signoPAD-API Linux

Documentation

Software components for communication with
signotec Sigma, Zeta, Omega, Gamma, Delta and Alpha LCD pads

Version: 8.5.2030
Date: 18.06.2021

© signotec GmbH
www.signotec.de
Tel.: +49 (0) 2102 53575 10
E-mail: info@signotec.de

Contents

Legal notice

1 Document history

Version	Date	Person responsible	Status/note
1.0	22 April 2016	Paul Grütter	Newly created document
1.1	11 January 2017	Valentin Tkachev	Miscellaneous changes
1.2	2 June 2017	Valentin Tkachev	Changes in chapters 7.36, 7.52, 7.57
1.3	26 June 2017	Valentin Tkachev	Changes in chapters 7.47, 7.48, 7.50, 7.51
1.4	29 June 2018	Valentin Tkachev	Sections 7.71, 7.8, 7.9, 7.10, 7.11, 7.12, 7.13, 7.14, 7.15, 7.16, 7.27, 7.28, 7.32, 7.44, 7.45, 7.60, 7.62, 7.63, 7.64, 7.65, 7.66, 7.22 added Hyperlinks adjusted
8.5.2030	18 June 2021	Valentin Tkachev, Paul Grütter	Sections 2, 3, 4, 5, 6.1.3, 6.2.3, 6.2.4, 6.2.6, 7.1, 7.6, 7.8, 7.9, 7.10, 7.16, 7.18, 7.19, 7.21, 7.23, 7.42, 7.54, 7.55, 7.56, 7.57, 7.59, 7.64, 7.65, 7.67, 7.68, 7.70, 7.71, 8.2, 8.3 changed Sections 6.1.2, 6.1.5, 6.2.2, 6.2.5, 6.6 7.25, 7.26, 7.29, 7.49, 7.66 added Various code examples revised

2 Functional overview

The signoPAD API contains a non-visual interface, allowing programmers to implement a wide range of functions for capturing electronic signatures and displaying graphics, text and buttons on a signotec LCD pad in their own programs.

The following table provides an overview of the components included in the signoPAD API.

File name	Short description
libSTPadLib.so	Non-visual native library for activating the Sigma, Zeta, Omega, Gamma, Delta and Alpha model types.
libSTCPImageEngine.so	Library that is used internally for graphics functionality. It is not possible to use this library directly.
STPad.ini	Control file to set different kind of parameters for the pad communication. In addition, debug logging can be activated for the components listed above.
STPadStores.ini	Control file to assign non-volatile image memories exclusively to an application. Please refer to section 'Exclusive use of non-volatile memory' for details.
Sample application	Applications and source code in C++ to demonstrate the functions of the libSTPadLib.so

3 System requirements

3.1 signoPAD API components for Linux

The signoPAD API for Linux can be run on all Linux versions from Kernel 3.19.0 onwards. It was tested under the following systems and development environments:

- Ubuntu 15.04
- Eclipse Luna 4.4.2
- Microsoft Visual Studio 2017–2019

3.1.1 Dependencies

- libX11
- libusb-1.0
- libpthread
- libtiff
- libpng12
- libcairo
- libpangocairo-1.0

3.2 signoPAD API components for Java

Please use the signoPAD API Java, which can be downloaded at www.signotec.de.

3.3 signoPAD API components for Windows

Please use the signoPAD API Windows, which can be downloaded at www.signotec.de.

4 General information on the signoPAD-API components

4.1 libSTPadLib.so

PadLib.so is a native and dynamically loadable library. A C header file (STPadLib.h) is included. Initialisation is performed automatically as soon as the library is activated; before it is unloaded again, the `STControlExit()` method must be called to release resources used internally.

4.2 Image formats

Signatures can be returned in JPEG, PNG and TIFF format. Generally speaking, you should use PNG as it offers the best results with the smallest file size. JPEG is an image format with lossy compression and is not recommended.

4.3 SignData structures

The signoPAD-API components can return a captured signature as a **SignData** data structure. This format is an encrypted, compressed, biometric structure that can be stored in a database or as a tag in a TIFF or PDF document.

4.4 Notes for redistribution

You can, of course, redistribute individual files from the signoPAD API in a separate package. Only libSTPadLib.so and libSTCPImageEngine.so, and possibly the files STPad.ini and STPadStores.ini, are required for the basic support of the signotec Sigma, Zeta, Omega, Gamma, Delta and Alpha LCD signature pads:

Component	Installation path
libSTPasLib.so	/usr/local/lib für Ubuntu /usr/lib64 für CentOS x64 /usr/lib für CentOS x86
libSTCPImageEngine.so	/usr/local/lib für Ubuntu /usr/lib64 für CentOS x64 /usr/lib für CentOS x86
STPad.ini	Working directory of the application (optional)
STPadStores.ini	Working directory of the application (optional)

The STPadLibDemoApp, which is also included in the package, serves an example of how the signoPAD API should be used. The demo is included with the C++ source code in the signoPAD API package. It is a simple console program. At the beginning, the demo searches for signotec LCD pads, and the first pad found is used. The signature process is initiated, and the signature is captured and saved as PNG and SignData files.

5 Description of possible error messages

Most of the library methods return an integer value, which is negative in the case of an error. A description of the respective error messages is provided in the following table. An error description can be retrieved at runtime by calling the `STControlGetErrorString()` method.

The error descriptions are available in the components in German, English, French and Italian.

Error	Description
-1	A NULL pointer was passed.
-3	One of the parameters that were passed contains an invalid value.
-4	The signing pad is already being used by another application.
-5	No connection has been opened to this signature pad.
-6	A connection has already been opened.
-8	No device with this ID is connected.
-9	The LED colour that was passed cannot be set.
-12	The function could not be executed because the signature capture process is running.
-13	No further hotspots can be added.
-14	The function could not be executed because the coordinates of an active hotspot overlap with the signature window or another active hotspot.
-15	The function could not be executed because no signature capture area has been set.
-17	The function could not be executed because no signature capture process was started.
-18	An error occurred while attempting to reserve memory.
-19	An error occurred while initialising a system resource.
-20	An error occurred while communicating with the signing pad.
-21	The rectangle that was passed is invalid.
-22	No compatible devices connected or the connection to a device has been cut.
-25	The connected device does not support this function or one of the parameters.
-26	Error while reading or writing a file.
-39	Signature capture could not be started because pen-controlled scrolling is activated.
-40	The function could not be executed because the device does not feature an NFC reader.
-41	The function could not be executed due to an unknown error. It may be necessary to update the software.
-45	Signature capture could not be started because a keypad hotspot is defined.
-84	The function could not be executed because the secure sign mode is active.
-93	The function could not be executed because an overlay rectangle is set.
-94	The function could not be executed because the screen content is scrolled or pen-controlled scrolling is activated.
-95	The function could not be executed because it would have activated the scroll mode that is not possible if a hotspot outside the overlay rectangle is defined.
-97	An error occurred during initialisation. Please restart the software.
-99	The function could not be executed because keypad hotspots can only be inverted if the foreground memory is set as active device memory.

6 Information about the available image memory

The signotec LCD Signature Pads have several image memory, which can be used by different methods. An image memory has at least the size of the display and can store one picture in a maximum of this size. Adding another image overrides the areas it overlaps with the existing memory content. Adding multiple images to one memory can therefore create a collage.

Depending on the model, a different number of volatile and non-volatile memories are available.

6.1 Volatile image memories

All signotec LCD Signature Pads have at least two volatile image memories, one foreground memory containing the current display content and one background memory, which can be used to prepare the display content. It can be written in both of the memories.

The content of the volatile image memory is lost when you close the connection to the device.

6.1.1 Model type Sigma

The two volatile image memories have the size of the display (320 x 160 pixels).

The transmission and representation of images is usually so fast that there is no visible lag. For more complex representations that consist of several individual images, it may be useful to first save them in the background memory before copying them into the foreground memory.

6.1.2 Model type Zeta

The two volatile image memories have the size of the display (320 x 200 pixels).

The transmission and representation of images is usually so fast that there is no visible lag. For more complex representations that consist of several individual images, it may be useful to first save them in the background memory before copying them into the foreground memory.

6.1.3 Model type Omega

The Omega model has three volatile image memories, two that have the doubled size of the display (640 x 960 pixels) to be used as foreground and background memory and one that has the size of the display (640 x 480 pixels) to be used as overlay memory. Its contents can be overlaid over the current display content.

The speed of displaying an image in the Omega model with firmware up to Version 1.40 depends on the size and content of the images. The image composition is usually visible. Therefore, images should always be stored first in the background memory and then moved into the foreground memory.

An image is displayed in the Omega model with firmware 2.0 or newer only after it has been transferred; the image composition is not visible. The speed of the image transmission depends on the size and content of the images and the Connection-Type mode. For more complex representations that consist of several individual images, it can generally be useful to first save them in the background memory before copying them into the foreground memory.

6.1.4 Model type Gamma

The Gamma model has three volatile image memories, two that are larger than the display (800 x 1440 pixels) to be used as foreground and background memory and one that has the size of the display (800 x 480 pixels) to be used as overlay memory. Its contents can be overlaid over the current display content.

With the Gamma model, an image is only displayed after it has been transferred; the image composition is not visible. The speed of the image transmission depends on the size and content of the images and the Connection-Type mode. For more complex representations that consist of several individual images, it can generally be useful to first save them in the background memory before copying them into the foreground memory.

6.1.5 Model type Delta

The Delta model has three volatile image memories, two that are larger than the display (1280 x 37,600 pixels) to be used as foreground and background memory one that is the size of the display (1280 x 800 pixels) to be used as overlay memory. Its contents can be overlaid over the current display content.

The speed of displaying a picture in the Delta model depends on the size and content of the images as well as the Connection-Type mode. The image composition is usually visible. Therefore, images should always be stored first in the background memory and then moved into the foreground memory.

6.1.6 Model type Alpha

The Alpha model has three volatile image memories, two that are larger than the display (2048 x 2048 pixels) to be used as foreground and background memory one that is the size of the display (768 x 1366 pixels) to be used as overlay memory. Its contents can be overlaid over the current display content.

With the Alpha model, an image is only displayed after it has been transferred; the image composition is not visible. The speed of the image transmission depends on the size and content of the images and the Connection-Type mode. For more complex representations that consist of several individual images, it can generally be useful to first save them in the background memory before copying them into the foreground memory.

6.2 Non-volatile image memories

Depending on the model, a different number of non-volatile memories are available. The saving of images in non-volatile image memory lasts longer than storing in volatile image memory, but the content remains unchanged even after switching off the device. An intelligent memory management detects whether an image to be stored is already stored in the device so that only the first time it’s stored it comes to a delay.

6.2.1 Model type Sigma

The Sigma model has one non-volatile image memory in the size of the display (320 x 160 pixels), which can only be used for the standby image. Due to the rapid transmission and display of pictures, it is not necessary to be able to save other images permanently.

6.2.2 Model type Zeta

The Zeta model has one non-volatile image memory in the size of the display (320 x 200 pixels), which can only be used for the standby image. Due to the rapid transmission and display of pictures, it is not necessary to be able to save other images permanently.

6.2.3 Model type Omega

The Omega model has eleven non-volatile image memories, which can be used for the standby image, the slide show and optimizations of the program. The memories, used for the standby image or the slide show, are read-only and can be freed only by disabling the standby image or the slide show.

One non-volatile image memory has the doubled size of the display (640 x 960 pixels), ten memories have the size of the display (640 x 480 pixels).

To use a non-volatile memory, this must be reserved first. This is done by calling the `STDisplaySetTarget()` method. The size of the currently selected memory can be queried using the `STDisplayGetTargetWidth()` and `STDisplayGetTargetHeight()` methods.

USB Bulk Transfer, which is available from Firmware 2.0, does not require the use of non-volatile memory to optimise the program, as image transmission is very fast. However, it depends on the individual case at hand and the developer should make the final decision.

6.2.4 Model type Gamma

The Gamma model has ten non-volatile image memories, which can be used for the standby image, the slide show and optimisations of the program. The memories, used for the standby image or the slide show, are read-only and can be freed only by disabling the standby image or the slide show.

The ten non-volatile memories are the same size as the display (800 x 480 pixels).

To use a non-volatile memory, this must be reserved first. This is done by calling the `STDisplaySetTarget()` method. The size of the currently selected memory can be queried using the `STDisplayGetTargetWidth()` and `STDisplayGetTargetHeight()` methods.

USB Bulk Transfer does not require the use of non-volatile memory to optimise the program, as image transmission is very fast. However, it depends on the individual case at hand and the developer should make the final decision.

6.2.5 Model type Delta

The Delta model has 32 non-volatile image memories, which can be used for the standby image, the slide show and optimisations of the program. The memories, used for the standby image or the slide show, are read-only and can be freed only by disabling the standby image or the slide show.

The 32 non-volatile memories are the same size as the display (1280 x 800 pixels).

To use a non-volatile memory, this must be reserved first. This is done by calling the `STDisplaySetTarget()` method. The size of the currently selected memory can be queried using the `STDisplayGetTargetWidth()` and `STDisplayGetTargetHeight()` methods.

USB Bulk Transfer does not require the use of non-volatile memory to optimise the program, as image transmission is very fast. However, it depends on the individual case at hand and the developer should make the final decision.

6.2.6 Model type Alpha

The Alpha model has ten non-volatile image memories, which can be used for the standby image, the slide show and optimisations of the program. The memories, used for the standby image or the slide show, are read-only and can be freed only by disabling the standby image or the slide show.

The ten non-volatile memories are the same size as the volatile memories (2048 x 2048 pixels).

To use a non-volatile memory, this must be reserved first. This is done by calling the `STDisplaySetTarget()` method. The size of the currently selected memory can be queried using the `STDisplayGetTargetWidth()` and `STDisplayGetTargetHeight()` methods.

USB Bulk Transfer does not require the use of non-volatile memory to optimise the program, as image transmission is very fast. However, it depends on the individual case at hand and the developer should make the final decision.

6.3 Copying between image memories

The contents can be copied between the most of the available image stores. The content of background memory cannot be copied to the foreground memory; it can only be moved. The contents of the overlay memory cannot be copied but only overlaid over the display content.

Typical copy operations are copying from a non-volatile image memory in a volatile image memory and moving from the volatile background memory into the foreground memory. Copying an image within the device is always faster than sending this image from the PC to the device. Please refer to the descriptions of the `STDisplaySetImageFromStore()` and `STDisplaySetOverlayRect()` methods for details.

6.4 The typical process

Most applications use the same images with possibly variable units (such as document-related texts) for the signature process. It therefore makes sense to store images that are the same each time in one of the non-volatile memory if possible. The following is the typical work flow for this scenario

First, the images are loaded, which will be permanently stored in the device, since they change rarely. A memory is reserved by calling the `STDisplaySetTarget()` method with the `STPAD_TARGET_STANDARDSTORE` value. The return value of the method is the ID of the memory used. If no non-volatile image memory is available, the ID is returned as `STPAD_TARGET_BACKGROUND`, which means that the background memory is set as an image memory. This is always the case when using the Sigma or Zeta models. When using the Omega, Gamma and Alpha models, the number of available memories can be less than expected when a slide show is configured.

Text and images that are added to a non-volatile memory are only saved locally to begin with and are sent to the device only when `STDisplaySetImageFromStore()` or `STDisplayConfigSlideShow()` is called in order to be able to compare the image (which may be composed of several texts and images) with the image already stored in the device. Thus only when one of these methods is called, there will be a noticeable delay.

```
LONG nTarget = STPAD_TARGET_STANDARDSTORE;
LONG nRc = STDisplaySetTarget(nTarget);
if (nRc < 0)
   return nRc;
nTarget = nRc;
nRc = STDisplaySetImageFromFile(10, 10, L"./1.png");
if (nRc < 0)
   return nRc;
nRc = STDisplaySetText(200, 160, kLeft, L"Signature:");
if (nRc < 0)
   return nRc;
nRc = STDisplaySetImageFromFile(220, 400, L"./2.png");
if (nRc < 0)
   return nRc;
```

The content can now be copied to a volatile image memory, typically the background memory (`STDisplaySetTarget(STPAD_TARGET_BACKGROUND)`). If the images have already been written to the background memory because no non-volatile memory was available (see above), the `STDisplaySetImageFromStore()` method will not function, however it will also not produce any errors and can therefore be safely called.

```
nRc = STDisplaySetTarget(STPAD_TARGET_BACKGROUND);
if (nRc < 0)
   return nRc;
nRc = STDisplaySetImageFromStore(nTarget);
if (nRc < 0)
   return nRc;
```

Now content, that change with every signature process, can be added to the background memory.

```
nRc = STDisplaySetImageFromFile(120, 400, L"./3.png");
if (nRc < 0)
   return nRc;
nRc = STDisplaySetText(200, 160, kLeft, L"01.01.2010");
if (nRc < 0)
   return nRc;
```

In the background memory there is now a collage of two images and a text copied from a non-volatile memory and an image and a text that have been sent from the PC. This collage can now be moved into the foreground memory and thus displayed on the screen. The total composition has happened before in the background buffer and thus "invisible".

```
nRc = STDisplaySetTarget(STPAD_TARGET_FOREGROUND);
if (nRc < 0)
   return nRc;
nRc = STDisplaySetImageFromStore(STPAD_TARGET_BACKGROUND);
if (nRc < 0)
   return nRc;
```

The process described must be performed every time a connection is opened. When a connection is closed all information about reserved memories is lost. Only information regarding which display content is stored in which non-volatile memory remains saved on the device (even when it is switched off).

6.5 The standby feature

The signotec LCD signature pads (Omega, Gamma, Delta and Alpha models only) can display one or more images automatically when not in use (no established connection). These images are stored permanently in the device and they are displayed without launching any application on the PC.

Image memories that are used by the standby feature are write protected and cannot be used for by an application.

6.5.1 Displaying a logo

In all devices, an image that is displayed automatically in standby can be stored permanently. Please refer to the descriptions of the `STDisplaySetStandbyImage()` and `STDisplaySetStandbyImageFromFile()` methods for details.

6.5.2 Displaying a slide show

As an alternative to a logo, the Omega, Gamma, Delta and Alpha models can display a slide show containing up to ten (Gamma and Alpha), 11 (Omega) or 32 (Delta) images. To configure a slide show, please follow these steps:

First, a standby mode that may be configured must be disabled by calling `STDisplayConfigSlideShow()` in order to remove write protection from all images. The current configuration can be queried with the `STDisplayGetStandbyId()` method.

Then any contents can be written to one or more of the non-volatile image memories (as described in 7.4). When all the contents have been written, the desired image memories must be configured using the `STDisplayConfigSlideShow()` or `STDisplayConfigSlideShowEx()` method.

6.6 Exclusive use of non-volatile memory

Since after closing the connection to a device all the information about reserved stores is lost, two applications may use the same memory and thus always overwrite it. As a result, the use of non-volatile memories can even cause slower program execution.

To avoid this problem, a user may exclusively reserve up to ten image memories of an Omega device for a particular application. These memories are not available to other applications and thus cannot be overwritten accidently.

6.6.1 Implementation in an application

Applications must provide the component with their name in order to support this functionality. This is done with the `STControlSetAppName()` property. Users can enter this name in the configuration file and reserve a certain number of memories for this application.

6.6.2 Assign non-volatile memory to an application

Memories are assigned in the STPadStores.ini file, which must be located in the working directory of the application. The configuration in this file only applies to this workstation; if the pad is connected to another PC without configuration, all memories will be available there again.

The file can contain up to ten sections (for ten applications). The names of the sections are `[Application1]`, `[Application2]`, etc.

Each section contains two keys named `Name` and `StoreCount`. The `Name` key contains the name given by the application (see above). The `StoreCount` key contains the number of memories to be reserved. It does not matter if memory is reserved for one or several applications, but the sum of the reserved memories must not exceed 10. The Omega memory with a size of 640 x 960 pixels cannot be used exclusively.

```
[Application1]
Name=MyGreatApp
StoreCount=2

[Application2]
Name=Another Great App
StoreCount=4
```

If not all of the available memory is assigned exclusively, all further image memories are available for all applications. If a standby image or a slide show has already been configured for the device, the maximum number of available memories is reduced accordingly. If a standby image or slide show is configured by an application on a workstation with a memory configuration, only memories can be used in this context that have been reserved for the respective application or have not been reserved at all.

7 Methods

Methods are named according to the following naming convention:

- Methods that set or query general hardware properties begin with '**STDevice**'
- Methods that set or query sensor properties begin with '**STSensor**'
- Methods that apply to the signature begin with '**STSignature**'
- Methods that set or query LCD properties begin with '**STDisplay**'
- Methods that set or query component properties begin with '**STControl**'

7.1 STDeviceSetComPort method

This method defines the interfaces to be searched for devices when `STDeviceGetCount()` is called up. A search will only be made for USB devices unless this method is called up.

In order to query the connection to which an existing device is connected please use the `STDeviceGetComPort()` and `STDeviceGetIPAddress()` methods.

Parameter	Values	I/O	Description
`LPCWSTR szPortList`	A list separated by semi-colons with at least one of the following members:		
	`"HID"`	I	A search will be made for USB devices.
	`"IP=<Address>:<Port>"`	I	An Alpha or Delta with an Ethernet connection or an USB device that is connected to a signotec Ethernet USB adapter will be searched for at the specified IP address; `<Address>` is the IP address of the device and has the format `x.x.x.x` or the host name, `<Port>` is the port via which the signature device should be activated, generally `1002`. A valid value would therefore be, for example, `'IP=192.168.100.100:1002'` or `'IP=host1:1002'`; for details about the signotec Ethernet USB adapter, please refer to your contact at signotec.
	`"all"`	I	A search will be made for devices on all COM ports; this search may take an extremely long time depending on the hardware configuration.
	Whole number	I	A search will be made for a serial device on the COM port with the specified number (>=0); this search may take an extremely long time if no signotec device is connected to the port
	`"LowSpeed"`	I	A search will be made on the COM ports only for devices that communicate at a baud rate of 115200 (Sigma Zeta, Omega, Gamma plus Delta and Alpha in 'Low Speed' mode). The search is accelerated if COM ports to which no signotec LCD signature pads are connected are also searched; this does not influence the search for USB or IP devices.
	`"High Speed"`	I	A search will be made on the COM ports only for devices that communicate at a baud rate of 2 MBaud (Delta and Alpha in 'High Speed' mode), which accelerates the search; this does not influence the search for USB or IP devices.

Return value	Values	Description
LONG	>= 0	Number of detected members in the transferred list (for the member "all" the return value 256 will be added to old versions instead of 1 for compatibility reasons)
	< 0	Error

Available from Version 8.5.2.2.

```
LONG STDeviceSetComPort(LPCWSTR szPortList)
```

7.1.1 Implementation

```
LONG nPortCount = STDeviceSetComPort(L"HID;1;4;IP=192.168.100.100:1002");
if (nPortCount < 0)
    wprintf(L"Error %d", nPortCount);
else
    wprintf(L"%d ports set", nPortCount);
```

7.2 STDeviceGetCount method

This method searches for connected devices, generates an internal index beginning with 0 and returns the number of devices detected. This value should be cached so that the method only needs to be called, if the number of connected devices has changed. A device's index is retained until the method is called again. The index can be assigned to a device via the information returned by `STDeviceGetInfo()`.

By default, a search will only be made for USB devices that are locally connected. A search will only be made for other devices if this has been configured previously by calling up `STDeviceSetComPort()`.

Parameter	Values	I/O	Description
-	-	-	-
Return value	**Values**	**Description**	
LONG	>= 0	Number of devices detected	
	< 0	Error	

Available from Version 8.2.0.

```
LONG STDeviceGetCount()
```

7.2.1 Implementation in C++

```
LONG nDeviceCount = STDeviceGetCount();
if (nDeviceCount < 0)
    wprintf(L"Error %d", nDeviceCount);
else
    wprintf(L"%d devices detected.", nDeviceCount);
```

7.3 STDeviceGetConnectionType method

This method returns the type of connection via which a device is connected.

Parameter	Values	I/O	Description
LONG nIndex	>= 0	I	Index of the device whose Connection-Type number is to be queried
Return value	**Values**	**Description**	
LONG	0	USB interrupt transfer (HID)	
	1	USB bulk transfer	
	2	Serial or virtual channel	
	3	Ethernet	
	< 0	Error	

Available from Version 8.2.0.

```
LONG STDeviceGetConnectionType(LONG nIndex)
```

7.3.1 Implementation in C++

```
LONG nType = STDeviceGetConnectionType(0);
switch (nType)
{
    case 0:
        wprintf(L"The device is connected via HID.");
        break;
    case 1:
        wprintf(L"The device is connected via USB.");
        break;
    case 2:
        wprintf(L"The device is connected to a serial port.");
        break;
    case 3:
        wprintf(L" The device is connected via IP.");
        break;
    default:
        wprintf(L"Error %d", nType);
        break;
}
```

7.4 STDeviceGetComPort method

This method returns the number of the COM port to which a device is connected.

Parameter	Values	I/O	Description
LONG nIndex	>= 0	I	Index of the device whose port number is to be queried
Return value	**Values**	**Description**	
LONG	>= 0	Number of the COM port for serial devices. (Note: Devices that are not serially connected return 0; use `STDeviceGetConnectionType()` to ascertain the type of connection)	
	< 0	Error	

Available from Version 8.5.2.2.

```
LONG STDeviceGetComPort(LONG nIndex)
```

7.4.1 Implementation

```
LONG nPort = STDeviceGetComPort(0);
if (nPort < 0)
    wprintf(L"Error %d", nPort);
else
    wprintf(L"This device is connected to COM port %d", nPort);
```

7.5 STDeviceGetIPAddress method

You can use this method to retrieve the IP address of a Delta or Alpha with Ethernet connection or a signotec Ethernet USB adapter to which a device is connected.

Parameter	Values	I/O	Description
WCHAR szAddress[32]	""	O	The device is not connected via IP
	max. 32 characters	O	Device IP address and port ("x.x.x.x:x")
LONG nIndex	>= 0	I	Index of the device whose information is to be queried
Return value	**Values**	**Description**	
LONG	0	Method was executed successfully	
	< 0	Error	

Available from Version 8.5.2.2.

```
LONG STDeviceGetIPAddress(WCHAR szAddress[32], LONG nIndex)
```

7.5.1 Implementation

```
WCHAR szAddress[32];
LONG nResult = STDeviceGetIPAddress(szAddress, 0);
if (nResult < 0)
    wprintf(L"Error %d", nResult);
else
    wprintf(L"The device is connected to: %s", szAddress);
```

7.6 STDeviceGetInfo method

You can use this method to retrieve the serial number and model type of a connected device in order to uniquely identify it.

Parameter	Values	I/O	Description
WCHAR szSerial[16]	max. 16 chars	O	Serial number

LONG* pnType	1	O	'Sigma USB' model type
	2	O	'Sigma serial' model type
	5	O	'Zeta USB' model type
	6	O	'Zeta serial' model type
	11	O	'Omega USB' model type
	12	O	'Omega serial' model type
	15	O	'Gamma USB' model type
	16	O	'Gamma serial' model type
	21	O	'Delta USB' model type
	22	O	'Delta serial' model type
	23	O	'Delta IP' model type
	31	O	'Alpha USB' model type
	32	O	'Alpha serial' model type
	33	O	'Alpha IP' model type
	other	O	Reserved for further model types
LONG nIndex	>= 0	I	Index of the device whose information is to be queried
Return value	**Values**	**Description**	
LONG	0	Method was executed successfully	
	< 0	Error	

Available from Version 8.5.2.2.

```
LONG STDeviceGetInfo(WCHAR szSerial[16], LONG* pnType, LONG nIndex)
```

7.6.1 Implementation

```
WCHAR szSerial[16];
LONG nType = 0;
LONG nRc = STDeviceGetInfo(szSerial, &nType, 0);
if (nRc < 0)
    wprintf(L"Error %d", nRc);
else
    wprintf(L"Type: %d, Serial: %s", nType, szSerial);
```

7.7 STDeviceGetVersion method

You can use this method to retrieve the version number of a connected device's firmware. It is intended primarily for support purposes.

Parameter	Values	I/O	Description
WCHAR szVersion[16]	max. 16 chars	O	Firmware version number (major.minor)
LONG nIndex	>= 0	I	Index of the device whose information is to be queried
Return value	**Values**	**Description**	
LONG	0	Method was executed successfully	
	< 0	Error	

Available from Version 8.2.0.

```
LONG STDeviceGetVersion(WCHAR szVersion[16], LONG nIndex)
```

7.7.1 Implementation

```
WCHAR szVersion[16];
LONG nRc = STDeviceGetVersion(szVersion, 0);
if (nRc < 0)
    wprintf(L"Error %d", nRc);
else
    wprintf(L"Firmware:  %s", szVersion);
```

7.8 STDeviceGetCapabilities method

You can use this method to retrieve various properties of a connected device.

Parameter	Values	I/O	Description
LONG nIndex	>= 0	I	Index of the device whose information is to be queried
Return value	**Values**	**Description**	
LONG	0	Bitmask from which the properties can be read (for details see below)	
	< 0	Error	

Support for a property is indicated by a set bit in the returned bitmask. Some of the device properties cannot be used with the present version of the API, but they are still listed here. The following properties can be currently supported:

Bit	Definition in STPdLib.h	Description
0x00000001	STPAD_CAP_COLORDISPLAY	Device has a colour screen
0x00000002	STPAD_CAP_BACKLIGHT	Device has a backlit screen
0x00000004	STPAD_CAP_VERTICALSCROLLING	Device supports vertical scrolling
0x00000008	STPAD_CAP_HORIZONTALSCROLLING	Device supports horizontal scrolling
0x00000010	STPAD_CAP_PENSCROLLING	Device supports scrolling with the pen
0x00000020	STPAD_CAP_SERVICEMENU	The service menu can be called using the `STDeviceStartService()` method
0x00000040	STPAD_CAP_RSA	Device supports the RSA functions
0x00000080	STPAD_CAP_CONTENTSIGNING	Device supports content signing
0x00000100	STPAD_CAP_H2CONTENTSIGNING	Device supports content signing where only hash 2 is signed
0x00000200	STPAD_CAP_GENERATESIGNKEY	Device can generate a signature key pair
0x00000400	STPAD_CAP_STORESIGNKEY	Device can save an externally supplied signature key pair
0x00000800	STPAD_CAP_STOREENCRYPTKEY	Device can save an externally supplied biometric key
0x00001000	STPAD_CAP_SIGNEXTERNALHASH	Device can sign a hash calculated externally (otherwise only a hash calculated through content signing)
0x00002000	STPAD_CAP_RSAPASSWORD	RSA keys can be protected through a device password.
0x00004000	STPAD_CAP_SECUREMODEPASSWORD	The ‘Secure mode’ can be protected through a device password.
0x00008000	STPAD_CAP_4096BITKEY	Device supports RSA keys with a length of up to 4096 bits (otherwise up to 2048 bits)
0x00010000	STPAD_CAP_NFCREADER	Device has an NFC reader

0x00020000	STPAD_CAP_KEYPAD	Device supports keypad encryption with a length of up to 8 characters
0x00040000	STPAD_CAP_KEYPAD32	Device supports keypad encryption with a length of up to 32 characters
0x00080000	STPAD_CAP_DISPLAY	Device has a screen (please note: Support for this property is always displayed even with a Sigma Lite up to firmware 2.3)
0x00100000	STPAD_CAP_RSASIGNPASSWORD	The signature key can be protected by a key password

Available from Version 8.5.2.2.

```
LONG STDeviceGetCapabilities(LONG nIndex)
```

7.8.1 Implementation

```
LONG nCapabilities = STDeviceGetCapabilities(0);
if (nCapabilities < 0)
    wprintf(L"Error %d", nCapabilities);
else if (nCapabilities & 0x40)
    wprintf(L"Device supports RSA");
```

7.9 STDeviceOpen method

This method opens a connection to a device. The backlight is switched on if this is configured in the STPad.ini file (see method `STDisplaySetBacklight()`).

Parameter	Values	I/O	Description
LONG nIndex	>= 0	I	Index of the device to which a connection is to be opened
BOOL bEraseDisplay	TRUE	I	The device display screen will be erased (Default)
	FALSE	I	The content of the display screen will not change; the screen content displayed cannot be copied into another image memory at a later stage (optional)
Return value	**Values**	**Description**	
LONG	0	Method was executed successfully	
	< 0	Error	

Available from Version 8.2.0.

```
LONG STDeviceOpen(LONG nIndex, BOOL bEraseDisplay=TRUE)
```

7.9.1 Implementation

```
LONG nRc = STDeviceOpen(0);
if (nRc < 0)
    wprintf(L"Error %d", nRc);
```

7.10 STDeviceClose method

This method closes the connection to a device. Before closing, a currently running signature capture process is terminated and the backlight switched off, where appropriate, if so configured in the STPad.ini file (see method `STDisplaySetBacklight()`).

Captured signature data is discarded.

Parameter	Values	I/O	Description
`LONG nIndex`	`>= 0`	I	Index of the device whose connection is to be closed
Return value	**Values**	**Description**	
`LONG`	`0`	Method was executed successfully	
	`< 0`	Error	

Available from Version 8.2.0.

```
LONG STDeviceClose(LONG nIndex)
```

7.10.1 Implementation

```
LONG nRc = STDeviceClose(0);
if (nRc < 0)
    wprintf(L"Error %d", nRc);
```

7.11 STDeviceSetLed method

This method sets the colour of the LED on the front of the pad. The `STDeviceSetLedDefaultFlag()` method should be called with FALSE before when this method is used to ensure that the colour is not changed when `STSignatureStart()`, `STSignatureCancel()` and `STSignatureConfirm()` are called. The LED always lights up yellow as soon as the device has been detected by the PC operating system and is ready for use.

Parameter	Values	I/O	Description
`LONG nLedColor`	Bitmask containing one or more hexadecimal values from the following list:		
	`0x01`	I	Yellow
	`0x02`	I	Green
Return value	**Values**	**Description**	
`LONG`	`0`	Method was executed successfully	
	`< 0`	Error	

Available from Version 8.2.0.

```
LONG STDeviceSetLed(LONG nLedColor)
```

The following values defined in the header file can be used for the `nLedColor` parameter:

```
#define STPAD_LED_YELLOW 0x01
#define STPAD_LED_GREEN  0x02
```

7.11.1 Implementation

```
LONG nRc = STDeviceSetLed(STPAD_LED_YELLOW);
if (nRc < 0)
    wprintf(L"Error %d", nRc);
```

7.12 STDeviceGetLedDefaultFlag method

This method can be used to query whether the LED on the front of the pad automatically changes to green when the device is in signature capture mode. The default setting is TRUE. The LED always lights up yellow as soon as the device has been detected by the PC operating system and is ready for use.

Parameter	Values	I/O	Description
-	-	-	-
Return value	**Values**	**Description**	
BOOL	TRUE	LED lights up green for a capture process	
	FALSE	LED does not change	

Available from Version 8.5.2.2.

```
BOOL STDeviceGetLedDefaultFlag()
```

7.12.1 Implementation

```
BOOL nLedDefaultFlag = STDeviceGetLedDefaultFlag();
```

7.13 STDeviceSetLedDefaultFlag method

This method determines whether the LED on the front of the pad automatically changes to green when the device is in signature capture mode. The default setting is TRUE. The LED always lights up yellow as soon as the device has been detected by the PC operating system and is ready for use.

Parameter	Values	I/O	Description
BOOL bFlag	TRUE	I	LED lights up green for a capture process
	FALSE	I	LED does not change
Return value	**Values**	**Description**	
VOID	-	-	

Available from Version 8.5.2.2.

```
VOID STDeviceSetLedDefaultFlag(BOOL bFlag)
```

7.13.1 Implementation

```
STDeviceSetLedDefaultFlag(FALSE);
```

7.14 STDeviceGetNFCMode method

This method reads out the operating mode of the optionally installed NFC reader. Whether the connected device has an NFC reader can be queried using the `STDeviceGetCapabilities()` method.

Parameter	Values	I/O	Description
LONG nIndex	>= 0	I	Index of the device whose information is to be queried
Return value	**Values**	**Description**	
LONG	0–1	Operating mode (see also STDeviceSetNFCMode())	
	< 0	Error	

Available from Version 8.5.2.2.

```
LONG STDeviceGetNFCMode(LONG nIndex)
```

The following values defined in the header file can be used for the nMode parameter:

```
#define STPAD_NFC_OFF   0
#define STPAD_NFC_ON    1
```

7.14.1 Implementation

```
LONG nMode = STDeviceGetNFCMode(0);
switch (nMode)
{
    case STPAD_NFC_OFF:
        wprintf(L"The NFC reader is currently switched off.");
        break;
    case STPAD_NFC_ON:
        wprintf(L"The NFC reader is currently switched on.");
        break;
    default:
        wprintf(L"Error %d", nMode);
        break;
}
```

7.15 STDeviceSetNFCMode method

This method changes the operating mode of the optionally installed NFC reader. It can only be called if a connection to the device has not been opened in another application. Whether the connected device has an NFC reader can be queried using the STDeviceGetCapabilities() method.

Parameter	Values	I/O	Description
LONG nMode	0	I	The NFC reader is turned off; after a restart, it is in standard operating mode again.
	1	I	The NFC reader is turned on; after a restart, it is in standard operating mode again.
	2	I	The NFC reader is turned off; the standard operating mode is likewise set to 'off'.
	3	I	The NFC reader is turned on; the standard operating mode is likewise set to 'on'.
LONG nIndex	>= 0	I	Index of the device whose mode is to be changed.
Return value	**Values**	**Description**	
LONG	0	Method was executed successfully	
	< 0	Error	

Available from Version 8.5.2.2.

```
LONG STDeviceSetNFCMode(LONG nMode, LONG nIndex)
```

The following values defined in the header file can be used for the `nMode` parameter:

```
#define STPAD_NFC_OFF               0
#define STPAD_NFC_ON                1
#define STPAD_NFC_PERMANENTLYOFF    2
#define STPAD_NFC_PERMANENTLYON     3
```

7.15.1 Implementation

```
LONG nRc = STDeviceSetNFCMode(STPAD_NFC_ON, 0);
if (nRc < 0)
    wprintf(L"Error %d", nRc);
```

7.16 STDeviceStartService method

This method starts one of the configuration dialog boxes on the signature device. The device cannot be reached as long as the dialog is displayed. The device restarts if the type of connection or the IP configuration is adjusted in the configuration dialog box.

Parameter	Values	I/O	Description
LONG nType	0	I	Starts the service menu in which the type of connection, the IP configuration and the screen brightness can be changed; this is only supported by the Sigma model from firmware 2.10, the Zeta model, the Omega model from firmware 2.0, the Gamma model from firmware 1.6, the Delta model and the Alpha model, provided that the device has a backlit screen
	1	I	Starts screen calibration
Return value	**Values**	**Description**	
LONG	0	Method was executed successfully	
	< 0	Error	

Available from Version 8.5.2.2.

```
LONG STDeviceStartService(LONG nType)
```

7.16.1 Implementation

```
LONG nRc = STDeviceStartService(1);
if (nRc < 0)
    wprintf(L"Error %d", nRc);
```

7.17 STSensorGetSampleRateMode method

This method returns the configured sample rate with which the signature is captured.

Parameter	Values	I/O	Description
–	–	-	-

Return value	Values	Description
LONG	3	280 Hz
	2	500 Hz
	1	250 Hz
	0	125 Hz
	< 0	Error

Available from Version 8.2.0.

```
LONG STSensorGetSampleRateMode()
```

7.17.1 Implementation

```
LONG nMode = STSensorGetSampleRateMode();
switch (nMode)
{
    case 0:
        wprintf(L"Sample rate is 125 Hz.");
        break;
    case 1:
        wprintf(L"Sample rate is 250 Hz.");
        break;
    case 2:
        wprintf(L"Sample rate is 500 Hz.");
        break;
    case 3:
        wprintf(L"Sample rate is 280 Hz.");
        break;
    default:
        wprintf(L"Error %d", nMode);
        break;
}
```

7.18 STSensorSetSampleRateMode method

This method sets the sample rate with which the signature is captured. The default setting is mode 1 (250 Hz) or mode 3 (280 Hz) when using the Alpha model. This mode provides high-quality signature data while at the same time ensures that the data record is of moderate size. When using the Sigma, Zeta, Gamma and Omega models, this value can easily be set to 2 (500 Hz) for high-speed data lines.

Parameter	Values	I/O	Description
LONG nMode	0	I	125 Hz (Sigma, Zeta, Omega, Gamma and Delta only)
	1	I	250 Hz (Sigma, Zeta, Omega, Gamma and Delta only)
	2	I	500 Hz (Sigma, Zeta, Omega, Gamma and Delta only)
	3	I	280 Hz (Alpha only)
Return value	**Values**		**Description**
LONG	0		Method was executed successfully
	< 0		Error

Available from Version 8.2.0.

```
LONG STSensorSetSampleRateMode(LONG nMode)
```

7.18.1 Implementation

```
LONG nRc = STSensorSetSampleRateMode(1);
if (nRc < 0)
    wprintf(L"Error %d", nRc);
```

7.19 STSensorSetSignRect method

This method defines the rectangle in which the signature is captured. If the rectangle overlaps with one of the fixed hotspots (see `STSensorAddHotSpot()`), an error is returned.

Parameter	Values	I/O	Description
`LONG nLeft`	`>= 0`	I	Left boundary; 0 is on the far left of the display
`LONG nTop`	`>= 0`	I	Upper boundary; 0 is at the top of the display
`LONG nWidth`	`> 3`	I	Width; `STDisplayGetWidth()` returns the width of the LCD used
	0	I	Right boundary is automatically set to the maximum value (right margin of the LCD)
`LONG nHeight`	`> 3`	I	Height; `STDisplayGetHeight()` returns the height of the LCD used
	0	I	Lower boundary is automatically set to the maximum value (lower margin of the LCD)
Return value	**Values**	**Description**	
`LONG`	0	Method was executed successfully	
	`< 0`	Error	

Available from Version 8.2.0. The status described is available from Version 8.5.2.2.

```
LONG STSensorSetSignRect(LONG nLeft, LONG nTop, LONG nWidth, LONG nHeight)
```

7.19.1 Implementation

```
LONG nRc = STSensorSetSignRect(0, 40, 0, 0);
if (nRc < 0)
    wprintf(L"Error %d", nRc);
```

7.20 STSensorClearSignRect method

This method erases the signature window.

Parameter	Values	I/O	Description
-	-	-	-
Return value	**Values**	**Description**	
`LONG`	0	Method was executed successfully	
	`< 0`	Error	

Available from Version 8.2.0.

```
LONG STSensorClearSignRect()
```

7.20.1 Implementation

```
LONG nRc = STSensorClearSignRect();
if (nRc < 0)
    wprintf(L"Error %d", nRc);
```

7.21 STSensorSetScrollArea method

This method defines a rectangular subarea of the non-volatile memory whose content can be scrolled. The subarea must be at least as big as the display and fully encompass the displayed area. Once a connection has been opened, the entire memory is set as the scroll area.

Parameter	Values	I/O	Description
`LONG nLeft`	`>= 0`	I	Left boundary; 0 is on the far left of the memory
`LONG nTop`	`>= 0`	I	Upper boundary; 0 is at the top of the memory
`LONG nWidth`	`> 0`	I	Width, must be >= the value provided by `STDisplayGetWidth();` `STDisplayGetTargetWidth()` provides the width of the currently set memory
	0	I	Right boundary is automatically set to the maximum value (right margin of the memory)
`LONG nHeight`	`> 0`	I	Height, must be >= the value provided by `STDisplayGetHeight();` `STDisplayGetTargetHeight()` provides the height of the currently set memory
	0	I	Lower boundary is automatically set to the maximum value (lower margin of the memory)
Return value	**Values**	**Description**	
`LONG`	0	Method was executed successfully	
	`< 0`	Error	

Available from Version 8.5.2.2.

```
LONG STSensorSetScrollArea(LONG nLeft, LONG nTop, LONG nWidth, LONG nHeight)
```

7.21.1 Implementation

```
LONG nRc = STSensorSetScrollArea(0, 0, 0, 960);
if (nRc < 0)
    wprintf(L"Error %d", nRc);
```

7.22 STSensorSetPenScrollingEnabled method

This method activates scrolling with the pen. In this mode, the memory contents can be offset by moving the pen on the display within the area set via `STSensorSetScrollArea()`. The application is informed about this via the `DisplayScrollPosChanged()` event.

This method only works with the Delta model.

Parameter	Values	I/O	Description
BOOL bEnable	TRUE	I	Scrolling with the pen is activated; signature capture must not have been started.
	FALSE	I	Deactivates scrolling with the pen
Return value	**Values**	**Description**	
LONG	0	Method was executed successfully	
	< 0	Error	

Available from Version 8.5.2.2.

```
LONG STSensorSetPenScrollingEnabled(BOOL bEnable)
```

7.22.1 Implementation

```
LONG nRc = STSensorSetPenScrollingEnabled(TRUE);
if (nRc < 0)
    wprintf(L"Error %d", nRc);
```

7.23 STSensorAddHotSpot method

This method defines a rectangular subarea of the sensor surface that responds to user clicks. See also `SensorHotSpotPressed()`. If a scroll hotspot has already been defined, scrolling has been performed or pen-controlled scrolling (see also `STSensorSetPenScrollingEnabled()`) is active, the rectangle must be in the area defined via `STDisplaySetOverlayRect()`. It should not overlap with the defined signature window (see `STSensorSetSignRect()`) or a hotspot that was previously set.

Parameter	Values	I/O	Description
LONG nLeft	>= 0	I	Left boundary; 0 is on the far left of the display
LONG nTop	>= 0	I	Upper boundary; 0 is at the top of the display
LONG nWidth	> 0	I	Width; `STDisplayGetWidth()` returns the width of the LCD used
	0	I	Right boundary is automatically set to the maximum value (right edge of the LCD)
LONG nHeight	> 0	I	Height; `STDisplayGetHeight()` returns the height of the LCD used
	0	I	Lower boundary is automatically set to the maximum value (lower edge of the LCD)
Return value	**Values**	**Description**	
LONG	>= 0	ID of the hotspot that was generated	
	< 0	Error	

Available from Version 8.2.0.

```
LONG STSensorAddHotSpot(LONG nLeft, LONG nTop, LONG nWidth, LONG nHeight)
```

7.23.1 Implementation

```
LONG nRc = STSensorAddHotSpot(0, 0, 0, 40);
if (nRc < 0)
    wprintf(L"Error %d", nRc);
```

7.24 STSensorAddScrollHotSpot method

This method defines a rectangular subarea of the sensor surface that responds to user clicks. Depending on the option that is used, the subarea is either created as a scroll hotspot or as a scrollable hotspot.

If a scroll hotspot is activated by the user, the screen content is scrolled to the left, to the right, up or down at the speed defined by the `STDisplaySetScrollSpeed()` method and the `DisplayScrollPosChanged()` event is called. A scroll hotspot only responds to clicks if it lies in the area defined by `STDisplaySetOverlayRect()`; otherwise it is inactive.

The rectangle should not overlap with the defined signature window (see `STSensorSetSignRect()`) or a fixed hotspot that was previously set.

A scrollable hotspot behaves like a normal hotspot (also see `STSensorAddHotSpot()`), however, it is moved with the displayed content during scrolling. A scrollable hotspot does not respond to clicks if it lies in the area defined by `STDisplaySetOverlayRect()`.

The rectangle should not overlap with a scrollable hotspot that was previously set. The rectangle should not overlap with the defined signature window (see `STSensorSetSignRect()`) while a signature capture process is currently running.

This method only works with the Omega, Gamma and Delta models as well as with Alpha models with firmware 1.8 or a newer version.

Parameter	Values	I/O	Description
`LONG nLeft`	`>= 0`	I	Left boundary; 0 is on the far left of the display
`LONG nTop`	`>= 0`	I	Upper boundary; 0 is at the top of the display
`LONG nWidth`	`> 0`	I	Width; `STDisplayGetWidth()` returns the width of the LCD used
	0	I	Right boundary is automatically set to the maximum value (right edge of the LCD)
`LONG nHeight`	`> 0`	I	Height; `STDisplayGetHeight()` returns the height of the LCD used
	0	I	Lower boundary is automatically set to the maximum value (lower edge of the LCD)
`LONG nType`	0	I	Scroll hotspot: Pressing the hotspot moves the screen content up (scrolls down)
	1	I	Scroll hotspot: Pressing the hotspot moves the screen content down (scrolls up)
	2	I	Scroll hotspot: Pressing the hotspot moves the screen content to the left (scrolls right); only available with the Alpha model
	3	I	Scroll hotspot: Pressing the hotspot moves the screen content to the right (scrolls left); only available with the Alpha model
	4	I	Scrollable hotspot; only available with the Delta model
Return value	**Values**	**Description**	
`LONG`	`>= 0`	ID of the hotspot that was generated	
	`< 0`	Error	

Available from Version 8.2.0. The status described is available from Version 8.5.2.2.

```
LONG STSensorAddScrollHotSpot(LONG nLeft, LONG nTop, LONG nWidth, LONG nHeight,
HOTSPOTTYPE nType)
```

The `HOTSPOTTYPE` enumeration is defined as follows:

```
kScrollDown = 0,
kScrollUp = 1,
kScrollRight = 2,
kScrollLeft = 3,
kScrollable = 4
```

7.24.1 Implementation

```
LONG nRc = STSensorAddScrollHotSpot(0, 0, 0, 40, kScrollDown);
if (nRc < 0)
    wprintf(L"Error %d", nRc);
```

7.25 STSensorAddKeypadHotSpot method

This method defines a rectangular subarea of the sensor surface that responds to user clicks. This hotspot type is subject to a special security aspect: Although the `SensorHotSpotPressed()` event is triggered when clicked, the ID of the clicked hotspot is not transmitted but is instead kept in a list in the signature device. This list can be read in encrypted form using the `STSensorGetKeypadEntries()` method or deleted using the `STSensorClearKeypadEntries()` method. This hotspot type is therefore suitable for sensitive input such as PINs.

Depending on the device properties, the list in the device may hold up to eight or 32 entries before an overflow occurs; hotspots can still be clicked when a list is filled, however, the information will be lost. The application must therefore read out the list in good time if more than eight or 32 characters need to be entered. See also `STDeviceGetCapabilities()`.

If a scroll hotspot has already been defined, scrolling has been performed or pen-controlled scrolling (see also `STSensorSetPenScrollingEnabled()`) is active, the rectangle must be in the area defined via `STDisplaySetOverlayRect()`. It should not overlap with the defined signature window (see `STSensorSetSignRect()`) or a hotspot that was previously set.

Please note that adding a keypad hotspot empties the list in the device. Therefore, if necessary, make sure to read out the list first using the `STSensorGetKeypadEntries()` method.

This method works with the Sigma model from firmware 2.10, the Zeta model from firmware 1.0, the Omega model from firmware 2.14, the Gamma model from firmware 1.20 and the Delta model from firmware 1.22. For the Sigma model up to firmware 2.14, the Omega model up to firmware 2.18, the Gamma model up to firmware 1.33 and the Delta model up to firmware 1.35, the restriction applies that these methods only work if the memory defined with the `STDisplaySetTarget()` method is the foreground memory.

Parameter	Values	I/O	Description
`LONG nLeft`	`>= 0`	I	Left boundary; 0 is on the far left of the display
`LONG nTop`	`>= 0`	I	Upper boundary; 0 is at the top of the display

LONG nWidth	> 0	I	Width; STDisplayGetWidth() returns the width of the LCD used
	0	I	Right boundary is automatically set to the maximum value (right edge of the LCD)
LONG nHeight	> 0	I	Height; STDisplayGetHeight() returns the height of the LCD used
	0	I	Lower boundary is automatically set to the maximum value (lower edge of the LCD)
LPCWSTR szChars	!= NULL	I	Character string that is stored for the hotspot and returned when STSensorGetKeypadEntries() is called
Return value	**Values**	**Description**	
LONG	>= 0	ID of the hotspot that was generated	
	< 0	Error	

Available from Version 8.5.2.2.

```
LONG STSensorAddKeypadHotSpot(LONG nLeft, LONG nTop, LONG nWidth, LONG nHeight,
LPCWSTR szChars)
```

7.25.1 Implementation

```
LONG nRc = STSensorAddKeypadHotSpot(0, 0, 0, 40, L"1");
if (nRc < 0)
    wprintf(L"Error %d", nRc);
```

7.26 STSensorGetKeypadEntries method

With this method, any number of keypad entries signalled by the SensorHotSpotPressed() event can be read out of the signature device in encrypted form and retrieved as contiguous text consisting of the characters stored when calling STSensorAddKeyPadHotSpot(). The entries that are read out are deleted from the signature device.

This method works with the Sigma model from firmware 2.10, the Zeta model from firmware 1.0, the Omega model from firmware 2.14, the Gamma model from firmware 1.20 and the Delta model from firmware 1.22.

Parameter	Values	I/O	Description
LPCWSTR szEntries	NULL	I	The method returns the length of the character string in the pnStringLength parameter
	!= NULL	I/O	Array in which the character string is to be written
LONG* pnStringLength	>= 0	I/O	Length of the character string incl. terminated 0 or size of the szEntries array in bytes
LONG nMaxEntries	0	I	All existing list elements are read out
	1 - 32	I	Maximum number of list elements to be read from the signature device
Return value	**Values**	**Description**	
LONG	>= 0	Number of list items that have been read out	
	< 0	Error	

Available from Version 8.5.2.2.

```
LONG STSensorGetKeypadEntries(LPCWSTR szEntries, LONG* pnStringLength, LONG
nMaxEntries)
```

7.26.1 Implementation

```
LONG nLen = 0;
LONG nRc = STSensorGetKeypadEntries(NULL, &nLen, 1);
if (nRc == 0)
    wprintf(L"No button has been pressed since last call");
else if (nRc > 0)
{
    WCHAR* szKeypadEntries = new WCHAR[nLen / sizeof(WCHAR)];
    nRc = STSensorGetKeypadEntries(szKeypadEntries, &nLen, 1);
    if (nRc > 0)
        // process entries...
    delete [] szKeypadEntries;
}
if (nRc < 0)
    wprintf(L"Error %d", nRc);
```

7.27 STSensorSetHotspotMode method

This method defines the behaviour of a monitored area (hotspot).

Parameter	Values	I/O	Description
`HOTSPOTMODE nMode`	0	I	Deactivates the monitored area
	1	I	Activates the monitored area (default after calling `STSensorAddHotSpot()` or `STSensorAddScrollHotSpot()`, `STSensorAddKeypadHotSpot()`); scrollable hotspots can only be activated during an ongoing signature capture process if they do not overlap with the defined signature window (see `STSensorSetSignRect()`).
	2	I	Activates the monitored area but disables the automatic inverting when the area is clicked
`LONG nHotSpotId`	>= 0	I	ID of the hotspot that is to be changed
Return value	**Values**	**Description**	
`LONG`	0	Method was executed successfully	
	< 0	Error	

Available from Version 8.2.0. The status described is available from Version 8.5.2.2.

```
LONG STSensorSetHotSpotMode(HOTSPOTMODE nMode, LONG nHotSpotId)
```

The `HOTSPOTMODE` enumeration is defined as follows:

```
kInactive = 0,
kActive = 1,
kInvertOff = 2
```

7.27.1 Implementation

```
LONG nRc = STSensorSetHotspotMode(kActive, 0);
if (nRc < 0)
    wprintf(L"Error %d", nRc);
```

7.28 STSensorClearHotSpots method

This method removes all monitored areas (hotspots). However, it does not delete the list of actuated keypad hotspots in the signature device so that it can still be read out.

Parameter	Values	I/O	Description
-	-	-	-
Return value	**Values**	**Description**	
LONG	0	Method was executed successfully	
	< 0	Error	

Available from Version 8.2.0.

```
LONG STSensorClearHotSpots()
```

7.28.1 Implementation

```
LONG nRc = STSensorClearHotSpots();
if (nRc < 0)
    wprintf(L"Error %d", nRc);
```

7.29 STSensorClearKeypadEntries method

This method deletes the list of actuated keypad hotspots in the signature device, but does not remove the hotspots themselves. To do this, please use the `STSensorClearHotSpots`() method.

Keypad hotspots work with the Sigma model from firmware 2.10, the Zeta model from firmware 1.0, the Omega model from firmware 2.14, the Gamma model from firmware 1.20 and the Delta model from firmware 1.22.

Parameter	Values	I/O	Description
-	-	-	-
Return value	**Values**	**Description**	
LONG	0	Method was executed successfully	
	< 0	Error	

Available from Version 8.5.2.2.

```
LONG STSensorClearKeypadEntries()
```

7.29.1 Implementation

```
LONG nRc = STSensorClearKeypadEntries();
if (nRc < 0)
    wprintf(L"Error %d", nRc);
```

7.30 STSensorStartTimer method

This method starts a Timer, which starts a defined function, if there was no interaction on the sensor of the pad for the given time periods. This Functionality is intended primarily to capture a signature without user interaction, but it can also be used to get a confirmation for a displayed text, if the belonging hotspot is not pressed for a given time period.

Parameter	Values	I/O	Description
LONG nWaitBeforeAction	0	I	No timer waiting for the first interaction is started
	> 0	I	Maximum time to wait for the first interaction (in milliseconds) before the defined function is triggered (for example, before the start of a signature); the timer is restarted with this value after the calling of STSignatureRetry().
LONG nWaitAfterAction	0	I	After the first interaction no timer waiting for the next interaction is started
	> 0	I	Maximum time to wait after the last interaction was noticed in millisecond. If the given time period elapsed without a new interaction, the wanted function will be called (usually this takes places after the signing is finished).
LONG nOptions	0	I	If the timer has expired, the SensorTimeoutOccured() event is called.
	1	I	If the time period of nWaitBeforeAction has elapsed, STSignatureCancel() is called; if the time period of nWaitAfterAction has elapsed, STSignatureConfirm() is called
	2	I	If the timer has expired, STSignatureCancel() is called
Return value	**Values**	**Description**	
LONG	0	Method was executed successfully	
	< 0	Error	

Available from Version 8.2.0.

```
LONG STSensorStartTimer(LONG nWaitBeforeAction, LONG nWaitAfterAction, LONG
nOptions)
```

7.30.1 Implementation

```
LONG nRc = STSensorStartTimer(10000, 1000, 0);
if (nRc < 0)
    wprintf(L"Error %d", nRc);
```

7.31 STSensorStopTimer method

This method stops a timer started with STSensorStartTimer() without triggering the function defined there. The method is called automatically if STSignatureConfirm() or STSignatureCancel() is called.

Parameter	Values	I/O	Description
–	–	-	-
Return value	**Values**	**Description**	
LONG	0	Method was executed successfully (will be returned also, if no timer was set before)	
	< 0	Error	

Available from Version 8.2.0.

```
LONG STSensorStopTimer()
```

7.31.1 Implementation

```
LONG nRc = STSensorStopTimer();
if (nRc < 0)
    wprintf(L"Error %d", nRc);
```

7.32 STSignatureStart method

This method starts the signature capture process provided a connection has been opened via `STDeviceOpen()`. The entire sensor is used as a writing surface provided no signature window has been defined. Signature data is only received, if a signature is actually entered on the pad. The method sets the colour of the LED to green unless the method `STDeviceSetLedDefaultFlag()` was previously called with FALSE. This method automatically restores the previous content of the LCD unless this has been explicitly erased by calling `STDisplayErase()`.

The method cannot be called if an active scrollable hotspot overlaps with the defined signature window (see `STSensorSetSignRect()`).

Parameter	Values	I/O	Description
-	-	-	-
Return value	**Values**	**Description**	
LONG	0	Method was executed successfully	
	< 0	Error	

Available from Version 8.2.0. The status described is available from Version 8.5.2.2.

```
LONG STSignatureStart()
```

7.32.1 Implementation

```
LONG nRc = STSignatureStart();
if (nRc < 0)
    wprintf(L"Error %d", nRc);
```

7.33 STSignatureStop method

This method terminates the signature capture process that is currently running, and caches the captured signature data. Unlike the `STSignatureConfirm()` method, it does not change the display content. `STSignatureStop()` sets the colour of the LED to yellow, unless the method `STDeviceSetLedDefaultFlag()` was previously called with FALSE.

Parameter	Values	I/O	Description
-	-	-	-
Return value	**Values**	**Description**	
LONG	>= 0	Number of points captured	
	< 0	Error	

Available from Version 8.2.0.

```
LONG STSignatureStop()
```

7.33.1 Implementation

```
LONG nRc = STSignatureStop();
if (nRc < 0)
    wprintf(L"Error %d", nRc);
else
    wprintf(L"%d points captured.", nRc);
```

7.34 STSignatureConfirm method

This methods terminates the signature capture process that is currently running (if any), caches the captured signature data and, unlike `STSignatureStop()`, erases the entire LCD. `STSignatureConfirm()` sets the colour of the LED to yellow, unless the method `STDeviceSetLedDefaultFlag()` was previously called with FALSE.

Parameter	Values	I/O	Description
–	–	-	-

Return value	Values	Description
LONG int Integer	>= 0	Number of points captured
	< 0	Error

Available from Version 8.2.0.

```
LONG STSignatureConfirm()
```

7.34.1 Implementation

```
LONG nRc = STSignatureConfirm();
if (nRc < 0)
    wprintf(L"Error %d", nRc);
else
    wprintf(L"%d points captured.", nRc);
```

7.35 STSignatureRetry method

This method discards the signature data without ending the signature capture process, and deletes the rendered signature and in the LCD. This method starts a new capture process if the signature capture process was terminated beforehand with `STSignatureStop()`.

Parameter	Values	I/O	Description
–	–	-	-

Return value	Values	Description
LONG	0	Method was executed successfully
	< 0	Error

Available from Version 8.2.0.

```
LONG STSignatureRetry()
```

7.35.1 Implementation

```
LONG nRc = STSignatureRetry();
if (nRc < 0)
    wprintf(L"Error %d", nRc);
```

7.36 STSignatureCancel method

This method ends the capture process, discards the signature data and deletes the entire LCD or just the signature. The colour of the LED is set to yellow, unless the method `STDeviceSetLedDefaultFlag()` was previously called with FALSE. This method is called automatically when `STDeviceClose()` is called.

Parameter	Values	I/O	Description
`ERASEOPTION nErase`	0	I	The entire LCD will be deleted (Default)
	1	I	Only the signature will be deleted (Optional)
Return value	**Values**	**Description**	
`LONG`	0	Method was executed successfully	
	< 0	Error	

Available from Version 8.2.0.

```
LONG STSignatureCancel(ERASEOPTION nErase=kComplete)
```

The `ERASEOPTION` enumeration is defined as follows:

```
kComplete = 0,
kSignature = 1
```

7.36.1 Implementation

```
LONG nRc = STSignatureCancel();
if (nRc < 0)
    wprintf(L"Error %d", nRc);
```

7.37 STSignatureGetState method

This method returns the current state of the signature capture process.

Value	I/O	Description
`TRUE`	O	Signature capture process is running
`FALSE`	O	Signature capture process is not running

Available from Version 8.2.0.

```
BOOL STSignatureGetState()
```

7.37.1 Implementation

```
if (!STSignatureGetState())
    STSignatureStart();
else
    STSignatureConfirm();
```

7.38 STSignatureGetSignData method

This method returns the digitalised signature in SignData format.

Parameter	Values	I/O	Description
BYTE* pbtSignData	NULL	I	The method returns the required size of the array in the pnSize parameter.
	other	I/O	Array (in the required size) in which the SignData is written; pnSize must correspond to the value returned for the previous call.
LONG* pnSize	> 0	I/O	Size of the array (in bytes) in which the SignData is to be written
Return value	**Values**	**Description**	
LONG	0	Method was executed successfully	
	< 0	Error	

Available from Version 8.2.0.

```
LONG STSignatureGetSignData(BYTE* pbtSignData, LONG* pnSize)
```

7.38.1 Implementation

```
LONG nSize = 0;
LONG nRc = STSignatureGetSignData(NULL, &nSize);
BYTE* pbtSignData = NULL;
if (nRc == 0)
{
    pbtSignData = new BYTE[nSize];
    nRc = STSignatureGetSignData(pbtSignData, &nSize);
}
if (nRc < 0)
    wprintf(L"Error %d", nRc);
```

7.39 STSignatureSaveAsFileEx method

This method can be used to save a captured signature as an image file on a hard disk. The colour depth depends on the file type, the device used and the settings. If no further settings are made (see nOptions parameter), the image is created with the aspect ratio of the rectangle that surrounds the signature.

Parameter	Values	I/O	Description
LPCWSTR szPath	!= NULL	I	Storage location for the image file as a full path that includes the file name
LONG nResolution	>=75 <=600	I	Resolution of the image file in pixels per inch (ppi); for the signature to be displayed in its original size, this value must be identical to the resolution of the document, in which the signature is to be integrated
LONG nWidth	0	I	The image will be created in original size; the nHeight parameter is ignored.
	> 0	I	Maximum width of the image in pixels
LONG nHeight	0	I	The image will be created in original size; the nWidth parameter is ignored
	> 0	I	Maximum height of the image in pixels

FILETYPE nFileType	0	I	Use TIFF with CCITT4 compression (b/w image) or LZW compression (colour image) as the file format (recommended)
	1	I	Use PNG file format
	3	I	Use JPEG with a quality setting of 75 as the file format
LONG nPenWidth	< 0	I	Fixed pen width in pixels (absolute value); the pressure values are visualised by drawing in variable brightness
	0	I	A variable pen width is used that is dependent on the resolution and the pressure values
	> 0	I	Fixed pen width in pixels
COLORREF clrPen	>= 0	I	Signature colour

LONG nOptions	Bitmask containing one or more hexadecimal values from the following list:		
	0x0001	I	A visual timestamp is added to the image beneath the signature
	0x0002	I	The signature is rendered in the image displayed during capture; the image always has the aspect ratio of the display that is used, nWidth or nHeight may be ignored
	0x0004	I	The defined hotspot areas are whitened in the image (only if 0x0002 is set).
	0x0008	I	White areas at the sides of the signature are not removed; if nWidth and nHeight are greater than 0, the signature is scaled to the defined height or width depending on aspect ratio, and the image to be created has the exact defined size (only if 0x0002 is not set).
	0x0010	I	The signature will be aligned to the left (only if 0x0008 is set).
	0x0020	I	The signature will be aligned to the right (only if 0x0008 is set)
	0x0040	I	The signature will be aligned to the top (only if 0x0008 is set)
	0x0080	I	The signature will be aligned to the bottom (only if 0x0008 is set)
	0x0100	I	The timestamp size is relative to the height of the created image, not to the height of the display; this setting is useful if the signature is scaled to a given image size to make sure that the timestamp size is independent from the size of the actual signature (only if 0x0001 is set)
	0x0200	I	The signature is never smoothed independent from all other settings; this will create small files
	0x0400	I	The signature is always smoothed independent from all other settings
	0x0800	I	The image includes the overlay rectangle if displayed (only if 0x0002 is set)
	0x1000	I	White areas are stored as transparent (only if 0x0002 is not set and PNG is selected as the file format)
	0x2000	I	The current display content and not the content displayed during capture is used as the background image (only if 0x0002 is set)
	0x4000	I	The pen width will vary by the value indicated depending on the pressure values
Return value	**Values**	**Description**	
LONG	0	Method was executed successfully	
	< 0	Error	

Available from Version 8.2.0.

```
LONG STSignatureSaveAsFileEx(LPCWSTR szPath, LONG nResolution, LONG nWidth,
LONG nHeight, FILETYPE nFileType, LONG nPenWidth, COLORREF clrPen, LONG
nOptions)
```

The `FILETYPE` enumeration is defined as follows:

```
kTiff = 0,
kPng = 1,
kJpeg = 3,
```

The following values defined in the header file can be used for the `nOptions` parameter:

```
#define STPAD_SIMG_TIMESTAMP         0x0001
#define STPAD_SIMG_BACKIMAGE         0x0002
#define STPAD_SIMG_HOTSPOTS          0x0004
#define STPAD_SIMG_NOCROPPING        0x0008
#define STPAD_SIMG_ALIGNLEFT         0x0010
#define STPAD_SIMG_ALIGNRIGHT        0x0020
#define STPAD_SIMG_ALIGNTOP          0x0040
#define STPAD_SIMG_ALIGNBOTTOM       0x0080
#define STPAD_SIMG_TIMESTAMPIMGREL   0x0100
#define STPAD_SIMG_DONTSMOOTH        0x0200
#define STPAD_SIMG_SMOOTH            0x0400
#define STPAD_SIMG_OVERLAYIMAGE      0x0800
#define STPAD_SIMG_TRANSPARENT       0x1000
#define STPAD_SIMG_CURRENTIMAGES     0x2000
#define STPAD_SIMG_VARIABLEPENWIDTH 0x4000
```

7.39.1 Implementation

```
LONG nRc = STSignatureSaveAsFileEx(L"./Signature.png", 300, 0, 0, kPng,
                                   0, RGB(0, 0, 255), 0);
if (nRc < 0)
    wprintf(L"Error %d", nRc);
```

7.40 STSignatureGetBounds method

This method delivers the coordinates of the rectangle in which the captured signature is given.

Parameter	Values	I/O	Description
`LONG* pnLeft`	`>= 0`	O	Left border of the signature rectangle
`LONG* pnTop`	`>= 0`	O	Upper border of the signature rectangle
`LONG* pnRight`	`>= 0`	O	Right border of the signature rectangle
`LONG* pnBottom`	`>= 0`	O	Bottom border of the signature rectangle
`LONG nOptions`	0	I	The coordinates will be delivered relative to the size of the used LCD
	1	I	The coordinates will be returned relative to the defined size of the signature rectangle (see `STSensorSetSignRect()`).
Return value	**Values**	**Description**	
`LONG`	0	Method was executed successfully	
	`< 0`	Error	

Available from Version 8.2.0.

```
LONG STSignatureGetBounds(LONG* pnLeft, LONG* pnTop, LONG* pnRight, LONG*
pnBottom, LONG nOptions)
```

The following values defined in the header file can be used for the `nOptions` parameter:

```
#define STPAD_BOUNDS_DISPLAY   0
#define STPAD_BOUNDS_SIGNRECT  1
```

7.40.1 Implementation

```
LONG nLeft, nTop, nRight, nBottom;
LONG nRc = STSignatureGetBounds(&nLeft, &nTop, &nRight, &nBottom,
                                STPAD_BOUNDS_DISPLAY);
if (nRc < 0)
    wprintf(L"Error %d", nRc);
else
{
    wprintf(L"The Bounds of the Signature are: %d (left), %d (top), %d
            (right) & %d (bottom).", nLeft, nTop, nRight, nBottom);
}
```

7.41 STSignatureScaleToDisplay method

This method converts the sensor coordinates delivered by the `SignatureDataReceived()` event into display coordinates.

Parameter	Values	I/O	Description
LONG nSensorValue	>= 0	I	x or y value of a sensor coordinate
Return value	**Values**	**Description**	
LONG	0	x or y value of a display coordinate	
	< 0	Error	

Available from Version 8.2.0.

```
LONG STSignatureScaleToDisplay(LONG nSensorValue)
```

7.41.1 Implementation

```
LONG nRc = STSignatureScaleToDisplay(1000);
if (nRc < 0)
    wprintf(L"Error %d", nRc);
else
    wprintf(L"Display Value: %d", nRc);
```

7.42 STDisplayErase method

This method erases both the foreground and the background memory and removes the overlay rectangle if set. Thus the entire contents of the LCD are erased. To erase only parts of the memory defined with `STDisplaySetTarget()`, please use `STDisplayEraseRect()`.

Parameter	Values	I/O	Description
-	-	-	-
Return value	**Values**	**Description**	
LONG	0	Method was executed successfully	
	< 0	Error	

Available from Version 8.2.0.

```
LONG STDisplayErase()
```

7.42.1 Implementation

```
LONG nRc = STDisplayErase();
if (nRc < 0)
    wprintf(L"Error %d", nRc);
```

7.43 STDisplayEraseRect method

This method erases a rectangle in the memory defined with `STDisplaySetTarget()`.

Parameter	Values	I/O	Description
`LONG nLeft`	`>= 0`	I	Left boundary; 0 is on the far left of the display
`LONG nTop`	`>= 0`	I	Upper boundary; 0 is at the top of the display
`LONG nWidth`	`> 0`	I	Width; `STDisplayGetWidth()` returns the width of the LCD used
	0	I	Right boundary is automatically set to the maximum value (right edge of the LCD)
`LONG nHeight`	`> 0`	I	Height; `STDisplayGetHeight()` returns the height of the LCD used
	0	I	Lower boundary is automatically set to the maximum value (lower edge of the LCD)
Return value	**Values**	**Description**	
`LONG`	0	Method was executed successfully	
	`< 0`	Error	

Available from Version 8.2.0.

```
LONG STDisplayEraseRect(LONG nLeft, LONG nTop, LONG nWidth, LONG nHeight)
```

7.43.1 Implementation

```
LONG nRc = STDisplayEraseRect(10, 50, 30, 20);
if (nRc < 0)
    wprintf(L"Error %d", nRc);
```

7.44 STDisplayConfigPen method

This method sets the pen width and colour used to display a signature on the LCD. The pen width is always stored permanently in the device; the pen colour is stored permanently only on Omega devices with firmware 1.4 or later.

Parameter	Values	I/O	Description
`LONG nWidth`	`1 - 3`	I	Width in pixels
`COLORREF clrPen`	`>= 0`	I	Colour; this parameter is ignored for the Sigma and Zeta models.
Return value	**Values**	**Description**	
`LONG`	0	Method was executed successfully	
	`< 0`	Error	

Available from Version 8.2.0.

```
LONG STDisplayConfigPen(LONG nWidth, COLORREF clrPen)
```

7.44.1 Implementation

```
LONG nRc = STDisplayConfigPen(2, RGB(0, 0, 255));
if (nRc < 0)
    wprintf(L"Error %d", nRc);
```

7.45 STDisplaySetFont method

This method permanently sets the font that is used to output text to the LCD. Text that has already been output is not modified. Ubuntu 20 pt (Sigma and Zeta models) or 40 pt (Omega, Gamma, Delta and Alpha models) is set when `STDeviceOpen()` is called.

Parameter	Values	I/O	Description
`LPCWSTR szName`	`!= NULL`	I	Full name of the font, which must be installed on the PC
`LONG nSize`	`12 - 200`	I	Font size
`LONG nOptions`	Bitmask containing one or more hexadecimal values from the following list:		
	`0x01`	I	Bold
	`0x02`	I	Underline
	`0x04`	I	Italic
Return value	**Values**	**Description**	
`LONG`	`0`	Method was executed successfully	
	`< 0`	Error	

Available from Version 8.2.0.

```
LONG STDisplaySetFont(LPCWSTR szName, LONG nSize, LONG nOptions)
```

The following values defined in the header file can be used for the `nOptions` parameter:

```
#define STPAD_FONT_NORMAL     0x00
#define STPAD_FONT_BOLD       0x01
#define STPAD_FONT_UNDERLINE  0x02
#define STPAD_FONT_ITALIC     0x04
```

7.45.1 Implementation

```
LONG nRc = STDisplaySetFont(L"Ubuntu Mono", 20, STPAD_FONT_NORMAL);
if (nRc < 0)
    wprintf(L"Error %d", nRc);
```

7.46 STDisplaySetFontColor method

This method permanently sets the colour in which the text is displayed on the LCD. Text that has already been output is not modified. The given values will be ignored, if a pad without a colour LCD is used. The colour black is set when the component is initialised.

Parameter	Values	I/O	Description
`COLORREF clrFont`	`>= 0`	I	Text colour

Return value	Values	Description
LONG	0	Method was executed successfully
	< 0	Error

Available from Version 8.2.0.

```
LONG STDisplaySetFontColor(COLORREF clrFont)
```

7.46.1 Implementation

```
LONG nRc = STDisplaySetFontColor(RGB(238, 121, 0));
if (nRc < 0)
    wprintf(L"Error %d", nRc);
```

7.47 STDisplayGetWidth method

This method returns the width of the LCD. It can only be queried after a device has been opened.

Parameter	Values	I/O	Description
–	–	-	-
Return value	**Values**	**Description**	
LONG	>= 0	Width of the display in pixels	
	< 0	Error	

Available from Version 8.2.0.

```
LONG STDisplayGetWidth()
```

7.47.1 Implementation

```
wprintf(L"Display width is %d", STDisplayGetWidth());
```

7.48 STDisplayGetHeight method

This method holds the height of the LCD It can only be queried after a device has been opened.

Parameter	Values	I/O	Description
–	–	-	-
Return value	**Values**	**Description**	
LONG	>= 0	Height of the display in pixels	
	< 0	Error	

Available from Version 8.2.0.

```
LONG STDisplayGetHeight()
```

7.48.1 Implementation

```
wprintf(L"Display height is %d", STDisplayGetHeight());
```

7.49 STDisplayGetResolution method

This property contains the resolution of the LCD. It can only be queried after a device has been opened.

Parameter	Values	I/O	Description
-	-	-	-
Return value	**Values**	**Description**	
LONG	>= 0	Resolution of the display in pixels per inch (ppi).	
	< 0	Error	

Available from Version 8.5.2.2.

```
DOUBLE STDisplayGetResolution()
```

7.49.1 Implementation

```
wprintf(L"Display resolution is %d", STDisplayGetResolution());
```

7.50 STDisplayGetRotation method

This method returns the set rotation with which the image data is transferred to the signature device and the sensor areas are defined.

Parameter	Values	I/O	Description
-	-	-	-
Return value	**Values**	**Description**	
LONG	0, 180	Rotation in degrees (clockwise)	
	< 0	Error	

Available from Version 8.5.2.2.

```
LONG STDisplayGetRotation()
```

7.50.1 Implementation

```
LONG nRotation = STDisplayGetRotation();
```

7.51 STDisplaySetRotation method

This method specifies the set rotation with which the image data is transferred to the signature device and the sensor areas are defined. A change to the value only affects images and sensor areas which are transferred after the change is made and not images which are copied from one pad memory to another. The default setting is 0.

Parameter	Values	I/O	Description
LONG nRotation	0, 180	I	Rotation in degrees (clockwise)
Return value	**Values**	**Description**	
LONG	0, 180	Actual set value	
	< 0	Error	

Available from Version 8.0.29.

```
LONG STDisplaySetRotation(LONG nRotation)
```

7.51.1 Implementation

```
STDisplaySetRotation(180);
```

7.52 STDisplayGetTargetWidth method

This property holds the width of the memory defined with the `STDisplaySetTarget()` method. It can only be queried after a device has been opened.

Parameter	Values	I/O	Description
-	-	-	-
Return value	**Values**	**Description**	
LONG	>= 0	Width of the memory in pixels	
	< 0	Error	

Available from Version 8.2.0.

```
LONG STDisplayGetTargetWidth()
```

7.52.1 Implementation

```
wprintf(L"Target width is %d", STDisplayGetTargetWidth());
```

7.53 STDisplayGetTargetHeight method

This property holds the height of the memory defined with the `STDisplaySetTarget()` method. It can only be queried after a device has been opened.

Parameter	Values	I/O	Description
-	-	-	-
Return value	**Values**	**Description**	
LONG	>= 0	Height of the memory in pixels	
	< 0	Error	

Available from Version 8.2.0.

```
LONG STDisplayGetTargetHeight()
```

7.53.1 Implementation

```
wprintf(L"Target height is %d", STDisplayGetTargetHeight());
```

7.54 STDisplaySet Target method

This method defines the device memory that is used by the following methods and properties: `STDisplayEraseRect()`, `STDisplaySetText()`, `STDisplaySetTextInRect()`, `STDisplaySetImage()`, `STDisplaySetImageFromFile()`, `STDisplaySetImageFromStore()`, `STDisplaySetScrollPos()`, `STDisplayGetScrollPos()`, `STDisplayTargetWidth` and `STDisplayTargetHeight`. The set memory remains valid until the next call of this method or

of `STDeviceClose()`. Contents stored in a non-visible memory can be displayed with the `STDisplaySetImageFromStore()` method. For more details, see Chapter 6.

After calling `STDeviceOpen()`, the methods specified above are all executed directly on the LCD (foreground memory) as long as `STDisplaySetTarget()` is not called.

Parameter	Values	I/O	Description
`LONG nTarget`	`-2`	I	A permanent memory that can hold an image of the same width and double the height of the display is reserved inside the device; the memory can be used for writing from now on; if there is no permanent memory available, the return value will be 1 (see below); the value -2 is handled as -1 for the Gamma, Delta and Alpha models (see there for details)
	`-1`	I	A permanent memory that can hold an image in the size of the display (Omega, Gamma and Delta models) or up to a size of 2048 x 2048 pixels (Alpha model) is reserved inside the device; the memory can be used for writing from now on; if there is no permanent memory available, the return value will be 1 (see below)
	`0`	I	All content is displayed directly on the LCD and stored in the foreground memory; the content is lost if the device is switched off or if `STDisplayErase()` or `STDeviceClose()` is called
	`1`	I	All content is written to the non-visible background memory; the background memory is used internally during the signature process, so content is no longer available after `STSignatureStart()` has been called; the content is also lost if the device is switched off or if `STDisplayErase()` or `STDeviceClose()` is called.
	`2`	I	All content is written to the overlay memory; it is visible immediately if an overlay rectangle has already been defined; the content is lost if the device is switched off or if `STDisplayErase()` or `STDeviceClose()` is called; this value cannot be used for the Sigma and Zeta models.
	other	I	Enables direct access to a permanent storage; The storage must be reserved before it can be used (see above for value -1 and -2)
Return value	**Values**	**Description**	
`LONG`	`>= 0`	ID of the store which is now selected; all the above referred methods will now be applied to this store; this ID can be used when calling this method again to specifically address this store	
	`< 0`	Error	

Available from Version 8.2.0.

```
LONG STDisplaySetTarget(LONG nTarget)
```

The following values defined in the header file can be used for the `nTarget` parameter:

```
#define STPAD_TARGET_LARGESTORE    -2
#define STPAD_TARGET_STANDARDSTORE -1
#define STPAD_TARGET_FOREGROUND     0
#define STPAD_TARGET_BACKGROUND     1
#define STPAD_TARGET_OVERLAY        2
```

7.54.1 Implementation

```
LONG nStoreId = STDisplaySetTarget(STPAD_TARGET_STANDARDSTORE);
if (nStoreId < 0)
    wprintf(L"Error %d", nStoreId);
```

7.55 STDisplaySetText method

This method can be used to write any text to the memory defined with the `STDisplaySetTarget()` method. The rectangle enclosing the text overlays existing information in the memory. The text can also appear outside of the display and it is not wrapped unless it already contains breaks. Ubuntu 20 pt (Sigma and Zeta models) or 40 pt (Omega, Gamma, Delta and Alpha models) is used, unless another font has been set using the `STDisplaySetFont()` method. The font colour will be black unless another colour has been set using the `STDisplaySetFontColor()` method.

Parameter	Values	I/O	Description
`LONG nXPos`	all	I	X coordinate of the starting point; 0 is on the far left of the display; `STDisplayGetWidth()` returns the point on the far right of the display
`LONG nYPos`	all	I	Y coordinate of the starting point; 0 is at the top of the display; `STDisplayGetHeight()` returns the point at the very bottom of the display
`ALIGN nAlignment`	0	I	Text is aligned to the right of the starting point
	1	I	Text is centred horizontally at the starting point
	2	I	Text is aligned to the left of the starting point
`LPCWSTR szText`	`!= NULL`	I	Text to be output
Return value	**Values**	**Description**	
`LONG`	`>= 0`	Width of the rectangle enclosing the text	
	`< 0`	Error	

Available from Version 8.2.0.

```
LONG STDisplaySetText(LONG nXPos, LONG nYPos, ALIGN nAlignment, LPCWSTR szText)
```

The `ALIGN` enumeration is defined as follows:

```
kLeft = 0,
kCenter = 1,
kRight = 2,
kLeftCenteredVertically = 3,
kCenterCenteredVertically = 4,
kRightCenteredVertically = 5
```

7.55.1 Implementation

```
LONG nRc = STDisplaySetText(50, 20, kLeft, L"Text");
if (nRc < 0)
    wprintf(L"Error %d", nRc);
```

7.56 STDisplaySetTextInRect method

This method can be used to write any text to the memory defined with the `STDisplaySetTarget()` method. The specified rectangle overlays existing information in the memory. The text is placed in the rectangle and can optionally be wrapped automatically. No check is made regarding whether the rectangle is within the display. Ubuntu 20 pt (Sigma and Zeta models) or 40 pt (Omega, Gamma, Delta and Alpha models) is used, unless another font has been set using the `STDisplaySetFont()` method. If the text is too long, the font size is automatically reduced to a minimum of 12 pt (see also options parameter). The font colour will be black unless another colour has been set using the `STDisplaySetFontColor()` method.

Parameter	Values	I/O	Description
`LONG nLeft`	all	I	Left boundary; 0 is on the far left of the display
`LONG nTop`	all	I	Upper boundary; 0 is at the top of the display
`LONG nWidth`	`> 0`	I	Width; `STDisplayGetWidth()` returns the width of the LCD used
	0	I	Right boundary is automatically set to the maximum value (right margin of the LCD)
`LONG nHeight`	`> 0`	I	Height; `STDisplayGetHeight()` returns the height of the LCD used
	0	I	Lower boundary is automatically set to the maximum value (lower margin of the LCD)
`ALIGN nAlignment`	0	I	Text is left-aligned and wrapped automatically
	1	I	Text is centred and wrapped automatically
	2	I	Text is right-aligned and wrapped automatically
	3	I	Text is left-aligned and centred vertically in the rectangle with no wrapping (breaks are ignored)
	4	I	Text is centred vertically and horizontally in the rectangle with no wrapping (breaks are ignored); this setting is ideal for button text
	5	I	Text is right-aligned and centred vertically in the rectangle with no wrapping (breaks are ignored)
	6	I	Text is left-aligned and not wrapped automatically (breaks are retained)
	7	I	Text is centred and not wrapped automatically (breaks are retained)
	8	I	Text is right-aligned and not wrapped automatically (breaks are retained)
`LPCWSTR szText`	`!= NULL`	I	Text to be output

LONG nOptions	Bitmask containing one or more hexadecimal values from the following list (optional):		
	0x01	I	Instead of the font size, the height of the text block in pixels is returned; if the text does not fit in the given rectangle, it is not output, and the height that is necessary to output the text in the desired font size is returned; an error is returned if the longest word in the text does not fit in a line of the given rectangle.
Return value	**Values**	**Description**	
LONG	>=0	The font size that is actually used or the height of the text in pixels (see also options parameter)	
	< 0	Error	

Available from Version 8.2.0. The status described is available from Version 8.5.2.2.

```
LONG STDisplaySetTextInRect(LONG nLeft, LONG nTop, LONG nWidth, LONG nHeight,
ALIGN nAlignment, LPCWSTR szText, LONG nOptions=0)
```

The `ALIGN` enumeration is defined as follows:

```
kLeft = 0,
kCenter = 1,
kRight = 2,
kLeftCenteredVertically = 3,
kCenterCenteredVertically = 4,
kRightCenteredVertically = 5,
kLeftNoWrap = 6,
kCenterNoWrap = 7,
kRightNoWrap = 8
```

The following values defined in the header file can be used for the `nOptions` parameter:

```
#define STPAD_TEXT_NORESIZE   0x01
```

7.56.1 Implementation

```
LONG nRc = STDisplaySetTextInRect(0, 0, 20, 40, kLeft, L"Text");
if (nRc < 0)
    wprintf(L"Error %d", nRc);
```

7.57 STDisplaySetImageFromFile method

This method can be used to write an image to the memory defined with the `STDisplaySetTarget()` method. Although the colour depth is automatically adjusted to the connected LCD, it is still advisable to correctly generate the image beforehand (for example, a 1-bit monochrome image is required for the Sigma and Zeta models). If there is an existing Alpha channel, it is ignored. The transfer time for the Omega, Gamma, Delta and Alpha models depends on the image material; the best pictures have few colours, so they can be compressed well. The image overlays the existing information in the memory and any signature that is present is completely erased. The image may also be positioned outside of the display.

Parameter	Values	I/O	Description
LONG nXPos	all	I	X coordinate of the point from which the bitmap is output to the right; 0 is on the far left of the display; STDisplayGetWidth() returns the point on the far right of the display
LONG nYPos	all	I	Y coordinate of the point from which the bitmap is output downwards; 0 is at the top of the display; STDisplayGetHeight() returns the point at the very bottom of the display
LPCWSTR szPath	!= NULL	I	Full file path of the image; PNG and TIFF can be used as image formats
Return value	**Values**	**Description**	
LONG	0	Method was executed successfully	
	< 0	Error	

Available from Version 8.2.0.

```
LONG STDisplaySetImageFromFile(LONG nXPos, LONG nYPos, LPCWSTR szPath)
```

7.57.1 Implementation

```
LONG nRc = STDisplaySetImageFromFile(0, 0, L"./Image.png");
if (nRc < 0)
    wprintf(L"Error %d", nRc);
```

7.58 STDisplaySetImageFromStore method

This method allows an image that has been stored in a device memory to be written to the memory defined by STDisplaySetTarget(). The content to copy will overlay the content which is currently stored in the target storage. For more details, see Chapter 6.

If the memory defined by nStoreId was not reserved beforehand by calling STDisplaySetTarget(), the content is copied as desired; however, it is not available within the component for storing with the STDisplaySaveImageFile() or STSignatureSaveFileEx() method. To differentiate this case from a call with a reserved nStoreId, nStoreId is returned instead of 0.

The scroll position of the source memory will be assigned to the destination memory, if both have the same size, else it will be set to 0 / 0.

Parameter	Values	I/O	Description
LONG nStoreId	>= 0	I	ID of the memory from which the image is to be read; the ID is the value returned by DisplaySetTarget().
Return value	**Values**	**Description**	
LONG	> 2	The memory defined by nStoreId has not been reserved beforehand; the content was successfully copied, but is not available within the component; the returned value is identical to the value of nStoreId	
	0	Method was executed successfully	
	< 0	Error	

Available from Version 8.2.0.

```
LONG STDisplaySetImageFromStore(LONG nStoreId)
```

The following values defined in the header file or the ID of a reserved, non-volatile memory can be used for the `nStoreId` parameter:

```
#define STPAD_TARGET_FOREGROUND     0
#define STPAD_TARGET_BACKGROUND     1
```

7.58.1 Implementation

```
LONG nRc = STDisplaySetImageFromStore(STPAD_TARGET_BACKGROUND);
if (nRc < 0)
    wprintf(L"Error %d", nRc);
```

7.59 STDisplaySetOverlayRect method

This method overlays a rectangular section of the overlay memory on the content of the foreground memory. The foreground memory is covered within this rectangle until it is removed again or until `STDisplayErase()`, `STSignatureConfirm()` or `STSignatureCancel()` is called. This functionality is ideal for a toolbar that displays hotspots, for example, to scroll.

If the storage defined with `STDisplaySetTarget()` is not the foreground memory, the rectangle is not set until `STDisplaySetImageFromStore()` (with the foreground memory as the destination) is called to synchronize the display.

The parameters must be multiples of eight when using Omega (with firmware 1.x) and Alpha models and are rounded if necessary.

This method cannot be called when a both a standard hotpot lying outside of the given rectangle and a scroll hotspot has been defined previously.

This method only works with the Omega, Gamma, Delta and Alpha models!

Parameter	Values	I/O	Description
`LONG nLeft`	`>= 0`	I	Left boundary; 0 is on the far left of the display
`LONG nTop`	`>= 0`	I	Upper boundary; 0 is at the top of the display
`LONG nWidth`	`>= 8`	I	Width; `STDisplayGetWidth()` returns the width of the LCD used
	0	I	The overlay rectangle is removed; the complete contents of the foreground memory will be visible again.
`LONG nHeight`	`>= 8`	I	Height; `STDisplayGetHeight()` returns the height of the LCD used
	0	I	The overlay rectangle is removed; the complete contents of the foreground memory will be visible again.
Return value	**Values**	**Description**	
`LONG`	0	Method was executed successfully	
	`< 0`	Error	

Available from Version 8.2.0. The status described is available from Version 8.5.2.2.

```
LONG STDisplaySetOverlayRect(LONG nLeft, LONG nTop, LONG nWidth, LONG nHeight)
```

7.59.1 Implementation

```
LONG nRc = STDisplaySetOverlayRect(0, 400, 640, 80);
if (nRc < 0)
    wprintf(L"Error %d", nRc);
```

7.60 STDisplayGetScrollPos method

This method returns the X/Y position where the content of the memory defined using the `STDisplaySetTarget()` method are displayed on the screen.

Parameter	Values	I/O	Description
`LONG* pnXPos`	`!= NULL`	O	Horizontal offset of the memory contents to the left, in pixels
`LONG* pnYPos`	`!= NULL`	O	Vertical offset of the memory contents to the top, in pixels
Return value	**Values**	**Description**	
`LONG`	`0`	Method was executed successfully	
	`< 0`	Error	

Available from Version 8.2.0.

```
LONG STDisplayGetScrollPos(LONG* pnXPos, LONG* pnYPos)
```

7.60.1 Implementation

```
LONG nXPos, nYPos;
LONG nRc = STDisplayGetScrollPos(&nXPos, &nYPos);
if (nRc < 0)
    wprintf(L"Error %d", nRc);
else
    wprintf(L"Scroll pos: %d / %d", nXPos, nYPos);
```

7.61 STDisplaySetScrollPos method

This method defines the X/Y position where the contents of the storage defined with `STDisplaySetTarget()` will be displayed. This method only works for image memories with a size larger than the display size. Please refer to the descriptions of the `STDisplayGetTargetWidth()` and `STDisplayGetTargetHeight()` methods.

Parameter	Values	I/O	Description
`LONG nXPos`	`>= 0`	I	Horizontal offset of the memory contents to the left, in pixels; while the maximum possible value is calculated from `STDisplayGetTargetWidth()` - `STDisplayGetWidth()`, it is possible to impose limits on this value by calling the `STSensorSetScrollArea()` method.
`LONG nYPos`	`>= 0`	I	Vertical offset of the memory contents to the top, in pixels; the maximum possible value is calculated from `STDisplayGetTargetHeight()` - `STDisplayGetHeight()`, it is possible to impose limits on this value by calling the `STSensorSetScrollArea()` method.

Return value	Values	Description
LONG	0	Method was executed successfully
	< 0	Error

Available from Version 8.2.0. The status described is available from Version 8.5.2.2.

```
LONG STDisplaySetScrollPos(LONG nXPos, LONG nYPos)
```

7.61.1 Implementation

```
LONG nRc = STDisplaySetScrollPos(0, 100);
if (nRc < 0)
    wprintf(L"Error %d", nRc);
```

7.62 STDisplayGetScrollSpeed method

This method returns the speed at which the screen content is scrolled when a hotspot generated by `STSensorAddScrollHotSpot()` is triggered.

Parameter	Values	I/O	Description
-	-	-	-
Return value	**Values**	**Description**	
LONG	1 – 1000	Scroll speed in lines per second	
	< 0	Error	

Available from Version 8.2.0.

```
LONG STDisplayGetScrollSpeed()
```

7.62.1 Implementation

```
LONG nScrollSpeed = STDisplayGetScrollSpeed();
```

7.63 STDisplaySetScrollSpeed method

This method returns the speed at which the screen content is scrolled when a hotspot generated by `STSensorAddScrollHotSpot()` is triggered. The default setting is 100.

Parameter	Values	I/O	Description
LONG nSpeed	1–1000	I	Scroll speed in lines / second; not all values are possible; invalid values are rounded to the next valid value
Return value	**Values**	**Description**	
LONG	1 – 1000	Actual set scroll speed	
	< 0	Error	

Available from Version 8.2.0.

```
LONG STDisplaySetScrollSpeed(LONG nSpeed)
```

7.63.1 Implementation

```
STDisplaySetScrollSpeed(100);
```

7.64 STDisplaySaveImageAsFile method

This method can be used to save the contents of an image memory as an image file on the hard dis. Any existing signature will be ignored for saving. The image has the size and resolution of the screen on the device used and, depending on the option, the size of the screen or image memory. The colour depth depends on the file type and the device used.

Parameter	Values	I/O	Description
`LPCWSTR szPath`	`!= NULL`	I	Storage location for the image file as a full path that includes the file name
`FILETYPE nFileType`	`0`	I	Use TIFF with CCITT4 compression (b/w image) or LZW compression (colour image) as the file format (recommended)
	`1`	I	Use PNG file format
	`3`	I	Use JPEG with a quality setting of 75 as the file format
`LONG nOptions`	Bitmask containing one or more hexadecimal values from the following list:		
	`0x01`	I	The whole content of the display will be stored; The hotspot areas (buttons) stay white in this mode.
	`0x02`	I	Instead of the current display content the content of the whole foreground memory without the overlay rectangle is saved
	`0x04`	I	Instead of the current display content, the content of the entire memory defined beforehand with `STDisplaySetTarget()` is saved.
Return value	**Values**	**Description**	
`LONG`	`0`	Method was executed successfully	
	`< 0`	Error	

Available from Version 8.2.0. The status described is available from Version 8.5.2.2.

```
LONG STDisplaySaveImageAsFile(LPCWSTR szPath, FILETYPE nFileType, LONG
nOptions)
```

The `FILETYPE` enumeration is defined as follows:

```
kTiff = 0,
kPng = 1,
kJpeg = 3
```

The following values defined in the header file can be used for the `nOptions` parameter:

```
#define STPAD_DIMG_HOTSPOTS          0x01
#define STPAD_DIMG_BUFFER            0x02
#define STPAD_DIMG_CURRENTTARGET     0x04
```

7.64.1 Implementation

```
LONG nRc = STDisplaySaveImageAsFile(L"./Image.png", kPng, 0);
if (nRc < 0)
    wprintf(L"Error %d", nRc);
```

7.65 STDisplaySetStandbyImageFromFile method

This method permanently stores an image in the selected device. The image is automatically displayed when a connection to the device has not yet been opened (while a connection is being established, for example). Although the colour depth is automatically adjusted to the connected LCD, it is still advisable to correctly generate the image beforehand (for example, a 1-bit monochrome image is required for the Sigma and Zeta models). If there is an existing Alpha channel, it is ignored. If the image is too small, it is centred. If the image is too large, it is cropped on the right and at the bottom.

The image is only transmitted when the memory management determines that the image is not yet stored in the device. A slide show configuration is removed by calling this method.

Parameter	Values	I/O	Description
`LPCWSTR szPath`	`!= NULL`	I	Full file path of the image; PNG and TIFF can be used as image formats
Return value	**Values**	**Description**	
`LONG`	`0`	Method was executed successfully	
	`< 0`	Error	

Available from Version 8.2.0.

```
LONG STDisplaySetStandbyImageFromFile(LPCWSTR szPath)
```

7.65.1 Implementation

```
LONG nRc = STDisplaySetStandbyImageFromFile(L"./Image.png");
if (nRc < 0)
    wprintf(L"Error %d", nRc);
```

7.66 STDisplaySetStandbyImageFromFileEx method

This method permanently stores an image in the selected device. The image is automatically displayed when a connection to the device has not yet been opened (while a connection is being established, for example). Unlike `STDisplaySetStandbyImageFromFile()`, the backlight is switched off and the image is removed from the display after a previously specified time period has expired.

The colour depth of the image is automatically adjusted to suit the connected LCD. Although, it is still advisable to correctly generate the image beforehand (for example, a 1-bit monochrome image is required for the Sigma and Zeta models). If there is an existing Alpha channel, it is ignored. If the image is too small, it is centred. If the image is too large, it is cropped on the right and at the bottom.

The image is only transmitted when the memory management determines that the image is not yet stored in the device.

A slide show configuration is removed by calling this method.

The time-controlled switching off is not supported by the models Sigma, Omega up to firmware 1.40 or Alpha. The time value is ignored in these devices.

Parameter	Values	I/O	Description
LPCWSTR szPath	!= NULL	I	Full file path of the image; PNG and TIFF can be used as image formats
LONG nDuration	0, 100 - 300000	I	Time in milliseconds that the image is displayed until the display is switched off; a maximum value of 255000 can be transmitted for the Gamma and Delta models. If the value is 0, no switching off will occur.
Return value	**Values**	**Description**	
LONG	0	Method was executed successfully	
	< 0	Error	

Available from Version 8.2.0.

```
LONG STDisplaySetStandbyImageFromFileEx(LPCWSTR szPath, LONG nDuration)
```

7.66.1 Implementation

```
LONG nRc = STDisplaySetStandbyImageFromFileEx(L"./Image.png", 5000);
if (nRc < 0)
    wprintf(L"Error %d", nRc);
```

7.67 STDisplayConfigSlideShow method

With this method, a slide show of permanently stored images can be configured to be played automatically on the target device, if the device is not in use. A possibly saved standby image is removed.

This method only works with the Omega, Gamma, Delta and Alpha models. It is necessary to use the `STDisplayConfigSlideShowEx()` method in order to fully exploit the functionality of the Gamma and Delta models.

Parameter	Values	I/O	Description
LPCWSTR szSlideList	NULL, ""	I	The slide show will be disabled
	other	I	A list of up to 16 (Gamma up to firmware 1.9 and Delta up to firmware 1.7) or 32 (Omega, Gamma from firmware 1.10, Delta from firmware 1.8 and Alpha) IDs of image stores separated by semicolons; these IDs must be reserved previously by the `STDisplaySetTarget()` method and must be filled with text or images; an ID of an image store can be included multiple times; the ID 35 cannot be used for a slide show; the slide show will be displayed in the given order.
LONG nDuration	100 - 300000	I	Time in milliseconds that each image is displayed; a maximum value of 255000 can be passed for the Gamma and Delta models.
Return value	**Values**	**Description**	
LONG	>= 0	Number of images in the slide show	
	< 0	Error	

Available from Version 8.2.0. The status described is available from Version 8.5.2.2.

```
LONG STDisplayConfigSlideShow(LPCWSTR szSlideList, LONG nDuration)
```

7.67.1 Implementation

```
LONG nRc = STDisplayConfigSlideShow(L"5;6;8;5;7", 2000);
if (nRc < 0)
    wprintf(L"Error %d", nRc);
```

7.68 STDisplayConfigSlideShowEx method

With this method, a slide show of permanently stored images can be configured to be played automatically on the target device, if the device is not in use. A possibly saved standby image is removed.

This method only offers the functionality of the `STDisplayConfigSlideShow()` method with the Omega (up to firmware 1.40) and Alpha models.

Parameter	Values	I/O	Description
`LPCWSTR szSlideList`	`NULL, ""`	I	The slide show will be disabled
	other	I	A list of up to 16 (Gamma up to firmware 1.9 and Delta up to firmware 1.7) or 32 (Omega, Gamma from firmware 1.10, Delta from firmware 1.8 and Alpha) IDs of image stores separated by semicolons; these IDs must have been previously reserved through the `STDisplaySetTarget()` method and must have been filled with text or images; an ID of an image store can be included multiple times; the ID 35 cannot be used for a slide show; the slide show will be displayed in the given order; a negative number can also be included when using the Omega (from firmware 2.0), Gamma and Delta models, the image from the Store ID that corresponds to the absolute value of this number will then only be displayed during the first run of the slide show; a value of 0 can also be included with the Omega (from firmware 2.0), Gamma and Delta models, the slide show will then end at this point and the backlight will be disabled.

LPCWSTR szDurationList	100 - 300000	I	A list of a maximum of one (Omega up to firmware 1.40 and Alpha) or 16 (Omega from firmware 2.0, Gamma up to firmware 1.9 and Delta up to firmware 1.7) or 32 (Gamma from firmware 1.10 and Delta from firmware 1.8) times in milliseconds separated by semicolon for which each individual image is to be displayed; if this list contains fewer values than the list of Store IDs, the last value in the list is used for all further images; permitted values are all those from "100" to "255000" for the Gamma and Delta models and "300000" for all other models.
Return value	**Values**	**Description**	
LONG	>= 0	Number of images in the slide show	
	< 0	Error	

Available from Version 8.5.2.2.

```
LONG STDisplayConfigSlideShowEx(LPCWSTR szSlideList, LPCWSTR szDurationList)
```

7.68.1 Implementation

```
LONG nRc = STDisplayConfigSlideShowEx(L"-5;6;8;6;7;0",
                                      L"5000;1000;2000;1000;2000");
if (nRc < 0)
    wprintf(L"Error %d", nRc);
```

7.69 STDisplayGetStandbyId method

This method returns the number of images configured for standby operation, as well as a hexadecimal character string that identifies the standby image currently set or the slide show currently configured. Thus it can be checked, for example, whether the current configuration matches the desired one.

Parameter	Values	I/O	Description
LPCWSTR szId	NULL	I	The method returns the length of the character string in the pnStringLength parameter
	!= NULL	I/O	Array in which the character string that identifies the current configuration is written; if the array is too small, the end characters are cut off
LONG* pnStringLength	>= 0	I/O	Length of the character string incl. terminated 0 or size of the szId array in bytes
Return value	**Values**	**Description**	
LONG	>= 0	Number of reserved permanent stores used for the standby image or the slide show	
	< 0	Error	

Available from Version 8.2.0. The status described is available from Version 8.5.2.2.

```
LONG STDisplayGetStandbyId(LPCWSTR szId, LONG* pnStringLength)
```

7.69.1 Implementation

```
LONG nLen = 0;
LONG nRc = STDisplayGetStandbyId(NULL, &nLen);
if (nRc == 0)
    wprintf(L"No standby mode configured!");
else if (nRc > 0)
{
    WCHAR* szId = new WCHAR[nLen / sizeof(WCHAR)];
    nRc = STDisplayGetStandbyId(szId, &nLen);
    if (nRc > 0)
        wprintf(L"%d stores configured, ID is: %s", nRc, szId);
    delete [] szId;
}
if (nRc < 0)
    wprintf(L"Error %d", nRc);
```

7.70 STDisplaySetBacklight method

This method controls the display backlight. The backlight is always set to the default value when the device is switched on. A default behaviour for opening and closing can be additionally defined in the STPad.ini file (see there for details).

With the Sigma model this method only works from firmware 1.10. With the Omega model, it only works from firmware 1.7. In Omega models with a firmware version that is older than 1.12, the values 1, 2 and 3 all set the default brightness.

Parameter	Values	I/O	Description
BACKLIGHT nMode	0	I	The backlight is switched off
	1	I	The backlight is set to the default value
	2	I	The backlight is set to medium brightness
	3	I	The backlight is set to maximum brightness
Return value	**Values**	**Description**	
LONG	0	Method was executed successfully	
	< 0	Error	

Available from Version 8.2.0.

```
LONG STDisplaySetBacklight(BACKLIGHT nMode)
```

The BACKLIGHT enumeration is defined as follows:

```
kOff = 0,
kOn = 1,
kMedium = 2,
kMaximum = 3
```

7.70.1 Implementation

```
LONG nRc = STDisplaySetBacklight(kOff);
if (nRc < 0)
    wprintf(L"Error %d", nRc);
```

7.71 STControlSetLogDirectory method

This method allows logging to be controlled in any folder regardless of the settings in the STPad.ini file or in the registry. The component must have write permissions in this folder. Accessing this method always closes the current log file. If a valid path is transferred, information is immediately written to an existing or new log file in the specified folder.

Parameter	Values	I/O	Description
LPCWSTR szPath	NULL	I	Logging is disabled.
	other	I	Absolute path or environment variable to an existing folder where the logging is to take place.
LONG nOptions	Bitmask containing one or more hexadecimal values from the following list:		
	0x01	I	Activate standard logging
	0x02	I	Activate communication logging; should only be activated on request from signotec, as it has a negative effect on performance
	0x04	I	Activate logging for Virtual Channel (in client)
Return value	**Values**	**Description**	
LONG	0	Method was executed successfully	
	< 0	Error	

Available from Version 8.5.2.2.

```
LONG STControlSetLogDirectory(LPCWSTR szPath, LONG nOptions=STPAD_LOG_STPADLIB)
```

The following values defined in the header file can be used for the `nOptions` parameter:

```
#define STPAD_LOG_STPADLIB    0x01
#define STPAD_LOG_STPAD       0x02
#define STPAD_LOG_STVCPAD     0x04
```

7.71.1 Implementation

```
LONG nRc = STControlSetLogDirectory(L"/home/user/documents");
if (nRc < 0)
    wprintf(L"Error %d", nRc);
```

7.72 STControlSetAppName method

This method can be used to specify the name of the application that uses the component. Users can use this name to exclusively assign one or more image memories. Please refer to section 'Exclusive use of non-volatile memory' for details.

Parameter	Values	I/O	Description
LPCWSTR szName	NULL	I	Application does not use any memories exclusively
	!= NULL	I	Name of the application (may contain spaces)
Return value	**Values**	**Description**	
LONG	0	Method was executed successfully	
	< 0	Error	

Available from Version 8.2.0. The status described is available from Version 8.5.2.2.

```
VOID STControlSetAppName(LPCWSTR szName)
```

7.72.1 Implementation

```
STControlSetAppName(L"My Great App");
```

7.73 STControlGetVersion method

This method returns the version number of the component.

Parameter	Values	I/O	Description
WCHAR szVersion[16]	!= NULL	O	Buffer where the version number is written
Return value	**Values**	**Description**	
LONG	0	Method was executed successfully	
	< 0	Error	

Available from Version 8.2.0.

```
LONG STControlGetVersion(WCHAR szVersion[16])
```

7.73.1 Implementation

```
WCHAR szVersion[16];
LONG nRc = STControlGetVersion(szVersion);
if (nRc < 0)
    wprintf(L"Error %d", nRc);
else
    wprintf(L"Version: %s", szVersion);
```

7.74 STControlGetErrorString method

This method returns an error description in German, English, French or Italian, depending on the system language.

Parameter	Values	I/O	Description
LPCWSTR szError	NULL	I	The method returns the length of the error description in the `pnStringLength` parameter
	!= NULL	I/O	Array in which the error description is written; if the array is too small, the end characters are cut off
LONG* pnStringLength	>= 0	I/O	Length of the error description incl. terminated 0 or size of the `szError` array in bytes
LONG nErrorId	0	I	The description of the last error that occurred will be returned.
	< 0	I	Error number, for which the description should be returned.
Return value	**Values**	**Description**	
LONG	0	Method was executed successfully	
	< 0	Error	

Available from Version 8.2.0.

```
LONG STControlGetErrorString(LPCWSTR szError, LONG* pnStringLength, LONG
nErrorId)
```

7.74.1 Implementation

```
LONG nLen = 0;
LONG nRc = STControlGetErrorString(NULL, &nLen, 0);
if (nRc == 0)
{
    WCHAR* szError = new WCHAR[nLen / sizeof(WCHAR)];
    nRc = STControlGetErrorString(szError, &nLen, 0);
    if (nRc == 0)
        wprintf(szError);
    delete [] szError;
}
```

7.75 STControlSetCallback method

This method defines a callback routine that is called if one of the events is triggered. For more information, see the section 'Events'.

Parameter	Values	I/O	Description
`CBPTR pCallback`	`NULL`	I	No callback is used
	`!= NULL`	I	Pointer to the callback routine
`LPVOID pCustomPar`	all	I	Any parameter that is passed when the callback routine is called; normally a pointer to the class whose methods are called from the callback routine
Return value	**Values**	**Description**	
-	-	-	

Available from Version 8.2.0.

```
VOID STControlSetCallback(CBPTR pCallback, LPVOID pCustomPar)
```

The `CBPTR` type is defined as follows:

```
typedef VOID (*CBPTR)(LONG nEvent, LPVOID pData, LONG nDataSize, LPVOID
pCustomPar);
```

Parameter	Values	I/O	Description
`LONG nEvent`	Index of the triggered event from the following list:		
	0	I	`DeviceDisconnected()`
	1	I	`SensorHotSpotPressed()`
	2	I	`SensorTimeoutOccured()`
	3	I	`DisplayScrollPosChanged()`
	4	I	`SignatureDataReceived()`
`LPVOID pData`	`!= NULL`	I	Array of data that is given as a parameter to the event; please refer to the respective event for a description of the parameters
`LONG nDataSize`	`> 0`	I	Size of the `pData` array in bytes
`LPVOID pCustomPar`	all	I	Parameter that was passed when calling `STControlSetCallback()`; normally a pointer to the class whose methods are called from the callback routine

Return value	Values	Description
-	-	-

The following values defined in the header file can be used for the `nEvent` parameter:

```
#define STPAD_CALLBACK_DISCONNECT   0
#define STPAD_CALLBACK_HOTSPOT      1
#define STPAD_CALLBACK_TIMEOUT      2
#define STPAD_CALLBACK_SCROLL       3
#define STPAD_CALLBACK_SIGNATURE    4
```

7.75.1 Implementation

```
VOID Callback(LONG nEvent, LPVOID pData, LONG nDataSize, LPVOID
              pCustomPar)
{
    if (!pCustomPar)
        return;

    CMyClass* pCls = (CMyClass*)pCustomPar;
    switch (nEvent)
    {
        case STPAD_CALLBACK_DISCONNECT:
            if (nDataSize >= sizeof(LONG))
                pCls->DeviceDisconnected(*(LONG*)pData);
            break;
        case STPAD_CALLBACK_HOTSPOT:
            if (nDataSize >= sizeof(LONG))
                pCls->SensorHotSpotPressed(*(LONG*)pData);
            break;
        case STPAD_CALLBACK_TIMEOUT:
            if (nDataSize >= sizeof(LONG))
                pCls->SensorTimeoutOccured(*(LONG*)pData);
            break;
        case STPAD_CALLBACK_SCROLL:
            if (nDataSize >= (2 * sizeof(LONG)))
                pCls->DisplayScrollPosChanged(*(LONG*)pData,
                                              *((LONG*)pData + 1));
            break;
        case STPAD_CALLBACK_SIGNATURE:
            if (nDataSize >= (4 * sizeof(LONG)))
                pCls->SignatureDataReceived(*(LONG*)pData,
                                            *((LONG*)pData + 1),
                                            *((LONG*)pData + 2),
                                            *((LONG*)pData + 3));
            break;
      }
}

CMyClass::CMyClass()
{
    STControlSetCallback(&Callback, (VOID*)this);
}
```

7.76 STControlExit method

This method releases used resources; it must be called before the component is de-initialised.

Parameter	Values	I/O	Description
–	–	-	-
Return value	**Values**	**Description**	
–	–	-	

Available from Version 8.2.0.

```
VOID STControlExit()
```

7.76.1 Implementation

```
STControlExit();
```

8 Events

Events are named according to the following naming convention:

- General hardware events begin with '**Device**'
- Events that apply to the signature begin with '**Signature**'
- Sensor events begin with '**Sensor**'
- Display events begin with '**Display**'

The component uses a callback mechanism to pass events through to the application. For more information, see the `STControlSetCallback()` method.

8.1 DeviceDisconnected event

This event is called as soon as a device is disconnected through an external event (e. g. unplugging the device).

Parameter	Values	Description
`LONG nIndex`	`>= 0`	Index of the disconnected device
Return value	**Values**	**Description**
-	-	-

Available from Version 8.2.0.

```
VOID DeviceDisconnected(LONG nIndex)
```

8.1.1 Implementation

```
VOID CMyClass::DeviceDisconnected(LONG nIndex)
{
    wprintf(L"Device %d disconnected!", nIndex);
}
```

8.2 SignatureDataReceived event

This event is called when signature data is received from the pad.

Parameter	Values	Description
`LONG nXPos`	`>= 0`	x value of the received data record
`LONG nYPos`	`>= 0`	y value of the received data record
`LONG nPressure`	`>= 0`	Pressure value of the received data record
`LONG nTimestamp`	`>= 0`	Timestamp of the received data record
Return value	**Values**	**Description**
-	-	-

Available from Version 8.2.0.

```
VOID SignatureDataReceived(LONG nXPos, LONG nYPos, LONG nPressure, LONG
nTimestamp)
```

8.2.1 Implementation

```
void CMyClass::SignatureDataReceived(long nXPos, long nYPos, long
nPressure, long nTimestamp)
{
    WCHAR szText[64];
    swprintf_s(szText, 64, L"X: %d; Y: %d; P: %d; T: %d", nXPos, nYPos,
               nPressure, nTimestamp);
    AfxMessageBox(szText);
}
```

8.3 SensorHotSpotPressed event

This event is called as soon as the user lifts the pen off a rectangle defined with `STSensorAddHotSpot()`, `STSensorAddScrollHotSpot()` or `STSensorAddKeyPadHotSpot()` after placing it in this rectangle.

Parameter	Values	Description
`LONG nHotSpotId`	`>= 0`	ID of the operated hotspot
	`-1`	An area set with `STSensorAddKeypadHotSpot()` was clicked once and added to the list stored in the signature device.
	`-2`	An area set with `STSensorAddKeypadHotSpot()` has been clicked at least once but could not be added to the list stored in the signature device because it is full
Return value	**Values**	**Description**
-	-	-

Available from Version 8.2.0. The status described is available from Version 8.5.2.2.

```
VOID SensorHotSpotPressed(LONG nHotSpotId)
```

8.3.1 Implementation

```
VOID CMyClass::SensorHotSpotPressed(LONG nHotSpotId)
{
    if (e.nHotSpotId >= 0)
        wprintf(L"Hotspot %d!", nHotSpotId);
    else
        // process keypad hotspot...
}
```

8.4 SensorTimeoutOccured Event

This event is called as soon as the timer started with `STSensorStartTimer()` has expired.

Parameter	Values	Description
`LONG nPointsCount`	`>= 0`	Number of points captured if any
Return value	**Values**	**Description**
-	-	-

Available from Version 8.2.0.

```
VOID SensorTimeoutOccured(LONG nPointsCount)
```

8.4.1 Implementation

```
VOID CMyClass::SensorTimeoutOccured(LONG nPointsCount)
{
    wprintf(L"Timeout, captured points: %d!", nPointsCount);
}
```

8.5 DisplayScrollPosChanged event

This event is called as soon as the scroll position of the display contents has changed.

Parameter	Values	Description
LONG nXPos	>= 0	Horizontal offset of the display contents to the left, in pixels
LONG nYPos	>= 0	Vertical offset of the display contents to the top, in pixels
Return value	**Values**	**Description**
-	-	-

Available from Version 8.2.0.

```
VOID DisplayScrollPosChanged(LONG nXPos, LONG nYPos)
```

8.5.1 Implementation

```
VOID CMyClass::DisplayScrollPosChanged(LONG nXPos, LONG nYPos)
{
    wprintf(L"Scroll pos: %d / %d", nXPos, nYPos);
}
```