Documentation signoMobileCapture API Win

Software components for capturing
signatures on mobile devices

signotec GmbH
Am Gierath 20 b
40885 Ratingen, Germany

(+49-2102) 53575-10
www.signotec.com
support@signotec.de

Version 1.4 as of 18.10.2018

Contents

Legal notice

1 Document history

Version	Date	Person responsible	Status/note
1.0	13 April 2016	Thorsten Stelter	Document created
1.1	2 May 2016	Thorsten Stelter	COM interface additions made
1.2	20 July 2016	Thorsten Stelter	Events added and amended
1.3	29 July 2016	Thorsten Stelter	Changes to "SignatureDataReceived"
1.4	10 Oct. 2018	Thorsten Stelter	Added properties "SignaturePenColor" and "SignatureTransparency" of the MobileDevice class

2 Function overview

signoMobileCapture API for Windows consists of an assembly for .NET 4.0 Client Profile. The assembly offers the classes that are needed to facilitate communication with mobile devices. In addition to this, it contains a Windows Forms control, which makes it possible to display the signature on a computer in real time.

The assembly also offers a COM interface. The two main classes of the .NET interface have been outlined here in four interfaces that have the same range of functions. Instead of the Windows Forms control, an ActiveX control with the same function has been made available here.

The use of 'Bonjour' means that no configuration is required. The mobile device just needs to be in the same network as the PC and the necessary app needs to have been installed and launched on the mobile device.

Inserting the signatures that have been captured using this API into PDF documents requires the use of additional signoAPI components (e.g., SignPDF3). For more information about using these components, please refer to the relevant documentation.

3 System requirements and integration

For the development of applications using signoMobileCapture API, the requirements are as follows:

- A Windows PC (XP or higher, with SP3)
- .NET 4.0 Client Profile
- VC++ Runtime 10.0 (VS 2010)
- Bonjour

Please note: The signoAPI only contains x86 (32-bit) components. This means that applications using this API must also be compiled for x86 (32-bit). Please make the necessary adjustments to your development environment settings.

3.1 .NET

signoMobileCapture API consists of the following files:

- signoMobileCapture.dll
- signoMobileCapture.xml
- signoSignatureUtils.dll

The XML file is only required on the development PC. It contains information for Visual Studio's IntelliSense features.

The project settings simply require that a reference is set to 'signoMobileCapture.dll'. The 'signoSignatureUtils.dll' file is only used internally by this API. For this reason, it is not documented and there is no need to set a reference to this DLL.

3.2 COM

signoMobileCapture API consists of the following files:

- signoMobileCapture.dll
- signoMobileCapture.tlb
- signoSignatureUtils.dll

For developing using the COM interface, there are two special features to be considered.

The first is to do with the components for a .NET assembly. Thus `regasm.exe` is to be used for registration, instead of `regsvr32.exe`. When using `regasm.exe`, ensure that you are using the correct version, i.e., the 32-bit version of .NET 4 Framework. On the development PC, the setup registration is performed by the signoAPI, which means it does not need to be performed manually.

The second is the fact that numerous events are used in these components and it is imperative that these events are evaluated. For this purpose, a custom EventSink must be implemented in C++. An implementation example can be found in the C++ demo.

4 MobileCapture class

The `MobileCapture` class is the starting point for signoMobileCapture API. This class is responsible for Bonjour communication and makes appropriate instances of the `MobileDevice` class available to the located mobile devices.

```
class MobileCapture : IDisposable
```

Usage:

```
MobileCapture mobileCapture = new MobileCapture();
```

When using COM, instead of this class, the two interfaces `IMobileCapture` and `IMobileCaptureEvents` are available. These interfaces offer an identical range of functions.

Usage:

```
HRESULT hresult =
pMobileCapture.CreateInstance(signoMobileCapture::CLSID_MobileCaptureCOM);
if (SUCCEEDED(hresult))
{
    pMobileCapture->StartService();
}
```

4.1 (Static) SetLicenseKey method

This method can be used to pass the acquired licence key for the 'signoAPI' and thus to enable the API. If this method is not called or if an invalid licence key is passed, the signature is given a demo stamp, both in `MobileControl` and in the properties of the `SignatureConfirmed` events in the `MobileDevice` class.

Parameter	Description
`string licenseKey`	Ein gültiger Lizenzschlüssel für das „signoAPI".
Return value	**Description**
-	-

4.1.1 .NET

```
static void SetLicenseKey(string licenseKey)
```

Usage:

```
MobileCapture.SetLicenseKey("abc123def456");
```

4.1.2 COM

```
void SetLicenseKey(BSTR licenseKey)
```

Usage:

```
pMobileCapture->SetLicenseKey(CComBSTR("abc123def456"));
```

4.2 Dispose method

In cases where the `MobileCapture` class is not instantiated via a 'using' block (only .NET), the `Dispose` method must be called as soon as the instance is no longer required. As a result, the Bonjour service is stopped (if it has been running) and other cleanup operations are performed.

Parameter	Description
-	-
Return value	**Description**
-	-

4.2.1 .NET

```
void Dispose()
```

Usage:

```
mobileCapture.Dispose();
```

4.2.2 COM

```
void Dispose()
```

Usage:

```
pMobileCapture->Dispose();
```

4.3 StartService method

This method launches the Bonjour service. Once this has taken place, the respective events will be triggered for all located mobile devices and for any errors that may occur.

Please note: It is highly recommended that appropriate handlers are registered for all events in this class before running the service!

Parameter	Description
-	-
Return value	**Description**
-	-

4.3.1 .NET

```
void StartService()
```

Usage:

```
mobileCapture.StartService();
```

4.3.2 COM

```
void StartService()
```

Usage:

```
pMobileCapture->StartService();
```

4.4 MobileDeviceFound event

This event is called whenever the current service locates a mobile device.

4.4.1 .NET

```
event MobileDeviceFoundHandler MobileDeviceFound;
```

The `MobileDeviceEventArgs` contains the following property:

Property	Description
`MobileDevice Device`	Instance of the `MobileDevice` class for the located mobile device.

Usage:

```
private void MobileCapture_MobileDeviceFound(Object sender,
    MobileDeviceEventArgs e)
{
    MessageBox.Show(String.Format("New mobile device found: {0}",
        e.Device.Name));
}
```

4.4.2 COM

The dispatcher ID of the event is 1.

```
void OnMobileDeviceFound(IDispatch* device)
```

Property	Description
`IDispatch* device`	`IMobileDevice` instance for the located mobile device.

Usage:

```
HRESULT CCaptureEventSink::Invoke(DISPID dispIdMember, ...)
{
    switch(dispIdMember)
    {
    case 1L: // Mobile Device Found
        {
            IDispatch* mobileDevice = pDispParams->rgvarg[0].pdispVal;

            OnMobileDeviceFound(mobileDevice);
            break;
        }
    }

    return S_OK;
}
```

4.5 MobileDeviceLost event

This event is called whenever the current service can no longer contact a mobile device that it had previously located.

4.5.1 .NET

```
event MobileDeviceLostHandler MobileDeviceLost;
```

The `MobileDeviceEventArgs` contains the following property:

Property	Description
`MobileDevice Device`	Instance of the `MobileDevice` class for the mobile device that can no longer be contacted.

Usage:

```
private void MobileCapture_MobileDeviceLost(Object sender,
    MobileDeviceEventArgs e)
{
    MessageBox.Show(String.Format("Mobile device lost: {0}", e.Device.Name));
}
```

4.5.2 COM

The dispatcher ID of the event is 2.

```
void OnMobileDeviceLost(IDispatch* device)
```

Property	Description
`IDispatch* device`	`IMobileDevice` instance for the mobile device that can no longer be contacted.

Usage:

```
HRESULT CCaptureEventSink::Invoke(DISPID dispIdMember, ...)
{
    switch(dispIdMember)
    {
    case 2L: // Mobile Device Lost
        {
            IDispatch* mobileDevice = pDispParams->rgvarg[0].pdispVal;

            OnMobileDeviceLost(mobileDevice);
            break;
        }
    }

    return S_OK;
}
```

4.6 BonjourError event

This event is called whenever one of the asynchronous Bonjour methods reports a specific error.

The possible error codes are listed in the chapter entitled 'BonjourErrorCode list'.

4.6.1 .NET

```
event BonjourErrorHandler BonjourError;
```

The `BonjourErrorEventArgs` contains the following property:

Property	Description
`BonjourErrorCode ErrorCode`	The error code reported by Bonjour.

Usage:

```
private void MobileCapture_BonjourError(object sender,
    BonjourErrorEventArgs e)
{
    MessageBox.Show(String.Format("Bonjour error code: {0} ({1})",
        e.ErrorCode.ToString(), (int)e.ErrorCode));
}
```

4.6.2 COM

The dispatcher ID of the event is 3.

```
void OnBonjourError(int errorCode);
```

Property	Description
`int errorCode`	The error code reported by Bonjour.

Usage:

```
HRESULT CCaptureEventSink::Invoke(DISPID dispIdMember, ...)
{
    switch(dispIdMember)
    {
    case 3L: // Bonjour error
        {
            int errorCode = pDispParams->rgvarg[0].intVal;

            // Do something with the error code...
            break;
        }
    }

    return S_OK;
}
```

4.7 MobileCaptureError event

This event is called whenever one of the current service's asynchronous internal methods throws an exception.

4.7.1 .NET

```
event MobileCaptureErrorHandler MobileCaptureError;
```

The `MobileCaptureErrorEventArgs` class contains the following property:

Property	Description
`Exception Exception`	The exception thrown by the internal method.

Usage:

```
private void MobileCapture_MobileCaptureError(object sender,
    MobileCaptureErrorEventArgs e)
{
    MessageBox.Show(e.Exception.Message);
}
```

4.7.2 COM

The dispatcher ID of the event is 4.

```
void OnMobileCaptureError(BSTR errorMessage)
```

Property	Description
`BSTR errorMessage`	The error message from the exception thrown by the internal method.

Usage:

```
HRESULT CCaptureEventSink::Invoke(DISPID dispIdMember, ...)
{
    switch(dispIdMember)
    {
    case 4L: // Mobile Capture error
        {
            BSTR errorMessage = pDispParams->rgvarg[0].bstrVal;

            // Do something with the error message...
            break;
        }
    }

    return S_OK;
}
```

4.8 BonjourErrorCode list

The `BonjourErrorCode` list contains the following values:

Code	Value
`PollingMode`	-65567
`NoRouter`	-65566
`NATPortMappingDisabled`	-65565
`NATPortMappingUnsupported`	-65564
`ServiceNotRunning`	-65563
`Transient`	-65562
`BadKey`	-65561
`BadSig`	-65560
`BadTime`	-65559
`DoubleNAT`	-65558
`NATTraversal`	-65557
`NoSuchKey`	-65556

NoAuth	-65555
NoSuchRecord	-65554
Refused	-65553
BadInterfaceIndex	-65552
Incompatible	-65551
Firewall	-65550
Invalid	-65549
NameConflict	-65548
AlreadyRegistered	-65547
NotInitialized	-65545
Unsupported	-65544
BadFlags	-65543
BadState	-65542
BadReference	-65541
BadParam	-65540
NoMemory	-65539
NoSuchName	-65538
Unknown	-65537
NoError	0

5 MobileDevice class

The `MobileDevice` class represents a mobile device. It contains methods for connecting and disconnecting a device and for starting and controlling the signature process. Furthermore, it offers a range of properties and events.

When using COM, the two interfaces `IMobileDevice` and `IMobileDeviceEvents` are available instead of the class. These interfaces offer an identical range of functions.

Please note: It is not possibly to directly create instances of this class. Corresponding instances for mobile devices are provided via the `MobileDeviceFound` event in the `MobileCapture` class.

Usage:

```
private void MobileCapture_MobileDeviceFound(Object sender,
    MobileDeviceEventArgs e)
{
    MobileDevice mobileDevice = e.Device;
}
```

In COM:

```
void CCaptureEventSink::OnMobileDeviceFound(IDispatch* mobileDevice)
{
    signoMobileCapture::IMobileDevicePtr pMobileDevice = mobileDevice;
}
```

5.1 Name property

This property makes the name of the mobile device available.

```
string Name { get; }
```

Usage:

```
MessageBox.Show(mobileDevice.Name);
```

In COM:

```
BSTR* name;
pMobileDevice->get_Name(name);
```

5.2 Width property

This property makes the width (in pixels) of the mobile device's display available.

Please note: This property is only available if the device has been successfully connected. If this property is requested without a connection to the device, an exception is thrown.

```
int Width { get; }
```

Usage:

```
Control.Width = mobileDevice.Width;
```

In COM:

```
long* width;
pMobileDevice->get_Width(width);
```

5.3 Height property

This property makes the height (in pixels) of the mobile device's display available.

Please note: This property is only available if the device has been successfully connected. If this property is requested without a connection to the device, an exception is thrown.

```
int Height { get; }
```

Usage:

```
Control.Height = mobileDevice.Height;
```

In COM:

```
long* height;
pMobileDevice->get_Height(height);
```

5.4 SignatureResolution property

This property makes it possible to read and set the desired resolution of the signature image.

Please note: The signature image is sent by the `SignatureConfirmed` event.

```
double SignatureResolution { get; set; }
```

Usage:

```
SignatureResolution = 250.0;
```

In COM:

```
pMobileDevice->put_SignatureResolution(250.0);
```

5.5 SignatureWidth property

This property makes it possible to read and set the desired width of the signature image.

Please note: The signature image is sent by the `SignatureConfirmed` event.

```
int SignatureWidth { get; set; }
```

Usage:

```
SignatureWidth = 300;
```

In COM:

```
pMobileDevice->put_SignatureWidth(300);
```

5.6 SignatureHeight property

This property makes it possible to read and set the desired height of the signature image.

Please note: The signature image is sent by the `SignatureConfirmed` event.

```
int SignatureHeight { get; set; }
```

Usage:

```
SignatureHeight = 100;
```

In COM:

```
pMobileDevice->put_SignatureHeight(100);
```

5.7 SignaturePenColor property

This property makes it possible to read and set the desired color of the signature. The chosen color will be used for the real-time view of the apps (Android and iOS since v2.2.0) as well as the control and the rendered signature image.

Available since: v1.2.6.0

Please note: The property must be set before the method `CaptureSignature` is called since the color information will be interchanged with the app by this call.

```
System.Drawing.Color SignaturePenColor { get; set; }
```

Usage:

```
SignaturePenColor = Color.FromKnownColor(KnownColor.Blue);
```

In COM:

```
pMobileDevice->put_SignaturePenColor(0xFF0000);
```

5.8 SignatureTransparency property

This property makes it possible to choose, if the background of the rendered signature image should be either transparent or white. If the value of this property is true (default) the image will be rendered with a transparent background.

Available since: v1.2.6.0

Please note: The property must be set before the method `ConfirmSignature` is called since the signature image will be rendered by this call and returned as an event parameter.

```
bool SignatureTransparency { get; set; }
```

Usage:

```
SignatureTransparency = false;
```

In COM:

```
pMobileDevice->put_SignatureTransparency(FALSE);
```

5.9 Dispose method

The `Dispose` method should be called as soon as the instance is no longer required. As a result, the connection to the device (if still in place) is terminated and other cleanup operations are performed.

Parameter	Description
-	-
Return value	**Description**
-	-

5.9.1 .NET

```
void Dispose()
```

Usage:

```
mobileDevice.Dispose();
```

5.9.2 COM

```
void Dispose()
```

Usage:

```
pMobileDevice->Dispose();
```

5.10 Connect method

The `Connect` method creates a connection to the mobile device and performs a handshaking process. Since these processes are asynchronous, this method does not send a return value and does not block the program run (or the thread). After this, the `DeviceConnected` event can be used to check if and when a connection was successfully made.

Parameter	Description
`string pairingCode` `BSTR pairingCode`	The pairing code displayed on the mobile device via the app.
Return value	**Description**
-	-

5.10.1 .NET

```
void Connect(string pairingCode)
```

Usage:

```
mobileDevice.Connect("ABC123");
```

5.10.2 COM

```
void Connect(BSTR pairingCode)
```

Usage:

```
pMobileDevice->Connect(CComBSTR("ABC123"));
```

5.11 Disconnect method

The `Disconnect` method ends the connection to the mobile device.

Parameter	Description
-	-
Return value	**Description**
-	-

5.11.1 .NET

```
void Disconnect()
```

Usage:

```
mobileDevice.Disconnect();
```

5.11.2 COM

```
void Disconnect()
```

Usage:

```
pMobileDevice->Disconnect();
```

5.12 captureSignature method

The `CaptureSignature` method starts the signature process on the mobile device. If a confirmation text is passed, this is then displayed on the mobile device. In such cases, the signature process itself only starts once the text has been accepted.

If this method is called without a connection to the device being in place, an exception is thrown.

Please note: The `SigningStarted` event is always called at the exact point at which the signature process begins on the mobile device, i.e., in cases where there is an upstream confirmation text, the exact moment when the text is accepted (if it is accepted). It may therefore be advisable to implement appropriate additional program steps in the relevant event handler. Such steps should not be implemented immediately after this method is called.

Parameter	Description
`string confirmationText` `BSTR confirmationText`	The confirmation text that is to be displayed on the mobile device.
Return value	**Description**
-	-

5.12.1 .NET

```
void CaptureSignature()
void CaptureSignature(string confirmationText)
```

Usage:

```
mobileDevice.CaptureSignature();
```

Or:

```
mobileDevice.CaptureSignature("Your confirmation text...");
```

5.12.2 COM

```
void CaptureSignature(BSTR confirmationText)
```

Usage:

```
pMobileDevice->CaptureSignature(CComBSTR("Your confirmation text..."));
```

5.13 RetrySignature method

The `RetrySignature` method discards all signature data that has been previously captured and removes the signature from the mobile device's display. The mobile device remains in the signature process, however.

If this method is called without a connection to the device being in place, an exception is thrown.

Parameter	Description
-	-
Return value	**Description**
-	-

5.13.1 .NET

```
void RetrySignature()
```

Usage:

```
mobileDevice.RetrySignature();
```

5.13.2 COM

```
void RetrySignature()
```

Usage:

```
pMobileDevice->RetrySignature();
```

5.14 CancelSignature method

The `CancelSignature` method discards all signature data that has been previously captured. The mobile device quits the signature process.

If this method is called without a connection to the device being in place, an exception is thrown.

Please note: The `SigningCanceled` event is called after this method has been called. Furthermore, it is also called when the 'cancel' button is pressed on the mobile device. Implementing appropriate program sequences in the relevant event handler is therefore highly recommended because it will mean that the sequences are processed in both scenarios.

Parameter	Description
-	-
Return value	**Description**
-	-

5.14.1 .NET

```
void CancelSignature()
```

Usage:

```
mobileDevice.CancelSignature();
```

5.14.2 COM

```
void CancelSignature()
```

Usage:

```
pMobileDevice->CancelSignature();
```

5.15 ConfirmSignature method

The `ConfirmSignature` method accepts the captured signature data. The mobile device quits the signature process. The captured signature data, along with a rendered image of the signature, are then made available via the `SignatureConfirmed` event.

If this method is called without a connection to the device being in place, an exception is thrown.

Please note: The `SignatureConfirmed` event is called after this method has been called. Furthermore, it is also called when the 'OK' button is pressed on the mobile device. Implementing appropriate program sequences in the relevant event handler is therefore highly recommended because it will mean that the sequences are processed in both scenarios.

Parameter	Description
-	-
Return value	**Description**
-	-

5.15.1 .NET

```
void ConfirmSignature()
```

Usage:

```
mobileDevice.ConfirmSignature();
```

5.15.2 COM

```
void ConfirmSignature()
```

Usage:

```
pMobileDevice->ConfirmSignature();
```

5.16 DeviceConnected event

This event is called whenever the attempt to establish a connection with the mobile device is ended.

The possible result codes are listed in the chapter entitled 'DeviceConnectResult list'.

5.16.1 .NET

```
event DeviceConnectedHandler DeviceConnected;
```

The `DeviceConnectedEventArgs` class contains the following property:

Property	Description
`DeviceConnectResult Result`	The result of the attempt to establish a connection with the mobile device.

Usage:

```
private void MobileDevice_Connected(object sender,
    DeviceConnectedEventArgs e)
{
    if (e.Result != DeviceConnectResult.Successful)
    {
        MessageBox.Show(e.Result.ToString());
    }
}
```

5.16.2 COM

The dispatcher ID of the event is 1.

```
void OnDeviceConnected(int result)
```

Property	Description
int result	The result of the attempt to establish a connection with the mobile device.

Usage:

```
HRESULT CDeviceEventSink::Invoke(DISPID dispIdMember, ...)
{
    switch(dispIdMember)
    {
    case 1L: // Device connected
        {
            int result = pDispParams->rgvarg[0].intVal;

            // Do something with the result code...
            break;
        }
    }

    return S_OK;
}
```

5.17 SigningStarted event

This event is called whenever the signature process is initiated on the mobile device. Thus, in cases where there is an upstream confirmation text, it also provides information about if and when the user accepts the text.

5.17.1 .NET

```
event EventHandler SigningStarted;
```

Usage:

```
private void MobileDevice_SigningStarted(object sender, EventArgs e)
{
    MessageBox.Show("Signature capture on the mobile device has started.");
}
```

5.17.2 COM

The dispatcher ID of the event is 2.

```
void OnSigningStarted()
```

Usage:

```
HRESULT CDeviceEventSink::Invoke(DISPID dispIdMember, ...)
{
    switch(dispIdMember)
    {
    case 2L: // Signing started
        {
            // Do something...
            break;
        }
    }

    return S_OK;
}
```

5.18 SignatureRetry event

This event is called whenever the signature process is repeated on the mobile device.

5.18.1 .NET

```
event EventHandler SigningStarted;
```

Usage:

```
private void MobileDevice_SignatureRetry(object sender, EventArgs e)
{
    MessageBox.Show("Signature capture on the mobile device was restarted.");
}
```

5.18.2 COM

The dispatcher ID of the event is 3.

```
void OnSignatureRetry()
```

Usage:

```
HRESULT CDeviceEventSink::Invoke(DISPID dispIdMember, ...)
{
    switch(dispIdMember)
    {
    case 3L: // SignatureRetry
        {
            // Do something...
            break;
        }
    }

    return S_OK;
}
```

5.19 SigningCanceled event

This event is called whenever the signature process is cancelled on the mobile device, regardless of whether this happened via the button on the mobile device or due to the `CancelSignature` method being called.

5.19.1 .NET

```
event EventHandler SigningCanceled;
```

Usage:

```
private void MobileDevice_SigningCanceled(object sender, EventArgs e)
{
    MessageBox.Show("Signature capture on the mobile device was canceled.");
}
```

5.19.2 COM

The dispatcher ID of the event is 4.

```
void OnSigningCanceled()
```

Usage:

```
HRESULT CDeviceEventSink::Invoke(DISPID dispIdMember, ...)
{
    switch(dispIdMember)
    {
    case 4L: // Signing canceled
        {
            // Do something...
            break;
        }
    }

    return S_OK;
}
```

5.20 SignatureConfirmed event

This event is called whenever the signature captured on the mobile device is confirmed, regardless of whether this happened via the button on the mobile device or due to the `ConfirmSignature` method being called.

Please note: The resolution, height and width of the signature image delivered by this event can be adjusted using `SignatureResolution`, `SignatureWidth` and `SignatureHeight`, which are the properties of this class.

5.20.1 .NET

```
event SignatureConfirmedHandler SignatureConfirmed;
```

The `SignatureConfirmedEventArgs` class contains the following properties:

Property	Description
byte[] SignData	The captured signature data.
int NumSignaturePoints	The number of points captured in the signature.
Image Image	A rendered image of the signature.

Usage:

```
private void MobileDevice_SignatureConfirmed(object sender,
    SignatureConfirmedEventArgs e)
{
    MessageBox.Show(e.NumSignaturePoints + " points have been captured.");
    File.WriteAllBytes(Path.Combine(Environment.GetFolderPath(
        Environment.SpecialFolder.Desktop), "SignData.sdb"), e.SignData);
    e.Image.Save(Path.Combine(Environment.GetFolderPath(
        Environment.SpecialFolder.Desktop), "Signature.png"),
ImageFormat.Png);
}
```

5.20.2 COM

The dispatcher ID of the event is 5.

```
void OnSignatureConfirmed(byte[] signData, int numSignaturePoints,
    byte[] imageData)
```

Property	Description
byte[] signData	The captured signature data.
int numSignaturePoints	The number of points captured in the signature.
byte[] image	A rendered image of the signature.

Usage:

```
HRESULT CDeviceEventSink::Invoke(DISPID dispIdMember, ...)
{
    switch(dispIdMember)
    {
    case 5L: // Signature confirmed
        {
            // Do something...
            break;
        }
    }

    return S_OK;
}
```

5.21 SignatureDataReceived event

This event is called whenever new points are captured using the mobile device and transmitted to the PC during the signature capture process. The newly captured points are delivered as a list (or, for COM, as an array) in the event parameters.

Please note: For security reasons, no time values are assigned to the points provided via this event. This means that it is not possible to reconstruct biometrical data by recording this unencrypted information.

5.21.1 .NET

```
event SignatureDataReceivedHandler SignatureDataReceived;
```

The `SignatureDataReceivedEventArgs` class contains the following property:

Property	Description
`List<SigPoint> SigPoints`	List of the newly captured signature points.

The points are represented via the `SigPoint` class, which has the following properties:

Property	Description
`double X`	Horizontal coordinates of the signature point.
`double Y`	Vertical coordinates of the signature point.
`double P`	Pressure of the signature point.

Usage:

```
private void MobileDevice_ SignatureDataReceived (object sender,
    SignatureDataReceivedEventArgs e)
{
    MessageBox.Show(String.Format("{0} new signature points captured.",
        e.SigPoints.Count);
}
```

5.21.2 COM

The dispatcher ID of the event is 6.

```
void OnSignatureDataReceived(SigPoint[] newPoints);
```

Property	Description
`byte[] newPoints`	Array of the newly captured signature points.
`int newPointsCount`	Count of the newly captured signature points.

The byte array contains 3 double values for each signature point making a total of 24 bytes for each point. The values are always stored in the order X (horizontal coordinate), Y (vertical coordinate) and P (pressure). An example, who all values can be extracted from the byte array, can be found in the COM demo.

Usage:

```
HRESULT CDeviceEventSink::Invoke(DISPID dispIdMember, ...)
{
    switch(dispIdMember)
    {
    case 6L: // SignatureDataReceived
        {
            CComVariant newPoints = pDispParams->rgvarg[1];
            int newPointsCount = pDispParams->rgvarg[0].intVal;

            // Do something...
            break;
        }
    }

    return S_OK;
}
```

5.22 BonjourError event

This event is called whenever one of the asynchronous Bonjour methods reports a specific error.

The possible error codes are listed in the chapter entitled 'BonjourErrorCode list'.

5.22.1 .NET

```
event BonjourErrorHandler BonjourError;
```

The `BonjourErrorEventArgs` contains the following property:

Property	Description
`BonjourErrorCode ErrorCode`	The error code reported by Bonjour.

Usage:

```
private void MobileDevice_BonjourError(object sender,
    BonjourErrorEventArgs e)
{
    MessageBox.Show(String.Format("Bonjour error code: {0} ({1})",
        e.ErrorCode.ToString(), (int)e.ErrorCode));
}
```

5.22.2 COM

The dispatcher ID of the event is 7.

```
void OnBonjourError(int errorCode);
```

Property	Description
`int errorCode`	The error code reported by Bonjour.

Usage:

```
HRESULT CDeviceEventSink::Invoke(DISPID dispIdMember, ...)
{
    switch(dispIdMember)
    {
    case 7L: // BonjourError
        {
            int errorCode = pDispParams->rgvarg[0].intVal;

            // Do something with the error code...
            break;
        }
    }

    return S_OK;
}
```

5.23 MobileCaptureError event

This event is called whenever one of the current service's asynchronous internal methods throws an exception.

5.23.1 .NET

```
event MobileCaptureErrorHandler MobileCaptureError;
```

The `MobileCaptureErrorEventArgs` class contains the following property:

Property	Description
`Exception Exception`	The exception thrown by the internal method.

Usage:

```
private void MobileDevice_MobileCaptureError(object sender,
    MobileCaptureErrorEventArgs e)
{
    MessageBox.Show(e.Exception.Message);
}
```

5.23.2 COM

The dispatcher ID of the event is 8.

```
void OnMobileCaptureError(BSTR errorMessage)
```

Property	Description
`BSTR errorMessage`	The error message from the exception thrown by the internal method.

Usage:

```
HRESULT CDeviceEventSink::Invoke(DISPID dispIdMember, ...)
{
    switch(dispIdMember)
    {
    case 8L: // Mobile Capture error
        {
            BSTR errorMessage = pDispParams->rgvarg[0].bstrVal;

            // Do something with the error message...
            break;
        }
    }

    return S_OK;
}
```

5.24 SignaturePointsChanged event

This event is called whenever the new points are added to the internal list of signature data. This event is required so that the API can internally communicate with the control, to inform it that the display needs to be updated. The event is generally not required for any other purposes.

Please note: The COM interface does not contain this event.

```
event EventHandler SignaturePointsChanged;
```

5.25 DeviceConnectResult list

The `DeviceConnectResult` list contains the following values:

Code	Value	Description
`Successful`	1	The handshaking process with the mobile device was successful. Consequently, a connection was established.
`WrongPairingCode`	2	The pairing code passed to the Connect method did not correspond to the code for this mobile device.
`AnotherDeviceConnected`	3	The mobile device is already connected to another device (PC).
`UnknownResultCode`	99	The mobile device reported an unknown result code. If you receive this value, please first check whether an updated version of this API is available.
`None`	0	The initial value used in this list.

6 MobileControl class

The `MobileControl` class is a Windows Forms control that displays the signature captured on the mobile device in real time. It can be placed on the custom program window in the usual way by using Visual Studio Designer.

If using COM, it is possible to revert to the ActiveX control, which has the same function.

6.1 MobileDevice property

This property makes it possible to read and set the mobile device from which this control should display the signatures.

```
MobileDevice MobileDevice { get; set; }
```

Usage:

```
mobileControl.MobileDevice = mobileDevice;
```

In COM:

```
pMobileControl->put_MobileDevice(pMobileDevice);
```