00}}$ color errors. We use inequality constraints to keep the curves monotonic, and linear equation constraints when curve endpoints are fixed. These constraints are set by the **add_li_constraints** and **add_le_constraints** functions.

Levmar requires a function to evaluate the color errors. This function (**func_opt_curves**) is called repeatedly until the color error is minimized. This often requires hundreds of levmar iterations. Most of the function calls are made to compute the numerical Jacobian.

The curves are Bernstein polynomials, and levmar optimizes the Bernstein coefficients. Each curve is implemented as an **ICC::Support::bern** object. The object coefficients are provided to levmar as aliases, using the

Data::Alias module. The degree of the Bernstein polynomials is determined from the sample data by the function **make_cvst_auto**. The **degree:** setting, if any, overrides the determined value.

The alternate curve methods, selected in the ink map by A, B, C, D, E, F, S, or N, also use levmar and Bernstein polynomials. These curves are computed individually by the **make_TVI_curve**, **make_SCTV_curve**, and **make_G7K_curve** functions. The optimization targets for these functions are formulas, not the reference profile.

Curve Output

The **ICC::Support::bern** objects are contained in an **ICC::Profile::cvst** object (curve set). After the curves are optimized, they're output in various formats, according to the **output:** setting. This setting is a string of tokens, with optional parameters, for different curve formats. The formats are actually **ICC::Profile::cvst** methods. To add additional curve format(s), just add new method(s) to the **ICC::Profile::cvst** module. The existing output methods may be used as a guide.

Keep in mind that output involves computing discrete points using the **_transform** method of the **ICC::Support::bern** objects. This method has a direction parameter – 0 for forward, 1 for inverse. PressCal curves, as computed, are actually inverse functions, so you'll output them with this parameter set to 1.

Curve Graphs

The solid ink colors are graphed as an a*/b* plot. Then, the individual and composite color curves are plotted. Finally, the cumulative error distribution, before and after optimization, is plotted with fitted gamma curves.

The graphs are HTML files using the **RGraph** JavaScript library. These files are created using the Perl **Template** module, which merges a static template with variable data. The HTML files are created at different points in the **main** function, and their paths saved until just before the program exits. They are then opened as a group using the default web browser.

Ink Balance

As mentioned earlier, the ink balance tool is a branch of the **main** function. The settings, reference profile, and press measurements are loaded. The solid ink colors and errors are computed and printed. At about line 360, a **goto** command branches to the **ink_balance** function.

The **ink_balance** function creates a spectral model of ink layers of varying thickness, based on the current solid ink measurements. Then, the ink layer thicknesses are optimized to minimize the weighted color errors, relative to the reference profile. With spectral data, we can compute the ink densities before and after optimization, which provide the pressman with relative density adjustments.

In addition to computing density adjustments, the trapping values for the spectral model are displayed for each of the overprint colors. Ideally, all of these values would be 1, indicating perfect trapping. Normally, trapping values will be less than one. The spectral trapping model needs to know the ink sequence, entered with the **ink_sequence:** setting. The **ink_weight:** setting sets the weight for each of the color errors during optimization. By default, the CMY and RGB errors all have the same weight. With this setting, the CMY overprint has a weight or 0. If you plan to make G7 curves, you may want to change this value, to pull the CMY overprint towards neutral. This will increase the error of the other solids, but may be a good trade off with G7 curves.

Ink Balance Graphs

The solid ink colors are graphed as an a*/b* plot, before and after optimization. The colored shapes, either circles or ellipses, are computed from the target color, and the ΔE metric. The black dots are the actual measurements. If you touch them with the cursor, the L*a*b* values are displayed, along with the color error.

The graphs are HTML files using the **RGraph** JavaScript library. These files are created using the Perl **Template** module, which merges a static template with variable data. The HTML files are created at different

points in the **ink_balance** function, and their paths saved until just before the program exits. They are then opened as a group using the default web browser.

Grading

As mentioned earlier, the grading tool is a branch of the **main** function. The settings, reference profile, and press measurements are loaded. The solid ink colors and errors are computed and printed. At about line 360, a **goto** command branches to the **grade** function.

The **grade** function uses a database created by a separate script, **PressCal_grade_database.plx**, and saved as a **Storable** file. The database is actually just a Perl hash, using compound keys to obtain a list of grading functions and parameters, which comprise the selected grading rules. For example,

```
$grade = $db->{'g7'}{'36'}{'colorspace_proof'};
```

This returns the node for G7 pass/fail version 36, colorspace proof, as a hash reference. The hash keys correspond to function names that perform tests, e.g. **'substrate'**, **'cmyk_solids'**, **'rgb_solids'**, **'cmy_ramp'**, **'k_ramp'**, **'all_samples'**. The hash values are an array of parameters for that test function, typically the error types and limits. These functions and parameters are explained in the **Custom Grading** section of the user manual.

Each of the test functions must locate the specified samples, evaluate the color errors, and determine if the test is passed or failed. The number of tests, and the number of tests failed, is tallied and displayed at the end.

It is possible to make a custom grading set using the **`grade:`** setting. This is a hash reference, like the grade node described above. You could also add custom function(s). If you want to alter the grade database, you'll need the **PressCal_grade_database.plx** script, available from the authors by request.

Grading Graphs

The solid ink colors are graphed as an a*/b* plot. Then, the individual color ramps are plotted, either as SCTV or TVI values. These plots are meant to give a visual indication of the process linearity. If applicable ramps of CMY gray, CMYK gray, and black inks are plotted as L*a*b* errors. Finally, the cumulative error distribution is plotted with a fitted gamma curve.

The graphs are HTML files using the **RGraph** JavaScript library. These files are created using the Perl **Template** module, which merges a static template with variable data. The HTML files are created at different points in the **grade** function, and their paths saved until just before the program exits. They are then opened as a group using the default web browser.

Program Data

PressCal is distributed with a large assortment of data files. These are located in the **ICC** folder,

```
open ~/Library/Application\ Support/TextMate/Pristine\ Copy/Bundles/
PressCal.tmbundle/Support/lib/perl5/ICC
```

The **Data** folder contains several YAML files used by the **ICC::Support::Color** module. There is no reason to alter these files. The **Grade** folder contains the grading database, **grade_db.sto**, and several key collections used by the grade functions. The **Test** folder contains the built-in test data – measurements and ICC profiles.

Returning to the **ICC** folder, the **JavaScripts** folder contains files used to create the HTML graphs, notably the **RGraph** library. Several of the RGraph files have been slightly modified. The **Templates** folder contains XML and Template Toolkit files, used by the **ICC::Profile::cvst** module to output curves and by PressCal to make the HTML graphs.

PressCal assumes the **Data**, **JavaScripts**, and **Templates** folders are located within the **ICC** folder. If you move them, program modifications will be required.

Porting to Another OS

The distributed version of PressCal uses TextMate as the user interface. TextMate is Mac-only software, so an alternative user interface would be required for another OS. Likely candidates are Electron or Visual Studio Code, both of which utilize Node.js.

PressCal also requires Perl, which is built into many, but not all, OS. On Windows, we like the Strawberry Perl distribution. PressCal was developed using Perl v5.18.4, which is built into MacOS 10.10 through 10.15. Another version using Perl v5.30.3 exists for compatibilty with macOS 11 and newer. This version contains Universal Perl modules compiled for Intel and ARM processors. The PressCal.plx code is very portable, and should run without modification on any version of Perl v10.0 or later.

PressCal requires a list of Perl modules, some of which are XS modules that require compiling. On a Mac, you will need to install Command Line Tools. Most of the required modules are built-in or can be installed automatically using CPAN. Here is a list of required CPAN modules,

```
ICC::Profile
Math::Matrix
YAML::Tiny
Data::Alias
List::Util
Scalar::Util
Sub::Util
Template
```

Three XS modules are only available from the authors by request,

```
ICC::Support::Lapack
ICC::Support::Levmar
ICC::Support::Rmath
```

These are interfaces to the math libraries used by PressCal. MacOS has built-in versions of BLAS and LAPACK (Accelerate framework). You may need to install these libraries on other OS. The levmar software does not need to be installed. But it links to the LAPACK library. The Rmath library must also be installed. These modules include build scripts for macOS, which will require some modification for other OS.