Betaface SDK manual

VERSION 3.2.3

WWW.BETAFACE.COM

Introduction

Betaface Face Detection and Recognition SDK is a x64 Windows DLL library or Linux SO library containing set of the algorithms trained and tuned to detect and compare human face patterns and facial features on images and in video streams.
SDK detects frontal and partially profile faces (with some tolerance to the maximum head rotation) under any in-plane angle rotation and returns coordinates of faces found and all detected face features. Detected face information can be converted to facial recognition key which can be compared with other keys for the purposes of similarity measurements, matching face search in database (identification) or verification tasks. Additionally SDK provides faces classification (age, gender, ethnicity, emotion) as well as extended geometric and color measurements giving you information about hairstyle, skin/eyes color, head or facial features shape and clothes color.
SDK can be accessed from different environments, as long as you can make a call to DLL/SO functions or .Net assembly. Standard C and .Net interfaces together with code samples are included in SDK package, however you can call library from other environments, such as Java with mapped functions declarations, SDK core library uses only primitive parameter types.
SDK features summary:
- Detect multiple faces on images or in video streams, returning position, size and duration of tracked face appearance (video).
- Most image formats are supported including recognition of EXIF rotation flags. Uncompressed images (pixel buffers) can be loaded directly from memory.
- Built in video capture engine works with most local cameras, IP streaming cameras and video files. Capture engine allows to connect external video capture via callback function, which should provide raw video frame with timestamp (in seconds).
- Crop faces from the images based on detection information.
- Compare faces (similarity score, identification (1:N), and verification (1:1)).
- Support of very large RAM-cached face indexes for quick search in large face collections.
- Detect and track (on video) face landmarks coordinates (22 basic robust points and 101 advanced points).
- Classification, such as gender, age, ethnicity estimation; smile, glasses, mustache and beard detection.
- Morph, warp or generate average face images.
- Extended face measurements such as face and face features shape description; hairstyle shape estimation; skin, hair, facial hair, eyes, clothes color detection.
- 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 supports both CPU only and GPU (Nvidia CUDA 8.0, CUDNN 6.0) accelerated execution. For GPU acceleration you need to have compatible video card(s), Maxwell, Pascal or higher architecture from Nvidia, such as Nvidia GTX 1080 or Titan X.
SDK package includes all necessary runtime files, this documentation and precompiled samples of usage with source code in C++ (Windows, Linux) and C#.Net (Windows). SDK includes software license protection system. Our typical licensing option is a hardware-locked software license key file without the need of internet connection. Other options (USB Dongle, network license) may be provided on a per project basis depending on volume.
Betaface can offer additional components, ready or customized solutions on request, such as complete API/Server solution including distributed processing service and database, such as http://www.betafaceapi.com.

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 further SDK function call. It is required to initialize separate instance for each execution thread. At the application exit, or to save RAM allocated for instance when face recognition is not needed de-initialization function should be called.

Betaface_Init (BetafaceInitFlags initFlags, int DeviceID, BetafaceInternalState* pState);
Call this functions to initialize the library and retrieve internal state value.
Parameters:
initFlags – initialization flags specifying which versions of algorithms to prepare and which engine (CPU or GPU) to use for each.
Use BETAFACE_INIT_DEFAULT_CPU or BETAFACE_INIT_DEFAULT_GPU to enable all default selection of algorithms for CPU or GPU. Notice by default BETAFACE_INIT_DEFAULT_CPU will use older generation CPU based face detector as new face detector might be too slow on CPU for some applications.
You can specify exactly which algorithms you need by combining flags below, for each algorithm you want to enable select only CPU or GPU flag:

Select one or none for image content detection
BETAFACE_INIT_CONTENT_NSFW_CPU
BETAFACE_INIT_CONTENT_NSFW_CPU

Select one or none of detectors:
New face detector: BETAFACE_INIT_DETECT_CPU or BETAFACE_INIT_DETECT_GPU
Previous gen CPU-only fast face detector: BETAFACE_INIT_DETECT_LEGACY_CPU

Select Basic, Pro or no points detection (require face detector):
BETAFACE_INIT_POINTS_BASIC_CPU, BETAFACE_INIT_POINTS_BASIC_GPU
BETAFACE_INIT_POINTS_PRO_CPU, BETAFACE_INIT_POINTS_PRO_GPU

Face classifiers:
BETAFACE_INIT_CLASSIFIERS_CPU, BETAFACE_INIT_CLASSIFIERS_GPU

Recognition key generator:

BETAFACE_INIT_RECKEY_DEEPIDGRAY_CPU, BETAFACE_INIT_RECKEY_DEEPIDGRAY_GPU – CPU/GPU recognition algorithm first introduced in SDK 2.1, 0.9982 accuracy on LFW.

BETAFACE_INIT_RECKEY_DEEPIDGRAY_FAST_CPU, BETAFACE_INIT_RECKEY_DEEPIDGRAY_FAST_GPU – 2x faster version of previous algorithm with small accuracy tradeoff. Keys are compatible with DEEPIDGRAY.
BETAFACE_INIT_RECKEY_DEEPIDGRAY3_CPU, BETAFACE_INIT_RECKEY_DEEPIDGRAY3_GPU – Our latest recognition algorithm introduced in SDK 3.2, 0.9953 accuracy on LFW. Can be run both on CPU and GPU. Uses more memory than DEEPIDGRAY algorithm and slightly slower, especially on CPU.

DeviceID - When using GPU accelerated SDK version DeviceID correspond to the GPU card index.
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 clone 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 named 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

Betaface_RemoveDisplay(BetafaceInternalState State, char* strWindowName);

This function destroy the window with specific name previously created by Betaface_DisplayImage
Parameters:
State - Betaface library internal state value, obtained from Betaface_Init function
strWindowName - Window caption name
Return value:

Function returns BETAFACE_OK if it is successful, error code otherwise

Faces 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 – Enable basic points detection
BETAFACE_DETECTFACES_NOFEATURES = 0x10 – Disable basic points detection, points are required for recognition key
BETAFACE_DETECTFACES_PRO_POINTS = 0x1 – Enable 101 pro points detection
BETAFACE_DETECTFACES_BASIC_PROFILE = 0x4 – Enable profile (non-frontal) faces detection
BETAFACE_DETECTFACES_BASIC_NOFRONTAL = 0x16 – Disable frontal detector (when using BETAFACE_INIT_DETECT_LEGACY_CPU only)
BETAFACE_DETECTFACES_BESTFACEONLY = 0x100 – If this flag is specified, only single face with highest detection score is processed and returned.
dSettings .iMaxImageWidthPix
dSettings .iMaxImageHeightPix
Downscale image to certain max resolution before processing to increase speed. 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 800 or 640 for iMaxImageWidthPix and 600 or 480 for iMaxImageHeightPix. If one or both values are 0 no downscaling is performed, however this option is not recommended and detection can be very slow on hi-res images.
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 to remove limit. Typical value for group photos is 0.05 and 0.3 for portrait photos.
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, in this case faces as small as 25x25 pixels can be returned. Recommended minimal face size for face recognition is 100x100 pixels.
dSettings .dAngleDegrees
Central angle of the faces to detect. If dAngleToleranceDegrees parameter is 0 or greater/equal to 180 degrees this parameter is ignored. Use it if you expect certain rotation of the images. Rotation flags written by cameras in EXIF are automatically recognized.
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. For CPU face detector

(BETAFACE_DETECTFACES_BASIC_LEGACY) angle limit increases detection speed. For new face detector speed is the same regardless of angle range but faces outside angle limits are not returned.
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 0.5-1.0, 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:

Feature BETAFACE_FEATURE_FACE or BETAFACE_FEATURE_FACE_RAW (non-aligned face bounding box) can be combined with following parameters to retrieve basic face properties. Pitch/Yaw are available for new CPU/GPU face detector only.
BETAFACE_PARAM_SCORE = 0x0001 – detection score
BETAFACE_PARAM_X = 0x0002
BETAFACE_PARAM_Y = 0x0003
BETAFACE_PARAM_PROFILE = 0x0008
BETAFACE_PARAM_WIDTH = 0x0004
BETAFACE_PARAM_HEIGHT = 0x0005
BETAFACE_PARAM_ANGLE = 0x0006
BETAFACE_PARAM_YAW = 0x0009
BETAFACE_PARAM_PITCH = 0x000A
Point features (BETAFACE_FEATURE_*, BETAFACE_FEATURE_PRO_*) can be combined with X,Y parameters to get coordinates.
BETAFACE_PARAM_X = 0x0002
BETAFACE_PARAM_Y = 0x0003
Measurement features can be combined with following parameters to get range and value of measurement
BETAFACE_PARAM_VALUE = 0x0010
BETAFACE_PARAM_MIN = 0x0011
BETAFACE_PARAM_MAX = 0x0012
For complete list of extended 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 2 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, 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. After key creation image and face info is no longer needed. It is recommended first to crop face image with sufficient resolution and generate key from cropped face for consistency. It is also recommended to store cropped face image and cropped face info in the database together with a key, so you have an option to update face keys when new recognition algorithm is available.
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
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 is 0.005 for maximum accuracy. For verification tasks use 0.001 or lower.
pbIsSamePerson - Returned identification decision based on false alarm rate – 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

Face recognition in large face collections

When dealing with large faces collections, where you need to find closest matches for a given input face (1:N search) pairwise comparison of input face with every face in collection is not very efficient. SDK offers you a possibility to prepare your faces collection (collection of recognition keys) if form of an optimized face index, which then can be searched efficiently. Face index can be very large (> 1M of faces) and on each search call you will get back sorted list of best matching faces. Face index can store recognition keys generated by the same method only (see BetafaceRecognitionFlags flags parameter in Betaface_GenerateFaceKey function).

Betaface_SearchIndex_Init(int nFaces, BetafaceSearchState* pState, char* pFaceKeyData)
This function prepares face index structure and allocates required memory to store face index data.
Parameters:
nFaces – size of the face index (number of faces it will store).
pState – pointer to a BetafaceSearchState variable where allocated index value will be written.
pFaceKeyData – example recognition key, as returned by Betaface_GenerateFaceKey function. It will be used to determine recognition method, key size and to later verify that all recognition keys written to the face index have same size and generation method.
Function returns BETAFACE_OK if it is successful, error code otherwise
Betaface_SearchIndex_Deinit(BetafaceSearchState* pState)
This function releases previously prepared face index and frees the memory.
Parameters:
pState – pointer to a BetafaceSearchState variable which contains allocated face index state value.
Function returns BETAFACE_OK if it is successful, error code otherwise

Betaface_SearchIndex_SetKey(BetafaceSearchState State, int iFaceIdx, char* pFaceKeyData)
This function stores or overwrites given recognition key at specified iFaceIdx offset inside face index.
Parameters:

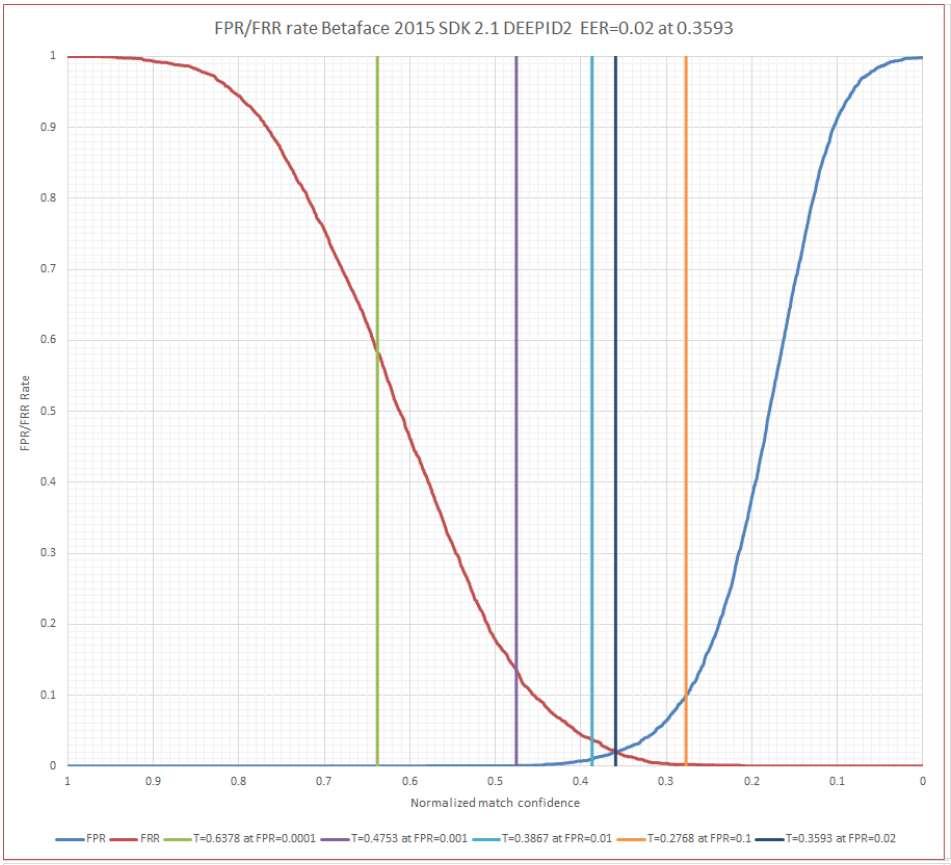
State –allocated face index state value as returned by Betaface_SearchIndex_Init function.
iFaceIdx – Offset at which write or overwrite recognition key inside face idex, range is from 0 (first position) to nFaces-1 (last face).
pFaceKeyData –recognition key, as returned by Betaface_GenerateFaceKey function.
Function returns BETAFACE_OK if it is successful, error code otherwise

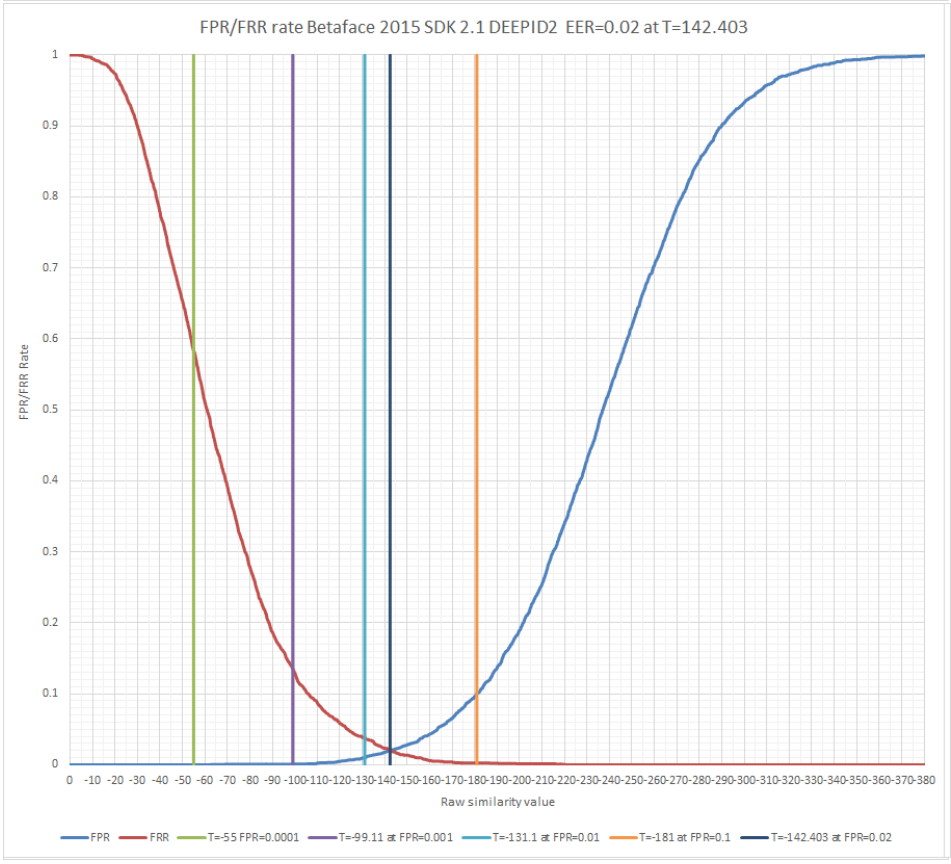
Betaface_SearchIndex_Search(BetafaceSearchState State, char* pSearchFilter, char* pFaceKeyData, int iStartIdx, int nFaces, int nMatches, int iRank, double dFalseAlarmRate, int* piIndexes, double* pdScores, int* pbIsMatches)

This function executes search inside face index and returns sorted list of best matching faces.
Parameters:
State – allocated face index state value as returned by Betaface_SearchIndex_Init function.
pSearchFilter – optional filter mask, a byte array of same size as face index (nFaces parameter in Betaface_SearchIndex_Init function). Can be NULL. If specified only faces with filter byte value >0 will participate in the search. This is typically used if you only want to search filtered set of faces, not complete index.
pFaceKeyData – input recognition key that will be used for comparison.
iStartIdx – starting face index for search, usually 0
nFaces – how many faces to compare starting from iStartIdx. iStartIdx + nFaces should not exceed face index size.
nMatches – how many best matches to return. Can be integer value from 1 up to total face index size.
iRank – currently 0
dFalseAlarmRate - Target false alarm rate, default is 0.005 for maximum accuracy. For verification tasks use 0.001 or lower. Corresponding to same parameter in Betaface_CompareFaceKeysEx function.
piIndexes, pdScores, pbIsMatches – separate (int, double, int) arrays that caller should allocate, all should have size of nMatches, where best matches corresponding face index, comparison score and decision same person or not, based on dFalseAlarmRate parameter, will be returned. Comparison score and match decision are the same as in Betaface_CompareFaceKeysEx function.
Function returns BETAFACE_OK if it is successful, error code otherwise

Following characteristics graphs can help understand tradeoff between false alarm and true positive rate for different similarity values returned by CompareFaceKeys and CompareFaceKeysEx functions:

DEEPIDGRAY2 (SDK 2.1+) algorithm:





DEEPIDGRAY3 (SDK 3.2+) algorithm

Image content detection

Betaface_AnalyseContent(BetafaceInternalState State, BetafaceImage Img, BetafaceAnalyseContentFlags flags, BetafaceImageInfo* pImageInfo)

This function can analyze image and detect its general content. Currently only adult content detection (NSFW) is available.

Parameters:
State - Betaface library internal state value, obtained from Betaface_Init function
Img - Image which content is to be analyzed
Flags - Flag defining which detectors to run, should be
BETAFACE_ANALYSE_CONTENT_NSFW = 0x00000010

Results are written to pImageInfo. Currently ImageInfo is just an alias for FaceInfo.
Return value:
Function returns BETAFACE_OK if it is successful, error code otherwise

You can retrieve detection results using same functions like for retrieving values from FaceInfo - Betaface_GetFaceInfoBoolParam and Betaface_GetFaceInfoDoubleParam.

Combine BETAFACE_CONTENT_NSFW feature flag with BETAFACE_PARAM_VALUE and BETAFACE_PARAM_SCORE parameter flags to check if content is detected and to get detection score. For NSFW detector values below 0.2 mean no adult content present with very high probability and values above 0.8 indicate adult content with high probability.

Classifying faces

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, can be one of:
BETAFACE_ANALYSE_ATTRIBUTES = 0x00000010
BETAFACE_ANALYSE_EXTENDED = 0x00000020

Results are written to FaceInfo. You have to use flags above combined with BETAFACE_PARAM_VALUE and BETAFACE_PARAM_SCORE to retrieve double value of classifier and confidence. Please check code samples for values interpretation.
Return value:
Function returns BETAFACE_OK if it is successful, error code otherwise

You can retrieve attributes parameters combining following features flags with BETAFACE_PARAM_VALUE and BETAFACE_PARAM_SCORE flags (value range from -1 to +1, score from 0 to 1 (0-100%)):
BETAFACE_ATTRIBUTE_5_O_CLOCK_SHADOW
BETAFACE_ATTRIBUTE_ARCHED_EYEBROWS
BETAFACE_ATTRIBUTE_ATTRACTIVE
BETAFACE_ATTRIBUTE_BAGS_UNDER_EYES
BETAFACE_ATTRIBUTE_BALD
BETAFACE_ATTRIBUTE_BANGS
BETAFACE_ATTRIBUTE_BIG_LIPS
BETAFACE_ATTRIBUTE_BIG_NOSE
BETAFACE_ATTRIBUTE_BLACK_HAIR
BETAFACE_ATTRIBUTE_BLOND_HAIR
BETAFACE_ATTRIBUTE_BLURRY
BETAFACE_ATTRIBUTE_BROWN_HAIR
BETAFACE_ATTRIBUTE_BUSHY_EYEBROWS
BETAFACE_ATTRIBUTE_CHUBBY
BETAFACE_ATTRIBUTE_DOUBLE_CHIN
BETAFACE_ATTRIBUTE_EYEGLASSES
BETAFACE_ATTRIBUTE_GOATEE
BETAFACE_ATTRIBUTE_GRAY_HAIR
BETAFACE_ATTRIBUTE_HEAVY_MAKEUP
BETAFACE_ATTRIBUTE_HIGH_CHEEKBONES
BETAFACE_ATTRIBUTE_MALE
BETAFACE_ATTRIBUTE_MOUTH_SLIGHTLY_OPEN
BETAFACE_ATTRIBUTE_MUSTACHE
BETAFACE_ATTRIBUTE_NARROW_EYES
BETAFACE_ATTRIBUTE_BEARD
BETAFACE_ATTRIBUTE_OVAL_FACE
BETAFACE_ATTRIBUTE_PALE_SKIN
BETAFACE_ATTRIBUTE_POINTY_NOSE
BETAFACE_ATTRIBUTE_RECEDING_HAIRLINE
BETAFACE_ATTRIBUTE_ROSY_CHEEKS
BETAFACE_ATTRIBUTE_SIDEBURNS
BETAFACE_ATTRIBUTE_SMILING
BETAFACE_ATTRIBUTE_STRAIGHT_HAIR
BETAFACE_ATTRIBUTE_WAVY_HAIR
BETAFACE_ATTRIBUTE_WEARING_EARRINGS
BETAFACE_ATTRIBUTE_WEARING_HAT
BETAFACE_ATTRIBUTE_WEARING_LIPSTICK
BETAFACE_ATTRIBUTE_WEARING_NECKLACE
BETAFACE_ATTRIBUTE_WEARING_NECKTIE
BETAFACE_ATTRIBUTE_YOUNG
BETAFACE_ATTRIBUTE_AGE
BETAFACE_ATTRIBUTE_ETHNICITY

BETAFACE_ATTRIBUTE_AGE value contains age in years
BETAFACE_ATTRIBUTE_ETHNICITY value interpreted as follows: value 0 - white, 1 - black, 2 - asian, 3 - hispanic, 4 - mideast, 5 - indian, 6+ - other

For a full list and way to retrieve all extended measurements please contact Betaface support or check the C# sample application code.

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

Introduction

Betaface SDK video processing engine can analyze video files or video streams from local cameras, IP cameras or video files. SDK as well support custom video capturing via callback. 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).

Initialize SDK internal video capturing from video file, local camera or IP camera

Betaface_CaptureVideo(BetafaceInternalState State, char* strVideoFilename, bool bIsLive, int iCameraIdx, int iCaptureBackend, int iWidth, int iHeight, BetafaceVideo* pVideo);
This function initializes 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 or IP camera Url.
bIsLive – set to true when IP camera url is specified in strVideoFilename. Video capture will treat stream as live, will use realtime local time measurement as timestamps instead of timestamps read from frames and will not wait (drop frames) if processing cannot analyze each frame.
iCameraIdx – set strVideoFilename to empty string and this parameter to camera index to capture from local camera. Local camera is always treated as live regardless of bIsLive parameter.
iCaptureBackend – default 0 (automatic). Please contact Betaface if you have problems capturing from certain source.
iWidth – attempt to set X size of the video frame (should be supported by the camera)
iHeight – attempt to set 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

Betaface_CaptureVideoGetProp(BetafaceInternalState State, BetafaceVideo Video, int iPropId, double* pdPropValue);
Betaface_CaptureVideoSetProp(BetafaceInternalState State, BetafaceVideo Video, int iPropId, double dPropValue);
Please contact Betaface if you need to configure or read properties from your camera

Betaface_CaptureVideoCb(BetafaceInternalState State, CB_GrabNextFrame grabFunc, BetafaceVideoParam pArguments, bool bOnline, BetafaceVideo* pVideo);
This function initializes capturing from video using externally supplied frames 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 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 each time just before old frame is released from capturing buffer (i.e. fully processed)
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 – 101 Pro or only 22 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