



Betaface Face Detection and Recognition SDK

ver. 1.73

1. Introduction

Betaface Face Detection and Recognition SDK is a 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. These algorithms do not require color information and work on grayscale pixel values internally.

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 allows you also to crop faces from the images based on detection information, compare faces and get the resemblance score, morph and warp faces, as well as creating average face images (face composites).

SDK core is a protected Windows 32bit DLL. SDK package includes necessary additional runtime files, C++, COM, .Net interfaces, this documentation and samples of usage written in C++ and C#.

In order to use SDK you should have valid registration key file placed in the same folder together with all SDK DLL's or valid USB dongle installed in this PC USB slot.

You can find description of additional Betaface software plug-ins, like face shape and color analysis; iris detection; hairstyle and hair color detection; facial hair and glasses detection; age, gender and emotion recognition plug-ins in a separately supplied documentation.

2. Core SDK functions (all SDK editions)

2.1 Initialization

SDK requires initialization/de-initialization function calls. During them internal variables are being allocated, data is being pre-calculated and you receive an internal state value which you will have to supply to the rest SDK functions. For each initialization call you will get unique state value, which allows you to use parallel processing of different images. Storing initialized state between SDK functions calls allows you to eliminate per-image initialization delays in your application. The same effect may be achieved by using Betaface COM object interface with enabled COM+ object pooling (refer to section explaining Betaface COM installation).

BETAFACEFDRE_API **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 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.

BETAFACEFDRE_API **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 value.*

Return value:

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

2.2 Loading/saving images

BETAFACEFDRE_API **Betaface_LoadImage**(BetafaceInternalState State, char* strImageFilename, BetafaceImage* plmg);

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

Parameters:

State	<i>Betaface library internal state value, obtained from Betaface_Init function.</i>
strImageFilename	<i>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).</i>
plmg	<i>Pointer to BetafaceImage variable, by this address image in internal Betaface format will be returned.</i>

Return value:

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

BETAFACEFDRE_API **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	<i>Betaface library internal state value, obtained from Betaface_Init function.</i>
strImageFilename	<i>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).</i>
Img	<i>Betaface internal representation image in BetafaceImage type variable</i>
flags	<i>Optional combination of image saving parameter flags. Following flags currently supported: BETAFACE_SAVEIMAGE_GRAYSCALE – convert image to grayscale. BETAFACE_SAVEIMAGE_FLIP_HORISONTAL – mirrors image.</i>
strText	<i>Reserved for future use</i>

Return value:

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

BETAFACEFDRE_API **Betaface_LoadMemoryImage**(BetafaceInternalState State, char* plmageBytes, int iWidth, int iHeight, int nFormat, BetafaceImage* plmg);

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

Parameters:

State	<i>Betaface library internal state value, obtained from Betaface_Init function.</i>
plmageBytes	<i>Pointer to a memory buffer where uncompressed image pixel data is located according to supplied image size and format parameters.</i>
iWidth	<i>Image width in pixels</i>
iHeight	<i>Image height in pixels</i>
nFormat	<i>Format of the image pixel data in supplied memory buffer. Accepted format</i>

values are:

BETAFACE_MEMORYIMAGE_GG (8 bit grayscale)

BETAFACE_MEMORYIMAGE_RRGGBB (24 bit RGB)

BETAFACE_MEMORYIMAGE_BBGRR (24 bit BGR)

BETAFACE_MEMORYIMAGE_RRGGBBAA (32 bit RGB)

BETAFACE_MEMORYIMAGE_BBGRRRAA (32 bit BGR)

BETAFACE_MEMORYIMAGE_RGB32FLT (96 bit special RGB32FLT)

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.

BETAFACEFDRE_API Betaface_SaveMemoryImage(*BetafaceInternalState* State, *BetafaceImage* *Img*, *BetafaceMemoryFormat* *nFormat*, int* *piWidth*, int* *piHeight*, char** *pplmageBytes*, int* *pilmageBytesLen*);
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 (8 bit grayscale)

BETAFACE_MEMORYIMAGE_RRGGBB (24 bit RGB)

BETAFACE_MEMORYIMAGE_BBGRR (24 bit BGR)

BETAFACE_MEMORYIMAGE_RRGGBBAA (32 bit RGB)

BETAFACE_MEMORYIMAGE_BBGRRRAA (32 bit BGR)

BETAFACE_MEMORYIMAGE_RGB32FLT (96 bit special 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

pplmageBytes Pointer to a pointer variable where allocated memory buffer containing uncompressed image pixel data according to supplied format parameter will be returned.

pilmageBytesLen 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.

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

This function creates a copy of 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.

BETAFACEFDRE_API **Betaface_ReleaseImage**(BetafaceInternalState State, BetafaceImage* plmg);

This function releases image in internal Betaface representation.

Parameters:

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

plmg *Betaface internal representation image in BetafaceImage variable.*

Return value:

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

BETAFACEFDRE_API **Betaface_ReleaseMemoryImage**(BetafaceInternalState State, char**
pplmageBytes);

*This function releases memory buffer allocated via **Betaface_SaveMemoryImage** function.*

Parameters:

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

pplmageBytes *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.

2.3 Face Detection

BETAFACEFDRE_API **Betaface_DetectFaces**(BetafaceInternalState State, BetafaceImage Img, BetafaceDetectFacesFlags flags, int iMaxImageWidthPix, int iMaxImageHeightPix, double dMinFaceSizeOnImage, int iMinFaceSizePix, double dAngleDegrees, double dAngleToleranceDegrees, int* piFacesCount, BetafaceDetectionResult* pDetectionResult);

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

Parameters:

State	<i>Betaface library internal state value, obtained from Betaface_Init function.</i>
Img	<i>Source image in internal Betaface representation.</i>
Flags	<i>Combination of detection flags, currently following flags are supported: BETAFACE_DETECTFACES_BASIC or BETAFACE_DETECTFACES_PRO – specify one of those flags to indicate which feature points you need to detect – Basic (8 points) or Basic + Pro (8 + 86 facial points). BETAFACE_DETECTFACES_BESTFACEONLY – If this flag is specified, only single face with highest detection score processed and returned.</i>
iMaxImageWidthPix	<i>Limits of the source image resolution. First number of pixels 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.</i>
iMaxImageHeightPix	<i>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.</i>
dMinFaceSizeOnImage	<i>Limits the minimum face size on the image as the fraction to the whole image. Used to increase the speed of detection. 0.3 mean that faces which 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.</i>
iMinFaceSizePix	<i>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.</i>
dAngleDegrees	<i>Central angle of the faces detected. If dAngleToleranceDegrees parameter is 0 or greater or equal to 180 degrees this parameter is ignored.</i>
dAngleToleranceDegrees	<i>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 20 or 30 for portrait photos (+- 20-30 degrees from strictly vertical face, if dAngleDegrees is 0) or 0 for full image scan.</i>
piFacesCount	<i>Pointer to integer variable where number of faces detected will be returned.</i>
pDetectionResult	<i>Pointer to the BetafaceDetectionResult variable where the detection data in internal Betaface representation will be returned.</i>

Return value:

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

BETAFACEFDRE_API **Betaface_ReleaseDetectionResult**(BetafaceInternalState State, BetafaceDetectionResult* pDetectionResult);

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

Parameters:

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

Return value:

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

2.4 Detection result processing

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

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

Parameters:

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

Return value:

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

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

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

Parameters:

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

Return value:

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

BETAFACEFDRE_API **Betaface_CreateFaceInfo**(BetafaceInternalState State, BetafaceFaceInfo* pFaceInfo);

This function creates empty face information structure.

Parameters:

State	<i>Betaface library internal state value, obtained from Betaface_Init function.</i>
-------	---

pFacelInfo *Pointer to the BetafaceFacelInfo 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.

BETAFACEFDRE_API **Betaface_CopyFacelInfo**(BetafaceInternalState State, BetafaceFacelInfo FacelInfo, BetafaceFacelInfo* pFacelInfoCopy);

This function creates a complete copy of face information structure.

Parameters:

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

FacelInfo *BetafaceFacelInfo variable containing face information data to be copied.*

pFacelInfoCopy *Pointer to the BetafaceFacelInfo variable where complete copy of FacelInfo face information data in internal Betaface representation will be returned.*

Return value:

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

BETAFACEFDRE_API **Betaface_ReleaseFacelInfo**(BetafaceInternalState State, BetafaceFacelInfo* pFacelInfo);

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.*

pFacelInfo *Pointer to the BetafaceFacelInfo variable where stored valid face information data in internal Betaface representation.*

Return value:

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

BETAFACEFDRE_API **Betaface_GetFacelInfoBoolParam**(BetafaceInternalState State, BetafaceFacelInfo FacelInfo, BetafaceFeatureParam param, bool* pValue);

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

Parameters:

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

FacelInfo *BetafaceFacelInfo variable where stored valid face information data in internal Betaface representation.*

param *Parameter ID calculated as a combination of one of the face feature type flags and one of the Boolean information type flags, 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/angle 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.

BETAFACEFDRE_API **Betaface_GetFacelInfoDoubleParam**(BetafaceInternalState State, BetafaceFacelInfo FacelInfo, BetafaceFeatureParam param, double* pValue);
This function used to retrieve double type parameters values from face information data.

Parameters:

State	<i>Betaface library internal state value, obtained from Betaface_Init function.</i>
FacelInfo	<i>BetafaceFacelInfo variable where stored valid face information data in internal Betaface representation.</i>
param	<i>Parameter ID calculated as a combination of one of the face feature type flags and one of the Double information type flags, for example parameter BETAFACE_PARAM_WIDTH BETAFACE_FEATURE_EYE_L value will contain width in pixels of the left eye bounding rectangle.</i>
pValue	<i>Pointer to double variable where the requested parameter value will be returned.</i>

Return value:

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

Available features:

Basic features (SDK Standard, Pro, Xtreme)

BETAFACE_FEATURE_FACE
BETAFACE_FEATURE_EYE_L
BETAFACE_FEATURE_EYE_R
BETAFACE_FEATURE_EYE_LCO
BETAFACE_FEATURE_EYE_RCO
BETAFACE_FEATURE_EYE_LCI
BETAFACE_FEATURE_EYE_RCI
BETAFACE_FEATURE_MOUTH_LC
BETAFACE_FEATURE_MOUTH_RC

Pro features (SDK Pro, Xtreme)

BETAFACE_FEATURE_PRO_CHIN_EARCONN_L
BETAFACE_FEATURE_PRO_CHIN_P1_L
BETAFACE_FEATURE_PRO_CHIN_P2_L
BETAFACE_FEATURE_PRO_CHIN_P3_L
BETAFACE_FEATURE_PRO_CHIN_P4_L
BETAFACE_FEATURE_PRO_CHIN_P5_L
BETAFACE_FEATURE_PRO_CHIN_B
BETAFACE_FEATURE_PRO_CHIN_P5_R
BETAFACE_FEATURE_PRO_CHIN_P4_R
BETAFACE_FEATURE_PRO_CHIN_P3_R
BETAFACE_FEATURE_PRO_CHIN_P2_R
BETAFACE_FEATURE_PRO_CHIN_P1_R
BETAFACE_FEATURE_PRO_CHIN_EARCONN_R
BETAFACE_FEATURE_PRO_TEMPLE_P4_R
BETAFACE_FEATURE_PRO_TEMPLE_P3_R
BETAFACE_FEATURE_PRO_TEMPLE_P2_R
BETAFACE_FEATURE_PRO_TEMPLE_P1_R
BETAFACE_FEATURE_PRO_TEMPLE_R
BETAFACE_FEATURE_PRO_FOREHEAD_R
BETAFACE_FEATURE_PRO_FOREHEAD_P4
BETAFACE_FEATURE_PRO_FOREHEAD_P3
BETAFACE_FEATURE_PRO_FOREHEAD_M
BETAFACE_FEATURE_PRO_FOREHEAD_P2
BETAFACE_FEATURE_PRO_FOREHEAD_P1
BETAFACE_FEATURE_PRO_FOREHEAD_L
BETAFACE_FEATURE_PRO_TEMPLE_L
BETAFACE_FEATURE_PRO_TEMPLE_P1_L

BETAFACE_FEATURE_PRO_TEMPLE_P2_L
BETAFACE_FEATURE_PRO_TEMPLE_P3_L
BETAFACE_FEATURE_PRO_TEMPLE_P4_L
BETAFACE_FEATURE_PRO_EYE_O_R
BETAFACE_FEATURE_PRO_EYE_BO_R
BETAFACE_FEATURE_PRO_EYE_B_R
BETAFACE_FEATURE_PRO_EYE_BI_R
BETAFACE_FEATURE_PRO_EYE_I_R
BETAFACE_FEATURE_PRO_EYE_TI_R
BETAFACE_FEATURE_PRO_EYE_T_R
BETAFACE_FEATURE_PRO_EYE_TO_R
BETAFACE_FEATURE_PRO_EYE_O_L
BETAFACE_FEATURE_PRO_EYE_TO_L
BETAFACE_FEATURE_PRO_EYE_T_L
BETAFACE_FEATURE_PRO_EYE_TI_L
BETAFACE_FEATURE_PRO_EYE_I_L
BETAFACE_FEATURE_PRO_EYE_BI_L
BETAFACE_FEATURE_PRO_EYE_B_L
BETAFACE_FEATURE_PRO_EYE_BO_L
BETAFACE_FEATURE_PRO_EYEBROW_I_R
BETAFACE_FEATURE_PRO_EYEBROW_TI_R
BETAFACE_FEATURE_PRO_EYEBROW_T_R
BETAFACE_FEATURE_PRO_EYEBROW_TO_R
BETAFACE_FEATURE_PRO_EYEBROW_O_R
BETAFACE_FEATURE_PRO_EYEBROW_BO_R
BETAFACE_FEATURE_PRO_EYEBROW_B_R
BETAFACE_FEATURE_PRO_EYEBROW_BI_R
BETAFACE_FEATURE_PRO_EYEBROW_I_L
BETAFACE_FEATURE_PRO_EYEBROW_TI_L
BETAFACE_FEATURE_PRO_EYEBROW_T_L
BETAFACE_FEATURE_PRO_EYEBROW_TO_L
BETAFACE_FEATURE_PRO_EYEBROW_O_L
BETAFACE_FEATURE_PRO_EYEBROW_BO_L
BETAFACE_FEATURE_PRO_EYEBROW_B_L
BETAFACE_FEATURE_PRO_EYEBROW_BI_L
BETAFACE_FEATURE_PRO_MOUTH_L
BETAFACE_FEATURE_PRO_MOUTH_TL
BETAFACE_FEATURE_PRO_MOUTH_T
BETAFACE_FEATURE_PRO_MOUTH_TR
BETAFACE_FEATURE_PRO_MOUTH_R
BETAFACE_FEATURE_PRO_MOUTH_BR
BETAFACE_FEATURE_PRO_MOUTH_B
BETAFACE_FEATURE_PRO_MOUTH_BL
BETAFACE_FEATURE_PRO_NOSE_T_L
BETAFACE_FEATURE_PRO_NOSE_TI_NOSTRIL_L
BETAFACE_FEATURE_PRO_NOSE_TO_NOSTRIL_L
BETAFACE_FEATURE_PRO_NOSE_BO_NOSTRIL_L
BETAFACE_FEATURE_PRO_NOSE_B_NOSTRIL_L
BETAFACE_FEATURE_PRO_NOSE_B
BETAFACE_FEATURE_PRO_NOSE_B_NOSTRIL_R
BETAFACE_FEATURE_PRO_NOSE_BO_NOSTRIL_R
BETAFACE_FEATURE_PRO_NOSE_TO_NOSTRIL_R
BETAFACE_FEATURE_PRO_NOSE_TI_NOSTRIL_R
BETAFACE_FEATURE_PRO_NOSE_T_R
BETAFACE_FEATURE_PRO_EYE_IRIS_R
BETAFACE_FEATURE_PRO_EYE_IRIS_L
BETAFACE_FEATURE_PRO_NOSE_TIP
BETAFACE_FEATURE_PRO_CHEEKBONE_L
BETAFACE_FEATURE_PRO_CHEEKBONE_R

More features can be retrieved via this interface extensions to SDK are installed, such as facial hair, glasses and hairstyle detection, facial features and face geometry analysis.

BETAFACEFDRE_API **Betaface_SetFacelInfoBoolParam**(BetafaceInternalState State, BetafaceFacelInfo FacelInfo, BetafaceFeatureParam param, bool Value);

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

Parameters:

State	<i>Betaface library internal state value, obtained from Betaface_Init function.</i>
FacelInfo	<i>BetafaceFacelInfo variable where stored valid face information data in internal Betaface representation.</i>
param	<i>Parameter ID calculated as a combination of one of the face feature type flags and one of the Boolean information type flags, for example in order to set a flag that left eye information data is available specify BETAFACE_PARAM_EXISTS BETAFACE_FEATURE_EYE_L.</i>
Value	<i>bool variable containing parameter value to be set.</i>

Return value:

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

BETAFACEFDRE_API **Betaface_SetFacelInfoDoubleParam**(BetafaceInternalState State, BetafaceFacelInfo FacelInfo, BetafaceFeatureParam param, double Value);

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

Parameters:

State	<i>Betaface library internal state value, obtained from Betaface_Init function.</i>
FacelInfo	<i>BetafaceFacelInfo variable where stored valid face information data in internal Betaface representation.</i>
param	<i>Parameter ID calculated as a combination of one of the face feature type flags and one of the Double information type flags, for example in order to set the width in pixels of the left eye bounding rectangle specify BETAFACE_PARAM_WIDTH BETAFACE_FEATURE_EYE_L.</i>
Value	<i>double variable containing parameter value to be set.</i>

Return value:

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

2.5 Cropping faces

BETAFACEFDRE_API **Betaface_CropFacelImage**(BetafaceInternalState State, BetafaceImage Img, BetafaceFacelInfo FacelInfo, double dEyesDistance, double dEyeLineHeight, bool bDeRotate, int iCropWidth, int iCropHeight, double dAreaScale, int colorBackground, BetafaceImage* pCroppedFacelImg, BetafaceFacelInfo* pCroppedFacelInfo);

This function is cropping and de-rotating face from the image and is fitting it to specified output image dimensions.

Parameters:

State	<i>Betaface library internal state value, obtained from Betaface_Init function.</i>
Img	<i>Image containing face to crop and to which FacelInfo corresponds.</i>
FacelInfo	<i>BetafaceFacelInfo variable where valid face information data in internal Betaface representation is stored.</i>
dEyesDistance	<i>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.</i>
dEyeLineHeight	<i>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.</i>
bDeRotate	<i>Specify true if you want to de-rotate face image 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 usually set to true.</i>
iCropWidth	<i>Output image width.</i>
iCropHeight	<i>Output image height.</i>
dAreaScale	<i>Additional scale coefficient applied to the face rectangle before the area to crop is determined. If this coefficient >1.0 Then the rectangle scaled until the borders of the whole image are reached, if the face rectangle was already out of the image borders then this coefficient fixed to 1.0. If the specified coefficient <1.0 then no checks performed and scaling factor applied directly.</i>
colorBackground	<i>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.</i>
pCroppedFacelImg	<i>Pointer to the BetafaceImage variable where the cropped image in internal Betaface representation will be returned.</i>
pCroppedFacelInfo	<i>Pointer to the BetafaceFacelInfo variable where the face information data, converted to the cropped image coordinate system will be returned.</i>

Return value:

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

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

This function cropping 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	<i>Betaface library internal state value, obtained from Betaface_Init function.</i>
Img	<i>Image containing face to crop and to which FaceInfo corresponds.</i>
FaceInfo	<i>BetafaceFaceInfo variable where stored valid face information data in internal Betaface representation.</i>
dEyesDistance	<i>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.</i>
dEyeLineHeight	<i>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.</i>
dFaceAspectHW	<i>Aspect ratio of the face rectangle.</i>
dImageAspectHW	<i>Aspect ratio of the final image area to cut.</i>
colorBackground	<i>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.</i>
colorBackground	<i>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.</i>
pCroppedFaceImg	<i>Pointer to the BetafaceImage variable where the cropped image in internal Betaface representation will be returned.</i>
pCroppedFaceInfo	<i>Pointer to the BetafaceFaceInfo variable where the face information data, converted to the cropped image coordinate system will be returned.</i>

Return value:

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

2.6 Drawing faces

BETAFACEFDRE_API **Betaface_DrawFacelInfo**(BetafaceInternalState State, BetafaceImage Img, BetafaceFacelInfo FacelInfo);

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 source image.

Parameters:

State	<i>Betaface library internal state value, obtained from Betaface_Init function.</i>
Img	<i>Image containing face to crop and to which FacelInfo corresponds.</i>
FacelInfo	<i>BetafaceFacelInfo variable where valid face information data in internal Betaface representation is stored.</i>

Return value:

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

3.0 Face Recognition (comparison)

BETAFACEFDRE_API **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 to another. These keys usually stored in the database as BLOB (binary large object) fields.

Parameters:

State	<i>Betaface library internal state value, obtained from Betaface_Init function.</i>
Img	<i>Image containing face to crop and to which FaceInfo corresponds.</i>
FaceInfo	<i>BetafaceFaceInfo variable where valid face information data in internal Betaface representation is stored.</i>
ppFaceKeyData	<i>Pointer to the variable where the address of the key binary data will be returned.</i>
piFaceKeyLen	<i>Pointer to the integer variable where length of the key data in bytes will be returned.</i>

Return value:

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

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

When more than one image available for particular person this function allow you to join recognition keys created from multiple pictures into single strong key, to improve recognition quality. We recommend to create for each person in database strong keys that consist from minimum 3 and ideally 10 usual keys. When joining multiple keys call this function repeatedly to first join key 1 and key 2 into strong key12, then join key12 and key 3 to create strong key 123 and so on.

Parameters:

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

Return value:

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

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

This function compare two binary face “keys” of the same length and returns similarity score value. Consult Betaface support regarding which similarity values use for search, verification and identification tasks.

Parameters:

State	<i>Betaface library internal state value, obtained from Betaface_Init function.</i>
pFaceKeyData1	<i>Address of first face "key" binary data.</i>
pFaceKeyData2	<i>Address of second face "key" binary data.</i>
FaceKeysLen	<i>Length of the keys data in bytes.</i>
pdSimilarity	<i>Similarity score for these two faces. The higher the score the more similarity between two faces found. The range of similarity score at the moment</i> <i>Basic recognition –</i> <i>0 (100% similar, exactly the same faces) to an approximately -0.16.</i> <i>Pro recognition –</i> <i>0 (100% similar, exactly the same faces) to an approximately -1000</i>

Return value:

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

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

This function compares two binary face "keys" of the same length and returns similarity score value normalized to 0-100% and decision if it is same person or not. You have to specify target false alarm rate value.

Parameters:

State	<i>Betaface library internal state value, obtained from Betaface_Init function.</i>
pFaceKeyData1	<i>Address of first face "key" binary data.</i>
pFaceKeyData2	<i>Address of second face "key" binary data.</i>
FaceKeysLen	<i>Length of the keys data in bytes.</i>
iRank	<i>Not used, 0</i>
dFalseAlarmRate	<i>target false alarm rate (typical range 0.001-0.09, default 0.005).</i>
pblsSamePerson	<i>true if faces belong to the same person, with error defined by false alarm rate.</i>
pdNormalizedConfidence	<i>Similarity score for these two faces. The higher the score the more similarity between two faces found. The range of similarity score 0 (not similar, different persons) to 1.0 (same person).</i>

Return value:

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

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

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

Parameters:

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

Return value:

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

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

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

Parameters:

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

Return value:

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

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

This function 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.

4.0 Morphing and averaging faces (SDK Pro and Xtreme editions)

4.1. Face morphing and warping

“Face morphing” usually mean two different effects:

- **Morph**, is a transformation effect, when one face (source) smoothly transforms into another face (destination). Morphing effect can be used to mix two faces together (koeff. = 0.5) and create a face containing facial features of both source and destination faces, or to generate set of video frames showing transformation process (koeff. 0.0 – 1.0).
- **Warp**, which mean a geometrical distortion of one face (source) into a new shape, which can be done either in one step (koeff 1.0), showing final result, or in number of video frames, showing smooth transformation process.

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

This function morphs (blend and geometrically distort) between two faces, using feature points contained in the face info data as an anchor points of morphing.

Parameters:

State	<i>Betaface library internal state value, obtained from Betaface_Init function.</i>
SrcFaceImg	<i>Image containing source face and to which SrcFaceInfo corresponds.</i>
SrcFaceInfo	<i>BetafaceFaceInfo variable where valid face information of the source face data in internal Betaface representation is stored.</i>
DstFaceImg	<i>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.</i>
DstFaceInfo	<i>BetafaceFaceInfo variable where valid face information of the destination face data in internal Betaface representation is stored.</i>
dTransitionKoeff	<i>Transition coefficient ranges from 0.0 (100% source face) to 1.0 (100% destination face) to determine morphing stage.</i>
pMorphedImg	<i>Pointer to the BetafaceImage variable where the resulting morphed image in internal Betaface representation will be returned.</i>

Return value:

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

4.2 Averaging faces

Following section containing functions to cache and create face averages (face composites).

BETAFACEFDRE_API **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	<i>Betaface library internal state value, obtained from Betaface_Init function.</i>
dEyesDistance	<i>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.</i>
dEyeLineHeight	<i>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.</i>
iImageWidth	<i>Destination image width.</i>
iImageHeight	<i>Destination image height.</i>
pAverageFaceInfo	<i>Pointer to the BetafaceFaceInfo variable where the global average face information data, converted to the cropped image coordinate system will be returned.</i>

Return value:

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

BETAFACEFDRE_API **Betaface_UpdateAverageImage**(BetafaceInternalState State, BetafaceImage OldAvgImg, int OldAvgCount, bool bAppend, BetafaceImage Img, int iAvgCount, BetafaceImage* pAverageImg);

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

Parameters:

State	<i>Betaface library internal state value, obtained from Betaface_Init function.</i>
OldAvgImg	<i>Image containing accumulated average face texture, warped to a global face average shape obtained in Betaface_GetStoredAverageFaceInfo function.</i>
OldAvgCount	<i>Number of faces accumulated in average face texture image OldAvgImg.</i>
bAppend	<i>Type of update operation, set it to true to add texture image into accumulated image or false to subtract from it</i>
Img	<i>Image containing new single face texture to append into accumulated average face texture.</i>
iAvgCount	<i>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.</i>
pAverageImg	<i>Pointer to the BetafaceImage variable where the accumulated average face texture image in internal Betaface representation will be returned.</i>

Return value:

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

BETAFACEFDRE_API **Betaface_UpdateAverageFaceInfo**(BetafaceInternalState State, BetafaceFaceInfo OldAvgFaceInfo, int OldAvgCount, bool bAppend, BetafaceFaceInfo FaceInfo, int iAvgCount, BetafaceFaceInfo* pAverageFaceInfo);

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

Parameters:

State	<i>Betaface library internal state value, obtained from Betaface_Init function.</i>
OldAvgFaceInfo	<i>BetafaceFaceInfo variable where accumulated average face shape information, in internal Betaface representation is stored.</i>
OldAvgCount	<i>Number of face shapes accumulated in average face shape information OldAvgFaceInfo.</i>
bAppend	<i>Type of update operation, set it to true to add face shape information into accumulated face shape information or false to subtract from it</i>
FaceInfo	<i>BetafaceFaceInfo variable containing new single face shape information to append into accumulated average face shape information.</i>
iAvgCount	<i>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.</i>
pAverageFaceInfo	<i>Pointer to the BetafaceFaceInfo variable where the accumulated average face shape information in internal Betaface representation will be returned.</i>

Return value:

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

5. Video Processing (SDK Xtreme)

5.1. Introduction

Betaface SDK video processing engine is able to 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 will do all the capturing and tracking threads synchronization work for you. 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, 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, then de-initialize Betaface SDK by calling Betaface_Deinit function. You can find documentation for Betaface_Init/Betaface_Deinit functions in main documentation.

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 data (video frame, parameters) you 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.

Betaface .Net interface assembly offers extended capturing interface based on DirectShow.

5.2. Capturing

5.2.1 Initialize capturing from file

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

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

Parameters:

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

Return value:

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

5.2.2 Initialize capturing from camera (VFW)

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

Return value:

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

5.2.3 Initialize custom capturing via callbacks

BETAFACEFDRE_API **Betaface_CaptureVideoCb**(BetafaceInternalState State, CB_GrabNextFrame grabFunc, void* pArguments, int iWidth, int iHeight, int nFormat, double dFPS, bool bOnline, BetafaceVideo* pVideo);

This function initialize offline capturing from video file and return internal Betaface video capturing runtime state.

Parameters:

State	<i>Betaface library internal state value, obtained from Betaface_Init function.</i>
grabFunc	<i>Pointer to frame grabbing callback handler function.</i>
pArguments	<i>Pointer that will be passed back to you in every callback call.</i>
iWidth	<i>X size of the video frames you will supply.</i>
iHeight	<i>Y size of the video frames you will supply.</i>
dFPS	<i>Number of frames per second of your stream, needed to interpret parameters in seconds correctly.</i>

bOnline TRUE when it is a live camera stream, FALSE when it is offline capture from 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.

```
typedef BETAFACE_RETVAL (BETAFACEFDRE_CALLBACK *CB_GrabNextFrame)(BetafaceVideo Video,
BetafaceImage* pFrame, int* piFrameldx, void* pArguments);
```

This is a callback handler function prototype that will be called each time new frame is needed from the stream.

Parameters:

Video BetafaceVideo variable containing video capturing runtime state in internal Betaface format.

pFrame Pointer where to write BetafaceImage with frame data.

piFrameldx Pointer to write frame index.

pArguments Your pointer you supplied when called Betaface_CaptureVideoCb function.

Return value:

Function should return BETAFACE_OK if it is successful, error code otherwise.

5.2.4 Set capturing process callbacks

```
BETAFACEFDRE_API Betaface_SetFrameGrabbedCallback(BetafaceInternalState State, BetafaceVideo
Video, CB_OnFrameGrabbed callbackFunc, void* 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 grabbed callback handler function.

pArguments Pointer that will be passed back to you in every callback call.

Return value:

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

```
typedef BETAFACE_RETVAL (BETAFACEFDRE_CALLBACK *CB_OnFrameGrabbed)(BetafaceVideo
Video, BetafaceImage Frame, int iFrameldx, void* pArguments);
```

This is a callback handler function prototype that will be 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.

iFrameldx Captured frame index.

pArguments Your pointer you supplied when called Betaface_SetFrameGrabbedCallback function.

Return value:

Function should return BETAFACE_OK if it is successful, error code otherwise.

BETAFACEFDRE_API **Betaface_SetFrameReleasedCallback**(BetafaceInternalState State, BetafaceVideo Video, CB_OnFrameReleased callbackFunc, void* pArguments);
This function set callback called after each successfully grabbed frame.

Parameters:

State	<i>Betaface library internal state value, obtained from Betaface_Init function.</i>
Video	<i>Video capturing runtime state in internal Betaface format.</i>
callbackFunc	<i>Pointer to frame released callback handler function.</i>
pArguments	<i>Pointer that will be passed back to you in every callback call.</i>

Return value:

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

typedef BETAFACE_RETVAL (BETAFACEFDRE_CALLBACK ***CB_OnFrameReleased**)(BetafaceVideo Video, BetafaceImage Frame, int iFrameIdx, void* pArguments);
This is a callback handler function prototype that will be called each time before old frame is being released from capturing buffer (i.e. fully processed).

Parameters:

Video	<i>Betaface library internal state value.</i>
Frame	<i>BetafaceImage variable containing frame pixel data in internal Betaface format.</i>
iFrameIdx	<i>Captured frame index.</i>
pArguments	<i>Your pointer you supplied when called Betaface_SetFrameReleasedCallback function.</i>

Return value:

Function should return BETAFACE_OK if it is successful, error code otherwise.

5.2.5 Releasing capturing resources

BETAFACEFDRE_API **Betaface_ReleaseVideo**(BetafaceInternalState State, BetafaceVideo* pVideo);
This function stops capturing process and release all internal resources related to it.

Parameters:

State	<i>Betaface library internal state value, obtained from Betaface_Init function.</i>
pVideo	<i>Pointer to variable containing video capturing runtime state in internal Betaface format.</i>

Return value:

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

5.3. Tracking

5.3.1. Set tracking process callback

BETAFACEFDRE_API **Betaface_SetFaceInfoUpdatedCallback**(BetafaceInternalState State, BetafaceVideo Video, CB_OnFaceInfoUpdated callbackFunc, void* pArguments);
This function set callback called when new tracking information is available.

Parameters:

State	<i>Betaface library internal state value, obtained from Betaface_Init function.</i>
Video	<i>Video capturing runtime state in internal Betaface format.</i>
callbackFunc	<i>Pointer to tracking information callback handler function.</i>
pArguments	<i>Pointer that will be passed back to you in every callback call.</i>

Return value:

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

typedef BETAFACE_RETVAL (BETAFACEFDRE_CALLBACK ***CB_OnFaceInfoUpdated**)(BetafaceVideo Video, int iFaceInfoID, int iFrameIdx, int iUpdateType, void* pArguments);
This is a callback handler function prototype that will be called each new tracking information is available.

Parameters:

Video	<i>Betaface library internal state value.</i>
iFaceInfoID	<i>Face index.</i>
iFrameIdx	<i>Captured frame index.</i>
iUpdateType	<i>Type of information updated, can be FACEINFOUPDATE_INITIAL – initial face detection . FACEINFOUPDATE_FORWARD – forward tracking. FACEINFOUPDATE_BACKWARD – backward tracking</i>
pArguments	<i>Your pointer you supplied when called Betaface_SetFaceInfoUpdatedCallback function.</i>

Return value:

Function should return BETAFACE_OK if it is successful, error code otherwise.

5.3.2. Start/stop both capturing and tracking processes

BETAFACEFDRE_API **Betaface_StartDetectFaces**(BetafaceInternalState State, BetafaceVideo Video, CB_OnFaceInfoUpdated callbackFunc, void* pArguments);
This function starts capturing and tracking processes with specified parameters.

Parameters:

State	<i>Betaface library internal state value, obtained from Betaface_Init function.</i>
Video	<i>BetafaceVideo variable containing video capturing runtime state in internal Betaface format.</i>
dDetectionTickSeconds	<i>Detection tick time in seconds, how often stream will be scanned for new faces.</i>
dRecScoreTreshold	<i>Recognition score threshold value for recognition.</i>
dLearnScoreTreshold	<i>Recognition score threshold value for learning new person.</i>
dRecMinExposureSeconds	<i>Minimum amount of seconds person should be tracked before system try to recognize him.</i>
dLearnMinExposureSeconds	<i>Minimum amount of seconds person should be tracked before system</i>

start to learn him.

flags

iMaxImageWidthPix

iMaxImageHeightPix

dMinFaceSizeOnImage

Please refer to Betaface_DetectFaces function documentation for those parameters meaning.

iMinFaceSizePix

dAngleDegrees

dAngleToleranceDegrees

piFacesCount

pDetectionResult

Return value:

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

BETAFACEFDRE_API **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.

BETAFACEFDRE_API **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.

5.3.3. Runtime tracking state

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

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

Parameters:

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

Video *Video capturing runtime state in internal Betaface format.*

piFacesCount *Number of faces detected.*

pDetectionResult *Pointer where snapshot of detection result should be written.*

Return value:

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

5.4. Face recognition on video

BETAFACEFDRE_API **Betaface_AddPerson**(BetafaceInternalState State, BetafaceVideo Video, char* pFaceKeyData, int iFaceKeyLen, __int64 iFaceKeyCounter);

This function adds a person into tracker watchlist.

Parameters:

State	<i>Betaface library internal state value, obtained from Betaface_Init function.</i>
Video	<i>Video capturing runtime state in internal Betaface format.</i>
pFaceKeyData	<i>Facial recognition key data.</i>
iFaceKeyLen	<i>Facial recognition key length.</i>
iFaceKeyCounter	<i>Facial recognition key internal counter.</i>

Return value:

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

BETAFACEFDRE_API **Betaface_ResetPersons**(BetafaceInternalState State, BetafaceVideo Video);

This function removes all persons from tracker watchlist.

Parameters:

State	<i>Betaface library internal state value, obtained from Betaface_Init function.</i>
Video	<i>Video capturing runtime state in internal Betaface format.</i>
pFaceKeyData	<i>Facial recognition key data.</i>
iFaceKeyLen	<i>Facial recognition key length.</i>
iFaceKeyCounter	<i>Facial recognition key internal counter.</i>

Return value:

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

BETAFACEFDRE_API **Betaface_SetPersonUpdatedCallback**(BetafaceInternalState State, CB_OnPersonUpdated callbackFunc, void* pArguments);

This function set callback called after new person is learned or new information about existing person is learned.

Parameters:

State	<i>Betaface library internal state value, obtained from Betaface_Init function.</i>
Video	<i>Video capturing runtime state in internal Betaface format.</i>
callbackFunc	<i>Pointer to person updated callback handler function.</i>
pArguments	<i>Pointer that will be passed back to you in every callback call.</i>

Return value:

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

typedef BETAFACE_RETVAL (BETAFACEFDRE_CALLBACK ***CB_OnPersonUpdated**)(BetafaceVideo Video, int iPersonID, char* pFaceKeyData, int iFaceKeyLen, __int64 iFaceKeyCounter, void* pArguments);

This is a callback handler function prototype that will be called each time new person is learned or new information about existing person is learned.

Parameters:

Video	<i>BetafaceVideo variable containing video capturing runtime state in internal Betaface format.</i>
-------	---

iPersonID	<i>Person index in tracker watchlist (corresponding to the call index of Betaface_AddPerson function).</i>
pFaceKeyData	<i>Facial recognition key data.</i>
iFaceKeyLen	<i>Facial recognition key length.</i>
iFaceKeyCounter	<i>Facial recognition key internal counter.</i>

Return value:

Function should return BETAFACE_OK if it is successful, error code otherwise.

5.5. Displaying frames and draw debug information

BETAFACEFDRE_API **Betaface_DrawVideoFrame**(BetafaceInternalState State, BetafaceImage Img, BetafaceVideo Video, int iFrameldx, int iFaceID, BetafaceDrawVideoFrameFlags flags);
This function draws debug information on a video frame. Use Betaface_DisplayImage to display the frame after drawing.

Parameters:

State	<i>Betaface library internal state value, obtained from Betaface_Init function.</i>
Img	<i>Video frame in internal Betaface format.</i>
Video	<i>Video capturing runtime state in internal Betaface format.</i>
iFrameldx	<i>Index of video frame to draw.</i>
iFaceID	<i>Index of face to draw.</i>
flags	<i>Information flags, can be</i> BETAFACE_DRAWVIDEOFRAME_FACE – <i>draw tracked faces</i> BETAFACE_DRAWVIDEOFRAME_POINTS – <i>draw tracked points</i> BETAFACE_DRAWVIDEOFRAME_IDENTITY – <i>draw recognition info</i> BETAFACE_DRAWVIDEOFRAME_ALL – <i>draw all information</i> BETAFACE_DRAWVIDEOFRAME_INWHITE – <i>draw in white color</i>

Return value:

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

Appendix 1.

Betaface COM component installation as COM+ service application on (web) servers

Initial installation and recommended settings

1. Copy Betaface binary files and your registration key into server folder, for example "C:\Betaface\Bin".
2. Give Read and Execute permissions on all binary Betaface files for the web user (IIS) account.
3. Open Component Services from Start -> Settings -> Control Panel -> Administrative Tools or from the server management window.
4. Navigate to Component Services -> Computers -> My Computer -> COM+ Applications.
5. Right-click on COM+ Applications, select New -> Application -> Next -> Empty application.
6. Enter a name of Betaface application, for example "Betaface Service".
7. Select Server Application type, click Next.
8. Enter proper user account credentials under which Betaface Service will operate and click Finish.
9. Right click on newly created Betaface COM+ application and select Properties.
10. On tab Advanced set the check "Leave running when idle".
11. On tab Pooling and Recycling set Pool Size = 2, Lifetime Limit = 15, Memory Limit = 1048576, Expiration timeout = 10, Call limit = 500, Activation Limit = 200.
12. Open Betaface COM+ application, right click on Components and select New -> Component -> Next -> Install new components.
13. Select Betaface COM component DLL, BetafaceCom.dll, click ok. Click add and select BetafaceComPro.dll if you have Pro version of SDK and click Ok -> Next -> Finish.
14. Open Components folder of Betaface COM+ application and you should see all Betaface Com objects installed there.
15. Right click on each component in Betaface COM+ application, click Properties
16. On tab Activation set check "Enable Object Pooling", set Minimum pool size = 2, maximum pool size 4, creation timeout 120000.
17. If your server have only 1-2 gb of memory set Minimum pool size = 1, maximum pool size = 2 in step 16 and you may need to decrease pool size in step 11 from 2 to 1.
18. Right click on Betaface COM+ application and click Start.

Installing Betaface software updates

1. Open Component Services from Start -> Settings -> Control Panel -> Administrative Tools or from the server management window.
2. Navigate to Component Services -> Computers -> My Computer -> COM+ Applications.
3. Right click on Betaface COM+ application and select Disable.
4. Right click on Betaface COM+ application again and select Shut Down.
5. Create backup copy of old binaries.
6. Overwrite old binary files with new ones.
7. In Component Services right click on Betaface COM+ application and select Enable.
8. Right click on Betaface COM+ application again and select Start.