sgcBiometrics 2.1

Table of Contents

sgcBiometrics | Windows Biometric Framework

Every individual has unique characteristics that can be used for identification. Typically these characteristics are physical and include traits such as fingerprints, but they can also include behavioral traits such as gait and typing rhythm. The term biometrics encompasses both meanings. Biometric information is increasingly replacing passwords to identify and verify users. It is more secure and often more convenient for both user and administrator.

Sensors are used to capture biometric information. The information is captured by the sensor as a biometric sample. A single sample contains data that represents a single biometric characteristic for one individual. Multiple samples are averaged to create a biometric template, and the template is securely stored. Later, a sample from an unknown user is compared to the stored templates to establish and verify user identity. The Windows Biometric service, part of the Windows Biometric Framework (WBF), provides the following functionality. You can use the Windows Biometric Framework API to leverage this functionality.

- Captures biometric samples and uses them to create a template.
- Securely saves and retrieves biometric templates.
- Maps each template to a unique identifier such as a GUID or SID.

There are two types of sensor pools: **private** and **system**. The **fingerprint sensors** compatible with Windows Hello **support System pool** but **only some of them support private sensor pool**.

To support both Windows authentication scenarios and vendor defined authentication, the Windows Biometric service organizes biometric units into three possible sensor pools:

- **Private pool:** a collection of biometric units allocated for exclusive use by a client application. Private pools can support authentication scenarios that are not Windows-based, and they make it possible for an application to access the hardware of a biometric unit in a vendor-defined fashion. There can be as many private sensor pools on the system as there are biometric units.
- **System sensor pool:** a collection of sharable biometric units that provide access to Windows authentication services. This pool is used by Winlogon, UAC, and any other client that associates a SID with a specific biometric template. Each biometric service provider has one system sensor pool.

Applications can use the shared system pool or they can create a private pool made up of biometric units removed from the system or unassigned pools. When an application releases its private pool, the biometric units are reconfigured and returned to their original pools. To prevent denial of service attacks, only privileged users are permitted to remove the last sensor from the system pool. For more information, see the following topics.

sgcBiometrics currently has the following components:

1. **TsgcWinBioFingerPrint:** allows to login using a **fingerprint sensor**.
2. **TsgcWinBioFacial:** allows to detect face presence, **facial recognition** and more. (*only system sensor pool is supported).
3. **TsgcWinBioStorageFile:** allows to set a private pool where biometric units are allocated for use by a client application (*only Fingerprint sensors support StorageFile).
4. **TsgcWinBioUsersINI:** if you link this component to a TsgcWinBioFingerPrint object, when you try to enroll a new biometric sample, you cat save user data like username, userid... and when you identify a fingerprint, you can recover this data from INI file (*only Fingerprint sensors support StorageFile)

SensorPools

sgcBiometrics | System Sensor Pool

The system sensor pool is a collection of sharable biometric units that provide access to Windows authentication services. This pool is used by Winlogon, UAC, and any other client that associates a SID with a specific biometric template. Biometric units in the system pool:

- Can be shared by multiple client applications.
- Send event notices generated by the completion of biometric operations only to the application that has current window focus.
- Use account SIDs to represent the template identities. All of the templates associated with a single user account are tagged with the SID assigned to that account.

Depend on trustworthy template storage provided by the Windows Biometric Service.

A biometric unit can be included in the system pool if it can be:

- Configured to operate in basic mode and act only as a biometric capture device.
- Configured to operate in advanced mode but has no onboard template storage. That is, it must use the storage adapter and template store supplied by Microsoft.
- Configured to operate in advanced mode, contains onboard template storage, and can generate the required hashes.

Check TsgcWinBioFingerPrint or TsgcWinBioFacial to see how to use.

sgcBiometrics | Private Sensor Pool

A private sensor pool is a collection of biometric units reserved for exclusive use by a client application. Private pools support proprietary authentication methods and enable a client application to access a biometric unit by using vendor-specified control commands. Biometric units in a private sensor pool:

- Are available only to the client application that created the pool.
- Send event notices generated by the completion of biometric operations directly to the application without regard to current window focus.

- Use GUIDs to identify biometric templates.
- Share a single, application selected template database.

Private sensor pools must be used if the client application:

- Manages a collection of dedicated biometric units that use an application-specific template database. For an example, consider an employee attendance application where employees signal their arrival at work by swiping their finger on a fingerprint reader. The employees do not have Windows accounts on the computer running the application. Instead, their fingerprints are identified by GUIDs managed by the attendance application.
- Collects biometric samples rather than simply maps samples to SIDs.
- Places the biometric unit hardware into maintenance mode to update the firmware.
- Sends vendor-defined control commands to the biometric unit hardware or software.
- Attempts to configure a biometric unit with onboard storage to operate in advanced mode but the unit cannot perform the required hashing operations.
- Runs from a Remote Desktop client session.

*Currently, the private sensor pool is only supported by some fingerprints sensors.

Components

Components | TsgcWinBioFingerPrint

This component allows to login using a FingerPrint reader. By default uses System Sensor Pool, but if you attach a Storage Component, like TsgcWinBioStorageFile, you can setup your own Private Sensor Pool.

Requirements

- A compatible fingerprint sensor with support for Windows Hello.
- Windows 10+ (desktop apps only)
- Windows Server 2016+ (desktop apps only).

Basic Usage (System Sensor Pool)

1. Drop a **TsgcWinBioFingerPrint** in any form or datamodule.

2. Call **InitializeSensors** method to start to use your sensor.

```
TsgcWinBioFingerPrint1.InitializeSensors;
```

a. If initialization is successful, then **OnEnumBiometricUnit** event will be called.
b. If there is any error, **OnError** event is raised.

3. To save your FingerPrint in a Storage Template, call **EnrollBiometric** method

```
TsgcWinBioFingerPrint1.EnrollBiometric(aSubType);
```

Where aSubType can be any of the following:

```
WINBIO_ANSI_381_POS_RH_THUMB
WINBIO_ANSI_381_POS_RH_INDEX_FINGER
WINBIO_ANSI_381_POS_RH_MIDDLE_FINGER
WINBIO_ANSI_381_POS_RH_RING_FINGER
WINBIO_ANSI_381_POS_RH_LITTLE_FINGER

WINBIO_ANSI_381_POS_LH_THUMB
WINBIO_ANSI_381_POS_LH_INDEX_FINGER
WINBIO_ANSI_381_POS_LH_MIDDLE_FINGER
WINBIO_ANSI_381_POS_LH_RING_FINGER
WINBIO_ANSI_381_POS_LH_LITTLE_FINGER
```

Three Events will be called in Enroll process:

OnEnrollBegin: when enrollment starts.
OnEnrollCapture: every time you touch your sensor, you will get information about it, if it's correct sample, if it's bad...
OnErollCommit: when enrollment finishes.

You can check which fingerprints are already enrolled calling EnumEnrollments method:

OnEnumEnrollments: this event is called for every enrollment.

4. To verify if your fingerprint has been saved, just call **Identify** method

```
TsgcWinBioFingerPrint1.Identify;
```

OnIdentify event will be raised and If you a SID or GUID is provided, means sample exists.

```
  Case aIdentity._Type of
    WINBIO_ID_TYPE_SID:             vId             :=
 aIdentity.AccountSid.Data;
    WINBIO_ID_TYPE_GUID:             vId             :=
 aIdentity.TemplateGuid.Guid;
    else
      vId := '';
  End;

  case aSubfactor of
    WINBIO_ANSI_381_POS_RH_THUMB:     vSubFactor     :=
'RH_THUMB';
    WINBIO_ANSI_381_POS_RH_INDEX_FINGER: vSubFactor  :=
'RH_INDEX_FINGER';
    WINBIO_ANSI_381_POS_RH_MIDDLE_FINGER:    vSubFactor
:= 'RH_MIDDLE_FINGER';
    WINBIO_ANSI_381_POS_RH_RING_FINGER:  vSubFactor  :=
'RH_RING_FINGER';
    WINBIO_ANSI_381_POS_RH_LITTLE_FINGER:    vSubFactor
:= 'RH_LITTLE_FINGER';
    WINBIO_ANSI_381_POS_LH_THUMB:     vSubFactor     :=
'LH_THUMB';
    WINBIO_ANSI_381_POS_LH_INDEX_FINGER: vSubFactor  :=
'LH_INDEX_FINGER';
    WINBIO_ANSI_381_POS_LH_MIDDLE_FINGER:    vSubFactor
:= 'LH_MIDDLE_FINGER';
    WINBIO_ANSI_381_POS_LH_RING_FINGER:  vSubFactor  :=
'LH_RING_FINGER';
    WINBIO_ANSI_381_POS_LH_LITTLE_FINGER:    vSubFactor
:= 'LH_LITTLE_FINGER';
  end;
```

5. To capture a biometric sample just call CaptureSample method

```
TsgcWinBioFingerPrint1.CaptureSample;
```

OnCaptureSample event will be raised is result is successful.

The fingerprint image data is just a grayscale bitmap. Assuming you have a correctly-sized bitmap available, here's how to draw the bitmap. (Note, I'm assuming here that the fingerprint image is 8 bits per pixel. Actually, the pixel width is stored in the aAnsiBdbHeader, and a real application would have to use the value found there to calculate the proper X and Y values, based on pixel size.)

```
    for y := 0 to aAnsiBdbRecord.VerticalLineLength -
1 do
    begin
      for x := 0 to
aAnsiBdbRecord.HorizontalLineLength - 1 do
      begin
        vByte := aFirstPixel[(y *
aAnsiBdbRecord.HorizontalLineLength) + x];
        vRGB := RGB(vByte, vByte, vByte);
      end;
    end;
```

Asynchronous

By default, FingerPrint components uses **Synchronous** calls to request info about biometric devices, this mean that when you request user put fingerprint on sensor, application will wait till user has done the action and this locks application.

If you set **Asynchronous** property to True, all calls will be processed asynchronously, so application won't lock.

Components | TsgcWinBioFacial

Facial Recognition is supported using a compatible Windows Hello camera, this allows to monitor if there is a human person in front of the camera (you can know if there is someone, identify, arrives, departs and more). Facial Recognition and Identification is supported too.

Facial Recognition is **only supported by system sensor pool**, so you must first enroll a user using Windows Hello (WBF doesn't allow to enroll faces, only fingerprints are supported). You can register more than one face in your windows, you only must create a new windows account and attach the new face to this account.

Requirements

- A compatible camera with support for Windows Hello.
- Windows 10+ (desktop apps only)
- Windows Server 2016+ (desktop apps only).

Log in Windows 10 with your face

How to log in to Windows 10 with your face

- *Go to Settings > Accounts > Sign-in options.*
- *Set up an account password and PIN.*
- *Click the "Set up" button for Face under Windows Hello.*
- *Click the "Get started" button, enter your PIN, and sit in front of the camera while Windows takes a few seconds to scan your face.*
- *Click "Close" and you're all set.*

Basic Usage

1. Drop a **TsgcWinBioFacial** in any form or datamodule.

2. Check if there is a Session opened, if not, call **InitializeSensors** method to start to use your sensor. The Facial Recognition component uses Asynchronous mode, so set a Timeout to Initialize sensors (in milliseconds).

```
if not TsgcWinBioFacial1.SessionIsOpen then
  TsgcWinBioFacial1.InitializeSensors(10000);
```

a. If initialization is successful, then **OnEnumBiometricUnit** event will be called.
b. If there is any error, **OnError** event is raised.

3. Call **FacialRecognize** to try to recognize the face, if returns true, means that face has been recognized.

```
if TsgcWinBioFacial1.FacialRecognize then
  ShowMessage('Face Recognized')
else
  ShowMessage('Unknown Face');
```

4. You can get the Face Identification Id calling the method **FacialIdentify**, if successful returns the AccountSid as a string.

```
ShowMessage(TsgcWinBioFacial1.FacialIdentify);
```

Presence Monitor

This method activate the sensor of your windows camera and every time there is a new face, departs or arrives you will be notified.

1. Drop a **TsgcWinBioFacial** in any form or datamodule.

2. Check if there is a Session opened, if not, call **InitializeSensors** method to start to use your sensor. The Facial Recognition component uses Asynchronous mode, so set a Timeout to Initialize sensors (in milliseconds).

```
if not TsgcWinBioFacial1.SessionIsOpen then
  TsgcWinBioFacial1.InitializeSensors(10000);
```

3. Call **StartMonitorPresence** to start the presence monitor. Every time there is a change in your camera you will be notified

```
TsgcWinBioFacial1.StartMonitorPresence;
```

You can use the following events to track all changes:

- The type of event is unknown. This value is used for the uninitialized structure.

```
procedure OnFacialPresenceUnknown(Sender: TObject;
const aUnitId: Integer);
begin
  WriteLn('Monitor Presence: UNKNOWN');
end;
```

- Provides information about all of the faces current in the camera frame.

```
procedure        OnFacialPresenceUpdateAll(Sender:
TObject; const aUnitId: Integer);
begin
  WriteLn('Monitor Presence: UPDATE ALL');
end;
```

- A new face entered the camera frame.

```
procedure  OnFacialPresenceArrive(Sender:  TObject;
const aUnitId: Integer;
  const aPresence: WINBIO_PRESENCE);
begin
  WriteLn('Monitor       Presence       ['       +
IntToStr(aPresence.TrackingId) + ']: ARRIVE');
end;
```

- A face was matched to an enrolled user.

```
procedure        OnFacialPresenceRecognize(Sender:
TObject; const aUnitId:
  Integer; const aPresence: WINBIO_PRESENCE);
begin
  if aPresence.Identity._Type = WINBIO_ID_TYPE_SID
then
    WriteLn('Monitor      Presence      ['      +
IntToStr(aPresence.TrackingId) + ']: RECOGNIZE ' +
      aPresence.Identity.AccountSid.Data)
  else
    WriteLn('Monitor      Presence      ['      +
IntToStr(aPresence.TrackingId) + ']: RECOGNIZE');
end;
```

- A previously detected face has been out of the camera frame for a period of time.

```
procedure   TFRMFacial.FacialPresenceDepart(Sender:
TObject; const aUnitId:
    Integer; const aPresence: WINBIO_PRESENCE);
begin
  WriteLn('Monitor       Presence       ['       +
IntToStr(aPresence.TrackingId) + ']: DEPART');
end;
```

- Provides updates information about the bounding box and reject detail values for a subset of the faces that are currently in the camera frame.

```
procedure    TFRMFacial.FacialPresenceTrack(Sender:
TObject; const aUnitId:
    Integer; const aPresence: WINBIO_PRESENCE);
var
  vReject: string;
begin
  vReject := '';
  case aPresence.RejectDetail of
    WINBIO_FACE_TOO_BRIGHT:  vReject  :=  'FACE  Too
Bright';
    WINBIO_FACE_TOO_DARK:  vReject  :=  'FACE  Too
Dark';
    WINBIO_FACE_SPOOF_DETECTED:  vReject  :=  'FACE
Spoof Detected';
    WINBIO_FACE_AMBIGUOUS_TARGET: vReject := 'FACE
Ambiguous Target';
    WINBIO_FACE_EYES_OCCLUDED:  vReject  :=  'FACE
Eyes Occluded';
    WINBIO_FACE_WRONG_ORIENTATION:    vReject    :=
'FACE Wrong Orientation';
```

```
    WINBIO_FACE_TOO_HIGH:  vReject  :=  'FACE  Too
High';
    WINBIO_FACE_TOO_LOW:  vReject  :=  'FACE  Too
Low';
    WINBIO_FACE_TOO_LEFT:  vReject  :=  'FACE  Too
Left';
    WINBIO_FACE_TOO_RIGHT:  vReject  :=  'FACE  Too
Right';
    WINBIO_FACE_TOO_NEAR:  vReject  :=  'FACE  Too
Near';
    WINBIO_FACE_TOO_FAR:  vReject  :=  'FACE  Too
Far';
  end;
  WriteLn('Monitor       Presence       ['       +
IntToStr(aPresence.TrackingId)  +  ']:  TRACK  ['  +
vReject + ']');
end;
```

Multiple Facial Recognition

If you want to recognize more than 1 user on the same windows machine, first you must create one windows account for every user and for every new user created, enroll the face on the windows hello settings.

This way, every face will have a windows account attached.

After that, you can use the Facial Recognition component to Recognize ALL Faces enrolled in the windows machine (you can recognize faces of other accounts if they are saved on the same windows).

Components | TsgcWinBioStorageFile

This component allows to setup a private sensor pool if attached to a biometric component like TsgcWinBioFingerPrint.
Not all fingerprint sensors support private sensor pool, use our demo sample to test if your fingerprint sensor supports it.

Basic Usage (Private Sensor Pool)

1. Drop a **TsgcWinBioFingerPrint** in any form or datamodule.

2. Drop a **TsgcWinBioStorageFile** in any form or datamodule.

3. Link **TsgcWinBioFingerPrint.Storage** to **TsgcWinBioStorageFile**.

4. In **TsgcWinBioStorageFile** you can customize your own **DatabaseId** (must be a GUID).

5. Call InitializeSensors method to start to use your sensor.

```
TsgcWinBioFingerPrint1.InitializeSensors;
```

a. If initialization is successful, then **OnEnumBiometricUnit** event will be called.
b. If there is any error, **OnError** event is raised.

6. To save your FingerPrint in a Storage Template, call **EnrollBiometric** method

```
TsgcWinBioFingerPrint1.EnrollBiometric(aSubType);
```

Where aSubType can be any of the following:

```
WINBIO_ANSI_381_POS_RH_THUMB
WINBIO_ANSI_381_POS_RH_INDEX_FINGER
WINBIO_ANSI_381_POS_RH_MIDDLE_FINGER
WINBIO_ANSI_381_POS_RH_RING_FINGER
WINBIO_ANSI_381_POS_RH_LITTLE_FINGER

WINBIO_ANSI_381_POS_LH_THUMB
WINBIO_ANSI_381_POS_LH_INDEX_FINGER
WINBIO_ANSI_381_POS_LH_MIDDLE_FINGER
WINBIO_ANSI_381_POS_LH_RING_FINGER
WINBIO_ANSI_381_POS_LH_LITTLE_FINGER
```

Three Events will be called in Enroll process:

OnEnrollBegin: when enrollment starts.
OnEnrollCapture: every time you touch your sensor, you will get information about it, if it's correct sample, if it's bad...
OnErollCommit: when enrollment finishes.

7. To verify if your fingerprint has been saved, just call **Identify** method

```
TsgcWinBioFingerPrint1.Identify;
```

OnIdentify event will be raised and If you a SID or GUID is provided, means sample exists.

```
  Case aIdentity._Type of
    WINBIO_ID_TYPE_SID:              vId              :=
 aIdentity.AccountSid.Data;
    WINBIO_ID_TYPE_GUID:              vId              :=
 aIdentity.TemplateGuid.Guid;
    else
```

```
        vId := '';
    End;

    case aSubfactor of
      WINBIO_ANSI_381_POS_RH_THUMB:     vSubFactor     :=
  'RH_THUMB';
      WINBIO_ANSI_381_POS_RH_INDEX_FINGER: vSubFactor :=
  'RH_INDEX_FINGER';
      WINBIO_ANSI_381_POS_RH_MIDDLE_FINGER:    vSubFactor
  := 'RH_MIDDLE_FINGER';
      WINBIO_ANSI_381_POS_RH_RING_FINGER: vSubFactor  :=
  'RH_RING_FINGER';
      WINBIO_ANSI_381_POS_RH_LITTLE_FINGER:    vSubFactor
  := 'RH_LITTLE_FINGER';
      WINBIO_ANSI_381_POS_LH_THUMB:     vSubFactor     :=
  'LH_THUMB';
      WINBIO_ANSI_381_POS_LH_INDEX_FINGER: vSubFactor :=
  'LH_INDEX_FINGER';
      WINBIO_ANSI_381_POS_LH_MIDDLE_FINGER:    vSubFactor
  := 'LH_MIDDLE_FINGER';
      WINBIO_ANSI_381_POS_LH_RING_FINGER: vSubFactor  :=
  'LH_RING_FINGER';
      WINBIO_ANSI_381_POS_LH_LITTLE_FINGER:    vSubFactor
  := 'LH_LITTLE_FINGER';
    end;
```

Components | TsgcWinBioUsersINI

If you require save user data associated to a fingerprint, example: you can user System Pool Sensor to save fingerprints of different users (until the limit of fingerprints types associated to a single database). You can link this component to a TsgcWinBioFingerPrint component and every time you enroll a new biometric sample, you can save user data like: username, user id...

How works

1. Drop a TsgcWinBioFingerPrint component.
2. Drop a TsgcWinBioUsersINI component.
3. Link TsgcWinBioFingerPrint.Users property to TsgcWinBioUsersINI object.
4. Handle TsgcWinBioUsersINI events to set and get user data.

OnEnrollUser

```
procedure
TFRMFingerPrint.sgcWinBioUsersINI1EnrollUser(Sender:
TObject; const
    User: TsgcBiometrics_WinBio_User);
begin
  User.UserId := '0001';
  User.UserName := 'John';
  User.UserData := '<xml><phone>656545644</phone></xml>';
  User.UserSubType := WINBIO_ANSI_381_POS_RH_THUMB;
end;
```

OnIdentifyUser

```
procedure
TFRMFingerPrint.sgcWinBioUsersINI1IdentifyUser(Sender:
TObject; const
    aUnitId:  Integer;  const  aIdentity:  WINBIO_IDENTITY;
const aSubFactor:
    WINBIO_BIOMETRIC_SUBTYPE;      const      aRejectDetail:
WINBIO_REJECT_DETAIL; const
    aUser: TsgcBiometrics_WinBio_User);
var
  vSubFactor: String;
begin
  case aSubfactor of
    WINBIO_ANSI_381_POS_RH_THUMB: vSubFactor := 'RH_THUMB';
    WINBIO_ANSI_381_POS_RH_INDEX_FINGER:   vSubFactor   :=
'RH_INDEX_FINGER';
    WINBIO_ANSI_381_POS_RH_MIDDLE_FINGER:   vSubFactor   :=
'RH_MIDDLE_FINGER';
    WINBIO_ANSI_381_POS_RH_RING_FINGER:    vSubFactor    :=
'RH_RING_FINGER';
    WINBIO_ANSI_381_POS_RH_LITTLE_FINGER:   vSubFactor   :=
'RH_LITTLE_FINGER';
    WINBIO_ANSI_381_POS_LH_THUMB: vSubFactor := 'LH_THUMB';
    WINBIO_ANSI_381_POS_LH_INDEX_FINGER:   vSubFactor   :=
'LH_INDEX_FINGER';
    WINBIO_ANSI_381_POS_LH_MIDDLE_FINGER:   vSubFactor   :=
'LH_MIDDLE_FINGER';
    WINBIO_ANSI_381_POS_LH_RING_FINGER:    vSubFactor    :=
'LH_RING_FINGER';
    WINBIO_ANSI_381_POS_LH_LITTLE_FINGER:   vSubFactor   :=
'LH_LITTLE_FINGER';
  end;
end;
```

Index