Perfectly Clear
Intelligent Image Correction

PERFECTLY CLEAR API .NET ADAPTER REFERENCE

Version 7.4.1.0

For photos as vibrant as life itself

Background and Overview

Introduction

This document will explain how to use the Perfectly Clear .NET Adapter class 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 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 the adapter class 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 AutoCorrect into separate Calc and Apply calls, and the third example shows the usage for an interactive application where the correction parameters are altered by the user.

Supported Format

The following pixel formats of GDI bitmaps are supported in this version of adapter.

Format24bppRgb
Format32bppArgb
Format48bppRgb
Format64bppArgb

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 bRejectMonolith flag for enabling simple graphics rejection.

Added support for opacity in Apply function.

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:

bSkinTone   is 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.

Adapter Class Reference:

Name Space:

PerfectlyClearV7Adapter

Class:

PerfectlyClearV7x64 (x86)

Adapter class PerfectlyClearV7x64 encapsulates the details in deploying Perfectly Clear API version 7 in the .Net environment. (x86 for 32 bit version).

Class Methods:

PerfectlyClearV7x64

Syntax:

```
public unsafe class PerfectlyClearV7x64();
```

Return:

Instance of PerfectlyClearV7x64 class. If instantiation is successful, check class member *LastStatus.Status* for status (PFCENGINESTATUS see section Enums for details).

Description:

Constructor for PerfectlyClearV7x64 class.

SetParam

Syntax:

```
voidSetParam(PFCPRESETID id);
```

Parameters:

[in]	id	Preset id PFCPRESETID.

Description:

SetParam sets internal processing parameters in structure m_Param with values pertaining to Athentech preset as identified by the PFCPRESETID.

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

Calc

Syntax:

```
ADPTRRETURNCODE Calc(ref Bitmap bm);

ADPTRRETURNCODE Calc(ref Bitmap bm, ref Bitmap bmds);

ADPTRRETURNCODE Calc(ref Bitmap bm, PFCFEATURE feature);

ADPTRRETURNCODE Calc(ref Bitmap bm, ref Bitmap bmds, PFCFEATURE
feature);

ADPTRRETURNCODE Calc(ref Bitmap bm, ref Bitmapbmds,PFCFEATURE feature,
int ISO, string CameraModel, bool bRejectMonolith);

ADPTRRETURNCODE Calc(ref Bitmap bm, PFCFEATURE feature, int ISO,
string CameraModel, bool bRejectMonolith);
```

Return:

ADPTRRETURNCODE.

You should also query the *LastStatus* member of the class instance return codes in case of error.

The *Status* member contains top level return code of the last Calc action.

Syntax:

```
LastStatus.Status
```

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

Query individual status from the *LastStatus* class member.

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

Parameters:

[in/out]	bm	Bitmap instance that defines the image to be processed.	
[in]	bmds	Optional Bitmap instance that defines a supplementary down sampled image (approx. 1024 longest dimension) to aid in red eye detection.	
[in]	feature	Specify the type of calculations. Possible values are:	
		CALC_CORE	Calculates for Perfectly Clear Core correction.
		CALC_NR	Calculates for Perfectly Clear Noise Removal.
		CALC_FB	Calculates for Face Beautification.
		CALC_RE	Calculates for Red Eye Removal.
		CALC_ALL	Calculates for all of the above.
[in]	iISO	ISO value when the image is taken. Use -1 if not known.	
[in]	CameraModel	Text string of camera model which the picture is taken with. Set to NULL if not known.	
[in]	bRejectMonolith	Set to true to enable rejecting of simple graphics that may not look right with Perfectly Clear processing.	

Description:

Performs initial calculation of image specific profile parameters.

Apply

Syntax:

```
PFCAPPLYSTATUSApply(ref Bitmap bm);

PFCAPPLYSTATUS Apply(ref Bitmap bm, intiOpacity);
```

Return:

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

Parameters:

[in/out]	bm	Bitmap instance that defines the image to be processed.
[in]	iOpacity	Opacity level of Core Correction. From 0 (none applied) to 100 (fully applied).

Description:

Perform correction to image as defined by the Bitmap. Correction requires Calc() be called to establish internal image profile.

AutoCorrect

Syntax:

```
intAutoCorrect(ref Bitmap bm);

int AutoCorrect(ref Bitmap bm, ref Bitmap bmds);

int AutoCorrect(ref Bitmap bm, int ISO, string CameraModel);

int AutoCorrect(ref Bitmap bm, ref Bitmap bdms, int ISO, string
CameralModel);
```

Return:

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

Parameters:

[in/out]	bm	Bitmap instance that defines the image to be processed.
[in]	bmds	Optional Bitmap instance that defines a supplementary down sampled image (1024 x 768) to aid in red eye detection. However the use of such supplementary image is highly recommended.
[in]	ISO	Optional ISO value when the image is taken. Use -1 if not known.
[in]	CameraModel	Optional text string of camera model which the picture is taken with. Set to NULL if not known.

Description:

Composite function that takes an input picture and enhances it base on user parameters. This function encapsulates all the details involved in Perfectly Clear processing flow and is suitable for use in server type mass processing environment. See sample project "Sample1" for usage details.

ApplyLocal

Syntax:

```
PFCAPPLYSTATUSApplyLocal(ref Bitmap bm, intxOffset, intyOffset,
intwidthOrig, intheightOrig);
```

```
PFCAPPLYSTATUS ApplyLocal(ref Bitmap bm, intxOffset, intyOffset,
intwidthOrig, intheightOrig, intiOpacity);
```

Return:

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

Parameters:

[in/out]	bm	Bitmap instance 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]	iOpacity	Opacity level of Core Correction. From 0 (none applied) to 100 (fully applied).

Description:

Perform CORE correction only to partial image as defined by the Bitmap instance. Correction requires profile as calculated by the Calc() function.**IMPORTANT: Only the core correction is applied.**

HasFaceBeautification

Syntax:

```
boolHasFaceBeautification();
```

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.

Description:

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

FBFaceCount

Syntax:

```
intHasFaceBeautification();
```

Return:

Number of face(s) detected.

Description:

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

GetFaceInfo

Syntax:

```
boolGetFaceInfo(ref PFCFBFACEINFO info, int index);
```

Return:

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

Parameters:

[out]	info	Reference instance of 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:

```
if (Pfc.FBFaceCount() > 0)

{

for (inti = 0; i<Pfc.FBFaceCount(); i++)

{

PFCFBFACEINFO info = new PFCFBFACEINFO();

Pfc.GetFaceInfo(ref info, i);

}

}
```

AbnormalTintDetected

Syntax:

```
boolAbnormalTintDetected(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]	eTintMethod	TINTCORRECTION enum to specify tint detection method used.

Description:

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

Structures

PFCCOREPARAM

Parameter group for Perfectly Clear Core correction.

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 produce extreme adjustments, so a value of 0, 5, or perhaps a high as 10 is advised.	0 to 200 (5)
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.

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

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	Applied to only face area.
SKINMODE_BODY	Applied to all skin area.

SKINSMOOTHTYPE

SKINSMOOTHTYPE_SUBTLE	Removes wrinkles and spots while preserving skin texture.
SKINSMOOTHTYPE_DEFAULT	Even skin coverage, subtle in appearance.
SKINSMOOTHTYPE_SUPERSMOOTH	All skin defects are reduced.

SKINTONINGTYPE

SKINTONINGTYPE_WHITE	Whitens (bleaches) skin. Recommended mainly for darker skin.

SKINTONINGTYPE_PALE	Makes skin look lighter and more pale
SKINTONINGTYPE_WARM	Warms skin tone
SKINTONINGTYPE_TAN	Darkens skin, making it look naturally tanned
SKINTONINGTYPE_FOUNDATION	Colors the skin to a specific make-up color

LIPSHARPENTYPE

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

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	Source image is mono toned and cannot be processed.
PFC_CORE_BELOWMINSIZE	Source image dimension smallerthan 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

Scenerio #1 - Using 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:

```
PerfectlyClearV7x64 Pfc = new PerfectlyClearV7x64();
```

2. Perform full correction using the auto function:

```
int ret = Pfc.AutoCorrect(ref bm);
```

Example:

```
private void buttonTest_Click(object sender, EventArgs e)

{

OpenFileDialogofd = new OpenFileDialog();

ofd.Filter = "JPEG files (*.jpg)|*.jpg|All files (*.*)|*.*";

ofd.FilterIndex = 1;

if (ofd.ShowDialog() == DialogResult.OK)

{

string s = ofd.FileName;

// Instantiate class object

PerfectlyClearV7x64 Pfc = new PerfectlyClearV7x64();


// On successful instantiation

if (Pfc != null)

{

Trace.WriteLine((Pfc.HasFaceBeautification() ? "Has FB." : "No FB."));

// Optionally you can read image file with class method ReadImage.
Bitmap bm = Pfc.ReadImage(s);


// Use fully automated function for image correction

int ret = Pfc.AutoCorrect(ref bm, -1, null);


// That's it. The image in bm is now enhanced!

if (ret == 0)

{

bm.Save("Output.jpg");

}

else

{

// In case of error, query LastStatus for individual return code

Trace.WriteLine("Noise removal return code: " + Pfc.LastStatus.NR_Status.ToString());

Trace.WriteLine("Perfectly Clear core return code: " +
Pfc.LastStatus.CORE_Status.ToString());

Trace.WriteLine("Face beautification return code: " +
Pfc.LastStatus.FB_Status.ToString());

Trace.WriteLine("Red eye removal return code: " + Pfc.LastStatus.RE_Status.ToString());

}

}

}

}
```

Scenerio #2- Separate PFC_Calc and PFC_Apply

More advanced way to use Perfectly Clear library suite.

1. Instantiate the adapter class:

```
PerfectlyClearV7x64 Pfc = new PerfectlyClearV7x64()
```

2. Optionally one may choose to set processing parameters from a preset:

```
Pfc.SetParam(PFCPRESETID.PRESET_VIVID);
```

3. Perform pre-calculation of image specific profile:

```
ADPTRRETURNCODE ret = Pfc.Calc(ref bm);
```

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

```
int ret = Pfc.Apply(ref bm);
```

5. Release class instance (optional):

```
Pfc.Dispose();
```

Example:

```
private void buttonTest_Click(object sender, EventArgs e)
        {
OpenFileDialogofd = new OpenFileDialog();


ofd.Filter = "JPEG files (*.jpg)|*.jpg|All files (*.*)|*.*";
ofd.FilterIndex = 1;


if (ofd.ShowDialog() == DialogResult.OK)
            {
string s = ofd.FileName;
PerfectlyClearV7x64 Pfc = new PerfectlyClearV7x64();
```

```
if (Pfc != null)
                {
Trace.WriteLine("Constructor success.");
Trace.WriteLine((Pfc.HasFaceBeautification() ? "Has FB." : "No FB."));
                    Bitmap bm = Pfc.ReadImage(s);

ADPTRRETURNCODE ret =Pfc.Calc(ref bm, PFCFEATURE.CALC_ALL, -1, null);


Trace.WriteLine(Pfc.LastStatus.Status.ToString());
Trace.WriteLine(Pfc.LastStatus.NR_Status.ToString());
Trace.WriteLine(Pfc.LastStatus.CORE_Status.ToString());
Trace.WriteLine(Pfc.LastStatus.FB_Status.ToString());
Trace.WriteLine(Pfc.LastStatus.RE_Status.ToString());

                 // Optionally you can select a different preset
Pfc.SetParam(PFCPRESETID.PRESET_VIVID);
                    // Apply user settings and enhance image
Pfc.Apply(ref bm);
                    // You may check the return code for each process features.
Trace.WriteLine(Pfc.LastStatus.Status.ToString());
Trace.WriteLine(Pfc.LastStatus.NR_Status.ToString());
Trace.WriteLine(Pfc.LastStatus.CORE_Status.ToString());
Trace.WriteLine(Pfc.LastStatus.FB_Status.ToString());
Trace.WriteLine(Pfc.LastStatus.RE_Status.ToString());


                    // Optionally you may force resource disposal.
Pfc.Dispose();


bm.Save("output.jpg");
                }
```

```
else
                {
Trace.WriteLine("Class constructor failed.");
                }
            }
        }
```

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. Instantiate adapter class:

```
PerfectlyClearV7x64 Pfc = new PerfectlyClearV7x64()
```

2. Perform pre-calculation of image specific profile:

```
ADPTRRETURNCODE ret = Pfc.Calc(ref bm);
```

3. Optionally examine calculation results such as face detected, abnormal tint status etc.:

```
// Optionally display coordinates of detected faces (if any)

if (Pfc.FBFaceCount()> 0)

{

Trace.WriteLine("Face detected.");

for (inti = 0; i<Pfc.FBFaceCount(); i++)

{

PFCFBFACEINFO info = new PFCFBFACEINFO();

Pfc.GetFaceInfo(ref info, i);


Trace.WriteLine(info.face.left.ToString() + "  " + info.face.top.ToString() + "  " +
info.face.width.ToString() + "  " + info.face.height.ToString());

}

}
```

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

```
Pfc.m_Param.core.eBiasMode = BIASMODE.BIAS_BRIGHTER_PREFERENCE;

Pfc.m_Param.core.fBiasScale = 0.8f;


Pfc.m_Param.core.eContrastMode = CONTRASTMODE.HIGH_CONTRAST;


Pfc.m_Param.core.bAbnormalTintRemoval = true;
Pfc.m_Param.core.eTintMode = TINTCORRECTION.TINTCORRECT_DEFAULT;

Pfc.m_Param.core.fTintScale = 0.5f;
```

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

```
int ret = Apply(ref bm);
```

6. Release resources used by class instance (optional):

```
Pfc.Dispose();
```

Example:

```
private void buttonTest_Click(object sender, EventArgs e)
        {
OpenFileDialogofd = new OpenFileDialog();


ofd.Filter = "JPEG files (*.jpg)|*.jpg|All files (*.*)|*.*";
ofd.FilterIndex = 1;


if (ofd.ShowDialog() == DialogResult.OK)
            {
string s = ofd.FileName;
PerfectlyClearV7x64 Pfc = new PerfectlyClearV7x64();
if (Pfc != null)
                {
Trace.WriteLine("Constructor success.");


                    // Optionally check if Face Beautification is available
Trace.WriteLine((Pfc.HasFaceBeautification() ? "Has FB." : "No FB."));
                    Bitmap bm = Pfc.ReadImage(s);


ADPTRRETURNCODE ret = Pfc.Calc(ref bm, PFCFEATURE.CALC_ALL, -1, null);
Trace.WriteLine(Pfc.LastStatus.Status.ToString());
Trace.WriteLine(Pfc.LastStatus.NR_Status.ToString());

Trace.WriteLine(Pfc.LastStatus.CORE_Status.ToString());

Trace.WriteLine(Pfc.LastStatus.FB_Status.ToString());

Trace.WriteLine(Pfc.LastStatus.RE_Status.ToString());


                    // Optionally display coordinates of detected faces (if any)

if (Pfc.FBFaceCount()> 0)

                    {

Trace.WriteLine("Face detected.");

for (inti = 0; i<Pfc.FBFaceCount(); i++)

                        {

                            PFCFBFACEINFO info = new PFCFBFACEINFO();

Pfc.GetFaceInfo(ref info, i);


Trace.WriteLine(info.face.left.ToString() + "  " + info.face.top.ToString() + "  " +
info.face.width.ToString() + "  " + info.face.height.ToString());

                        }

                    }

                    // Optionally, customize preset parameters

Pfc.m_Param.core.eBiasMode = BIASMODE.BIAS_BRIGHTER_PREFERENCE;

Pfc.m_Param.core.fBiasScale = 0.8f;


Pfc.m_Param.core.eContrastMode = CONTRASTMODE.HIGH_CONTRAST;


Pfc.m_Param.core.bAbnormalTintRemoval = true;

Pfc.m_Param.core.eTintMode = TINTCORRECTION.TINTCORRECT_DEFAULT;

Pfc.m_Param.core.fTintScale = 0.5f;


                    // Apply and enhance image

Pfc.Apply(ref bm);
```

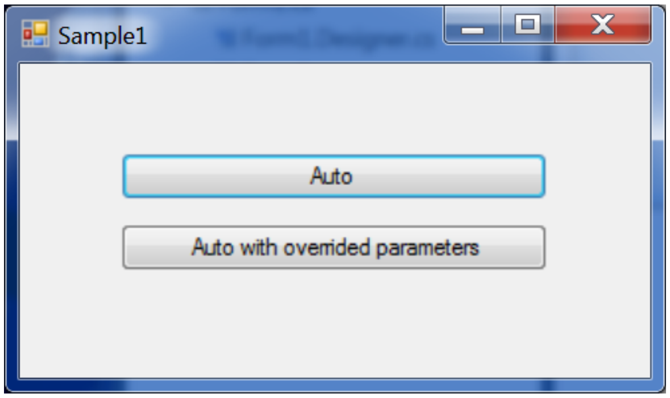
```
Trace.WriteLine(Pfc.LastStatus.Status.ToString());

Trace.WriteLine(Pfc.LastStatus.NR_Status.ToString());

Trace.WriteLine(Pfc.LastStatus.CORE_Status.ToString());

Trace.WriteLine(Pfc.LastStatus.FB_Status.ToString());

Trace.WriteLine(Pfc.LastStatus.RE_Status.ToString());


Pfc.Dispose();


bm.Save("output.jpg");

                }

else

                {

Trace.WriteLine("Class constructor failed.");

                }

            }

        }
```

Sample Projects

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.



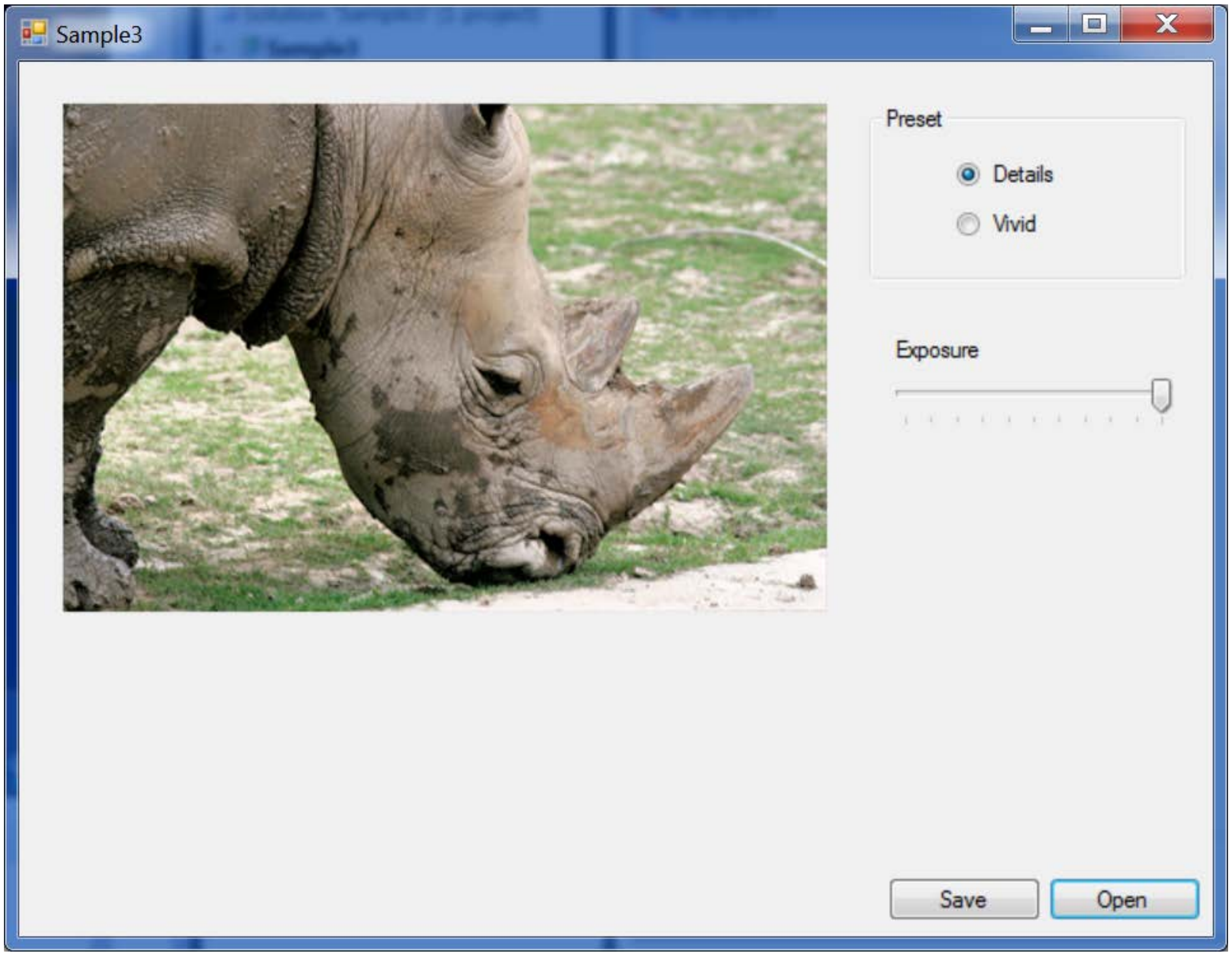
Press any of the two buttons to select JPEG picture for processing and the Perfectly Clear processed picture is saved as "output.jpg" in the work folder.

Sample 2

Similar to the basic sample, the sample shows the simple way of using the API but unlike the basic sample, sample 2 gives user more controls on return code handling. The sample first analyses the input image with Calc(). The results is used when Apply() is called 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.