The `parse_icc_tex` module – A TEX interface for luaicc

Marcel F. Krüger

October 23, 2021

The `parse_icc_tex` Lua module defines a plain TEX like interface consisting of three commands:

`\LoadProfile` `\LoadProfile` ⟨*csname*⟩ `{`⟨*filename*⟩`}`

⟨*csname*⟩ will be defined to represent the profile loaded from `{`⟨*filename*⟩`}` in other commands. If ⟨*csname*⟩ is already defined, it will be overwritten.

The defined control sequence ⟨*csname*⟩ can not be used on it's own but only in other commands from luaicc.

A simple example would be

```
\LoadProfile \sRGB {sRGB.icc}
```

`\ProfileInfo` ⋆ `\ProfileInfo components` ⟨*profile*⟩
`\ProfileInfo class` ⟨*profile*⟩

The profile provided in ⟨*profile*⟩ must be a control sequence defined with `\LoadProfile`.

When called with the `components` option, the function expands to the number of components in it's color space. (E.g. 3 for RGB spaces, 4 for CMYK spaces, etc.) The number will always be between 1 and 15.

When called with the `class` option, the expansion is the 4 character tag representing the profile class. The seven options are `scnr` for input device profiles, `mntr` for display device profiles, `prtr` for output device profiles, `spac` for color space profiles, `link` for device link profiles, `abst` for abstract profiles and `nmcl` for named color profiles.

The number of components for the `\sRGB` profile loaded by the previous example could e.g. be queried with

```
\ProfileInfo components\sRGB
```

\ApplyProfile `\ProfileInfo [delim⟨delimiter⟩] [gamut⟨out-of-gamut tag⟩] [⟨rendering intent⟩] [⟨interpolation space⟩ [inverse]] ⟨target profile⟩ ⟨n⟩ ⟨source profile$_1$⟩ ⟨source color$_1$⟩ ⟨weight$_1$⟩ ... ⟨source profile$_{n-1}$⟩ ⟨source color$_{n-1}$⟩ ⟨weight$_{n-1}$⟩ ⟨source profile$_n$⟩ ⟨source color$_n$⟩`

(The [] here do indicate optional arguments and not literal `[` and `]` to be written in the source code.)

To actually convert colors between profiles, `\ApplyProfile` is used. Beside just converting, it also allows interpolating between colors in different colorspaces.

It expands to the components of the color separated by ⟨*delimiter*⟩ (which must be a single token and defaults to spaces if it is not provided) in the profile given by ⟨*target profile*⟩. If `gamut` is given, the token provided as ⟨*out-of-gamut tag*⟩ is prepended if the color is outside of the gamut of ⟨*target profile*⟩.

The used rendering intent is given by ⟨*rendering intent*⟩. It must be one of `perceptual`, `colorimetric`, or `saturation`. If it is not provided, the default is unspecified and might change in later versions.

The number of source colors is given in ⟨*n*⟩. It must be a positive integer. If it is not 1, the source colors and interpolated based on the intergers given as ⟨*weight$_i$*⟩. The last weight is not explicitly provided but automatically determined such that the sum of all weights is 1000.

The colorspace the interpolation i done is is selected by the ⟨*interpolation space*⟩ option. The options are `lab` for CIELab, `xyz` for CIEXYZ, `xyy` for xyY, `luv` for CIELUV and the cylindrical options `lch` for CIELCh and `lchuv` for CIELCh(uv). The default in unspecified and subject to change.

The two cylindrical spaces can be followed by the `inverse` keyword to interpolate along the longer instead of the shorter path for the hue component.

The ⟨*source color$_i$*⟩'s are given by space separated components corresponding to the given ⟨*source profile$_i$*⟩.

An example for simple color conversion from the cyan primary in a CMYK space `\myCMYK` to a RGB space `\sRGB` with result components separated by commas would be

```
\ApplyProfile delim, \sRGB 1 \myCMYK 1 0 0 0
```

To mix 10% of `\sRGB`'s green with 30% of `\myCMYK`'s yellow and 60% of `\sRGB`'s red and return the (space separated) result in `\myCMYK`, while doing all calculations in CIELUV, the invocation is

```
\ApplyProfile luv \myCMYK 3
\sRGB     0 1 0   100
  \myCMYK 0 0 1 0 300
  \sRGB   1 0 0
```