Betaface

Betaface Face Detection and Recognition SDK

version 2.0

Introduction

Betaface Face Detection and Recognition SDK is a Windows DLL library containing set of the algorithms trained and tuned to detect human face pattern and some of the facial features on static images and in video streams. SDK detects frontal faces (with some tolerance to the head rotation) under any angle on the image and returns coordinates of faces found and all detected face features. SDK can be accessed from different environments, as long as you can make a call to DLL, COM object or .Net assembly.

SDK features summary:

- *Detect multiple faces on images (all SDK editions) or in video streams (SDK Xtreme).*
- *Crop faces from the images based on detection information.*
- *Compare faces (similarity score, identification, and verification).*
- *Morph, warp or generate average face images.*
- *Detect face landmarks coordinates (22 points in SDK Standard, 22+86=108 points in SDK Pro and Xtreme).*
- *Classification, such as gender, age, ethnicity estimation; smile, glasses, mustache and beard detection (SDK Pro and Xtreme).*
- *Additional face measurements such as face and face features shape description; hairstyle shape estimation; skin, hair, clothes color (SDK Pro and Xtreme).*

Most of the algorithms do not require color information and work on grayscale pixel values internally. Extended measurements that require color information (for example hairstyle detection) will return empty result for grayscale images.

Technical details:

SDK core is a Windows native DLL with additional COM and .Net interface wrappers. SDK package includes all necessary runtime files, this documentation and precompiled samples of usage with source code in C++ and C#. SDK includes software license protection system. Our typical licensing option is a hardware-locked license key file, which allow you to execute SDK runtime on a specific PC hardware.

Images can be loaded either from file system location (in various supported formats) or directly from memory (raw pixel data). SDK Xtreme also allows video input for face tracking (VFW compatible source, external video capture with raw video frames or, with additional sample project, from DirectShow compatible sources).

Betaface offers additional components, ready or customized solutions on request, such as complete web service infrastructure, including server-side and web-side assemblies, storage database and processing queue, as well as 64bit SDK builds.

Contents

SDK functions

SDK initialization and return values

Each SDK function returns integer value - 0 for no error and negative values for various error conditions. Following values are predefined for all functions:

BETAFACE_BAD_STATE	*NULL*
BETAFACE_OK	*0*
BETAFACE_ERROR_INTERNAL	*-1*
BETAFACE_ERROR_INVALIDPARAM	*-2*
BETAFACE_ERROR_LOADINGIMAGE	*-3*
BETAFACE_ERROR_NOTSUPPORTED	*-4*

SDK instance should be initialized prior to calling various SDK functions. Initialization function will return internal state reference value which should be passed back with each SDK function call. It is recommended to initialize separate instance for each execution thread. After initialization different images or video feeds can be repeatedly processed. At the application exit, or to save RAM allocated for instance de-initialization function should be called.

Betaface_Init *(BetafaceInternalState* pState)*

Call this function to initialize the library and retrieve internal state value.

Parameters:

pState	*Pointer to BetafaceInternalState variable, by this address internal state reference value will be returned. Value BETAFACE_BAD_STATE is returned if the function fails.*

Return value:

Function returns BETAFACE_OK if it is successful, error code otherwise.

Betaface_Deinit *(BetafaceInternalState* pState)*

Call this function to release internal library state and all associated resources.

Parameters:

pState	*Pointer to BetafaceInternalState variable, by this address should be stored valid internal state reference value.*

Return value:

Function returns BETAFACE_OK if it is successful, error code otherwise.

Loading, saving, copying, displaying and releasing images

Betaface_LoadImage *(BetafaceInternalState State, char* strImageFilename, BetafaceImage* pImg)*

This function loads the static image from the HDD to the memory and converts it to internal Betaface image representation format.

Parameters:

State	*Betaface library internal state value, obtained from Betaface_Init function.*
strImageFilename	*full pathname to the static image on the HDD. Accepted file formats are: JPEG files (*.jpeg;*.jpg;*.jpe), Windows bitmap (*.bmp;*.dib), Portable Network Graphics files (*.png), Portable image format (*.pbm;*.pgm;*.ppm), Sun raster files (*.sr;*.ras) and TIFF Files (*.tiff;*.tif).*
pImg	*pointer to BetafaceImage variable, by this address image in internal Betaface format will be returned.*

Return value:

Function returns BETAFACE_OK if it is successful, error code otherwise.

Betaface_SaveImage *(BetafaceInternalState State, char* strImageFilename, BetafaceImage Img, BetafaceSaveImageFlags flags, char* strText);*

This function converts and stores image from internal Betaface image representation to one of the common file formats.

Parameters:

State	*Betaface library internal state value, obtained from Betaface_Init function.*
strImageFilename	*Full pathname to the static image on the HDD. Accepted file formats are: JPEG files (*.jpeg;*.jpg;*.jpe), Windows bitmap (*.bmp;*.dib), Portable Network Graphics files (*.png), Portable image format (*.pbm;*.pgm;*.ppm), Sun raster files (*.sr;*.ras) and TIFF Files (*.tiff;*.tif).*
Img	*Betaface internal representation image in BetafaceImage type variable*
flags	*Optional combination of image saving parameter flags.* *Following flags are currently supported:* *BETAFACE_SAVEIMAGE_GRAYSCALE = 0x1 – convert image to grayscale.* *BETAFACE_SAVEIMAGE_FLIP_HORISONTAL = 0x2 – mirrors image.*
strText	*Reserved for future use*

Return value:

Function returns BETAFACE_OK if it is successful, error code otherwise.

Betaface_LoadMemoryImage *(BetafaceInternalState State, unsigned char* pImageBytes, int iWidth, int iHeight, double dPixelAspect, BetafaceMemoryFormat nFormat, bool bFlipVertical, BetafaceImage* pImg)*

This function loads the static image from the memory and converts it to internal Betaface image representation format.

Parameters:

State	*Betaface library internal state value, obtained from Betaface_Init function.*
pImageBytes	*Pointer to a memory buffer where uncompressed image pixel data is located according to supplied image size and format parameters.*
iWidth	*Image width in pixels*
iHeight	*Image height in pixels*
dPixelAspect	*Real x/y pixel aspect of the image pixels. For regular images stored as pixel buffers this value is 1.0. For uncompressed video frames in some specific formats like MPEG2 this value can differ from 1.0, i.e. pixels are not squares. SDK functions need square pixels as an input and this function can reshape them.*
nFormat	*Format of the image pixel data in supplied memory buffer. Accepted format values are:* *BETAFACE_MEMORYIMAGE_GG = 0x1 (8 bit grayscale)* *BETAFACE_MEMORYIMAGE_RRGGBB = 0x2 (24 bit RGB)* *BETAFACE_MEMORYIMAGE_BBGGRR = 0x3 (24 bit BGR)* *BETAFACE_MEMORYIMAGE_RRGGBBAA = 0x4 (32 bit RGB)* *BETAFACE_MEMORYIMAGE_BBGGRRAA = 0x5 (32 bit BGR)*
bFlipVertical	*If this parameter is true, image buffer will be flipped vertically during conversion. SDK assumes origin of the image is in top left corner.*
pImg	*Pointer to BetafaceImage variable, by this address image in internal Betaface format will be returned.*

Return value:

Function returns BETAFACE_OK if it is successful, error code otherwise.

Betaface_SaveMemoryImage *(BetafaceInternalState State, BetafaceImage Img, BetafaceMemoryFormat nFormat, int* piWidth, int* piHeight, char** ppImageBytes, int* piImageBytesLen)*

This function converts and exports the static image from the internal Betaface image representation format to memory format.

Parameters:

State	*Betaface library internal state value, obtained from Betaface_Init function.*
Img	*Betaface internal representation image in BetafaceImage type variable*
nFormat	*Format of the image pixel data that exported memory image should have. Accepted format values are:* *BETAFACE_MEMORYIMAGE_GG = 0x1 (8 bit grayscale)* *BETAFACE_MEMORYIMAGE_RRGGBB = 0x2 (24 bit RGB)* *BETAFACE_MEMORYIMAGE_BBGGRR = 0x3 (24 bit BGR)* *BETAFACE_MEMORYIMAGE_RRGGBBAA = 0x4 (32 bit RGB)*

	BETAFACE_MEMORYIMAGE_BBGGRRAA = 0x5 (32 bit BGR)
	BETAFACE_MEMORYIMAGE_RGB32FLT = 0x6 (96bit RGB32FLT)
piWidth	*Pointer to integer variable where image width in pixels will be returned*
piHeight	*Pointer to integer variable where image height in pixels will be returned*
ppImageBytes	*Pointer to a pointer variable where allocated memory buffer containing uncompressed image pixel data according to supplied format parameter will be returned.*
piImageBytesLen	*Pointer to integer variable where allocated memory buffer length in bytes will be returned.*

Return value:

Function returns BETAFACE_OK if it is successful, error code otherwise.

Betaface_CopyImage *(BetafaceInternalState State, BetafaceImage Img, BetafaceImage* pImgCopy)*

This function creates a copy of an image in internal Betaface representation.

Parameters:

State	*Betaface library internal state value, obtained from Betaface_Init function.*
Img	*Betaface internal representation image in BetafaceImage variable to be copied.*
pImg	*Pointer to BetafaceImage variable, by this address a copy of Img image in internal Betaface format will be returned.*

Return value:

Function returns BETAFACE_OK if it is successful, error code otherwise.

Betaface_ReleaseImage *(BetafaceInternalState State, BetafaceImage* pImg)*

This function releases image in internal Betaface representation

Parameters:

State	*Betaface library internal state value, obtained from Betaface_Init function*
pImg	*Betaface internal representation image in BetafaceImage variable*

Return value:

Function returns BETAFACE_OK if it is successful, error code otherwise.

Betaface_ReleaseMemoryImage *(BetafaceInternalState State, char** ppImageBytes)*

This function releases memory buffer allocated via Betaface_SaveMemoryImage function

Parameters:

State	*Betaface library internal state value, obtained from Betaface_Init function*
ppImageBytes	*Pointer to a variable containing memory buffer allocated and returned by Betaface_SaveMemoryImage function*

Return value:

Function returns BETAFACE_OK if it is successful, error code otherwise

Betaface_GetImageSize *(BetafaceInternalState State, BetafaceImage Img, int* piWidth, int* piHeight)*

This function returns image dimensions in pixels

Parameters:

State	*Betaface library internal state value, obtained from Betaface_Init function*
Img	*Image in internal Betaface format*
piWidth	*By this address width of the image in pixels will be returned*
piHeight	*By this address height of the image in pixels will be returned*

Return value:

Function returns BETAFACE_OK if it is successful, error code otherwise

Betaface_DisplayImage *(BetafaceInternalState State, BetafaceImage Img, char* strWindowName)*

This function returns display of the image in a separate window for debugging purposes

Parameters:

State	*Betaface library internal state value, obtained from Betaface_Init function*
Img	*Image in internal Betaface format*
strWindowName	*Window caption name for display*

Return value:

Function returns BETAFACE_OK if it is successful, error code otherwise

Face detection

Betaface_DetectFaces *(BetafaceInternalState State, BetafaceImage Img, BetafaceDetectionSettings dSettings, int* piFacesCount, BetafaceDetectionResult* pDetectionResult)*

This functions searches for the face pattern on the given images in all locations and under any angles.

Parameters:

State	*Betaface library internal state value, obtained from Betaface_Init function*
Img	*Source image in internal Betaface representation*
dSettings .Flags	*Combination of detection flags. Currently following flags are supported:* *BETAFACE_DETECTFACES_BASIC = 0x0 or* *BETAFACE_DETECTFACES_PRO = 0x1 – specify one of those flags to indicate which feature points you need to detect – Basic (22 points) or Basic + Pro (22 + 86 facial points).* *BETAFACE_DETECTFACES_BASIC_PROFILE = 0x4 – detect also profile faces (profile faces will have lower accuracy of recognition and Pro points detection)* *BETAFACE_DETECTFACES_BESTFACEONLY = 0x100 – If this flag is specified, only single face with highest detection score is processed and returned.* *BETAFACE_DETECTFACES_PROCALC_CALC_ALL = 0xFC03 – If this flag is supported by your SDK edition you can enable generation of 'extended measurements' which describe geometrical face properties, approximate hairstyle shape, detecting skin, hair, clothes colors and other measurements. Enabling this flag will slow down face detection function significantly. Extended measurements will be calculated only for the face with highest detection score.*
dSettings .iMaxImageWidthPix dSettings .iMaxImageHeightPix	*Limits of the source image resolution. First number of pixels is calculated as nPixels = iMaxImageWidthPix * iMaxImageHeightPix, then if actual source image resolution exceeds this number, image is downscaled before processing. Influences speed and quality of features detection. Recommended values are 640 or 320 for iMaxImageWidthPix and 480 or 240 for iMaxImageHeightPix. If one or both values are 0 no downscaling is performed, however this option is not recommended.*
dSettings .dMinFaceSizeOnImage	*Limits the minimum face size on the image as the fraction to the whole image. Used to increase the speed of detection. 0.3 means that faces that are 30% size of the whole picture or bigger will be detected. Value should be adjusted depending on your application. Value can be set to 0.*
dSettings .iMinFaceSizePix	*Limits the minimum face size as the number of pixels on the shortest face rectangle size. Used to limit minimum face resolution. Recommended value is 50. Value can be set to 0.*

dSettings .dAngleDegrees	*Central angle of the faces detected. If dAngleToleranceDegrees parameter is 0 or greater or equal to 180 degrees this parameter is ignored.*
dSettings .dAngleToleranceDegrees	*Limits the deviation of the face rotation angle from dAngleDegrees central angle. If this parameter is 0 or greater or equal to 180 degrees no angle limiting is performed. Recommended values are 30 for portrait photos (+-30 degrees from strictly vertical face, if dAngleDegrees is 0) or 0 for full image scan.*
dSettings. dMinDetectionScore	*You can reduce amount of false positive face detections by putting a limit on detection score. Use 0.0 to receive all faces detected. Recommended value for filtering is the range 0.1-0.25, which in most of the cases filters out all false detections including also blurred faces or faces with low resolution.*
piFacesCount	*Pointer to integer variable where number of faces detected will be returned.*
pDetectionResult	*Pointer to the BetafaceDetectionResult variable where the detection data in internal Betaface representation will be returned.*

Return value:

Function returns BETAFACE_OK if it is successful, error code otherwise.

Betaface_ReleaseDetectionResult *(BetafaceInternalState State, BetafaceDetectionResult* pDetectionResult)*

This function releases the detection result data and all resources associated with it.

Parameters:

State	*Betaface library internal state value, obtained from Betaface_Init function*
pDetectionResult	*Pointer to the BetafaceDetectionResult variable where valid detection data in internal Betaface representation is stored.*

Return value:

Function returns BETAFACE_OK if it is successful, error code otherwise

Processing detection result, retrieving faces information

Detection result is essentially a collection of metadata associated with each face detected. This collection of metadata is stored in a separate internal collection object for each face. You can query how many faces were detected, retrieve specific face metadata and later get/set individual parameters values in it.

Betaface_GetFacesCount *(BetafaceInternalState State, BetafaceDetectionResult DetectionResult, int* piFacesCount)*

This function returns the number of detected faces information contained in detection result

Parameters:

State	*Betaface library internal state value, obtained from Betaface_Init function*
DetectionResult	*BetafaceDetectionResult variable where valid detection data in internal Betaface representation is stored*
piFacesCount	*Pointer to integer variable where number of faces detected will be returned*

Return value:

Function returns BETAFACE_OK if it is successful, error code otherwise

Betaface_GetFaceInfo *(BetafaceInternalState State, BetafaceDetectionResult DetectionResult, int iFaceIndex, BetafaceFaceInfo* pFaceInfo)*

This function retrieves BetafaceFaceInfo face information in internal Betaface representation from the detection result by the given index.

Parameters:

State	*Betaface library internal state value, obtained from Betaface_Init function*
DetectionResult	*BetafaceDetectionResult variable, where valid detection data in internal Betaface representation is stored*
iFaceIndex	*Index of the face, starting from 0*
pFaceInfo	*Pointer to the BetafaceFaceInfo variable where the face information data in internal Betaface representation will be returned.*

Return value:

Function returns BETAFACE_OK if it is successful, error code otherwise

Betaface_CreateFaceInfo *(BetafaceInternalState State, BetafaceFaceInfo* pFaceInfo)*

This function creates empty face information structure, which can be further filled in with points and metadata. It is used when you need to create face information structure manually, for example to crop or further process face defined by serialized points information stored in external database.

Parameters:

State	*Betaface library internal state value, obtained from Betaface_Init function*
pFaceInfo	*Pointer to the BetafaceFaceInfo variable where empty face information data in internal Betaface representation will be returned.*

Return value:

Function returns BETAFACE_OK if it is successful, error code otherwise

Betaface_CopyFaceInfo *(BetafaceInternalState State, BetafaceFaceInfo FaceInfo, BetafaceFaceInfo* pFaceInfoCopy)*

This function creates a complete copy of face information structure.

Parameters:

State	*Betaface library internal state value, obtained from Betaface_Init function.*
FaceInfo	*BetafaceFaceInfo variable containing face information data to be copied.*
pFaceInfoCopy	*Pointer to the BetafaceFaceInfo variable where complete copy of FaceInfo face information data in internal Betaface representation will be returned.*

Return value:

Function returns BETAFACE_OK if it is successful, error code otherwise

Betaface_ReleaseFaceInfo *(BetafaceInternalState State, BetafaceFaceInfo* pFaceInfo)*

Call this function to release face information data and all resources associated with it.

Parameters:

State	*Betaface library internal state value, obtained from Betaface_Init function.*
pFaceInfo	*Pointer to the BetafaceFaceInfo variable where valid face information data in internal Betaface representation is stored.*

Return value:

Function returns BETAFACE_OK if it is successful, error code otherwise

Getting and setting face information property values

Face information is represented by collection of named property values. Each property value is referenced by combination (bitwise OR) of two integer constants – one of them defining specific Feature and another defining specific parameter of this feature. Each feature can be one of three types (Face, Point and Measurement) and have different parameters associated with it. For example: single Face type feature called "Face" represents rectangular area, where face is found on the image and has X,Y coordinates of its center as well as width/height of the face rectangle and detection score as parameters.

Features of Point type do not have detection scores or width/height information but only have X,Y coordinates. Measurements have single value parameter as well as min/max parameters defining a value within given range. Each parameter value can be either Boolean or Double, depends on the parameter, and corresponding function should be used to get/set it.

Betaface_GetFaceInfoBoolParam *(BetafaceInternalState State, BetafaceFaceInfo FaceInfo, BetafaceFeatureParam param, bool* pValue);*

This function used to retrieve Boolean type parameters values from face information data

Parameters:

State	*Betaface library internal state value, obtained from Betaface_Init function*
FaceInfo	*BetafaceFaceInfo variable with stored valid face information data in internal Betaface representation*
param	*Parameter ID (a combination of face feature flag and parameter flag), for example: parameter BETAFACE_PARAM_EXISTS | BETAFACE_FEATURE_EYE_L value will be set to true if left eye was detected on this face and coordinate information is available.*
pValue	*Pointer to bool variable where the requested parameter value will be returned*

Return value:

Function returns BETAFACE_OK if it is successful, error code otherwise

Betaface_GetFaceInfoDoubleParam *(BetafaceInternalState State, BetafaceFaceInfo FaceInfo, BetafaceFeatureParam param, double* pValue);*

This function is used to retrieve double type parameters values from face information data.

Parameters:

State	*Betaface library internal state value, obtained from Betaface_Init function*
FaceInfo	*BetafaceFaceInfo variable with stored valid face information data in internal Betaface representation*
param	*Parameter ID (a combination of face feature flag and parameter flag), for example parameter BETAFACE_PARAM_X | BETAFACE_FEATURE_EYE_L value will contain X*

	coordinate of the center of left eye in pixels.
pValue	*Pointer to double variable where the requested parameter value will be returned*

Return value:

Function returns BETAFACE_OK if it is successful, error code otherwise

Betaface_SetFaceInfoBoolParam *(BetafaceInternalState State, BetafaceFaceInfo FaceInfo, BetafaceFeatureParam param, bool Value);*

This function used to set bool type parameters values in face information data

Parameters:

State	*Betaface library internal state value, obtained from Betaface_Init function*
FaceInfo	*BetafaceFaceInfo variable where stored valid face information data in internal Betaface representation*
param	*Parameter ID (a combination of face feature flag and parameter flag)*
Value	*bool parameter value to be set*

Return value:

Function returns BETAFACE_OK if it is successful, error code otherwise

Betaface_SetFaceInfoDoubleParam *(BetafaceInternalState State, BetafaceFaceInfo FaceInfo, BetafaceFeatureParam param, double Value);*

This function used to set double type parameters values in face information data

Parameters:

State	*Betaface library internal state value, obtained from Betaface_Init function*
FaceInfo	*BetafaceFaceInfo variable with stored valid face information data in internal Betaface representation*
param	*Parameter ID (a combination of face feature flag and parameter flag)*
Value	*double parameter value to be set*

Return value:

Function returns BETAFACE_OK if it is successful, error code otherwise

Predefined parameters:

parameter name and constant		*feature types*	*value type*	*description*
BETAFACE_PARAM_DETECTED	*0x0*	*Face*	*Boolean*	*Indicating that corresponding feature information is present.*
BETAFACE_PARAM_SCORE	*0x1*	*Face*	*Double*	*Detection score of the face, representing confidence of the face detection algorithm.*
BETAFACE_PARAM_SCORE_PRO	*0x7*	*Face*	*Double*	*Detection score of 86 Pro facial points detection.*
BETAFACE_PARAM_WIDTH	*0x4*	*Face*	*Double*	*Width in pixels*
BETAFACE_PARAM_HEIGHT	*0x5*	*Face*	*Double*	*Height in pixels*
BETAFACE_PARAM_ANGLE	*0x6*	*Face*	*Double*	*Angle in degrees*
BETAFACE_PARAM_X	*0x2*	*Face or Point*	*Double*	*X coordinate in pixels*
BETAFACE_PARAM_Y	*0x3*	*Face or Point*	*Double*	*Y coordinate in pixels*
BETAFACE_PARAM_VALUE	*0x10*	*Measurement*	*Double*	*Measurement value*
BETAFACE_PARAM_MIN	*0x11*	*Measurement*	*Double*	*Minimum of the value range*
BETAFACE_PARAM_MAX	*0x12*	*Measurement*	*Double*	*Maximum of the value range*

Features of the Face type:

BETAFACE_FEATURE_FACE	*0x00000100*

Features of the Point type *(includes 8 basic and 86 Pro points):*

Available in all SDK editions

BETAFACE_FEATURE_EYE_L	*0x00000200*	*Center of left eye*
BETAFACE_FEATURE_EYE_R	*0x00000300*	*Center of right eye*
BETAFACE_FEATURE_EYE_LCO	*0x00000400*	*Left eye outer corner*
BETAFACE_FEATURE_EYE_RCO	*0x00000500*	*Right eye outer corner*
BETAFACE_FEATURE_EYE_LCI	*0x00000600*	*Left eye inner corner*
BETAFACE_FEATURE_EYE_RCI	*0x00000700*	*Right eye inner corner*
BETAFACE_FEATURE_MOUTH	*0x00000B00*	*Mouth center*
BETAFACE_FEATURE_MOUTH_LC	*0x00000800*	*Mouth left corner*
BETAFACE_FEATURE_MOUTH_RC	*0x00000900*	*Mouth right corner*
BETAFACE_FEATURE_NOSE_TIP	*0x00000A00*	*Tip of the nose*
BETAFACE_FEATURE_NOSE_L	*0x00001500*	*Left nostril*
BETAFACE_FEATURE_NOSE_R	*0x00001600*	*Right nostril*
BETAFACE_FEATURE_EYEBROW_L	*0x00000C00*	*Left eyebrow center*
BETAFACE_FEATURE_EYEBROW_R	*0x00000F00*	*Right eyebrow center*

BETAFACE_FEATURE_EYEBROW_LCI	*0x00000D00*	*Left eyebrow inner corner*
BETAFACE_FEATURE_EYEBROW_LCO	*0x00000E00*	*Left eyebrow outer corner*
BETAFACE_FEATURE_EYEBROW_RCI	*0x00001000*	*Right eyebrow inner corner*
BETAFACE_FEATURE_EYEBROW_RCO	*0x00001100*	*Right eyebrow outer corner*
BETAFACE_FEATURE_CHIN_B	*0x00001200*	*Chin bottom*
BETAFACE_FEATURE_CHIN_L	*0x00001300*	*Chin left*
BETAFACE_FEATURE_CHIN_R	*0x00001400*	*Chin right*

Available in Pro and Xtreme SDK editions

BETAFACE_FEATURE_PRO_CHIN_EARCONN_L	*0x00010000*
BETAFACE_FEATURE_PRO_CHIN_P1_L	*0x00020000*
BETAFACE_FEATURE_PRO_CHIN_P2_L	*0x00030000*
BETAFACE_FEATURE_PRO_CHIN_P3_L	*0x00040000*
BETAFACE_FEATURE_PRO_CHIN_P4_L	*0x00050000*
BETAFACE_FEATURE_PRO_CHIN_P5_L	*0x00060000*
BETAFACE_FEATURE_PRO_CHIN_B	*0x00070000*
BETAFACE_FEATURE_PRO_CHIN_P5_R	*0x00080000*
BETAFACE_FEATURE_PRO_CHIN_P4_R	*0x00090000*
BETAFACE_FEATURE_PRO_CHIN_P3_R	*0x000A0000*
BETAFACE_FEATURE_PRO_CHIN_P2_R	*0x000B0000*
BETAFACE_FEATURE_PRO_CHIN_P1_R	*0x000C0000*
BETAFACE_FEATURE_PRO_CHIN_EARCONN_R	*0x000D0000*
BETAFACE_FEATURE_PRO_TEMPLE_P4_R	*0x000E0000*
BETAFACE_FEATURE_PRO_TEMPLE_P3_R	*0x000F0000*
BETAFACE_FEATURE_PRO_TEMPLE_P2_R	*0x00100000*
BETAFACE_FEATURE_PRO_TEMPLE_P1_R	*0x00110000*
BETAFACE_FEATURE_PRO_TEMPLE_R	*0x00120000*
BETAFACE_FEATURE_PRO_FOREHEAD_R	*0x00130000*
BETAFACE_FEATURE_PRO_FOREHEAD_P4	*0x00140000*
BETAFACE_FEATURE_PRO_FOREHEAD_P3	*0x00150000*
BETAFACE_FEATURE_PRO_FOREHEAD_M	*0x00160000*
BETAFACE_FEATURE_PRO_FOREHEAD_P2	*0x00170000*
BETAFACE_FEATURE_PRO_FOREHEAD_P1	*0x00180000*
BETAFACE_FEATURE_PRO_FOREHEAD_L	*0x00190000*
BETAFACE_FEATURE_PRO_TEMPLE_L	*0x001A0000*
BETAFACE_FEATURE_PRO_TEMPLE_P1_L	*0x001B0000*
BETAFACE_FEATURE_PRO_TEMPLE_P2_L	*0x001C0000*
BETAFACE_FEATURE_PRO_TEMPLE_P3_L	*0x001D0000*
BETAFACE_FEATURE_PRO_TEMPLE_P4_L	*0x001E0000*
BETAFACE_FEATURE_PRO_EYE_O_R	*0x001F0000*
BETAFACE_FEATURE_PRO_EYE_BO_R	*0x00200000*
BETAFACE_FEATURE_PRO_EYE_B_R	*0x00210000*

BETAFACE_FEATURE_PRO_EYE_BI_R	*0x00220000*
BETAFACE_FEATURE_PRO_EYE_I_R	*0x00230000*
BETAFACE_FEATURE_PRO_EYE_TI_R	*0x00240000*
BETAFACE_FEATURE_PRO_EYE_T_R	*0x00250000*
BETAFACE_FEATURE_PRO_EYE_TO_R	*0x00260000*
BETAFACE_FEATURE_PRO_EYE_O_L	*0x00270000*
BETAFACE_FEATURE_PRO_EYE_TO_L	*0x00280000*
BETAFACE_FEATURE_PRO_EYE_T_L	*0x00290000*
BETAFACE_FEATURE_PRO_EYE_TI_L	*0x002A0000*
BETAFACE_FEATURE_PRO_EYE_I_L	*0x002B0000*
BETAFACE_FEATURE_PRO_EYE_BI_L	*0x002C0000*
BETAFACE_FEATURE_PRO_EYE_B_L	*0x002D0000*
BETAFACE_FEATURE_PRO_EYE_BO_L	*0x002E0000*
BETAFACE_FEATURE_PRO_EYEBROW_I_R	*0x002F0000*
BETAFACE_FEATURE_PRO_EYEBROW_TI_R	*0x00300000*
BETAFACE_FEATURE_PRO_EYEBROW_T_R	*0x00310000*
BETAFACE_FEATURE_PRO_EYEBROW_TO_R	*0x00320000*
BETAFACE_FEATURE_PRO_EYEBROW_O_R	*0x00330000*
BETAFACE_FEATURE_PRO_EYEBROW_BO_R	*0x00340000*
BETAFACE_FEATURE_PRO_EYEBROW_B_R	*0x00350000*
BETAFACE_FEATURE_PRO_EYEBROW_BI_R	*0x00360000*
BETAFACE_FEATURE_PRO_EYEBROW_I_L	*0x00370000*
BETAFACE_FEATURE_PRO_EYEBROW_TI_L	*0x00380000*
BETAFACE_FEATURE_PRO_EYEBROW_T_L	*0x00390000*
BETAFACE_FEATURE_PRO_EYEBROW_TO_L	*0x003A0000*
BETAFACE_FEATURE_PRO_EYEBROW_O_L	*0x003B0000*
BETAFACE_FEATURE_PRO_EYEBROW_BO_L	*0x003C0000*
BETAFACE_FEATURE_PRO_EYEBROW_B_L	*0x003D0000*
BETAFACE_FEATURE_PRO_EYEBROW_BI_L	*0x003E0000*
BETAFACE_FEATURE_PRO_MOUTH_L	*0x003F0000*
BETAFACE_FEATURE_PRO_MOUTH_TL	*0x00400000*
BETAFACE_FEATURE_PRO_MOUTH_T	*0x00410000*
BETAFACE_FEATURE_PRO_MOUTH_TR	*0x00420000*
BETAFACE_FEATURE_PRO_MOUTH_R	*0x00430000*
BETAFACE_FEATURE_PRO_MOUTH_BR	*0x00440000*
BETAFACE_FEATURE_PRO_MOUTH_B	*0x00450000*
BETAFACE_FEATURE_PRO_MOUTH_BL	*0x00460000*
BETAFACE_FEATURE_PRO_NOSE_T_L	*0x00470000*
BETAFACE_FEATURE_PRO_NOSE_TI_NOSTRIL_L	*0x00480000*
BETAFACE_FEATURE_PRO_NOSE_TO_NOSTRIL_L	*0x00490000*
BETAFACE_FEATURE_PRO_NOSE_BO_NOSTRIL_L	*0x004A0000*
BETAFACE_FEATURE_PRO_NOSE_B_NOSTRIL_L	*0x004B0000*
BETAFACE_FEATURE_PRO_NOSE_B	*0x004C0000*

BETAFACE_FEATURE_PRO_NOSE_B_NOSTRIL_R	*0x004D0000*
BETAFACE_FEATURE_PRO_NOSE_BO_NOSTRIL_R	*0x004E0000*
BETAFACE_FEATURE_PRO_NOSE_TO_NOSTRIL_R	*0x004F0000*
BETAFACE_FEATURE_PRO_NOSE_TI_NOSTRIL_R	*0x00500000*
BETAFACE_FEATURE_PRO_NOSE_T_R	*0x00510000*
BETAFACE_FEATURE_PRO_EYE_IRIS_R	*0x00520000*
BETAFACE_FEATURE_PRO_EYE_IRIS_L	*0x00530000*
BETAFACE_FEATURE_PRO_NOSE_TIP	*0x00540000*
BETAFACE_FEATURE_PRO_CHEEKBONE_L	*0x00550000*
BETAFACE_FEATURE_PRO_CHEEKBONE_R	*0x00560000*

Features of the Measurement type

Measurement features have id constants between

BETAFACE_FEATURE_PROPARAM_CALC_FIRST	*0x25000000*

and

BETAFACE_FEATURE_PROPARAM_CALC_LAST	*0x28FF0000*

with a step of

BETAFACE_FEATURE_PROPARAM_MULT	*0x00010000*

For complete list of Measurement features supported in your version of SDK please contact Betaface support.

Face recognition

Face recognition process involves converting face information and corresponding image into independent recognition binary 'key' which usually have a size of few kilobytes or less and can be stored in the database or any other storage like file system. Recognition key is essentially face and facial features description in compact binary form. Recognition keys can be compared in pairs giving similarity value. Similarity value can be used either directly to sort results of multiple faces comparison in the order of similarity or, with a conversion for specific False Alarm (FA) rate and rank, they give normalized confidence value - how likely two faces belong to the same person, in the range 0-100% as well as decision whether it is the same person or not. Accuracy of such identification/verification tasks strongly depends on the data used, size of the problem, particular algorithm and comparison strategy.

Betaface_GenerateFaceKey *(BetafaceInternalState State, BetafaceImage Img, BetafaceFaceInfo FaceInfo, BetafaceRecognitionFlags flags, char** ppFaceKeyData, int* piFaceKeyLen)*

Function converts the face to the special binary "key" representation, which can be quickly compared with any other "key(s)" to determine how similar one face is to another. These keys are usually stored in the database as BLOB (binary large object) fields.

Parameters:

State	*Betaface library internal state value, obtained from Betaface_Init function*
Img	*Image containing face to crop and to which FaceInfo corresponds*
FaceInfo	*BetafaceFaceInfo variable where valid face information data in internal Betaface representation is stored*
flags	*Flag defining which key to generate, depending on SDK edition can be one of:*
	BETAFACE_RECKEY_DEFAULT = 0xFF – best available key type
	BETAFACE_RECKEY_BASIC = 0x1
	BETAFACE_RECKEY_PRO = 0x2
ppFaceKeyData	*Pointer to the variable where the address of the key binary data will be returned*
piFaceKeyLen	*Pointer to the integer variable where length of the key data in bytes will be returned*

Return value:

Function returns BETAFACE_OK if it is successful, error code otherwise

Betaface_CompareFaceKeys *(BetafaceInternalState State, char* pFaceKeyData1, char* pFaceKeyData2, int FaceKeysLen, double* pdSimilarity)*

This function compares two binary face "keys" of the same length and returns raw similarity score value.

Parameters:

State	*Betaface library internal state value, obtained from Betaface_Init function*
pFaceKeyData1	*Address of first face "key" binary data*
pFaceKeyData2	*Address of second face "key" binary data*

FaceKeysLen	*Length of the keys data in bytes*
pdSimilarity	*Similarity score for these two faces. The higher the score the more similarities between two faces are found. The range of similarity score is undefined in general case and depends on the particular algorithm.*

Return value:

Function returns BETAFACE_OK if it is successful, error code otherwise

Betaface_CompareFaceKeysEx *(BetafaceInternalState State, char* pFaceKeyData1, char* pFaceKeyData2, int FaceKeysLen, int iRank, double dFalseAlarmRate, bool* pbIsSamePerson, double* pdNormalizedConfidence)*

This function compares two binary face "keys" of the same length and returns normalized confidence value as well as identification decision for specified False Alarm (FA) rate and rank.

Parameters:

State	*Betaface library internal state value, obtained from Betaface_Init function*
pFaceKeyData1	*Address of first face "key" binary data*
pFaceKeyData2	*Address of second face "key" binary data*
FaceKeysLen	*Length of the keys data in bytes*
iRank	*Target recognition rank (currently only 0 rank is supported)*
dFalseAlarmRate	*Target false alarm rate, default 0.02*
pbIsSamePerson	*Returned identification decision – true if those two faces belong to the same person.*
pdNormalizedConfidence	*Returned abstract confidence value that those two faces belong to the same person, normalized in the range 0-100%*

Return value:

Function returns BETAFACE_OK if it is successful, error code otherwise

Betaface_ReleaseFaceKey *(BetafaceInternalState State, char** ppFaceKeyData)*

This function release face "key" binary data and all associated resources

Parameters:

State	*Betaface library internal state value, obtained from Betaface_Init function*
ppFaceKeyData	*Pointer to variable containing address of face "key" binary data to be released*

Return value:

Function returns BETAFACE_OK if it is successful, error code otherwise

Betaface_JoinFaceKeys *(BetafaceInternalState State, char* pFaceKeyData1, char* pFaceKeyData2, int FaceKeysLen, bool bAppend, char** ppFaceKeyData, int* piFaceKeyLen)*

When more than one image is available for particular person usual strategy is to run comparison function using each face of one person and each face of another person, and select the best comparison result as the winning value. This function on the other hand will allow you to ' join' recognition keys created from multiple pictures into single 'strong' key, which might improve recognition quality for controlled environments input data (all faces are frontal, non-extreme lighting). In this case we recommend to create strong keys that consist of minimum 3 and ideally 10 usual keys for each person in database.

Parameters:

State	*Betaface library internal state value, obtained from Betaface_Init function*
pFaceKeyData1	*First recognition key for join operation*
pFaceKeyData2	*Second recognition key for join operation*
FaceKeysLen	*Length of the keys data in bytes*
bAppend	*Type of operation, when true then key1 and key2 are joined, when false key2 is subtracted from key1*
ppFaceKeyData	*Pointer to the variable where the address of the resulting strong key binary data will be returned*
piFaceKeyLen	*Pointer to the integer variable where length of the key data in bytes will be returned*

Return value:

Function returns BETAFACE_OK if it is successful, error code otherwise

Betaface_ReconstructFace *(BetafaceInternalState State, char* pFaceKeyData, int iFaceKeyLen, int iWidth, int iHeight, double dEyesDistance, double dEyeLineHeightK, BetafaceImage* pImg);*

This function can convert recognition key back into synthetic image representation of a human face. Use this function to control quality of face recognition data learned for each person

Parameters:

State	*Betaface library internal state value, obtained from Betaface_Init function*
pFaceKeyData	*Address of face "key" binary data*
iFaceKeyLen	*Length of the key data in bytes*
iWidth	*Output image width*
iHeight	*Output image height*
dEyesDistance	*Equivalent to the same parameter in Betaface_CropFaceImage function*
dEyeLineHeightK	*Equivalent to the same parameter in Betaface_CropFaceImage function*
pImg	*Resulting image with synthetic reconstructed face*

Return value:

Function returns BETAFACE_OK if it is successful, error code otherwise

Betaface_ReconstructFaceAvi *(BetafaceInternalState State, char* pFaceKeyData, int iFaceKeyLen, int iWidth, int iHeight, double dEyesDistance, double dEyeLineHeightK, char* strFilename)*

This function is equivalent to Betaface_ReconstructFace except that instead of static image it produces animated synthetic face, written in simple avi file format. Specify full filename, including .avi extension in strFilename parameter

Parameters:

State	*Betaface library internal state value, obtained from Betaface_Init function*
pFaceKeyData	*Address of face "key" binary data*
iFaceKeyLen	*Length of the key data in bytes*
iWidth	*Output image width*
iHeight	*Output image height*
dEyesDistance	*Equivalent to the same parameter in Betaface_CropFaceImage function*
dEyeLineHeightK	*Equivalent to the same parameter in Betaface_CropFaceImage function*
strFilename	*Resulting avi filename and path with synthetic reconstructed face*

Return value:

Function returns BETAFACE_OK if it is successful, error code otherwise

Classifying faces (Gender, Age, Ethnicity, Smile, Glasses or facial hair detection)

Betaface_AnalyseFace *(BetafaceInternalState State, BetafaceImage Img, BetafaceFaceInfo FaceInfo, BetafaceAnalyseFlags flags)*

This function analyzes the face and tries to classify it. Results are added to FaceInfo and can be retrieved via Betaface_GetFaceInfoDoubleParam with classifier flag combined with Value or Score flag. See BetafaceSampleCSharpApp sample project code.

Parameters:

State	*Betaface library internal state value, obtained from Betaface_Init function*
Img	*Image containing face to analyze and to which FaceInfo corresponds*
FaceInfo	*BetafaceFaceInfo variable where valid face information data in internal Betaface representation is stored*
flags	*Flag defining which classifiers to run, depending on SDK edition can be one of:* *BETAFACE_CLASSIFIER_ALL = 0xFFFFFFFF*

```
BETAFACE_CLASSIFIER_GENDER = 0x60010000
BETAFACE_CLASSIFIER_AGE = 0x60020000
BETAFACE_CLASSIFIER_ETHNICITY = 0x60040000
BETAFACE_CLASSIFIER_EXPRESSION = 0x60080000
BETAFACE_CLASSIFIER_GLASSES = 0x60100000
BETAFACE_CLASSIFIER_MUSTACHE = 0x60200000
BETAFACE_CLASSIFIER_BEARD = 0x60400000
```

Return value:

Function returns BETAFACE_OK if it is successful, error code otherwise

Cropping and drawing faces

Betaface_CropFaceImage *(BetafaceInternalState State, BetafaceImage Img, BetafaceFaceInfo FaceInfo, double dEyesDistance, double dEyeLineHeight, bool bDeRotate, int iCropWidth, int iCropHeight, double dAreaScale, int colorBackground, BetafaceImage* pCroppedFaceImg, BetafaceFaceInfo* pCroppedFaceInfo)*

This function is for cropping and de-rotating face from the source image and then fitting it to specified output image dimensions. Use this function to crop faces as aligned portrait images.

Parameters:

State	*Betaface library internal state value, obtained from Betaface_Init function*
Img	*Image containing face to crop and to which FaceInfo corresponds*
FaceInfo	*BetafaceFaceInfo variable where valid face information data in internal Betaface representation is stored*
dEyesDistance	*The distance between the eyes on the output face rectangle, as a fraction of the whole out image width, for example 0.3 means that the distance between the eyes on the output face rectangle will be 30% of the cropped image width*
dEyeLineHeight	*The distance between the top of the output face rectangle and the line between the eyes, as a fraction of the whole out image height, for example 0.4 means that the distance between the top of the output face rectangle and the line between the eyes will be 40% of the cropped image height*
bDeRotate	*Specify true if you want to de-rotate face into strict vertical portrait position and fit it into the output image dimensions. If this parameter is set to false face will be cropped under the angle it appears on the source image. It is usually set to true*
iCropWidth	*Output image width in pixels*
iCropHeight	*Output image height in pixels*
dAreaScale	*Additional scale coefficient applied to the face rectangle before the area to crop is determined. If this coefficient is >1.0 then the cropped rectangle is scaled by this factor or until the borders of the whole image are reached; if the face rectangle was already out of the image borders then this coefficient will be fixed to 1.0. If the specified coefficient is <1.0 then no checks are performed and scaling factor is applied directly.*
colorBackground	*Background color to fill in cropped image areas that fall outside of original image borders. Color value is in RGB format, represented as integer 0x00RRGGBB Commonly used color values:* *BETAFACE_COLOR_BLACK = 0x00000000* *BETAFACE_COLOR_WHITE = 0x00FFFFFF* *BETAFACE_COLOR_RED = 0x00FF0000* *BETAFACE_COLOR_GREEN = 0x0000FF00* *BETAFACE_COLOR_BLUE = 0x000000FF*
pCroppedFaceImg	*Pointer to the BetafaceImage variable where the cropped image in internal Betaface representation will be returned*
pCroppedFaceInfo	*Pointer to the BetafaceFaceInfo variable where the face information data, converted to the cropped image coordinate system will be returned*

Return value:

Function returns BETAFACE_OK if it is successful, error code otherwise

Betaface_CropImageAspect *(BetafaceInternalState State, BetafaceImage Img, BetafaceFaceInfo FaceInfo, double dEyesDistance, double dEyeLineHeight, double dFaceAspectHW, double dImageAspectHW, int colorBackground, BetafaceImage* pCroppedFaceImg, BetafaceFaceInfo* pCroppedFaceInfo)*

This function crops face image and all possible surrounding area keeping the specified target image aspect ratio. First face rectangle is calculated using dEyesDistance, dEyeLineHeight, dFaceAspectHW parameters, then the maximum fit bounding rectangle.

Parameters:

State	*Betaface library internal state value, obtained from Betaface_Init function*
Img	*Image containing face to crop and to which FaceInfo corresponds*
FaceInfo	*BetafaceFaceInfo variable where valid face information data in internal Betaface representation is stored.*
dEyesDistance	*The distance between the eyes on the face rectangle, as a fraction of the whole face rectangle width, for example 0.3 mean that the distance between the eyes on the output face rectangle will be 30% of the whole face rectangle width*
dEyeLineHeight	*The distance between the top of the output face rectangle and the line between the eyes, as a fraction of the whole face rectangle height, for example 0.4 mean that the distance between the top of the output face rectangle and the line between the eyes will be 40% of the whole face rectangle height*
dFaceAspectHW	*Aspect ratio of the face rectangle*
dImageAspectHW	*Aspect ratio of the final image area to cut*
colorBackground	*Background color to fill in cropped image areas that fall outside of original image borders. Color value is in RGB format, represented as integer 0x00RRGGBB. Commonly used color values:* *BETAFACE_COLOR_BLACK = 0x00000000* *BETAFACE_COLOR_WHITE = 0x00FFFFFF* *BETAFACE_COLOR_RED = 0x00FF0000* *BETAFACE_COLOR_GREEN = 0x0000FF00* *BETAFACE_COLOR_BLUE = 0x000000FF*
pCroppedFaceImg	*Pointer to the BetafaceImage variable where the cropped image in internal Betaface representation will be returned*
pCroppedFaceInfo	*Pointer to the BetafaceFaceInfo variable where the face information data, converted to the cropped image coordinate system will be returned*

Return value:

Function returns BETAFACE_OK if it is successful, error code otherwise

Betaface_DrawFaceInfo *(BetafaceInternalState State, BetafaceImage Img, BetafaceFaceInfo FaceInfo, BetafaceDrawFaceFlags flags)*

This function is used for debugging and visualization purposes and it draws rectangles and face feature points stored in the face information data on the corresponding image

Parameters:

State	*Betaface library internal state value, obtained from Betaface_Init function*
Img	*Image containing face to crop and to which FaceInfo corresponds*
FaceInfo	*BetafaceFaceInfo variable where valid face information data in internal Betaface representation is stored*
flags	*Flags that can be combined using bitwise OR operation, specifying what (rectangles, points) to draw:* *BETAFACE_DRAWFACE_FACERECT = 0x00000001* *BETAFACE_DRAWFACE_EYECROSSES = 0x00000002* *BETAFACE_DRAWFACE_FEATURES = 0x00000004* *BETAFACE_DRAWFACE_FEATURES_PRO = 0x00010000* *BETAFACE_DRAWFACE_CONTOURS_PRO = 0x00020000* *BETAFACE_DRAWFACE_ALL_BASIC = 0x0000FFFF* *BETAFACE_DRAWFACE_ALL_PRO = 0xFFFF0000* *BETAFACE_DRAWFACE_ALL = 0xFFFFFFFF*

Return value:

Function returns BETAFACE_OK if it is successful, error code otherwise

Face effects - morphing, warping, transporting to another image

Betaface_MorphFaces *(BetafaceInternalState State, BetafaceImage SrcFaceImg, BetafaceFaceInfo SrcFaceInfo, BetafaceImage DstFaceImg, BetafaceFaceInfo DstFaceInfo, double dTransitionKoeff, BetafaceImage* pMorphedImg)*

This function morphs or warps face image, using feature points contained in the face info data as an anchor points of morphing. Morph, is a transformation effect, when one face (source) shape and texture smoothly transforms into another face (destination) shape and texture. Morphing effect can be used to mix two faces together and create a face containing facial features of both source and destination faces in a proportion defined by transition coefficient, or to generate set of video frames showing transformation process (coeff. 0.0 – 1.0). Warp, is a geometrical distortion of one face (source) into a new shape, which can be done either in one step (coeff 1.0), or in number of video frames, showing smooth transformation process.

Parameters:

State	*Betaface library internal state value, obtained from Betaface_Init function*
SrcFaceImg	*Image containing source face and to which SrcFaceInfo corresponds*
SrcFaceInfo	*BetafaceFaceInfo variable where valid face information of the source face data in internal Betaface representation is stored*
DstFaceImg	*Image containing destination face and to which DstFaceInfo corresponds. This parameter can be NULL, in which case function performs a face Warp, using DstFaceInfo as a target face shape for geometrical distortion*
DstFaceInfo	*BetafaceFaceInfo variable where valid face information of the destination face data in internal Betaface representation is stored*
dTransitionKoeff	*Transition coefficient ranges from 0.0 (100% source face) to 0.5 (50% of each face) to 1.0 (100% destination face) to determine morphing stage*
pMorphedImg	*Pointer to the BetafaceImage variable where the resulting morphed image ininternal Betaface representation will be returned*

Return value:

Function returns BETAFACE_OK if it is successful, error code otherwise

Betaface_TransportFace *(BetafaceInternalState State, BetafaceImage SrcFaceImg, BetafaceFaceInfo SrcFaceInfo, BetafaceImage DstTemplateImg, BetafaceImage DstTemplateImgAlpha, double dLx, double dLy, double dRx, double dRy, int iContrast, int iBrightness, int iSkinFilter, BetafaceImage* pMorphedImg, BetafaceFaceInfo* pMorphedFaceInfo)*

This function can be used to extract face region from the source image and insert it in the destination image in the specified location with transparency mask and contrast/brightness adjustments. Function align source image according to eye coordinates of the face and targets eye coordinates in destination image, then blends two images using specified transparency mask.

Parameters:

State	*Betaface library internal state value, obtained from Betaface_Init function*

SrcFaceImg	*Image containing source face and to which SrcFaceInfo corresponds*
SrcFaceInfo	*BetafaceFaceInfo variable where valid face information of the source face data in internal Betaface representation is stored*
DstTemplateImg	*Destination image*
DstTemplateImgAlpha	*Destination image transparency mask, Pixels = 0 are not transparent (only destination image pixels will be visible), Pixels = 255 – transparent, Values between 0 and 255 define mix proportion between source and destination image.*
dLx	*Target left eye X position on the destination image, in pixels*
dLy	*Target left eye Y position on the destination image, in pixels*
dRx	*Target right eye X position on the destination image, in pixels*
dRy	*Target right eye Y position on the destination image, in pixels*
iContrast	*Contrast adjustment 0-200, 100 is middle value=no change*
iBrightness	*Brightness adjustment 0-200, 100 is middle value=no change*
iSkinFilter	*Skin filter – specify any value greater than 0 to apply transparency to the pixels close to detected face skin color*
pMorphedImg	*Pointer to the BetafaceImage variable where the resulting morphed image in internal Betaface representation will be returned*
pMorphedFaceInfo	*Pointer to the BetafaceFaceInfo variable where valid face information of the face data in destination image coordinates will be returned*

Return value:

Function returns BETAFACE_OK if it is successful, error code otherwise

Betaface_TransformFaceInfo *(BetafaceInternalState State, BetafaceImage Img, BetafaceFaceInfo FaceInfo, int iTransform, double dValue, BetafaceFaceInfo* pFaceInfo)*

This function applies different automatic face shape transformations of the face points.

Parameters:

State	*Betaface library internal state value, obtained from Betaface_Init function*
Img	*Image containing source face to which SrcFaceInfo corresponds*
FaceInfo	*BetafaceFaceInfo variable where valid face information of the source face data in internal Betaface representation is stored*
iTransform	*Transformation type index*
dValue	*Transformation strength*
pFaceInfo	*Pointer to the BetafaceFaceInfo variable where modified face information will be returned*

Return value:

Function returns BETAFACE_OK if it is successful, error code otherwise

Average faces – functions to create face composites

You can create 'averaged' face image from the group of input face images. This is done by warping each face image into average face shape, accumulate those warped images as well as original face shapes and then warping accumulated face image into accumulated face shape. Following functions and Betaface_MorphFaces function is all you need for this task.

Betaface_GetStoredAverageFaceInfo *(BetafaceInternalState State, double dEyesDistance, double dEyeLineHeight, int iImageWidth, int iImageHeight, BetafaceFaceInfo* pAverageFaceInfo)*

This function returns global (static) average face shape, with scale and position determined from cropping parameters you supply. Cropping parameters are equal to those in Betaface_CropFaceImage function

Parameters:

State	*Betaface library internal state value, obtained from Betaface_Init function*
dEyesDistance	*The distance between the eyes on the output face rectangle, as a fraction of the whole out image width, for example 0.3 means that the distance between the eyes on the output face rectangle will be 30% of the cropped image width*
dEyeLineHeight	*The distance between the top of the output face rectangle and the line between the eyes, as a fraction of the whole out image height, for example 0.4 means that the distance between the top of the output face rectangle and the line between the eyes will be 40% of the cropped image height*
iImageWidth	*Destination image width*
iImageHeight	*Destination image height*
pAverageFaceInfo	*Pointer to the BetafaceFaceInfo variable where the global average face information data, converted to the cropped image coordinate system will be returned*

Return value:

Function returns BETAFACE_OK if it is successful, error code otherwise

Betaface_UpdateAverageImage *(BetafaceInternalState State, BetafaceImage OldAvgImg, int OldAvgCount, bool bAppend, BetafaceImage Img, int AvgCount, BetafaceImage* pAverageImg)*

This function appends or subtracts single prepared face texture image into/from accumulated average face texture image.

Parameters:

State	*Betaface library internal state value, obtained from Betaface_Init function*
OldAvgImg	*Image containing accumulated average face texture, warped to a global face average shape obtained in Betaface_GetStoredAverageFaceInfo function*
OldAvgCount	*Number of faces accumulated in average face texture image OldAvgImg*
bAppend	*Type of update operation, set it to true to add texture image into accumulated image or false to subtract from it*
Img	*Image containing new single face texture to append into accumulated averageface texture*

AvgCount	*Number of faces already accumulated in image Img. If Img is not accumulated texture image i.e. contain only one face, set this parameter to 1*
pAverageImg	*Pointer to the BetafaceImage variable where the accumulated average face texture image in internal Betaface representation will be returned*

Return value:

Function returns BETAFACE_OK if it is successful, error code otherwise

Betaface_UpdateAverageFaceInfo *(BetafaceInternalState State, BetafaceFaceInfo OldAvgFaceInfo, int OldAvgCount, bool bAppend, BetafaceFaceInfo FaceInfo, int AvgCount, BetafaceFaceInfo* pAverageFaceInfo)*

This function appends or subtracts single prepared face shape into/from accumulated average face shape information structure

Parameters:

State	*Betaface library internal state value, obtained from Betaface_Init function*
OldAvgFaceInfo	*BetafaceFaceInfo variable where accumulated average face shape information, in internal Betaface representation is stored*
OldAvgCount	*Number of face shapes accumulated in average face shape information OldAvgFaceInfo*
bAppend	*Type of update operation, set it to true to add face shape information into accumulated face shape information or false to subtract from it*
FaceInfo	*BetafaceFaceInfo variable containing new single face shape information to append into accumulated average face shape information*
AvgCount	*Number of face shapes already accumulated in face shape information FaceInfo. If FaceInfo is not accumulated face shape i.e. contain only one face, set this parameter to 1*
pAverageFaceInfo	*Pointer to the BetafaceFaceInfo variable where the accumulated average face shape information in internal Betaface representation will be returned*

Return value:

Function returns BETAFACE_OK if it is successful, error code otherwise

Video Processing (SDK Xtreme)

Introduction

Betaface SDK video processing engine can analyze video files or video streams from cameras via VFW interface and support custom capturing via callbacks. Video processing chain split can be divided into two logical parts:

– capturing process, which retrieves video frames from the camera or video file, stores them in runtime buffer for processing and releases them, when they are no longer required.

- analyzing or tracking process, which analyzes captured data, detects and tracks faces + facial features and recognizes persons.

Betaface SDK does capturing and tracking threads synchronization internally. Your application should be aware that whole video processing chain is a multithreaded process, i.e callbacks can be called at any time from different threads.

When integrating into your application, typical sequence is first to initialize Betaface SDK (Betaface_Init), initialize capturing, and connect callback functions and then start/stop capturing and tracking processes using StartDetectFaces/StopDetectFaces functions. When exiting application first de-initialize video subsystem by calling Betaface_ReleaseVideo, and then de-initialize Betaface SDK by calling Betaface_Deinit function.

General and very important rule of integration is that no callback at any time can be delayed on your side in order not to disrupt tracking/capturing process. Do not call IO functions, wait for window messages, access GUI or use any kind of synchronized calls anywhere in handler function. Do not release data supplied to you in callback (video frames, parameters) or store any pointers. Maximum you can do inside your callback handler is to copy, using corresponding copy functions, the data (video frame, parameters) you are interested in somewhere in your local storage and immediately return. Don't forget callback can be called at once from multiple threads; therefore make sure you use locks to synchronize writing operations in your local storage. Make sure that your local storage can't be locked for significant time from other place. Refer to or use as a starting point supplied sample application(s).

Betaface .Net interface assembly and advanced sample video project FDCamStream offers extended capturing interface based on DirectShow.

Initialize video capturing from uncompressed video file or VFW source

Betaface_LoadVideo *(BetafaceInternalState State, char* strVideoFilename, BetafaceVideo* pVideo)*

This function initializes offline capturing from video file and returns internal Betaface video capturing runtime state

Parameters:

State	*Betaface library internal state value, obtained from Betaface_Init function*
strVideoFilename	*Full pathname to the video file on the HDD. Video will be decoded using standard Windows VFW interfaces, meaning only few video codecs and uncompressed frame formats are supported*
pVideo	*Pointer to BetafaceVideo variable, by this address video capturing runtime state in internal Betaface format will be returned*

Return value:

Function returns BETAFACE_OK if it is successful, error code otherwise

Betaface_CaptureVideo *(BetafaceInternalState State, int iCameraIdx, int iWidth, int iHeight, BetafaceVideo* pVideo)*

This function initializes live capturing from camera and return internal Betaface video capturing runtime state

Parameters:

State	*Betaface library internal state value, obtained from Betaface_Init function*
iCameraIdx	*Index of the camera to select from VFW interface*
iWidth	*X size of the video frame (should be supported by the camera)*
iHeight	*Y size of the video frame (should be supported by the camera)*
pVideo	*Pointer to BetafaceVideo variable, by this address video capturing runtime state in internal Betaface format will be returned*

Return value:

Function returns BETAFACE_OK if it is successful, error code otherwise

Initialize video capturing from custom external video source

Betaface_CaptureVideoCb *(BetafaceInternalState State, CB_GrabNextFrame grabFunc, __int64 pArguments, bool bOnline, BetafaceVideo* pVideo)*

This function initializes offline capturing from video file and returns internal Betaface video capturing runtime state

Parameters:

State	*Betaface library internal state value, obtained from Betaface_Init function*
grabFunc	*Pointer to frame grabbing callback handler function*
pArguments	*Pointer that will be passed back to you in every callback call*
bOnline	*TRUE when it is a live camera stream, FALSE when it is offline capture from video file. This parameter will affect tracking procedure and whether a tracker will or will not drop frames it cannot process in time.*
pVideo	*Pointer to BetafaceVideo variable, by this address video capturing runtime state in internal Betaface format will be returned*

Return value:

Function returns BETAFACE_OK if it is successful, error code otherwise

Callback CB_GrabNextFrame *(BetafaceVideo Video, int iFrameIdx, BetafaceImage* pFrame, double* pdFrameTime, __int64 pArguments)*

This is a callback handler function prototype that will be called each time new frame could be read from the stream. You can block this call if there are no new frames.

Parameters:

Video	*BetafaceVideo variable containing video capturing runtime state in internal Betaface format*
iFrameIdx	*Frame counter, starting from 0. Will increase with each callback call.*
pFrame	*Pointer where to write BetafaceImage with frame data*
pdFrameTime	*Pointer where to write frame relative time in seconds*
pArguments	*Your pointer supplied in Betaface_CaptureVideoCb function*

Return value:

Function returns BETAFACE_OK if it is successful, error code otherwise

De-initializing capturing and releasing allocated resources

Betaface_ReleaseVideo *(BetafaceInternalState State, BetafaceVideo* pVideo)*

This function stops capturing process and release all internal resources related to it

Parameters:

State	*Betaface library internal state value, obtained from Betaface_Init function*
pVideo	*Pointer to variable containing video capturing runtime state in internal Betaface format*

Return value:

Function returns BETAFACE_OK if it is successful, error code otherwise

Set capturing process callbacks

Betaface_SetFrameGrabbedCallback *(BetafaceInternalState State, BetafaceVideo Video, CB_OnFrameGrabbed callbackFunc, __int64 pArguments)*

This function set callback called after for each grabbed video frame just before it is delivered to face tracker.

Parameters:

State	*Betaface library internal state value, obtained from Betaface_Init function*
Video	*Video capturing runtime state in internal Betaface format*
callbackFunc	*Pointer to frame grabbed callback handler function*
pArguments	*Pointer that is passed back to you in every callback call*

Return value:

Function returns BETAFACE_OK if it is successful, error code otherwise

Callback CB_OnFrameGrabbed *(BetafaceVideo Video, BetafaceImage Frame, int iFrameIdx, double dFrameTime, __int64 pArguments);*

This is a callback handler function prototype that is called each time new frame is successfully grabbed from the stream

Parameters:

Video	*BetafaceVideo variable containing video capturing runtime state in internal Betaface format*
Frame	*BetafaceImage variable containing frame pixel data in internal Betaface format*
iFrameIdx	*Captured frame relative index*
dFrameTime	*Captured frame relative time in seconds*
pArguments	*Your pointer supplied in Betaface_SetFrameGrabbedCallback function*

Return value:

Return BETAFACE_OK

Betaface_SetFrameReleasedCallback *(BetafaceInternalState State, BetafaceVideo Video, CB_OnFrameReleased callbackFunc, __int64 pArguments)*

This function set callback called after each successfully grabbed frame

Parameters:

State	*Betaface library internal state value, obtained from Betaface_Init function*
Video	*Video capturing runtime state in internal Betaface format*
callbackFunc	*Pointer to frame released callback handler function*
pArguments	*Pointer that is passed back to you in every callback call*

Return value:

Function returns BETAFACE_OK if it is successful, error code otherwise

Callback CB_OnFrameReleased *(BetafaceVideo Video, BetafaceImage Frame, int iFrameIdx, double dFrameTime, __int64 pArguments);*

This is a callback handler function prototype that is called each time just before old frame is released from capturing buffer (i.e. fully processed)

Parameters:

Video	*Video capturing runtime state in internal Betaface format*
Frame	*BetafaceImage variable containing frame pixel data in internal Betaface format*
iFrameIdx	*Captured relative frame index*
dFrameTime	*Captured relative frame time in secnds*
pArguments	*Your pointer supplied in Betaface_SetFrameReleasedCallback function*

Return value:

Return BETAFACE_OK

Set tracking process callbacks

Betaface_SetFaceInfoUpdatedCallback *(BetafaceInternalState State, BetafaceVideo Video, CB_OnFaceInfoUpdated callbackFunc, __int64 pArguments)*

This function sets callback called after for each tracked face appearance update.

Parameters:

State	*Betaface library internal state value, obtained from Betaface_Init function*
Video	*Video capturing runtime state in internal Betaface format*
callbackFunc	*Pointer to tracking information callback handler function*
pArguments	*Pointer that is passed back to you in every callback call*

Return value:

Function returns BETAFACE_OK if it is successful, error code otherwise

Callback CB_OnFaceInfoUpdated *(BetafaceVideo Video, int iFaceInfoID, int iUpdateFromFaceInfoID, int iFrameIdx, double dFrameTime, double dStartTime, double dDurationTime, int iUpdateType, bool isTrackingChange, BetafaceImage Frame, BetafaceFaceInfo FaceInfo, __int64 pArguments)*

Description

Parameters:

Video	*Video capturing runtime state in internal Betaface format*
iFaceInfoID	*Face appearance index*
iUpdateFromFaceInfoID	*For updates of the type LenUpdate indicates which appearance was merged into current appearance. You can use this to append your data associated with merged appearance (recognition keys for example) into iFaceInfoID appearance.*
iFrameIdx	*Captured frame index*
dFrameTime	*Frame relative time in seconds*
dStartTime	*Face appearance start time*
dDurationTime	*Face appearance duration*
iUpdateType	*Type of information updated, can be* *FACEINFOUPDATE_INITIAL = 0 - initial face detection* *FACEINFOUPDATE_FORWARD = 1 - forward tracking* *FACEINFOUPDATE_BACKWARD=2 - backward tracking* *FACEINFOUPDATE_LENUPDATE=3 - total duration has changed* *FACEINFOUPDATE_DELETE=4 face appearance is discarded* *FACEINFOUPDATE_FINISHED=5 face appearance is finalized*
isTrackingChange	*True if this update is a result of the face tracking process*

Frame	*Current frame, when provided can be used together with FaceInfo to generate recognition keys, analyze faces. Only provided for Initial and Lenupdate updates and non-tracking updates (i.e. full scans).*
FaceInfo	*Face info corresponding to the current frame*
Face	*Cropped face image of this appearance (currently NULL)*
pArguments	*Your pointer supplied in Betaface_SetFaceInfoUpdatedCallback function*

Return value:

Return BETAFACE_OK

Starting/stopping both capturing and tracking processes

Betaface_StartDetectFaces *(BetafaceInternalState State, BetafaceVideo Video, BetafaceTrackSettings tSettings, BetafaceDetectionSettings dSettings)*

This function starts capturing and tracking processes with specified parameters. Tracker will also try to match each face appearance to the current persons watch list and, when enabled, will also try to learn new persons in the video stream that do not match anyone in current watch list. Each person entry may have multiple face appearances assigned to it.

Parameters:

State	*Betaface library internal state value, obtained from Betaface_Init function*
Video	*BetafaceVideo variable containing video capturing runtime state in internal Betaface format*
tSettings. dDetectionTickSeconds	*Detection tick time in seconds, how often stream will be scanned for the new faces*
tSettings. dMinExposureSeconds	*Minimum amount of seconds face should be tracked not to be rejected as too short appearance*
dSettings	*Face detection settings. Please refer to Betaface_DetectFaces function documentation for description of those parameters. Two additional face detections flags can be set here:* *BETAFACE_DETECTFACES_TRACK_BASIC = 0x0* *BETAFACE_DETECTFACES_TRACK_PRO=0x9* *This suggests tracker which points to use to track faces – 86 Pro or only 8 basic.*

Return value:

Function returns BETAFACE_OK if it is successful, error code otherwise

Betaface_StopDetectFaces *(BetafaceInternalState State, BetafaceVideo Video)*

This function stops capturing process immediately

Parameters:

State	*Betaface library internal state value, obtained from Betaface_Init function*
Video	*Video capturing runtime state in internal Betaface format*

Return value:

Function returns BETAFACE_OK if it is successful, error code otherwise

Betaface_WaitDetectFaces *(BetafaceInternalState State, BetafaceVideo Video, double dSeconds)*

This function stops capturing process after specified time

Parameters:

State	*Betaface library internal state value, obtained from Betaface_Init function*
Video	*Video capturing runtime state in internal Betaface format*
dSeconds	*Number of seconds to wait before stopping capturing/tracking process*

Return value:

Function returns BETAFACE_OK if it is successful, error code otherwise

Monitoring tracking progress and display debug information

Betaface_GetCurrentDetectionState *(BetafaceInternalState State, BetafaceVideo Video, int* piFacesCount, BetafaceDetectionResult* pDetectionResult)*

This function returns a snapshot of current capturing state and detected facial points

Parameters:

State	*Betaface library internal state value, obtained from Betaface_Init function*
Video	*Video capturing runtime state in internal Betaface format*
piFacesCount	*Pointer where number of faces detected is returned*
pDetectionResult	*Pointer where snapshot of detection result should be written*

Return value:

Function returns BETAFACE_OK if it is successful, error code otherwise

Betaface_DrawVideoFrame *(BetafaceInternalState State, BetafaceImage Img, BetafaceVideo Video, int iFrameIdx, int iFaceID, BetafaceDrawVideoFrameFlags flags)*

This function draws debug information on a video frame. Use Betaface_DisplayImage to display the frame after drawing Betaface_SaveImage to save it in an image file.

Parameters:

State	*Betaface library internal state value, obtained from Betaface_Init function*
Img	*Video frame in internal Betaface format*
iFrameIdx	*Index of video frame for which to draw available information*
iFaceID	*Index of the face for which to draw available information*
flags	*Flags, specifying what to draw – faces, points, IDs, in white or black and should tracker additional info be displayed:* *BETAFACE_DRAWVIDEOFRAME_FACE = 0x1* *BETAFACE_DRAWVIDEOFRAME_POINTS = 0x2* *BETAFACE_DRAWVIDEOFRAME_IDENTITY = 0x4* *BETAFACE_DRAWVIDEOFRAME_CENTER = 0x8* *BETAFACE_DRAWVIDEOFRAME_ALL = 0x0000FFFF* *BETAFACE_DRAWVIDEOFRAME_INWHITE =0x00010000* *BETAFACE_DRAWVIDEOFRAME_FILTER_SINGLES=0x00100000* *BETAFACE_DRAWVIDEOFRAME_FILTER_TAILS=0x00200000*

Return value:

Function returns BETAFACE_OK if it is successful, error code otherwise