PERFECTLY CLEAR API REFERENCE

Version 7.4.1.0

For photos as vibrant as life itself

Background and Overview 5

- Introduction 5
- Reduced Resolution Images in Calc 5
- Color Management 5
- Red Eye Correction 6
- Usage 6
- Minimum Version 6
- Change Log 7

API Function Reference: 9

- PFC_CreateEngine 9
- PFC_DestroyEngine 9
- PFC_SetParam 9
- PFC_Calc 10
- PFC_Apply 12
- PFC_ReleaseProfile 13
- PFC_AutoCorrect 13
- PFC_AutoCorrectPreset 14
- PFC_ApplyLocal 15
- PFC_HasFaceBeautification 16
- PFC_FBFaceCount 17
- PFC_GetFaceInfo 17
- PFC_FAEFaceCount 18
- PFC_EnumFAEFaceRect 19
- PFC_AbnormalTintDetected 19
- PFC_IsNoiseDetected 20

Structures 21

- PFCIMAGE 21
- PFCENGINE 22
- PFCCOREPARAM 22
- PFCFBPARAM 25
- PFCNRPARAM 26
- PFCREPARAM 27
- PFCPOINT 27
- PFCRECT 27
- PFCFBFACEINFO 27

Enums 28

- PFCPIXELFORMAT 28
- PFCENGINESTATUS 28
- PFCFEATURE 28
- PFCPRESETID 28
- TINTCORRECTION 29
- CONTRASTMODE 29
- DCFMODE 29
- AGGRESSIVENESS 29
- BIASMODE 29
- SKINMODE 29
- SKINSMOOTHTYPE 30
- SKINTONINGTYPE 30
- LIPSHARPENTYPE 30
- PFCAPPLYSTATUS 30
- PFCNR_STATUS 30
- PFCCORE_STATUS 31
- PFCFB_STATUS 31
- PFCRE_STATUS 31

Usage& Examples 32

- Scenerio #1 - Using PCF_AutoCorrect 32
- Scenerio #2- Separate PFC_Calc and PFC_Apply 34
- Scenario #3 - Interactive Corrections 36

Usage (OSX / Linux) 40

- Scenerio #1 - Using PCF_AutoCorrect 40
- Scenerio #2- Separate PFC_Calc and PFC_Apply 42

Sample Projects (Windows version) 47

- Sample 1 47
- Sample 2 47
- Sample3 47

Sample Projects (OSX / Linux) (Full Source) 49

- Sample 1 49
- Sample 2 52

Mapping parameters from version 6 CORE API 58
Mapping parameters from version 6 PRO API 59

Background and Overview

Introduction

This document will explain how to use the Perfectly Clear API to enable fully-automatic or interactive image corrections and analysis in your applications.

The API consists to four major components:

1. Perfectly Clear Core Corrections
2. Noise Corrections
3. Red-eye removal
4. “Beautify” Corrections

The first three are provided to all licensees and are delivered in the PerfectlyClearPro.dll, while the “Beautify” corrections are optional and are included in the Face Beautify enabled version of PerfectlyClearPro.dll. The Beautify features will not be available in non Face Beautify version – but no coding changes are needed to enable these corrections at a later date.

All four correction components include a *Calc* phase, where image analysis is performed, but no corrections are made. Corrections are made in a separate *Apply* phase. The *Calc* phase must be run once per image - and does not change based on image correction parameters, allowing you to speed up the correction results for use in interactive applications where users can alter the correction parameters and getting a preview image back to the customer is the primary goal. In fully automatic correction applications, this isn’t needed, so a single *AutoCorrect* function call will perform both *Calc* and *Apply*.

Reduced Resolution Images in Calc

The PFC_Calc function accepts two images as the first two arguments; a full-sized image and a reduced resolution image. The full resolution image is required to run Calc for Noise Removal and Beautify; omitting this parameter will disable Noise and Beautify corrections.

Perfectly Clear Core and Red-eye corrections can run on a reduced resolution image, speeding the Calc processing time without compromising quality. The image must be no smaller than 1024 px on the longer edge, and ideally should be the larger of one-third of the original image or 1024 px (one the longer edge). If a reduced resolution image is not passed into the Calc function, the PFC library will create this for you, using the down-scaling mentioned above.

Color Management

The Perfectly Clear Core corrections assume that the image data is in the sRGBcolor space. For best image quality, be certain to convert any images in other spaces to sRGB before processing with this API. Images in Adobe RGB or ProPhoto or other wide-gamut color spaces will appear overly red and overly dark once corrected with Perfectly Clear.

Red Eye Correction

Our red-eye correction technology leads the industry in its speed and accuracy. It automatically detects eyes, determines if “red-eye” is present, and applied a very natural image correction to remove this unsightly camera artifact. As a fully-automatic correction, it can mis-identify red-eyes occasionally. One method to lower the frequency of this is for you to only enable Red-Eye corrections on images where a flash was used – as determined by the EXIF Flash tag. As the libraries provided here only have access to the image content – not the file or EXIF data, you will need to implement this validation yourself.

Usage

You will find functional example code with this API that shows three different manners to use these libraries. The first is the most simple – load and image in memory and auto-correct it in a single function call. The second example shows splitting apart the AutoCorrectinto separate preCalc and Apply calls, and the third example shows the usage for an interactive application where the correction parameters are altered by the user.

Minimum Version

The minimum Mac OSX version is 10.6 (Snow Leopard).

Change Log

Version 7.1

Added support for new SFB parameters:

bSkinToning	BOOL	Set to TRUE to enable skin toning.
iSkinToning	int	Skin Toning level. (0 - 100)
eSkinToningMode	SKINMODE	Use SKINMODE_FACE to apply correction ONLY on skin regions included in faces. Use SKINMODE_BODY to apply correction on most skin regions regardless they are linked with a face or not.
eSkinToningType	SKINTONINGTYPE	See enum definition of SKINTONINGTYPE. P.24
bLipSharpen	BOOL	Set to TRUE to enable lip sharpening.
iLipSharpen	int	Lip sharpening level. (0 - 100)
eLipSharpenType	LIPSHARPENTYPE	See definition of LIPSHARPENTYPE. P.24
bBlush	BOOL	Set to TRUE to add blush.
iBlush	int	Blush level. (0 - 100)

Version 7.2

Added support for Face Aware Exposure.

bUseFAE	BOOL	Set to TRUE (recommended) to enable Face Aware Exposure selection. Recommended exposure will be calculated in favor of any human face detected from the image.

New utility functions:

PFC_FAEFaceCount

Return number of faces detected during the Face Aware Exposure analysis.

PFC_EnumFAEFaceRect

Enumerate faces detected during the FAE analysis.

PFC_IsNoiseDetected

Return true if noise detected in noise analysis.

Version 7.3

Added support for format

PFC_PixelFormat32bppARGB
PFC_PixelFormat48bppBGR
PFC_PixelFormat64bppABGR

Added support for opacity in Apply function.

Added preset "Intelligent Auto" support to SetParam().

Added new parameter bRejectMonolith in PFC_Calc().

Added new core error code PFC_CORE_BELOWMINSIZE when image side dimension smaller than 32 pixel.

Added support for variable DCF and Light Diffusion.

fDCF	float	Level of DCF. 0.0 (none applied) to 1.0 (full)
fLightDiffusion	float	Level of light diffusion correction. 0.0 (none applied) to 1.0 (full)

Two parameters changed in name:

bSkinToneis now: bInfrared

fSkinTone is now: fInfrared

Version 7.4

Replace vibrancy processing with a more reliable, gentle correction when value is at zero.

Fix problem with high contrast artifact.

API Function Reference:

PFC_CreateEngine

Syntax:

```
PFCENGINE* PFC_CreateEngine();
```

Return:

Pointer to PFCENGINE structure.

Description:

Create core engine for use in Perfectly Clear image processing pipeline. The PFCENGINE module created by this function can be used in the entire session before PFC_DestroyEngine() is called.

PFC_DestroyEngine

Syntax:

```
voidPFC_DestroyEngine(PFCENGINE *pEngine);
```

Parameters:

[in]	pEngine	Pointer to PFCENGINE structure to be released.

Description:

Release PFCENGINE module created by function PFC_CreateEngine() and release all resouces used by the PFCENGINE.

PFC_SetParam

Syntax:

```
voidPFC_SetParam(PFCPARAM *pParam, PFCPRESETID id);
```

Parameters:

[in/out]	pParam	Pointer to PFCPARAM structure to receive the parameters.
[in]	id	Preset id PFCPRESETID.

Description:

SetParam initializes a PFCPARAM structure with parameters pertaining to Athentech preset as identified by the PFCPRESETID. If preset id is not provided, the default preset values will be used.

PRESETID	Athentech Preset
PRESET_BEAUTIFY	Beautify
PRESET_BEAUTIFYPLUS	Beautify Plus
PRESET_DETAILS	Details
PRESET_VIVID	Vivid
PRESET_INTELLIGENTAUTO	Intelligent Auto

PFC_Calc

Syntax:

```
PFCPROFILE PFC_Calc(PFCIMAGE *pImage, PFCIMAGE *pImageds, PFCENGINE*
pEngine, unsignedint feature, intiISO=-1, char* pCameraModel=NULL,
PFCPROFILE pImageProfile=NULL, PFC_PROGRESS progfn=NULL, BOOL
bRejectMonolith=TRUE);
```

Return:

PFCPROFILE pointer to a profile which contains the analysis results for different features.

Query the Status member of the structure for high level return code.

<table>
<tr><td>0</td><td>Success.</td></tr>
<tr><td>> 0</td><td>The four least significant bits indicates which feature has not finished successfully.
<table>
<tr><td>Bit 0</td><td>Problem with Noise Removal.</td></tr>
<tr><td>Bit 1</td><td>Problem with Core analysis.</td></tr>
<tr><td>Bit 2</td><td>Problem with Face Beautification analysis.</td></tr>
<tr><td>Bit 3</td><td>Problem with Red Eye analysis.</td></tr>
</table>
</td></tr>
</table>

Query individual status from the PFCPROFILE structure.

NR_Status	PFCNR_STATUS enum.
CORE_Status	PFCCORE_STATUS enum.
FB_Status	PFCFB_STATUS enum.
RE_Status	PFCRE_STATUS enum.

Parameters:

<table>
<tr><td>[in]</td><td>pImage</td><td>Pointer to a PFCIMAGE structure that defines the image to be processed.</td></tr>
<tr><td>[in]</td><td>pImageds</td><td>Pointer to a PFCIMAGE structure that defines a supplementary down sampled image (approx. 1024 longest dimension) to aid in red eye detection. Set this parameter to NULL if you don't supply this image.</td></tr>
<tr><td>[in]</td><td>pEngine</td><td>Pointer returned from the PFC_CreateEngine() function.</td></tr>
<tr><td>[in]</td><td>feature</td><td>Specify the type of calculations. Possible values are:

<table><tr><td>CALC_CORE</td><td>Calculates for Perfectly Clear Core correction.</td></tr><tr><td>CALC_NR</td><td>Calculates for Perfectly Clear Noise Removal.</td></tr><tr><td>CALC_FB</td><td>Calculates for Face Beautification.</td></tr><tr><td>CALC_RE</td><td>Calculates for Red Eye Removal.</td></tr><tr><td>CALC_ALL</td><td>Calculates for all of the above.</td></tr></table></td></tr>
<tr><td>[in]</td><td>iISO</td><td>ISO value when the image is taken. Use -1 if not known.</td></tr>
<tr><td>[in]</td><td>pCameraModel</td><td>Text string of camera model which the picture is taken with. Set to NULL if not known.</td></tr>
<tr><td>[in]</td><td>pImageProfile</td><td>Pointer to PFCPROFILE. Leave as NULL if this is first time calling the PFC_Calc function.</td></tr>
<tr><td>[in]</td><td>progfn</td><td>Pointer to progress monitoring function. See definition of PFC_PROGRESS.</td></tr>
<tr><td>[in]</td><td>bRejectMonolith</td><td>Set to TRUE to enable rejection of simple graphics on which Perfectly Clear processing might not be desirable.</td></tr>
</table>

Description:

Performs initial calculation of image specific profile parameters.

Profile calculation can be done incrementally. For example if calculation is done initially on Perfectly Clear Core. Calculation of other feature, say Face Beautification, can be done in subsequent calls to PFC_Calc().

```
// Initial profile calculation only includes Perfectly Clear Core
PFCPROFILE *pProfile = PFC_Calc(&im, &imds, pEngine, CALC_PFC);
// Add calculation of Face Beautification to existing profile
pProfile = PFC_Calc(&im, &imds, pEngine, CALC_FB, -1, NULL, pProfile);
```

PFC_Apply

Syntax:

```
intPFC_Apply(PFCIMAGE *pImage, PFCENGINE *pEngine, PFCPROFILE
pImageProfile, PFCPARAM param, PFC_PROGRESS progfn=NULL, int
iOpacity=100);
```

Return:

<table>
<tr><td>0</td><td>The correction is successful.</td></tr>
<tr><td>> 0</td><td>Use macros to map out return code for each feature:
<table>
<tr><td>NRRETCODE</td><td>Returns PFCNR_STATUS Noise Removal correction status.</td></tr>
<tr><td>CORERETCODE</td><td>Returns PFCCORE_STATUS Core correction status.</td></tr>
<tr><td>FBRETCODE</td><td>Returns PFCFB_STATUS Face Beautification correction status.</td></tr>
<tr><td>RERETCODE</td><td>Returns PFCRE_STATUS Red Eye correction status.</td></tr>
</table></td></tr>
<tr><td>< 0</td><td>PFCAPPLYSTATUS enum.</td></tr>
</table>

Parameters:

[in/out]	pImage	Pointer to a PFCIMAGE structure that defines the image to be processed.
[in]	pEngine	Pointer returned from PFC_CreateEngine() function.
[in]	pImageProfile	Pointer to PFCPROFILE structure which is returned from the PFC_Calc() function.
[in]	param	PFCPARAM structure that carries the process parameters.
[in]	progfn	Pointer to progress monitoring function. See definition of PFC_PROGRESS.
[in]	iOpacity	Control opacity of NR and CORE corrections. Range from 0 (non applied) to 100 (fully applied).

Description:

Perform correction to image as defined by the PFCIMAGE structure using user parameters as defined in a PFCPARAM structure. Correction requires profile as calculated by the PFC_Calc() function.

Example:

```
PFCAPPLYSTATUS status = PFC_Apply(&im, m_pEngine, m_pProfile, m_Param);

if (status > 0)
    {
       int code;
       // Check return code for Noise Removal
       code = NRRETCODE(status);
       TRACE("Noise removal status  %d\n", code);

       // Check return code for Perfectly Clear Core correction
       code = CORERETCODE(status);
       TRACE("Perfectly Clear correction status  %d\n", code);

       // Check return code for Face Beautification
       code = FBRETCODE(status);
       TRACE("Face beautification status  %d\n", code);

       // Check return code for Red Eye Removal
       code = RERETCODE(status);
       TRACE("Red eye removal status  %d\n", code);
    }
```

PFC_ReleaseProfile

Syntax:

```
voidPFC_ReleaseProfile(PFCPROFILE pProfile);
```

Parameters:

[in]	pProfile	Pointer to PFCPROFILE structure to be released.

Description:

Release PFCPROFILE instance and its associated resources.

PFC_AutoCorrect

Syntax:

```
PFCAPPLYSTATUSPFC_AutoCorrect (PFCIMAGE *pImage, PFCIMAGE *pImageds,
PFCPARAM &param, intiISO=-1, char *pCameraModel=NULL, PFC_PROGRESS
progfn=NULL);
```

Return:

<table>
<tr><td>0</td><td>The correction is successful.</td></tr>
<tr><td>> 0</td><td>Use macros to map out return code for each feature:<table>
<tr><td>NRRETCODE</td><td>Returns PFCNR_STATUS Noise Removal correction status.</td></tr>
<tr><td>CORERETCODE</td><td>Returns PFCCORE_STATUS Core correction status.</td></tr>
<tr><td>FBRETCODE</td><td>Returns PFCFB_STATUS Face Beautification correction status.</td></tr>
<tr><td>RERETCODE</td><td>Returns PFCRE_STATUS Red Eye correction status.</td></tr>
</table></td></tr>
<tr><td>< 0</td><td>PFCAPPLYSTATUS enum.</td></tr>
</table>

Parameters:

[in]	pImage	Pointer to a PFCIMAGE structure that defines the image to be processed.
[in]	pImageds	Pointer to a PFCIMAGE structure that defines a supplementary down sampled image (1024 x 768) to aid in red eye detection. This parameter can be NULL. However the use of such supplementary image is highly recommended.
[in]	param	PFCPARAM structure that carries the process parameters.
[in]	iISO	ISO value when the image is taken. Use -1 if not known.
[in]	pCameraModel	Text string of camera model which the picture is taken with. Set to NULL if not known.
[in]	progfn	Pointer to progress monitoring function. See definition of PFC_PROGRESS.

Description:

Composite function that takes an input picture and enhances it base on user parameters. This function encapsulates all the details such as PFCENGINE, PFCPROFILE etc. and is suitable for use in server type mass processing environment. See sample project "SampleBasic" for usage details.

PFC_AutoCorrectPreset

Syntax:

```
voidPFC_AutoCorrectPreset(PFCIMAGE *pImage, PFCIMAGE *pImageds,
PFCPRESETID id, intiISO=-1, char *pCameraModel=NULL, PFC_PROGRESS
progfn=NULL);
```

Return:

<table>
<tr><td>0</td><td>The correction is successful.</td></tr>
<tr><td>> 0</td><td>Use macros to map out return code for each feature:
<table>
<tr><td>NRRETCODE</td><td>Returns PFCNR_STATUS Noise Removal correction status.</td></tr>
<tr><td>CORERETCODE</td><td>Returns PFCCORE_STATUS Core correction status.</td></tr>
<tr><td>FBRETCODE</td><td>Returns PFCFB_STATUS Face Beautification correction status.</td></tr>
<tr><td>RERETCODE</td><td>Returns PFCRE_STATUS Red Eye correction status.</td></tr>
</table></td></tr>
<tr><td>< 0</td><td>PFCAPPLYSTATUS enum.</td></tr>
</table>

Parameters:

[in]	pImage	Pointer to a PFCIMAGE structure that defines the image to be processed.
[in]	pImageds	Pointer to a PFCIMAGE structure that defines a supplementary down sampled image (1024 x 768) to aid in red eye detection. This parameter can be NULL. However the use of such supplementary image is highly recommended.
[in]	id	PFCPRESETID specifies parameters based on preset id.
[in]	iISO	ISO value when the image is taken. Use -1 if not known.
[in]	pCameraModel	Text string of camera model which the picture is taken with. Set to NULL if not known.
[in]	progfn	Pointer to progress monitoring function. See definition of PFC_PROGRESS.

Description:

Composite function that takes an input picture and enhances it using preset parameters specified by PFCPRESETID id.

PFC_ApplyLocal

Syntax:

```
PFCAPPLYSTATUSPFC_ApplyLocal(PFCIMAGE *pImage, intxOffset, intyOffset,
intwidthOrig, intheightOrig, PFCENGINE *pEngine, PFCPROFILE
pImageProfile, PFCPARAM param, int iOpacity=100);
```

Return:

0	The correction is successful.
> 0	PFCCORE_STATUS enum.
< 0	PFCAPPLYSTATUS enum.

Parameters:

[in/out]	pImage	Pointer to PFCIMAGE structure that defines the image to be processed.
[in]	xOffset	Horizontal (x) offset of image segment defined in pImage with respect to the original image.
[in]	yOffset	Vertical (y) offset of image segment defined in pImage with respect to the original image.
[in]	widthOrig	Pixel width of the original image.
[in]	heightOrig	Pixel height of the original image.
[in]	pEngine	Pointer returned from PFC_CreateEngine() function.
[in]	pImageProfile	Pointer to PFCPROFILE structure which is returned from the PFC_Calc() function.
[in]	param	PFCPARAM structure that carries the process parameters.
[in]	iOpacity	Control opacity of NR and CORE corrections. Range from 0 (non applied) to 100 (fully applied).

Description:

Perform CORE correction only to partial image as defined by the PFCIMAGE structure using user parameters as defined in a PFCPARAM structure. Correction requires profile as calculated by the PFC_Calc() function.**IMPORTANT: Only the core correction is applied.**

PFC_HasFaceBeautification

Syntax:

```
boolPFC_HasFaceBeautification(PFCENGINE pEngine);
```

Return:

True if feature is available for use. False otherwise. IF SFBEngine.dll is available and visible to the library the function should return true.

Parameters:

[in]	pEngine	Pointer returned from PFC_CreateEngine() function.

Description:

Utility function to query if Face Beautification feature is available at run time. This usually

PFC_FBFaceCount

Syntax:

```
intPFC_FBFaceCount(PFCPROFILE pProfile);
```

Return:

Number of face(s) detected.

Parameters:

[in]	pProfile	Pointer to PFCPROFILE structure which is returned from the PFC_Calc() function.

Description:

Utility function to query the number of faces detected during Face Beautification analysis.

PFC_GetFaceInfo

Syntax:

```
boolPFC_GetFaceInfo(PFCIMAGEPROFILE *pProfile, PFCFBFACEINFO *pFace,
int index);
```

Return:

True if face information is retrieved successfully. False if face is not detected or index is out of bound.

Parameters:

[in]	pProfile	Profile returned from PFC_Calc() function.
[out]	pFace	Pointer to PFCFBFACEINFO structure that carries face and eye details upon successful retrieval.
[in]	index	Index navigating the list. Must be >=0 and < number of faces detected.

Description:

Utility function to query geometry of detected faces.

Example:

```
// Declare PFCFBFACEINFO structure

PFCFBFACEINFO face;

int numFaces = PFC_FBFaceCount(pProfile);

for(int i = 0; i < numFaces; i++)

{

	if (PFC_GetFaceInfo(pProfile, &face, i))

	{

		// Do something with face info

	}

}
```

PFC_FAEFaceCount

Syntax:

```
intPFC_FAEFaceCount(PFCPROFILE pProfile);
```

Return:

Number of face(s) detected during Face Aware Exposure calculation.

Parameters:

[in]	pProfile	Pointer to PFCPROFILE structure which is returned from the PFC_Calc() function.

Description:

Utility function to query the number of faces detected during Face Aware Exposure analysis.

PFC_EnumFAEFaceRect

Syntax:

```
LPPFCFACERECTPFC_EnumFAEFaceRect(PFCPROFILE pProfile, LPPFCFACERECT p);
```

Return:

Pointer to the next face info as described in PFCFACERECT.

Parameters:

[in]	pProfile	Pointer to PFCPROFILE structure which is returned from the PFC_Calc() function.
[in]	p	LPPFCFACERECT type. Last face rect returned from this function.

Description:

Utility function to enumerate faces detected during Face Aware Exposure analysis.

Usage:

```
LPPFCFACERECT face = null;

while (face = PFC_EnumFAEFaceRect(pProfile, face))

{

    // Use face info in PFCFACERECT

}
```

PFC_AbnormalTintDetected

Syntax:

```
boolPFC_AbnormalTintDetected(PFCIMAGEPROFILE*pProfile, TINTCORRECTION
eTintMethod);
```

Return:

True if abnormal tint is detected when tint detection mode is eTintMethod. False otherwise. If Core calculation is not enabled during PFC_Calc(), this query returns False.

Parameters:

[in]	pProfile	Profile returned from PFC_Calc() function.
[in]	eTintMethod	TINTCORRECTION enum to specify tint detection method used.

Description:

Utility function to query if abnormal tint is detected at certain detection mode.

PFC_IsNoiseDetected

Syntax:

```
boolPFC_IsNoiseDetected(PFCIMAGEPROFILE *pProfile, int iPreset);
```

Return:

True if noise is detected when noise preset is iPreset. False otherwise. If Noise calculation is not enabled during PFC_Calc(), this query returns False.

Parameters:

[in]	pProfile	Profile returned from PFC_Calc() function.
[in]	iPreset	Noise preset to be checked.

Description:

Utility function to query if noise is detected at certain noise preset.

Structures

PFCIMAGE

<table>
<tr><td>width</td><td>int</td><td>Pixel width of image.</td></tr>
<tr><td>height</td><td>int</td><td>Pixel height of image.</td></tr>
<tr><td>stride</td><td>int</td><td>Byte width of each scanline.</td></tr>
<tr><td>format</td><td>PFCPIXELFORMAT</td><td>Defines byte order of input buffer. Possible values:
<table>
<tr><td>PFC_PixelFormat24bppRGB</td><td>24 bits per pixel; 8 bits each are used for the blue, green, and red components.
Byte order[B,G,R,B,G,R]</td></tr>
<tr><td>PFC_PixelFormat24bppBGR</td><td>24 bits per pixel; 8 bits each are used for the red, green, and blue components.
Byte order[R,G,B,R,G,B]</td></tr>
<tr><td>PFC_PixelFormat32bppABGR</td><td>32 bits per pixel; 8 bits each are used for the alpha, red, green, blue components.
Byte order[R,G,B,A]</td></tr>
<tr><td>PFC_PixelFormat48bppRGB</td><td>48 bits per pixel; 16 bits each are used for the red, green, blue components.
Byte order[B,G,R]</td></tr>
<tr><td>PFC_PixelFormat64bppARGB</td><td>64 bits per pixel; 16 bits each are used for the alpha, red, green, blue components.
Byte order[B,G,R,A]</td></tr>
<tr><td>PFC_PixelFormat32bppARGB</td><td>32 bits per pixel; 8 bits each are used for the alpha, red, green, blue components.
Byte order[B,G,R,A]</td></tr>
<tr><td>PFC_PixelFormat48bppBGR</td><td>48 bits per pixel; 16 bits each are used for the red, green, blue components.
Byte order[R,G,B]</td></tr>
<tr><td>PFC_PixelFormat64bppABGR</td><td>64 bits per pixel; 16 bits each are used for the alpha, red, green, blue</td></tr>
</table>
</td></tr>
</table>

			components. Byte order[R,G,B,A]
data	unsigned char*	Pointer pointing to the first byte of image data buffer.	

PFCENGINE

pEngine	PFCINTERNAL	Pointer to binary definition of the processing engine.
status	unsigned int	Status of engine.

PFCCOREPARAM

Parameter group for Perfectly Clear Core corrections. The “Range” column also includes recommended values for several parameters. However, we recommend starting with a Preset like Intelligent Auto, then adjust certain parameters to meet your needs, instead of starting with all parameters “off” and adding corrections. Our Workbench application is a great tool for visualizing these correction parameters, and will export the settings to a text file suitable for use with this SDK.

Parameter	Type	Description	Range
bEnabled	BOOL	Set to true to enable the entire Core correction.	
bAbnormalTintRemoval	BOOL	Set to true to enable Abnormal Tint Removal. Recommended default is FALSE.	
eTintMode	enum TINTCORRECTION	ENUM value defined in TINTCORRECTION. It sets the aggressiveness of Tint Removal.	
fTintScale	float	Scalar value of how much tint correction should be applied.	0.0 to 1.0
iBlackEnhancement	Int	Set luminance threshold for Noise Management	0 to 25 (12)
bVibrancy	BOOL	Set to true (recommended default) to enable Color Vibrancy in the library.	
iVibrancy	int	Degree of color vibrancy. This value will only be use when bVibrancy is TRUE. Very large values can	0 to 200 (5)

		produce extreme adjustments, so a value of 0, 5, or perhaps a high as 10 is advised.	
iStrength	int	Set the strength of exposure correction. If Automatic Strength Selection is enabled, the recommended value is put in this variable upon function return.	0 to 150 (100)
bContrast	BOOL	Set to TRUE to also apply Athentech‟s patented Medical Imaging contrast technology.	
eContrastMode	Enum CONTRASTMODE	Select contrast mode.	
iContrast	int	Intensity of contrast or depth	0 to 100
eBiasMode	Enum BIASMODE	Skin and depth bias control. Recommended value is BIAS_AVERAGE_PREFERENCE, unless you are printing to an indigo, iGen, or NexPress printer. If this is the case then use BIAS_BRIGHTER_PREFERENCE.	
fBiasScale	float	Scalar value of how much BIAS correction should be applied.	0.0 to 1.0
bSharpen	BOOL	Set to TRUE to enable sharpening.	
fSharpenScale	float	Sharpening intensity. This value controls how much sharpening to be applied.	0.0 to 3.0 (0.6)
bUseAutomaticStrengthSelection	BOOL	Set to TRUE (recommended default) to enable Automatic Strength Selection. Perfectly Clear will determine the optimum strength required for the input image. The value originally passed to the library in iStrength will be ignored. The strength recommended by Automatic Strength	

		Selection will be put in iStrength upon return to caller.	
bUseFAE	BOOL	Set to TRUE (recommended) to enable Face Aware Exposure selection. Recommended exposure will be calculated in favor of any human face detected from the image.	
eAggressiveness	enum AGGRESSIVENESS	The desired level of lighting for Automatic Strength Selection to target at the Exposure Correction.	
iMaxStrength	int	Use this value to limit the maximum Exposure to be applied on the Automatic Exposure Strength Selection algorithm.	0 to 150 (100)
bInfrared	BOOL	Set to TRUE (recommended default) to enable infrared correction. It‟s a good idea to turn Infrared Correction on if correcting lots of people in the picture.	
fInfrared	float	Scale value to control intensity of infrared correction.	0.0 to 1.0
bLightDiffusion	BOOL	Set to TRUE to enable light diffusion during DCF correction.	
fLightDiffusion	float	Scale value to control intensity of light diffusion correction.	0.0 to 1.0
bDCF	BOOL	Set to TRUE to enable Digital Color Fidelity. Recommended value is FALSE.	
eDCFMode	enum DCFMODE	Select different class of DCF.	
fDCF	float	Scale value to control intensity of Digital Color Fidelity correction.	0.0 to 1.0
bDynamicRange	BOOL	Set to TRUE to enable dynamic range correction.	

PFCFBPARAM

Parameter group for Face Beautification.

The parameter ranges for all “strength” settings expect for Catchlight are “relative to recommended values”. The Face Beautification library calculates recommended corrections for each face individually, and this is applied when the settings below are all enabled and set to 50. Higher or lower values apply more or less corrections.

Parameter	Type	Description	Range
bEnabled	BOOL	Set to TRUE to enable entire face beautification.	
bSmooth	BOOL	Set to TRUE to enable face smoothing.	
iSmoothLevel	int	Face smoothing level.	0 to 100
eSmoothMode	SKINMODE	Use SKINMODE_FACE to apply correction ONLY on skin regions included in faces. Use SKINMODE_BODY to apply correction on most skin regions regardless they are linked with a face or not.	
eSmoothType	SKINSMOOTHTYPE	See enum definition of SKINSMOOTHTYPE. P.28	
bEyeEnlarge	BOOL	Set to TRUE to enable eye enlargement.	
iEyeEnlargeLevel	int	Eye enlargement level.	0 to 100
bEyeEnhance	BOOL	Set to TRUE to enable eye enhancement.	
iEyeEnhanceLevel	int	Eye enhancement level.	
bEyeCircleRemoval	BOOL	Set to TRUE to enable eye circle removal.	
iEyeCircleRemovalLevel	int	Eye circle removal level.	0 to 100
bTeethWhiten	BOOL	Set to TRUE to enable teeth whitening.	
iTeethWhitenLevel	int	Teeth whitening level.	0 to 100
bBlemishRemoval	BOOL	Set to TRUE to enable blemish removal.	
iBlemishRemovalLevel	int	Blemish removal level.	0 to 100
bFaceSlim	BOOL	Set to TRUE to enable face slimming.	
iFaceSlimLevel	int	Face slimming level.	0 to 100
bDeFlash	BOOL	Set to TRUE to enable deflash.	
iDeFlashLevel	int	Deflash level.	0 to 100
bCatchLight	BOOL	Set to TRUE to enable	

		catchlight removal.	
iCatchLight	Int	Catchlight level.	0 to 100
iCatchLightType	int	1 Umbrella 2 Ringlight 3 Softbox 4 Beauty Dish 5 Outdoors	
bSkinToning	BOOL	Set to TRUE to enable skin toning.	
iSkinToning	int	Skin Toning level.	0 to 100
eSkinToningMode	SKINMODE	Use SKINMODE_FACE to apply correction ONLY on skin regions included in faces. Use SKINMODE_BODY to apply correction on most skin regions regardless they are linked with a face or not.	
eSkinToningType	SKINTONINGTYPE	See enum definition of SKINTONINGTYPE. P.28	
bLipSharpen	BOOL	Set to TRUE to enable lip sharpening.	
iLipSharpen	int	Lip sharpening level.	0 to 100
eLipSharpenType	LIPSHARPENTYPE	See definition of LIPSHARPENTYPE. P.28	
bBlush	BOOL	Set to TRUE to add blush.	
iBlush	int	Blush level.	0 to 100

PFCNRPARAM

Parameter group for Noise Removal.

Parameter	Type	Description	Range
bEnabled	BOOL	Set to TRUE to enable noise removal	
iPreset	int	Set preset number. 0 - default 1 - portrait 2 - night 3 - cameraphone 4 - force noise removal	0 to 4
iStrengthOffset	int	Offset to recommended level of noise removal strength	-5 to 5 (0)
iDetailOffset	int	Offset to recommended level of preservation of details	-30 to 30 (0)

PFCREPARAM

Parameter group for Red Eye correction.

bEnabled	BOOL	Set to TRUE to enable red eye removal.

PFCPOINT

x	int	X coordinate.
Y	Int	Y coordinate.

PFCRECT

left	int	Horizontal coordinate of left side of the rectangle.
Top	Int	Vertical coordinate of top of the rectangle.
Width	Int	Width of rectangle.
Height	Int	Height of rectangle.

PFCFBFACEINFO

face	PFCRECT	Bounding rectangle of detected face.
leftEye	PFCPOINT	Point of left eye in detected face.
rightEye	PFCPOINT	Point of right eye in detected face.

Enums

PFCPIXELFORMAT

PFC_PixelFormat24bppRGB	24 bits per pixel; 8 bits each are used for the red, green, and blue components. Byte order [BGRBGR]
PFC_PixelFormat24bppBGR	24 bits per pixel; 8 bits each are used for the red, green, and blue components. Byte order [RGBRGB]
PFC_PixelFormat32bppABGR	32 bits per pixel; 8 bits each are used for the alpha, red, green, and blue components. Byte order [RGBARGBA]
PFC_PixelFormat48bppRGB	48 bits per pixel; 16 bits each are used for the red, green, and blue components. Byte order [BBGGRRBBGGRR]
PFC_PixelFormat64bppARGB	64 bits per pixel; 16 bits each are used for the alpha, red, green, and blue components. Byte order [BBGGRRAABBGGRRAA]
PFC_PixelFormat32bppARGB	32 bits per pixel; 8 bits each are used for the alpha, red, green, and blue components. Byte order [BGRABGRA]
PFC_PixelFormat48bppBGR	48 bits per pixel; 16 bits each are used for the red, green, and blue components. Byte order [RRGGBBRRGGBB]
PFC_PixelFormat64bppABGR	64 bits per pixel; 16 bits each are used for the alpha, red, green, and blue components. Byte order [RRGGBBAARRGGBBAA]

PFCENGINESTATUS

ENGINESTATUS_OK	Engine successfully initialized.
ENGINESTATUS_FB_LIBRARY_LOAD_FAIL	Unable to load face beautification library. Check visibility of libSFBEngine.dll.
ENGINESTATUS_FB_FUNCTION_NOT_FOUND	Unable to load function from library. Check library version of libSFBEngine.dll.

PFCFEATURE

CALC_PFC	Enable calculation on Perfectly Clear Core correction.
CALC_NR	Enable calculation on Perfectly Clear Noise Removal.
CALC_FB	Enable calculation on Face Beautification.
CALC_RE	Enable calculation on Red Eye Removal.
CALC_ALL	Enable calculation on ALL of the above.

PFCPRESETID

PRESET_BEAUTIFY	Beautify
PRESET_BEAUTIFYPLUS	Beautify Plus

PRESET_DETAILS	Details
PRESET_VIVID	Vivid

TINTCORRECTION

TINTCORRECT_AGGRESSIVE	Aggressive detection of tint.
TINTCORRECT_DEFAULT	Moderate level of tint detection.
TINTCORRECT_CONSERVATIVE	Priority on minimum false positive detection.
TINTCORRECT_STRONGEST	Highest sensitivity level of tint detection.

CONTRASTMODE

HIGH_CONTRAST	Optimized to bring higher contrast to the image.
HIGH_DEFINITION	Optimized to bring out more details in the shadows, more details in the highlights, and more pleasing skin tones.

DCFMODE

DCF_STANDARD	For normal photo.
DCF_VIVID	For more color vibrancy.

AGGRESSIVENESS

AGGRESSIVENESS_CONSERVATIVE	Less aggressive in exposure correction.
AGGRESSIVENESS_MODERATE	Moderate level of exposure correction.
AGGRESSIVENESS_AGGRESSIVE	More aggressive in exposure correction.

BIASMODE

BIAS_NONE	Turn off bias correction.
BIAS_ASIAN_PREFERENCE	Fine tuned for Asian skin tone.
BIAS_AVERAGE_PREFERENCE	For average usage.
BIAS_BRIGHTER_PREFERENCE	Average usage with brighter tone.

SKINMODE

SKINMODE_FACE	Apply correction ONLY on skin regions included in faces.
SKINMODE_BODY	Apply correction on most skin regions regardless they are linked with

	a face or not.

SKINSMOOTHTYPE

SKINSMOOTHTYPE_SUBTLE	The subtle correction removes the wrinkles and spots alone while it keeps the texture of the face unchanged.
SKINSMOOTHTYPE_DEFAULT	This type of correction provides a more uniform appearance to the complexion. It combines the natural look of the SUBTLE mode with the improved efficiency of SUPERSMOOTH.
SKINSMOOTHTYPE_SUPERSMOOTH	This is a more aggressive and effective correction where the appearance of the entire skin (including wrinkles, spots, hot spots) is changed (softened).

SKINTONINGTYPE

SKINTONINGTYPE_WHITE	Whitens (bleaches) face. Recommended mainly for darker skin.
SKINTONINGTYPE_PALE	Makes skin look lighter and more pale.
SKINTONINGTYPE_WARM	Warms skin tone.
SKINTONINGTYPE_TAN	Darkens skin, makes it look naturally tanned.
SKINTONINGTYPE_FOUNDATION	Adjust skin to user defined foundation color.

LIPSHARPENTYPE

LIPSHARPENTYPE_FINE	Fine touch of sharpening.
LIPSHARPENTYPE_MEDIUM	Stronger sharpening. Details are more pronounced.
LIPSHARPENTYPE_COARSE	Lip details are coarsely pronounced.

PFCAPPLYSTATUS

APPLY_SUCCESS	Success.
APPLY_ERROR_PROFILE_MISSING	Pointer pProfile is NULL.
APPLY_ERROR_ENGINE_MISSING	Pointer pEngine is NULL.
APPLY_CANCELLED	Operation cancelled by user.
APPLY_NOSOURCE	Pointer pImage is NULL.
APPLY_BADFORMAT	Pixel format not supported.

PFCNR_STATUS

PFC_NR_SUCCESS	Success.
PFC_NR_NOTENABLED	Feature not enabled.
PFC_NR_FULLRES_REQUIRED	Full res image (pImage) is missing.

PFC_NR_CANCELLED	Process cancelled.
PFC_NR_ERRBITMAP	Error reading image data.
PFC_NR_ERRSETTINGS	Error in settings.
PFC_NR_MISC_ERROR	Misc. errors.
PFC_NR_NOTFOUND	Noise not found.

PFCCORE_STATUS

PFC_CORE_SUCCESS	Success.
PFC_CORE_NOTENABLED	Feature not enabled.
PFC_CORE_CANCELLED	Process cancelled.
PFC_CORE_NOSOURCEIMAGE	Full res source image (pImage) is missing.
PFC_CORE_INSUFFICIENTMEMORY	Process aborted because of insufficient memory.
PFC_CORE_MONOLITHIMAGE	Monolith source image.
PFC_CORE_BELOWMINSIZE	Image side dimension smaller than 32 pixels.

PFCFB_STATUS

PFC_FB_SUCCESS	Success.
PFC_FB_NOTENABLED	Feature not enabled.
PFC_FB_WARNING	Warning. e.g. face not detected.
PFC_FB_FULLRES_REQUIRED	Full res image (pImage) is missing.
PFC_FB_CANCELLED	Process cancelled.
PFC_FB_FUNCTION_ERROR	Unable to locate function in the SFBEngine library.
PFC_FB_CREATE_ENGINE_FAILED	Unable to create SFB Engine object for processing.
PFC_FB_ANALYSIS_FAILED	The face analysis did not complete successfully.
PFC_FB_NO_CORRECTION	No correction occur during process.
PFC_FB_NOT_EXECUTED	Not executed.
PFC_FB_NOT_AVAILABLE	Face beautification feature not available.

PFCRE_STATUS

PFC_RE_SUCCESS	Success.
PFC_RE_NOTENABLED	Feature not enabled.
PFC_RE_FULLRES_REQUIRED	Full res image (pImage) is missing.
PFC_RE_NOT_FOUND	Red eye not found.
PFC_RE_GEN_ERROR	General error.
PFC_RE_INVALID_PARAMETER	Invalid parameter.
PFC_RE_NO_MEMORY	Insufficient memory.
PFC_RE_CANCELLED	Process cancelled.
PFC_RE_NOT_SUPPORTED	Not supported.

Usage& Examples

(Windows)

Scenerio #1 - Using PCF_AutoCorrect

The simplest way to use Perfectly Clear library suite. This protocol is more suitable for developing a server type software project.

1. Initialize the parameter structure:

```
PFCPARAM param;

PFC_SetParam(param);
```

2. Perform full correction using the auto function:

```
int ret = PFC_AutoCorrect(&im, &imds, param);
```

Example:

```
// Create bitmap image

Bitmap *pBitmap = Bitmap::FromFile(dlg.m_ofn.lpstrFile);

// Create 1024 x 768 pixel bitmap image

Bitmap *pBMds = new Bitmap(1024, 768, pBitmap->GetPixelFormat());

{

	Graphics g(pBMds);

	g.SetInterpolationMode (InterpolationModeHighQualityBicubic);

	g.DrawImage(pBitmap, RectF(0, 0, 1024, 768));

}


BitmapData *pbd = new BitmapData;

pBitmap->LockBits(&Rect(0,0,pBitmap->GetWidth(),pBitmap->GetHeight()),
ImageLockModeWrite, pBitmap->GetPixelFormat(), pbd);

if ( pbd->PixelFormat == PixelFormat24bppRGB || pbd->PixelFormat ==
PixelFormat32bppARGB )

{

	LPBYTE lpBits = (LPBYTE)pbd->Scan0;
```

```
       IMAGE im, imds;

       im.width  =pbd->Width;

       im.height = pbd->Height;

       im.stride = pbd->Stride;

     im.format = (pbd->PixelFormat == PixelFormat24bppRGB)? PFC_PixelFormat24bppRGB :
PFC_PixelFormat32bppARGB;

       im.data = (unsigned char*)lpBits;



       // Use 1024 image for red eye detection

       BitmapData *pbdds = new BitmapData;

       pBMds->LockBits(&Rect(0,0,pBMds->GetWidth(),pBMds->GetHeight()),
ImageLockModeRead, pBitmap->GetPixelFormat(), pbdds);

       LPBYTE lpBitsds = (LPBYTE)pbdds->Scan0;

       imds.width  =pbdds->Width;

       imds.height = pbdds->Height;

       imds.stride = pbdds->Stride;

       imds.format = (pbdds->PixelFormat == PixelFormat24bppRGB)?
PFC_PixelFormat24bppRGB : PFC_PixelFormat32bppARGB;

       imds.data = (unsigned char*)lpBitsds;

       // Initialize parameter structure

       PFCPARAM param;

       PFC_SetParam(param);

       // Use the auto correct function.

       PFCAPPLYSTATUS ret = PFC_AutoCorrect(&im, &imds, param);

       TRACE("Apply returns %d\n", ret);

       pBMds->UnlockBits(pbdds);

       deletepBMds;

       deletepbdds;

}



pBitmap->UnlockBits(pbd);

SaveImage(pBitmap, CString("output.jpg"));



deletepBitmap;
```

```
deletepbd;
```

Scenerio #2- Separate PFC_Calc and PFC_Apply

More advanced way to use Perfectly Clear library suite.

1. Create process engine:

```
PFCENGINE *pEngine = PFC_CreateEngine();
```

2. Initialize the parameter structure (using default parameter values):

```
PFCPARAM param;

PFC_SetParam(param);
```

3. Perform pre-calculation of image specific profile:

```
PFCPROFILE pProfile = NULL;

pProfile = PFC_Calc(&im,&imds, pEngine);
```

(pEngine is created from step 1.)

4. Apply the calculated profile and parameters to the image.

```
PFC_Apply(&im, pEngine, pProfile, param);
```

5. Release resources used by PFCPROFILE:

```
PFC_ReleaseProfile(pProfile);
```

6. Release the engine to release resource:

```
PFC_DestroyEngine(pEngine);
```

Example:

```
// Create bitmap image
Bitmap *pBitmap = Bitmap::FromFile(dlg.m_ofn.lpstrFile);
// Create 1024 x 768 pixel bitmap image
Bitmap *pBMds = new Bitmap(1024, 768, pBitmap->GetPixelFormat());
{
	Graphics g(pBMds);
	g.SetInterpolationMode (InterpolationModeHighQualityBicubic);
```

```
	g.DrawImage(pBitmap, RectF(0, 0, 1024, 768));

}



BitmapData *pbd = new BitmapData;

pBitmap->LockBits(&Rect(0,0,pBitmap->GetWidth(),pBitmap->GetHeight()),
ImageLockModeWrite, pBitmap->GetPixelFormat(), pbd);

if ( pbd->PixelFormat == PixelFormat24bppRGB || pbd->PixelFormat ==
PixelFormat32bppARGB )

{

	LPBYTE lpBits = (LPBYTE)pbd->Scan0;

	IMAGE im, imds;

	im.width  =pbd->Width;

	im.height = pbd->Height;

	im.stride = pbd->Stride;

     im.format = (pbd->PixelFormat == PixelFormat24bppRGB)? PFC_PixelFormat24bppRGB :
PFC_PixelFormat32bppARGB;

	im.data = (unsigned char*)lpBits;



	// Use 1024 image for red eye detection

	BitmapData *pbdds = new BitmapData;

	pBMds->LockBits(&Rect(0,0,pBMds->GetWidth(),pBMds->GetHeight()),
ImageLockModeRead, pBitmap->GetPixelFormat(), pbdds);

	LPBYTE lpBitsds = (LPBYTE)pbdds->Scan0;

	imds.width  =pbdds->Width;

	imds.height = pbdds->Height;

	imds.stride = pbdds->Stride;

	imds.format = (pbdds->PixelFormat == PixelFormat24bppRGB)?
PFC_PixelFormat24bppRGB : PFC_PixelFormat32bppARGB;

	imds.data = (unsigned char*)lpBitsds;



	PFCENGINE *pEngine = PFC_CreateEngine();

	TRACE("Engine status %d\n", pEngine->status);



	PFCPARAM param;

	PFC_SetParam(param);
```

```
        PFCPROFILE pProfile = NULL;
        pProfile = PFC_Calc(&im, NULL, pEngine);
        PFCAPPLYSTATUS ret = PFC_Apply(&im, pEngine, pProfile, param);
        TRACE("Apply returns %d\n", ret);

        PFC_ReleaseProfile(pProfile);
        PFC_DestroyEngine(pEngine);

        pBMds->UnlockBits(pbdds);
        deletepBMds;
        deletepbdds;
}

pBitmap->UnlockBits(pbd);
SaveImage(pBitmap, CString("output.jpg"));

deletepBitmap;
deletepbd;
```

Scenario #3 - Interactive Corrections

User modification to preset parameters. User may want to use the Athentech preset "Vivid" parameters as a base and modify some of the parameters for specific need.

1. Create process engine:

```
PFCENGINE *pEngine = PFC_CreateEngine();
```

2. Initialize the parameter structure (using default parameter values):

```
PFCPARAM param;
PFC_SetParam(param, PRESET_VIVID);
```

3. Perform pre-calculation of image specific profile:

```
PFCPROFILE pProfile = NULL;

pProfile = PFC_Calc(&im,&imds, pEngine);
```

(pEngine is created from step 1.)

4. Modify process parameters as needed. For example, user wants to use a different contrast mode and enable abnormal tint removal.

```
param.core.eContrastMode = HIGH_CONTRAST;

param.core.bAbnormalTintRemoval = TRUE;

param.core.eTintMode = TINTCORRECT_DEFAULT;

param.core.fTintScale = 0.5f;
```

5. Apply the calculated profile and parameters to the image.

```
PFC_Apply(&im, pEngine, pProfile, param);
```

6. Release resources used by PFCPROFILE:

```
PFC_ReleaseProfile(pProfile);
```

7. Release the engine to release resource:

```
PFC_DestroyEngine(pEngine);
```

Example:

```
// Create bitmap image
Bitmap *pBitmap = Bitmap::FromFile(dlg.m_ofn.lpstrFile);
// Create 1024 x 768 pixel bitmap image
Bitmap *pBMds = new Bitmap(1024, 768, pBitmap->GetPixelFormat());
{
	Graphics g(pBMds);
	g.SetInterpolationMode (InterpolationModeHighQualityBicubic);
	g.DrawImage(pBitmap, RectF(0, 0, 1024, 768));
}

BitmapData *pbd = new BitmapData;
```

```
pBitmap->LockBits(&Rect(0,0,pBitmap->GetWidth(),pBitmap->GetHeight()),
ImageLockModeWrite, pBitmap->GetPixelFormat(), pbd);

if ( pbd->PixelFormat == PixelFormat24bppRGB || pbd->PixelFormat ==
PixelFormat32bppARGB )

{

       LPBYTE lpBits = (LPBYTE)pbd->Scan0;

       IMAGE im, imds;

       im.width  =pbd->Width;

       im.height = pbd->Height;

       im.stride = pbd->Stride;

      im.format = (pbd->PixelFormat == PixelFormat24bppRGB)? PFC_PixelFormat24bppRGB :
PFC_PixelFormat32bppARGB;

       im.data = (unsigned char*)lpBits;



       // Use 1024 image for red eye detection

       BitmapData *pbdds = new BitmapData;

       pBMds->LockBits(&Rect(0,0,pBMds->GetWidth(),pBMds->GetHeight()),
ImageLockModeRead, pBitmap->GetPixelFormat(), pbdds);

       LPBYTE lpBitsds = (LPBYTE)pbdds->Scan0;

       imds.width  =pbdds->Width;

       imds.height = pbdds->Height;

       imds.stride = pbdds->Stride;

       imds.format = (pbdds->PixelFormat == PixelFormat24bppRGB)?
PFC_PixelFormat24bppRGB : PFC_PixelFormat32bppARGB;

       imds.data = (unsigned char*)lpBitsds;



       PFCENGINE *pEngine = PFC_CreateEngine();

       TRACE("Engine status %d\n", pEngine->status);



       PFCPARAM param;

       PFC_SetParam(param);



       PFCPROFILE pProfile = NULL;

       pProfile = PFC_Calc(&im, NULL, pEngine, CALC_CORE);

       param.core.eContrastMode = HIGH_CONTRAST;
```

```
        param.core.bAbnormalTintRemoval = FALSE;
        param.core.eTintMode = TINTCORRECT_DEFAULT;
        param.core.fTintScale = 0.5F;
        PFCAPPLYSTATUS ret = PFC_Apply(&im, pEngine, pProfile, param);
        TRACE("Apply returns %d\n", ret);

        PFC_ReleaseProfile(pProfile);
        PFC_DestroyEngine(pEngine);

        pBMds->UnlockBits(pbdds);
        deletepBMds;
        deletepbdds;
}

pBitmap->UnlockBits(pbd);
SaveImage(pBitmap, CString("output.jpg"));

deletepBitmap;
deletepbd;
```

Usage (OSX / Linux)

Scenerio #1 - Using PCF_AutoCorrect

The simplest way to use Perfectly Clear library suite. This protocol is more suitable for developing a server type software project.

1. Initialize the parameter structure:

```
PFCPARAM param;

PFC_SetParam(param);
```

2. Perform full correction using the auto function:

```
int ret = PFC_AutoCorrect(&im, &imds, param);
```

Example:

```
// Populate image information into PFCIMAGE structure.

PFCIMAGE im;

im.width = pIT->width;

im.height = pIT->height;

im.stride = pIT->rowBytes;

im.format = PFC_PixelFormat24bppBGR;

im.data = pIT->data[0];


// Use auto correct function for image enhancement.

// Alternatively, you can use function PFC_AutoCorrect() if you prefer

// to supply your own process parameters.


int status = PFC_AutoCorrectPreset(&im, NULL, PRESET_VIVID);

printf("AutoCorrect returns %d\n", status);
```

```
// Check return status of auto correct function.
if (status == 0)
{
	// If successful write output file.
	WriteJPEG(pIT, "output1.jpg", 90);
}
else
{
	// In case of error, you can map out return status code for
	// each feature.

	int code;
	// Check return code for Noise Removal
	code = NRRETCODE(status);
	TRACE("Noise removal status  %d\n", code);

	// Check return code for Perfectly Clear Core correction
	code = CORERETCODE(status);
	TRACE("Perfectly Clear correction status  %d\n", code);

	// Check return code for Face Beautification
	code = FBRETCODE(status);
	TRACE("Face beautification status  %d\n", code);

	// Check return code for Red Eye Removal
```

```
        code = RERETCODE(status);

        TRACE("Red eye removal status  %d\n", code);

    }
```

Scenerio #2- Separate PFC_Calc and PFC_Apply

More advanced way to use Perfectly Clear library suite.

1. Create process engine:

```
PFCENGINE *pEngine = PFC_CreateEngine();
```

2. Initialize the parameter structure (using default parameter values):

```
PFCPARAM param;

PFC_SetParam(param);
```

3. Perform pre-calculation of image specific profile:

```
PFCPROFILE pProfile = NULL;

pProfile = PFC_Calc(&im,&imds, pEngine);
```

(pEngine is created from step 1.)

4. Apply the calculated profile and parameters to the image.

```
PFC_Apply(&im, pEngine, pProfile, param);
```

5. Release resources used by PFCPROFILE:

```
PFC_ReleaseProfile(pProfile);
```

6. Release the engine to release resource:

```
PFC_DestroyEngine(pEngine);
```

Example:

```
    // Initialize PFCPARAM structure with PFC_SetParam() function.

    PFCPARAM _param;

    PFC_SetParam(_param);
```

```
// Create PFCENGINE instance for use in this session.
PFCENGINE *pEngine = PFC_CreateEngine();

if (pEngine->status != ENGINESTATUS_OK)
{
	if (pEngine->status == ENGINESTATUS_FB_LIBRARY_LOAD_FAIL)
	{
		TRACE("Face Beautification library not available.");
	}
}

// Analyze image and obtain image specific profile.
PFCPROFILE pProfile = PFC_Calc(&im, NULL, pEngine, CALC_ALL, -1, NULL, NULL, myStatus);

if (pProfile->Status != 0)
{
	if (pProfile->Status & CALC_NR)
	{
		TRACE("  Pre-calculation on noise returns  %d\n", pProfile->NR_Status);
	}
	if (pProfile->Status & CALC_CORE)
	{
		TRACE("  Core pre-calculation returns  %d\n", pProfile->CORE_Status);
	}
	if (pProfile->Status & CALC_FB)
```

```
		{
			TRACE("  Pre-calculation on face details returns  %d\n", pProfile->FB_Status);
		}
		if (pProfile->Status & CALC_RE)
		{
			TRACE("  Pre-calculation on red eye returns  %d\n", pProfile->RE_Status);
		}
	}
	// Optionally you can check if Face Beautification is available in this version
	// of library.
	BOOL bFB = PFC_HasFaceBeautification(pEngine);
	if (bFB)
		TRACE("FB available.\n");
	else
		TRACE("FB NOT available.\n");

	// Optionally you can find geometry of faces detected.
	if (bFB)
	{
		int facecount = PFC_FBFaceCount(pProfile);
		for (int i = 0; i < facecount; i++)
		{
			PFCFBFACEINFO info;
			PFC_GetFaceInfo(pProfile, &info, i);
			TRACE("  Face  %d %d %d %d\n", info.face.left, info.face.top, info.face.width,
info.face.height);
```

```
        }
}

TRACE("Calc returns %d\n", pProfile->Status);

PFCAPPLYSTATUS status = PFC_Apply(&im, pEngine, pProfile, _param);
printf("Apply returns %d\n", status);

if (status == APPLY_SUCCESS)
{
        // Save image to file.
        WriteJPEG(pIT, "output.jpg", 90);
}
else
{
        int code;
        // Check return code for Noise Removal
        code = NRRETCODE(status);
        TRACE("Noise removal status  %d\n", code);

        // Check return code for Perfectly Clear Core correction
        code = CORERETCODE(status);
        TRACE("Perfectly Clear correction status  %d\n", code);

        // Check return code for Face Beautification
```

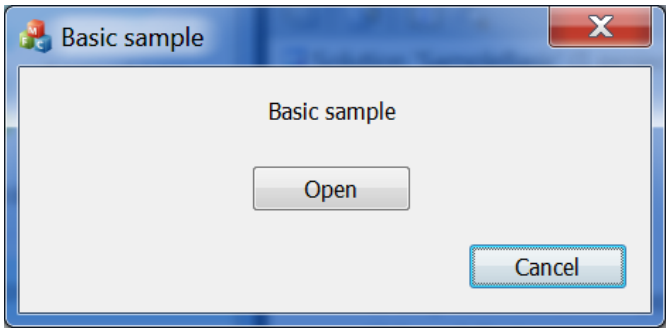
```
        code = FBRETCODE(status);
        TRACE("Face beautification status  %d\n", code);

        // Check return code for Red Eye Removal
        code = RERETCODE(status);
        TRACE("Red eye removal status  %d\n", code);
}

// Release Profile
PFC_ReleaseProfile(pProfile);

// Destroy engine. It is important that PFCENGINE is destroyed after
// all the profiles are released.
PFC_DestroyEngine(pEngine);
```

Sample Projects (Windows version)

Three sample projects are provided to showcase the usage of the API.

Sample 1

The Basic Sample demonstrates the most basic usage of the API. The sample application performs very basic image processing work flow: input, process, output.



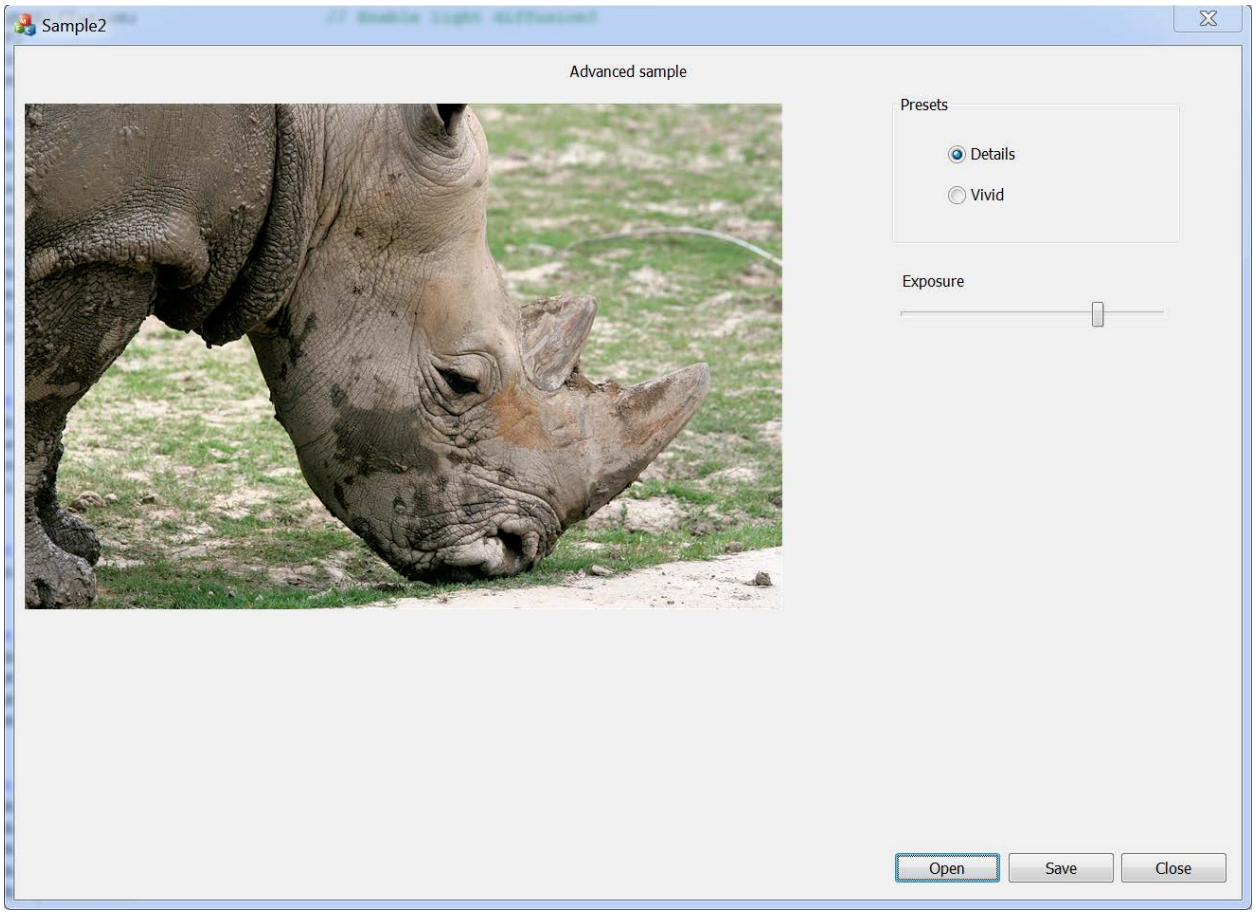
Press "Open" to select JPEG picture for processing and the Perfectly Clear processed picture is saved as "output.jpg" in the work folder.

Sample 2

Similar to the basic sample, the sample shows the simple way of using the API but unlike the basic sample, sample 1 gives user more controls on return code handling. The sample first analyses the input image with PFC_Calc(). The results (PFCPROFILE) is used in PFC_Apply() to carry out actual image enhancement.

Sample3

This sample shows advanced technique that streamlines the API usage in a image editing application.



Press "Open" to select JPEG picture for processing . The application proceeds to analyze the image and obtain the profile of the image. Toggle the preset radio buttons to select preset for processing. Slide the Exposure slider to adjust the exposure. Press the "Save" button to save the enhanced picture to file.

Sample Projects (OSX / Linux) (Full Source)

Three sample projects are provided to showcase the usage of the API.

Sample 1

Demonstrates the use of auto correction function.

```
int main(int argc, char* argv[])

{

	string inname;


	if ( argc > 1 )

	{

		inname = argv[1];

	}

	else

	{

		printf("sample2 <input file>\n");

		exit(0);

	}


	// Open input image.

	ImageTransfer *pIT = ReadJPEG(argv[1]);

	if (pIT == NULL)

	{

		printf("Unable to open input file.\n");

		exit(0);

	}
```

```
// Populate image information into PFCIMAGE structure.
PFCIMAGE im;
im.width = pIT->width;
im.height = pIT->height;
im.stride = pIT->rowBytes;
im.format = PFC_PixelFormat24bppBGR;
im.data = pIT->data[0];

// Use auto correct function for image enhancement.
// Alternatively, you can use function PFC_AutoCorrect() if you prefer
// to supply your own process parameters.

int status = PFC_AutoCorrectPreset(&im, NULL, PRESET_VIVID);
printf("AutoCorrect returns %d\n", status);

// Check return status of auto correct function.
if (status == 0)
{
	// If successful write output file.
	WriteJPEG(pIT, "output1.jpg", 90);
}
else
{
	// In case of error, you can map out return status code for
```

```
            // each feature.

            int code;
            // Check return code for Noise Removal
            code = NRRETCODE(status);
            TRACE("Noise removal status  %d\n", code);

            // Check return code for Perfectly Clear Core correction
            code = CORERETCODE(status);
            TRACE("Perfectly Clear correction status  %d\n", code);

            // Check return code for Face Beautification
            code = FBRETCODE(status);
            TRACE("Face beautification status  %d\n", code);

            // Check return code for Red Eye Removal
            code = RERETCODE(status);
            TRACE("Red eye removal status  %d\n", code);
        }


        FreeImageTransfer(pIT);

        return 1;
}
```

Sample 2

Demonstrates the use of Calc and Apply functions.

```
int main(int argc, char* argv[])
{
	string inname;

	if ( argc > 1 )
	{
		inname = argv[1];
	}
	else
	{
		printf("sample2 <input file>\n");
		exit(0);
	}

	ImageTransfer *pIT = ReadJPEG(argv[1]);
	if (pIT == NULL)
	{
		printf("Unable to open input file.\n");
		exit(0);
	}

	// Populate image information into PFCIMAGE structure.
```

```
PFCIMAGE im;
im.width = pIT->width;
im.height = pIT->height;
im.stride = pIT->rowBytes;
im.format = PFC_PixelFormat24bppBGR;
im.data = pIT->data[0];

// Initialize PFCPARAM structure with PFC_SetParam() function.
PFCPARAM _param;
PFC_SetParam(_param);

// Create PFCENGINE instance for use in this session.
PFCENGINE *pEngine = PFC_CreateEngine();

if (pEngine->status != ENGINESTATUS_OK)
{
	if (pEngine->status == ENGINESTATUS_FB_LIBRARY_LOAD_FAIL)
	{
		TRACE("Face Beautification library not available.");
	}
}

// Analyze image and obtain image specific profile.
PFCPROFILE pProfile = PFC_Calc(&im, NULL, pEngine, CALC_ALL, -1, NULL, NULL, myStatus);
```

```
if (pProfile->Status != 0)
{
	if (pProfile->Status & CALC_NR)
	{
		TRACE("  Pre-calculation on noise returns  %d\n", pProfile->NR_Status);
	}

	if (pProfile->Status & CALC_CORE)
	{
		TRACE("  Core pre-calculation returns  %d\n", pProfile->CORE_Status);
	}

	if (pProfile->Status & CALC_FB)
	{
		TRACE("  Pre-calculation on face details returns  %d\n", pProfile->FB_Status);
	}

	if (pProfile->Status & CALC_RE)
	{
		TRACE("  Pre-calculation on red eye returns  %d\n", pProfile->RE_Status);
	}
}

// Optionally you can check if Face Beautification is available in this version
// of library.
```

```
	BOOL bFB = PFC_HasFaceBeautification(pEngine);
	if (bFB)
		TRACE("FB available.\n");
	else
		TRACE("FB NOT available.\n");

	// Optionally you can find geometry of faces detected.
	if (bFB)
	{
		int facecount = PFC_FBFaceCount(pProfile);
		for (int i = 0; i < facecount; i++)
		{
			PFCFBFACEINFO info;
			PFC_GetFaceInfo(pProfile, &info, i);
			TRACE("  Face  %d %d %d %d\n", info.face.left, info.face.top, info.face.width,
info.face.height);
		}
	}

	TRACE("Calc returns %d\n", pProfile->Status);

	PFCAPPLYSTATUS status = PFC_Apply(&im, pEngine, pProfile, _param);
	printf("Apply returns %d\n", status);

	if (status == APPLY_SUCCESS)
```

```
{
        // Save image to file.
        WriteJPEG(pIT, "output.jpg", 90);
}
else
{
        int code;
        // Check return code for Noise Removal
        code = NRRETCODE(status);
        TRACE("Noise removal status  %d\n", code);

        // Check return code for Perfectly Clear Core correction
        code = CORERETCODE(status);
        TRACE("Perfectly Clear correction status  %d\n", code);

        // Check return code for Face Beautification
        code = FBRETCODE(status);
        TRACE("Face beautification status  %d\n", code);

        // Check return code for Red Eye Removal
        code = RERETCODE(status);
        TRACE("Red eye removal status  %d\n", code);
}

// Release Profile
```

```
        PFC_ReleaseProfile(pProfile);


        // Destroy engine. It is important that PFCENGINE is destroyed after
        // all the profiles are released.
        PFC_DestroyEngine(pEngine);



        FreeImageTransfer(pIT);


        return 1;
}
```

Mapping parameters from version 6 CORE API

V6	V7
* PerfectlyClearImageTransfer	* PerfectlyClearImageTransfer
bAbnormalTintRemoval	bAbnormalTintRemoval
eTint	eTintMode
fTintScale	fTintScale
iBlackEnhancement	iBlackEnhancement
bVibrancy	bVibrancy
iVibrancy	iVibrancy
bContrast	bContrast
eContrast	eContrastMode
iContrast	iContrast
eBias	eBiasMode
fBiasScale	fBiasScale
bSharpen	bSharpen
fImageRatio	fSharpenScale
bUseAutomaticStrengthSelection	bUseAutomaticStrengthSelection
eAggressiveness	eAggressiveness
iMaxStrength	iMaxStrength
bSkinTone	bSkinTone
fSkinTone	fSkinTone
bLightDiffusion	bLightDiffusion
bDCF	bDCF
eDCFMode	eDCFMode

In version 6, parameter pStrength is used as [in/out]. In version 7 it is changed to iStrength of integer (int) type and is [in] only.

Mapping parameters from version 6 PRO API

V6	V7
* PerfectlyClearImageTransfer	* PerfectlyClearImageTransfer
bTint	bAbnormalTintRemoval
eTintMethod	eTintMode
fTintScale	fTintScale
iBlackEnhance	iBlackEnhancement
bVibrancy	bVibrancy
iVibrancy	iVibrancy
bContrast	bContrast
eContrastMode	eContrastMode
fContrast (scale from 0.0 - 1.0)	iContrast (scale from 0 - 100)
eBias	eBiasMode
fBiasScale	fBiasScale
bSharpen	bSharpen
fSharpen	fSharpenScale
bASS	bUseAutomaticStrengthSelection
eAggressive	eAggressiveness
iGovernor	iMaxStrength
bSkinToneCorr	bSkinTone
eSkinToneVersion	(Not used)
fSkinToneScale	fSkinTone
bLightDiffusion	bLightDiffusion
bDCF	bDCF
eDCFMode	eDCFMode

In version 6, parameter pStrength is used as [in/out]. In version 7 it is changed to iStrength of integer (int) type and is [in] only.