

# **SigKit**

*Point of Sale Terminal Control Library Version 4.3*

## **User's Manual**

*For use with @pos.com POS terminals including PenWare™ 100, 1100, 1500, 2000, 3000, 3100, and the iPOS TC running posClassic*

*Supports the following target operating systems:*

*Microsoft 16 bit Windows 3.x and 32 bit Windows 95/NT;*

*IBM OS/2; DOS version 3.3 or higher*



Part No. PC17401  
Revision F

No part of this manual may be reproduced or transmitted in any form or by any means, electronic or mechanical, for any purpose, without the express written permission of @pos.com. @pos.com and its suppliers provides this product "as is" and make no representation or warranty regarding the content of this product and its documentation. All information in the product and documentation is subject to change without notice. @pos.com disclaims all warranties, either express or implied, including the warranties of merchantability and fitness for a particular purpose. In no event shall @pos.com or its suppliers be liable for any damages whatsoever including direct, indirect, incidental, consequential, loss of business profits or special damages.

Copyright © 1994-99 @pos.com. All rights reserved.

@pos.com and the @pos.com logo are registered trademarks and PenWare100, PenWare1100, PenWare1500, PenWare2000, PenWare3000, PenWare3100, iPOS TC, posClassic, posPortal, posBrowser, PadCom, SigKit and SigBox are trademarks of @pos.com.

Microsoft is a registered trademark and Windows and Visual C/C++, Visual Basic are trademarks of Microsoft Corporation.

Borland is a registered trademark and Borland C/C++ is a trademark of Borland International, Inc.

IBM, OS/2 and Visual Age C++ are trademarks or registered trademarks of International Business Machines Corporation.

# Table of contents

<b>INTRODUCTION.....</b>	<b>7</b>
<i>Introducing @pos.com SigKit library.....</i>	<i>7</i>
<i>Features of SigKit library.....</i>	<i>7</i>
<i>SigKit Interface.....</i>	<i>7</i>
<b>INSTALLATION.....</b>	<b>9</b>
<b>USING THE LIBRARY .....</b>	<b>11</b>
<i>Using SigKit library.....</i>	<i>11</i>
<i>Definition of a signature object.....</i>	<i>11</i>
<i>Creating and deleting a signature object.....</i>	<i>11</i>
<i>Adding pen strokes to the signature.....</i>	<i>12</i>
<i>Adding points obtained from the PadCom library2.....</i>	<i>12</i>
<i>Saving and loading signature objects.....</i>	<i>13</i>
<i>Implementation.....</i>	<i>13</i>
<i>Compiling and linking your applications.....</i>	<i>13</i>
<b>FUNCTIONS GROUPED BY CATEGORY .....</b>	<b>15</b>
<i>Signature object operations.....</i>	<i>15</i>
<i>Signature point operations.....</i>	<i>15</i>
<i>Page/frame area information.....</i>	<i>15</i>
<i>Image operations.....</i>	<i>15</i>
<i>Scaling functions.....</i>	<i>15</i>
<i>Signature file I/O.....</i>	<i>15</i>
<i>Signature Memory Object Commands.....</i>	<i>15</i>
<i>Signature format conversion.....</i>	<i>15</i>
<i>Signature object information.....</i>	<i>15</i>
<i>Point indexing information.....</i>	<i>16</i>
<i>Drawing functions.....</i>	<i>16</i>
<i>Pen attribute functions.....</i>	<i>16</i>
<i>Clipboard functions.....</i>	<i>16</i>
<b>ALPHABETIC LIST OF FUNCTIONS.....</b>	<b>17</b>
<i>sigAdd.....</i>	<i>17</i>
<i>sigAlignBottom.....</i>	<i>17</i>
<i>sigAlignLeft.....</i>	<i>18</i>
<i>sigAlignRight.....</i>	<i>18</i>
<i>sigAlignTop.....</i>	<i>19</i>
<i>sigAssert.....</i>	<i>20</i>
<i>sigCanModify.....</i>	<i>20</i>
<i>sigCat.....</i>	<i>21</i>
<i>sigCenter.....</i>	<i>21</i>
<i>sigCenterHorz.....</i>	<i>22</i>
<i>sigCenterVert.....</i>	<i>22</i>
<i>sigClean.....</i>	<i>23</i>
<i>sigCopy.....</i>	<i>23</i>
<i>sigCopyClip.....</i>	<i>24</i>
<i>sigCreateMetaFile.....</i>	<i>24</i>
<i>sigCutClip.....</i>	<i>25</i>
<i>sigCvtDPI.....</i>	<i>25</i>

<i>sigCvtScale</i> .....	26
<i>sigDelete</i> .....	26
<i>sigDestroyMetaFile</i> .....	27
<i>sigDisplay</i> .....	27
<i>sigDisplayDot</i> .....	27
<i>sigDisplayDots</i> .....	28
<i>sigDisplayLast</i> .....	28
<i>sigDisplayLine</i> .....	29
<i>sigDisplayStroke</i> .....	29
<i>sigDraw</i> .....	30
<i>sigDrawDot</i> .....	30
<i>sigDrawDots</i> .....	31
<i>sigDrawLast</i> .....	31
<i>sigDrawLine</i> .....	32
<i>sigDrawStroke</i> .....	32
<i>sigDup</i> .....	33
<i>sigEmpty</i> .....	33
<i>sigFirstLine</i> .....	34
<i>sigFirstPoint</i> .....	34
<i>sigFirstStroke</i> .....	34
<i>sigFitHeight</i> .....	35
<i>sigFitSize</i> .....	35
<i>sigFitWidth</i> .....	35
<i>sigFlipHorz</i> .....	36
<i>sigFlipVert</i> .....	36
<i>sigGet</i> .....	37
<i>sigGetBkColor</i> .....	37
<i>sigGetBounds</i> .....	38
<i>sigGetColor</i> .....	39
<i>sigGetPage</i> .....	39
<i>sigGetPen</i> .....	39
<i>sigGetPoint</i> .....	40
<i>sigGetScale</i> .....	40
<i>sigHeight</i> .....	41
<i>sigHorzDPI</i> .....	41
<i>sigIsEmpty</i> .....	41
<i>sigIsEnhanced</i> .....	42
<i>sigIsReduced</i> .....	42
<i>sigIsScaled</i> .....	42
<i>sigIsVoid</i> .....	43
<i>sigLastLine</i> .....	43
<i>sigLastPoint</i> .....	43
<i>sigLastStroke</i> .....	44
<i>sigLoad</i> .....	44
<i>sigMemError</i> .....	44
<i>sigNew</i> .....	45
<i>sigNextLine</i> .....	45
<i>sigNextPoint</i> .....	46
<i>sigNextStroke</i> .....	46
<i>sigNumPoints</i> .....	46
<i>sigNumLines</i> .....	47
<i>sigNumStrokes</i> .....	47
<i>sigOffset</i> .....	47
<i>sigPrevLine</i> .....	48

<i>sigPrevPoint</i> .....	48
<i>sigPrevStroke</i> .....	49
<i>sigRead</i> .....	49
<i>sigReadMem</i> .....	49
<i>sigReadCmpMem</i> .....	50
<i>sigReadNlcMem</i> .....	50
<i>sigReadSigMem</i> .....	51
<i>sigReadNLC</i> .....	51
<i>sigReadWIN</i> .....	52
<i>sigRemove</i> .....	52
<i>sigRender</i> .....	52
<i>sigRenderDot</i> .....	53
<i>sigRenderDots</i> .....	53
<i>sigRenderLast</i> .....	54
<i>sigRenderLine</i> .....	55
<i>sigRenderStroke</i> .....	55
<i>sigResetScale</i> .....	56
<i>sigSave</i> .....	56
<i>sigSaveAs</i> .....	56
<i>sigSaveAsBMP</i> .....	57
<i>sigSaveAsCGM</i> .....	58
<i>sigSaveAsDIB</i> .....	58
<i>sigSaveAsEPS</i> .....	59
<i>sigSaveAsMF</i> .....	59
<i>sigSaveAsNLC</i> .....	60
<i>sigSaveAsPCL</i> .....	60
<i>sigSaveAsPCX</i> .....	61
<i>sigSaveAsPLS</i> .....	61
<i>sigSaveAsTIF</i> .....	62
<i>sigSaveAsTXT</i> .....	62
<i>sigSaveAsWMF</i> .....	63
<i>sigSaveCmp</i> .....	63
<i>sigScale</i> .....	64
<i>sigScaleFrac</i> .....	64
<i>sigScaleToDPI</i> .....	64
<i>sigSetBkColor</i> .....	65
<i>sigSetBMPTyp</i> .....	65
<i>sigSetColor</i> .....	66
<i>sigSetEnhanced</i> .....	66
<i>sigSetPage</i> .....	66
<i>sigSetPen</i> .....	67
<i>sigSetReduced</i> .....	67
<i>sigSetScale</i> .....	68
<i>sigShrinkWrap</i> .....	68
<i>sigSmooth</i> .....	69
<i>sigSmoothingFilter</i> .....	69
<i>sigStretchSize</i> .....	70
<i>sigThinBy</i> .....	70
<i>sigThinTo</i> .....	70
<i>sigToHIENGLISH</i> .....	71
<i>sigToHIMETRIC</i> .....	71
<i>sigToLOENGLISH</i> .....	72
<i>sigToLOMETRIC</i> .....	72
<i>sigUndoPoint</i> .....	73

<i>sigUndoStroke</i> .....	73
<i>sigVertDPI</i> .....	73
<i>sigVoid</i> .....	74
<i>sigWidth</i> .....	74
<i>sigWrite</i> .....	74
<i>sigWriteCmp</i> .....	75
<i>sigWriteCmpMem</i> .....	75
<i>sigWriteCmpWIN</i> .....	76
<i>sigWriteNlcMem</i> .....	76
<i>sigWriteWIN</i> .....	77
<i>sigWriteMem</i> .....	77
<b>APPENDIX A</b> .....	<b>79</b>
SigKit Sample for DOS: .....	79
SigKit Sample for Win3.x: .....	81
SigKit Sample for Win95/NT: .....	84

# Introduction

## Introducing @pos.com SigKit library

SigKit library by @pos.com is a tool that allows developers to process collected Point Of Sale (POS) data (mainly signatures) easily and quickly. Developers can use SigKit library to shrink-wrap collected signatures, thin them, smoothen them and perform other image processing operations on them. By leveraging on @pos.com's position as a leader in signature capture technology, you can rest assured that your application will provide accurate and quality results with minimal programming efforts.

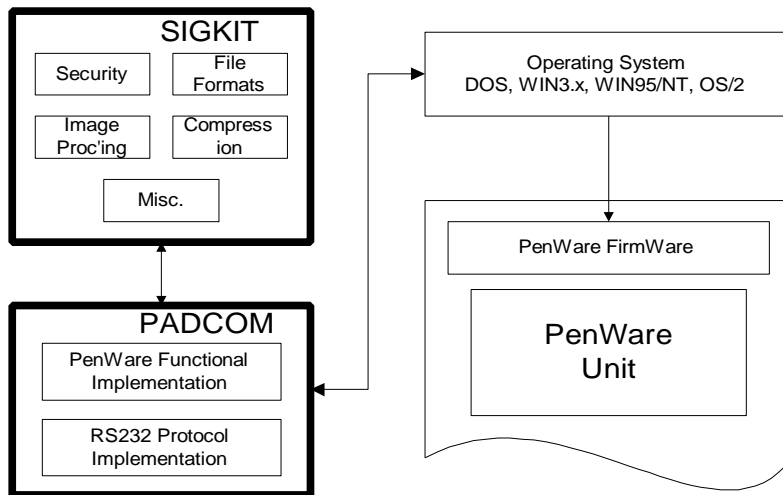
## Features of SigKit library

@pos.com SigKit library includes the following key features:

- Full interface to process data collected from all @pos.com units.
- Flexible architecture to support a wide variety of development needs.
- Simple, easy to use command set for rapid applications development.
- Consistent cross-platform APIs for enhanced portability.
- Accurate representation of image points.
- Reliable conversion to other standard image formats including Windows Bitmap and Metafiles.

In addition, SigKit library has been designed to work seamlessly with @pos.com PadCom<sup>1</sup>, the pad communication library for a complete signature capture solution.

## SigKit Interface



The diagram shows the hierarchy of SigKit library. SigKit is designed to provide a seamless interface with PadCom Library which is an Operating System dependent RS232 protocol implementation. Although, it is possible to use other RS232 implementations, @pos.com recommends that the

---

<sup>1</sup> For more information about the PadCom Library and other development tools, please contact your @pos.com sales representative.

programmers use the built-in driver. SigKit provides a varied functionality ranging from Signature Processing capabilities, file format conversions, and compression. SigKit libraries are also Operating System dependent and are compiled using industry leading compilers.

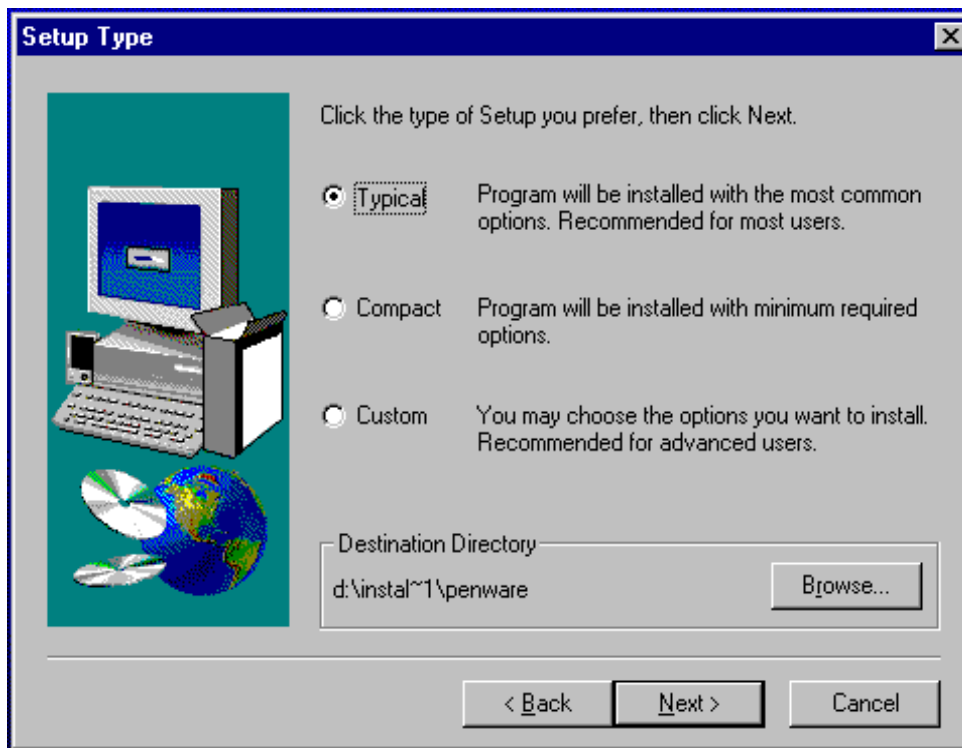
## Installation

@pos.com Library Installation Program is a simple to use InstallShield® application. It allows you to selectively install the desired components and the sample programs. Please note that the distribution diskettes allow you to install all of PadCom, SigKit and the Custom VBX control<sup>2</sup>.

Please follow the instructions given below:

1. Insert the diskette labeled “Library Installation Disk 1 of 2” in your 3 ½” disk drive.
2. From your Windows Manager, select the 3 ½” disk drive.
3. Find and execute setup.exe (You can execute the program by double-clicking on its icon).

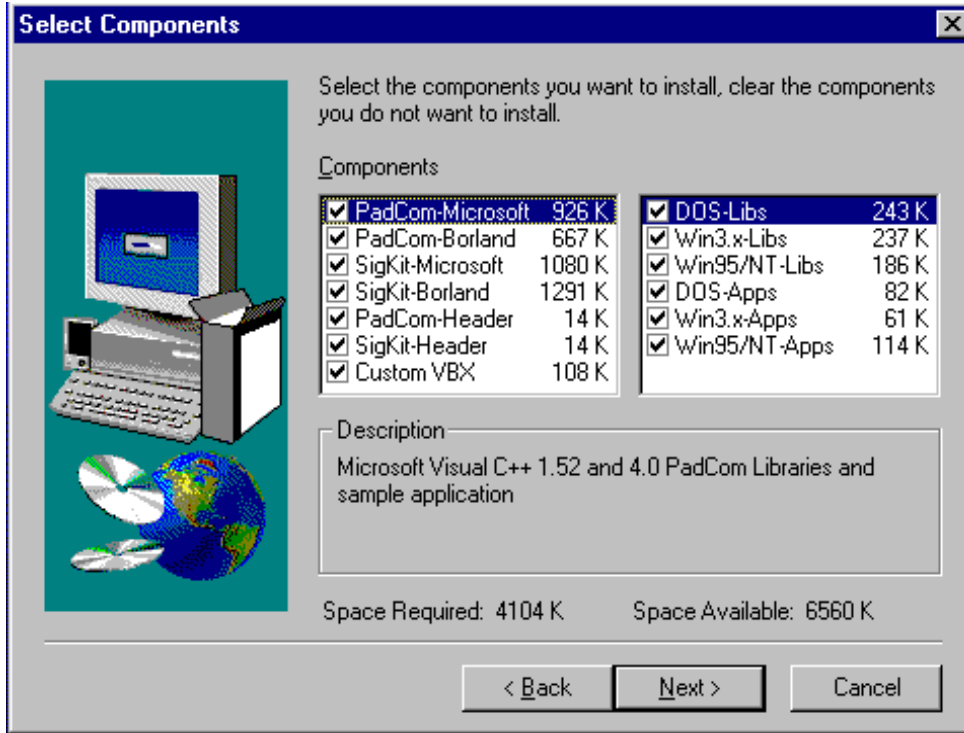
A series of simple screens are presented to the user with some relevant information and some selection options for customization. Please read the instructions on each of these screens carefully. The screen-shot shown below allows the user to customize the installation by selecting only the desired set of components.



Selecting the “Typical” option installs all of PadCom, SigKit and the custom VBX control. Selecting the “Compact” option installs all but the custom VBX control. Choosing the “Custom” option allows the user to selectively install components as desired. @pos.com recommends choosing the “Typical” option for beginners. The following screen shot shows the screen associated with the “Custom” option.

---

<sup>2</sup> For more information on PadCom and the custom VBX control, please contact your @pos.com sales representative.



The “Custom” option breaks down the installation into components. As shown, these components are grouped based on the type of compiler used. Recall that @pos.com supports Microsoft’s Visual C++ 1.52, Visual C++ 4.0 and Borland C++ 4.2 compilers.

The user has an option to select only the desired components. Each component (PadCom-Microsoft shown in the screen shot example), has an option to install the libraries and/or the samples applications. The library sub-components are post-fixed with “Lib” or “Libs” whereas the Application sub-components are post-fixed with “App” or “Apps.”

The rest of the installation is quite straightforward. Please follow the remaining instructions to complete the installation process. Please view the “Readme.txt” file supplied with the installation for details on installation. A brief description of some of the topics covered in the “Readme.txt” file is presented later on for convenience.

# Using the library

## Using SigKit library

SigKit library has many functions available to handle a wide variety of user needs, yet is simple enough to be used right away. By following the guidelines set forth in this chapter, you will be able to add robust signature processing to your applications quickly and easily.

## Definition of a signature object

The term *signature object* is used throughout this manual to refer to a signature as created and recognized by the SigKit library. Using signature objects is similar to the way a C programmer traditionally uses dynamic memory or standard file systems. A handle is used to reference an internal representation of the signature details. Member functions of the SigKit library use this handle to provide access to the signature data. The programmer uses the library to create a signature object, perform one or more operations on it, and finally, to delete it when it is no longer needed.

## Creating and deleting a signature object

Signature objects are references as **SIG** handles and are created and deleted by the functions **sigNew** and **sigDelete**, respectively. A signature is created with a “page” specification which provides the size and resolution details needed to maintain proper scaling of the original signature image. When used with @pos.com’s PadCom library, these specifications are most often set to match the pad characteristics at the time of signature entry. The following code illustrates this approach:

```
// Create a signature object using specifications obtained from PadCom functions
SIG InputSig = sigNew( padWidth(), padHeight(), padHorzDPI(), padVertDPI() );
if( InputSig )
{
    // Use the signature object ...
    // ...
    // Delete the signature object and free up memory
    sigDelete( InputSig );
}
```

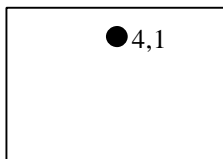
Alternatively, you may wish to specify constant sizes for the signature object and convert any pad points to match this specification. This method is not recommended because it reduces the integrity of the signature by forcing it to be scaled upon entry. This library is designed to provide an accurate representation of the original signature without the need for permanent image alterations or scaling. However, for purposes of illustration, this method can be helpful and may be used in some of the examples. The following line of code create a signature object using constant values:

```
// Create a low resolution signature object for purposes of illustration
// The signature will be contained in a 200 by 50 pixel area
// The signature will use a resolution of 90 dot-per-inch
SIG Sample = sigNew( 200, 50, 90, 90 );
```

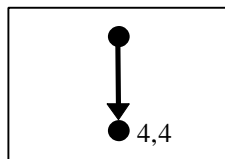
Note that you must always use **sigDelete** to delete signature objects when they are no longer needed. Otherwise memory leaks and other problems may occur. Of course, if you want to keep a signature permanently, you should save the signature object to a file in one of the available file formats.

## Adding pen strokes to the signature

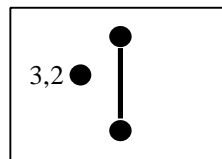
Pen strokes are generally added to a signature by using the **sigAdd** function. A pen stroke consists of the points received from the time the pen comes into contact with the pad through the time it is removed. A typical stroke contains many points. The first point, referred to as a “moveto” point, indicates where the pen was first placed onto the writing surface. Remaining points, referred to as “lineto” points, indicate that the pen is being “dragged” across the writing surface. Consequently, the **sigAdd** function requires a “moveto/lineto” pen indicator as well as the corresponding coordinates. For example, the following code adds two pen strokes as illustrated below. It also saves the resulting image as both a text dump and a reloadable signature file.



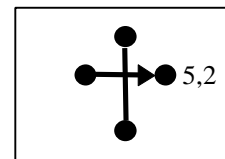
Move to 4,1  
*begin first stroke*



Line to 4,4  
*continue first stroke*



Move to 3,2  
*begin a new stroke*



Line to 5,2  
*continue new stroke*

```
// Create a low resolution signature object for purposes of illustration
SIG Sample = sigNew( 10, 10, 90, 90 );
if( Sample )
{
    // Add points to the signature
    sigAdd( Sample, 4,1,0 ); // Move to 4,1 - begin first stroke
    sigAdd( Sample, 4,4,1 ); // Line to 4,4 - continue first stroke
    sigAdd( Sample, 3,2,0 ); // Move to 3,2 - begin a new stroke
    sigAdd( Sample, 5,2,1 ); // Line to 5,2 - continue new stroke

    // Use the signature object ...
    // Lets save the result as both a text dump and a reloadable sig file!
    sigSaveAsTXT( Sample, "SAMPLE.TXT" );
    sigSave( Sample, "SAMPLE.SIG" );

    // Delete the signature object and free up memory
    sigDelete( Sample );
}
```

## Adding points obtained from the PadCom library2

In most cases you will be adding points to a signature object as they are obtained from a pen or pad input device. When using the @pos.com PadCom library to capture pad input, you will use the **sigAdd** function within a **padUpdate** loop. Consult the PadCom library documentation for information about using **padUpdate** and **padGet** functions. The following code receives a point from PadCom and adds it to a signature object:

```
// somewhere in the middle of a PadCom padUpdate sequence...
// SIG Sample;
// int x,y,p;
// ....
// Check if there is a new point available from the pad
if( padGet( &x, &y, &p ) )
{
    // If the signature object was not created to match the pad, scale it to
match
    // padScaleToDPI( &x, &y, sigHorzDPI( Sample ), sigVertDPI( Sample ) );
    // Add the new point to our signature object
    sigAdd( Sample, x, y, p );
}
```

}

## Saving and loading signature objects

Once a signature has been obtained, it can be saved and loaded back at a later time. Saving a signature using the **sigSave** function allows it to be read back in later using **sigLoad**. To save the signature image as an external file type, use one of the many **sigSaveAs** type functions. For example, you can save a signature as a Windows bitmap (.bmp) using **sigSaveAsBMP** or as a Windows metafile (.wmf) using **sigSaveAsWMF**. Note that external file types can not be read back in later using **sigLoad**. Therefore, we suggest using **sigSave** when saving a permanent copy of the signature, and only use external files types when required by the application. The following code loads a signature file and converts it to a Windows bitmap file. Note that when creating a signature object to be loaded from a saved file it is not necessary to provide any specifications to **sigNew**.

```
// Create a signature object with unknown size and resolution
SIG InputSig = sigNew( 0,0,0,0 );
if( InputSig )
{
    // Load a previously saved signature
    sigLoad( InputSig, "SAMPLE.SIG" );

    // Note that you can add to or manipulate the signature before saving
    // sigAdd( InputSig, 4,4,1 ); // Add more points to the signature
    // sigShrinkWrap( InputSig ); // Force the smallest possible "page" area

    // Save it as a Windows bitmap file as needed by the application
    sigSaveAsBMP( InputSig, "SAMPLE.BMP" );
}
```

## Implementation

For a complete listing of the available functions, see the Alphabetic list of functions on page 17. For various examples that use the SigKit library to capture a signature, please refer to Appendix A on page 79.

Note that the test application uses the @pos.com PadCom library to capture signature input and thus requires a pad device to be attached. The sample applications do not perform tasks of much importance; however, indicate how to integrate your signature capture applications with the PadCom communications library. These samples are provided with relevant makefiles for the compilers. It is strongly recommended that you study the compiler and linker settings and choose the correct libraries and paths.

## Compiling and linking your applications

Compiling and linking your application involves selecting the appropriate libraries and choosing the correct build environment on the compiler of your choice. As mentioned before, a complete operating system and compiler dependent solution is available for your use. The libraries are named using a standard naming convention.

### For DOS:

sigkitds.lib sigkitdm.lib and sigkitdl.lib are available to support the small, medium and the large memory models.

### For Windows 3.x:

sigkitws.lib, sigkitwm.lib and sigkitwl.lib are available to support the small, medium and the large memory models.

**For Windows 95/NT:**

sigkitw.lib and sigkitwt.lib<sup>3</sup> are available.

The user should select the supported library and add it to the project. Please do not forget to add the common header file padcom.h or else the program will not compile correctly. Please note that @pos.com libraries are not differentiated based on the type of compiler used. For example, sigkitds.lib is available for both the Borland C++ and Microsoft Visual C++ compilers. It is the responsibility of the programmer to use the correct library. The automatic installation program puts these libraries in clearly marked folders.

Please make sure that the compiler settings comply with the memory model (or the thread option wherever applicable). Failure to do so can result in unpredictable program failures. If you are using PadCom to communicate with @pos.com unit, please do not forget to link the appropriate PadCom libraries.

**Troubleshooting**

Application development always involves debugging your application code. @pos.com has put together a highly competent Technical Support Team to help you troubleshoot your applications quickly. There is however some simple troubleshooting that you can do on your end before calling the Technical Support at @pos.com.

@pos.com strongly recommends that you always run the sample test programs that you installed while installing the library components. Please call @pos.com immediately if any of these sample programs do not work on the intended Operating Systems.

If the sample test programs work but your applications do not, please re-check the compiler settings, used libraries and the sample code. Before you call @pos.com Technical Support, please note down the operating system, environment, compilers, compiler settings, used libraries and other information that you think may be useful or relevant. This will help the technical support personnel diagnose and/or make recommendations quickly.

---

<sup>3</sup> sigkitwt.lib is available for the Microsoft compiler only.

## Functions grouped by category

### Signature object operations

sigAssert	sigCopy	sigEmpty
sigCat	sigDelete	sigNew
sigClean	sigDup	

### Signature point operations

sigAdd	sigGetPoint	sigUndoPoint
sigGet	sigRemove	sigUndoStroke

### Page/frame area information

sigGetPage	sigSetPage	sigVertDPI
sigHeight	sigHorzDPI	sigWidth

### Image operations

sigAlignBottom	sigCenterVert	sigSmooth
sigAlignLeft	sigFlipHorz	sigSmoothingFilter
sigAlignRight	sigFlipVert	sigThinBy
sigAlignTop	sigGetBounds	sigThinTo
sigCenter	sigOffset	
sigCenterHorz	sigShrinkWrap	

### Scaling functions

sigCvtDPI	sigResetScale	sigToHIENGLISH
sigCvtScale	sigScale	sigToHIMETRIC
sigFitHeight	sigScaleFrac	sigToLOENGLISH
sigFitSize	sigScaleToDPI	sigToLOMETRIC
sigFitWidth	sigSetScale	
sigGetScale	sigStretchSize	

### Signature file I/O

sigLoad	sigSaveCmp	sigWriteCmpWIN
sigRead	sigWrite	sigSaveAsNLC
sigReadWIN	sigWriteWIN	sigReadNLC
sigSave	sigWriteCmp	

### Signature Memory Object Commands

sigReadMem	sigMemError	sigReadNlcMem
sigSaveMem	sigWriteNlcMem	sigReadCmpMem
sigSaveCmpMem	sigReadSigMem	

### Signature format conversion

sigSaveAs	sigSaveAsMF	sigSaveAsTXT
sigSaveAsBMP	sigSaveAsPCL	sigSaveAsWMF
sigSaveAsDIB	sigSaveAsPCX	sigSetBMptype
sigSaveAsCGM	sigSaveAsPLS	
sigSaveAsEPS	sigSaveAsTIF	

### Signature object information

sigCanModify  
 sigIsEmpty  
 sigIsEnhanced

sigIsReduced  
 sigIsScaled  
 sigIsVoid

sigSetEnhanced  
 sigSetReduced

**Point indexing information**

sigFirstLine  
 sigFirstPoint  
 sigFirstStroke  
 sigLastLine  
 sigLastPoint

sigLastStroke  
 sigNextLine  
 sigNextPoint  
 sigNextStroke  
 sigNumLines

sigNumPoints  
 sigNumStrokes  
 sigPrevLine  
 sigPrevPoint  
 sigPrevStroke

**Drawing functions**

sigDisplay  
 sigDisplayDot  
 sigDisplayDots  
 sigDisplayLast  
 sigDisplayLine  
 sigDisplayStroke

sigDraw  
 sigDrawDot  
 sigDrawDots  
 sigDrawLast  
 sigDrawLine  
 sigDrawStroke

sigRender  
 sigRenderDot  
 sigRenderDots  
 sigRenderLast  
 sigRenderLine  
 sigRenderStroke

**Pen attribute functions**

sigGetBkColor  
 sigGetColor

sigGetPen  
 sigSetBkColor

sigSetColor  
 sigSetPen

**Clipboard functions**

sigCopyClip  
 sigCreateMetaFile

sigCutClip  
 sigDestroyMetaFile

## Alphabetic list of functions

---

### sigAdd

**BOOL sigAdd( SIG aSig, int anX, int aY, int aPen )**

Adds a point to a signature object.

Parameter	Description
aSig	Signature object to receive the new point
anX	Horizontal coordinate of the new point
aY	Vertical coordinate of the new point
aPen	Pen status (0==moveto, 1==lineto)

#### Description

Adds the point defined by anX, aY and aPen to the signature object aSig. Duplicate points and duplicate “moveto”/new stroke indicators are ignored.

#### Returns

Returns TRUE if successful, FALSE otherwise.

#### Example

The following example adds points to creates a square:

```
// Create a 100x100 pixel signature object with 90 DPI resolution
SIG Sample = sigNew( 100, 100, 90, 90 );
if( Sample )
{
    // Add points to make a 50x50 "square" at offset 10,10
    sigAdd( Sample, 10,10,0 ); // Move to 10,10 - from top/left
    sigAdd( Sample, 60,10,1 ); // Line to 10,60 - to top/right
    sigAdd( Sample, 60,60,1 ); // Line to 60,60 - to bottom/right
    sigAdd( Sample, 10,60,1 ); // Line to 10,60 - to bottom/left
    sigAdd( Sample, 60,60,1 ); // Line to 10,10 - to top/right
    // Lets save the result as both Windows bitmap!
    sigSaveAsBMP( Sample, "SAMPLE.BMP" );
    // Delete the signature object and free up memory
    sigDelete( Sample );
}
```

#### See Also

sigRemove

---

### sigAlignBottom

**BOOL sigAlignBottom( SIG aSig )**

Align a signature at the bottom.

Parameter	Description
-----------	-------------

---

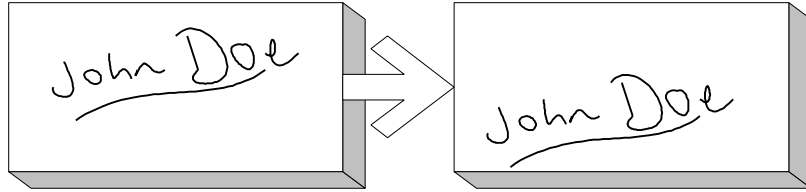
aSig                    Signature object to be aligned.

**Description**

Aligns the signature image to the bottom of the frame area.

**Returns**

Returns TRUE if successful, FALSE otherwise.

**Example**

The following example loads a signature, aligns it at the bottom, then saves the result.

```
SIG Sample = sigNew(0,0,0,0);  
sigLoad( "SAMPLE.SIG" );  
sigAlignBottom( Sample );  
sigSave( "SAMPLE.SIG" );  
sigDelete ( Sample );
```

**See Also**

sigAlignTop, sigAlignLeft, sigAlignRight, sigCenter

---

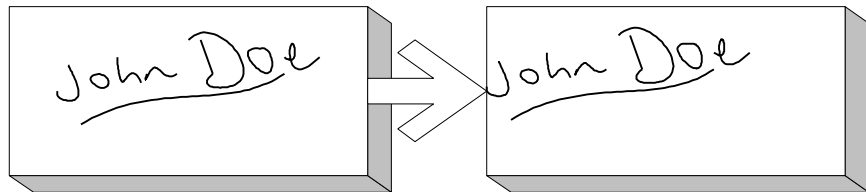
**sigAlignLeft**

**BOOL sigAlignLeft( SIG aSig )**

Align a signature to the left.

**Parameter Description**

aSig  
Signature object to be aligned.

**Description**

Aligns the signature image to the left of the frame area.

**Returns**

Returns TRUE if successful, FALSE otherwise.

**Example**

See sigAlignBottom for an example that uses a similar function.

**See Also**

sigAlignRight, sigAlignTop, sigAlignBottom, sigCenter

---

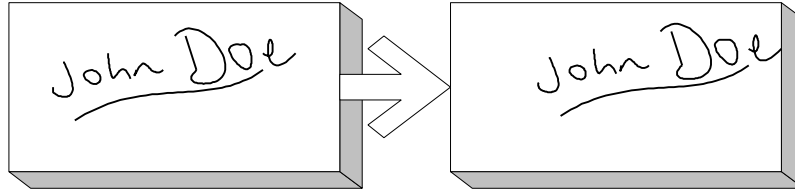
**sigAlignRight**

**BOOL sigAlignRight( SIG aSig )**

Align a signature to the right.

**Parameter**  
aSig      **Description**  
Signature object to be aligned.

**Description**  
Aligns the signature image to the right of the frame area.



**Returns**  
Returns TRUE if successful, FALSE otherwise.

**Example**  
See sigAlignBottom for an example that uses a similar function.

**See Also**  
sigAlignLeft, sigAlignTop, sigAlignBottom, sigCenter

---

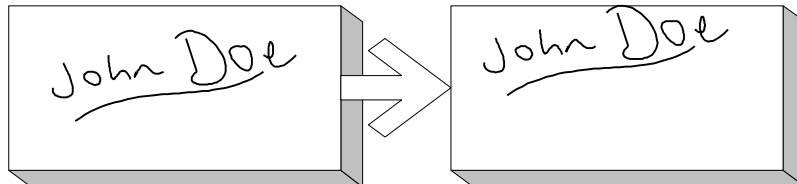
## sigAlignTop

**BOOL sigAlignTop( SIG aSig )**

Align a signature at the top.

**Parameter**  
aSig      **Description**  
Signature object to be aligned.

**Description**  
Aligns the signature image to the top of the frame area.



**Returns**  
Returns TRUE if successful, FALSE otherwise.

**Example**  
See sigAlignBottom for an example that uses a similar function.

**See Also**  
sigAlignBottom, sigAlignLeft, sigAlignRight, sigCenter

---

## sigAssert

**BOOL sigAssert( SIG aSig )**

Parameter	Description
aSig	Signature object to be checked.

### Description

Checks a signature object to see if it is valid. TRUE is returned by sigAssert only if aSig is a valid SIG. If the aSig is NULL or aSig does not appear to point to a valid SIG object then sigAssert returns FALSE.

### Returns

Returns TRUE if valid, FALSE otherwise.

### Example

The following function uses sigAssert to verify that it was given an valid SIG object.

```
BOOL SaveSig( SIG aSig )
{
    if( sigAssert( aSig ) )
    {
        // signature object is valid, save it
        sigSave( aSig, "SAMPLE.SIG" );
        return TRUE;
    }
    // fail if aSig is not valid object
    return FALSE;
}
```

---

## sigCanModify

**BOOL sigCanModify( SIG aSig )**

Checks if a signature object can be modified.

Parameter	Description
aSig	Signature object to be checked.

### Description

Checks if a signature object is in a state that can be modified.

### Returns

Returns TRUE if signature can be modified, FALSE otherwise.

### Example

The following function fails if the signature can not be modified:

```
BOOL AlignSig( SIG aSig )
{
    if( sigCanModify( aSig ) )
    {
        // signature can be modified, so do it
        sigAlignLeft( aSig );
        sigAlignBottom( aSig );
    }
}
```

```
        return TRUE;
    }
    // fail if aSig can not be modified
    return FALSE;
}
```

---

## sigCat

### **BOOL sigCat( SIG aDest, SIG aSrc )**

Concatenates two signatures.

Parameter	Description
aDest	Destination signature.
aSrc	Source signature.

### **Description**

Appends the image contained in *aSrc* onto the destination signature *aDest*. Converts point data from the source DPI resolution to the destination resolution.

### **Returns**

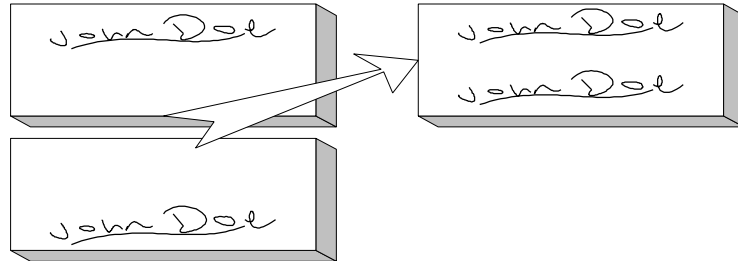
Returns TRUE if successful, FALSE otherwise.

### **Example**

The following function merges a mirrored signature image onto itself:

```
void SillyFun( SIG aSig )
{
    // create a temporary copy of the signature
    SIG tmp = sigDup( aSig );

    // flip the copy horizontally for a mirrored image
    sigFlipHorz( aSig );
    // merge the copy onto the original image
    sigCat( aSig, tmp );
    // delete the temporary
    sigDelete( tmp );
}
```



### **See Also**

sigAdd, sigRemove, sigCopy, sigDup

---

## sigCenter

### **BOOL sigCenter( SIG aSig )**

Center a signature horizontally and vertically within it's frame.

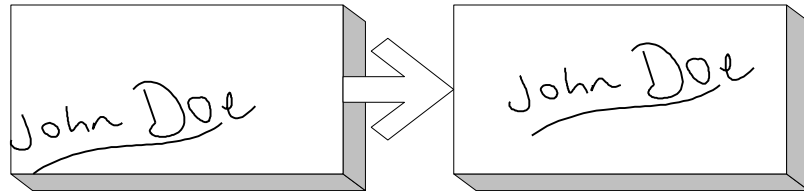
Parameter	Description
aSig	Signature to be centered.

**Description**

Positions the signature image so that it is both horizontally and vertically centered within it's frame area.

**Returns**

Returns TRUE if successful, FALSE otherwise.

**Example**

The following example loads a signature, centers it within it's frame area, then saves the result.

```
SIG Sample = sigNew(0,0,0,0);  
sigLoad( "SAMPLE.SIG" );  
sigCenter ( Sample )  
sigSave( "SAMPLE.SIG" );  
sigDelete ( Sample );
```

**See Also**

sigCenterHorz, sigCenterVert, sigAlignBottom, sigAlignLeft, sigAlignRight, sigAlignTop

---

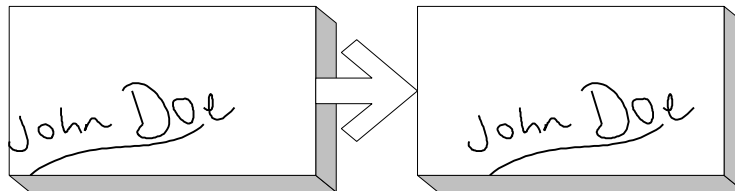
**sigCenterHorz****BOOL sigCenterHorz( SIG aSig )**

Center a signature horizontally within it's frame.

Parameter	Description
aSig	Signature to be centered.

**Description**

Positions the signature image so that it is horizontally centered within it's frame area.

**Returns**

Returns TRUE if successful, FALSE otherwise.

**Example**

See sigCenter for an example that uses a similar function.

**See Also**

sigCenter, sigCenterVert, sigAlignBottom, sigAlignLeft, sigAlignRight, sigAlignTop

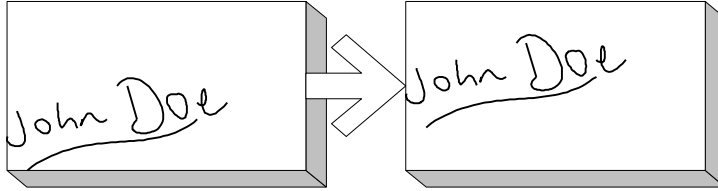
---

**sigCenterVert****BOOL sigCenterVert( SIG aSig )**

Center a signature vertically within it's frame.

<b>Parameter</b>	<b>Description</b>
aSig	Signature to be centered.

**Description**  
Positions the signature image so that it is vertically centered within it's frame area.



**Returns**  
Returns TRUE if successful, FALSE otherwise.

**Example**  
See sigCenter for an example that uses a similar function.

**See Also**  
sigCenter, sigCenterHorz, sigAlignBottom, sigAlignLeft, sigAlignRight, sigAlignTop

---

## sigClean

**unsigned sigClean( SIG aSig )**

Removes duplicate and unneeded points.

<b>Parameter</b>	<b>Description</b>
aSig	Signature to be cleaned.

**Description**  
Checks every point of the signature *aSig* and removes duplicate and unnecessary points. This function is general not needed because sigAdd automatically removes unnecessary points.

**Returns**  
Returns the number of points removed.

**See Also**  
sigAdd, sigRemove

---

## sigCopy

**BOOL sigCopy( SIG aDest, SIG aSrc )**

Copy a signature image from one signature object to another.

<b>Parameter</b>	<b>Description</b>
------------------	--------------------

aDest            Destination signature.  
aSrc             Source signature.

**Description**

Copies the signature image and frame information from *aSrc* to *aDest*. This function creates an exact copy. Note the original contents of the destination signature is lost.

**Returns**

Returns TRUE if successful, FALSE otherwise.

**Example**

The following function uses sigCopy when converting the resolution of an image before saving.

```
void DoSaveLowRes( SIG aSig, const char *aFileName )
{
    // create a new signature object
    SIG tmp = sigNew( 0,0,0,0 );
    // copy the given signature to the new sig
    sigCopy( tmp, aSig );
    // permanently change the new sig to 100 dpi resolution
    sigCvtDPI( tmp, 90,90 );
    // save the result as a Windows DIB file at 100 dpi
    sigSaveAsDIB( tmp, aFileName );
    // delete the temporary
    sigDelete( tmp );
}
```

**See Also**

sigCat, sigDup

---

**sigCopyClip**

**Win:** **BOOL sigCopyClip( HWND aWnd, SIG aSig )**

**DOS:** **Not available for DOS**

Copy a signature to the clipboard.

**Parameter      Description**

aWnd            Handle of the window that owns the signature object.  
aSig            Signature to be copied.

**Description**

Performs an “Edit/Copy” operation by copying the signature to the clipboard. Currently, only a metafile representation is copied.

**Returns**

Returns TRUE if successful, FALSE otherwise.

**See Also**

sigCutClip

---

**sigCreateMetaFile**

**Win:** HMETAFILE sigCreateMetaFile( HWND aWnd, SIG aSig )

**DOS:** Not available for DOS

Creates a metafile object.

Parameter	Description
aWnd	Handle of the window that owns the signature object.
aSig	Signature to be copied.

**Description**

Creates a metafile image that represents the signature. The handle returned must be deleted by using the function *sigDestroyMetaFile*.

**Returns**

Returns a handle the to metafile object or NULL if unsuccessful.

**See Also**

sigDestroyMetaFile

---

## sigCutClip

**Win:** BOOL sigCutClip( HWND aWnd, SIG aSig )

**DOS:** Not available for DOS

Copies a signature to the clipboard then removes all points.

Parameter	Description
aWnd	Handle of the window that owns the signature object.
aSig	Signature to be copied.

**Description**

Performs an “Edit/Cut” operation by copying the signature to the clipboard and then clearing it. Currently, only a metafile representation is copied.

**Returns**

Returns TRUE if successful, FALSE otherwise.

**See Also**

sigCopyClip

---

## sigCvtDPI

**BOOL sigCvtDPI( SIG aSIG, int aHorzDPI, int aVertDPI )**

Permanently convert dot-per-inch resolution.

Parameter	Description
aSig	Signature to be converted.
aHorzDPI	Desired horizontal dots-per-inch resolution.

---

aVertDPI          Desired vertical dots-per-inch resolution.

**Description**

Converts each point in the signature from its current dots-per-inch resolution to the desired dots-per-inch resolution. Removes any duplicate points that may result from the resolution being reduced

**Returns**

Returns TRUE if successful, FALSE otherwise.

**See Also**

sigScaleToDPI, sigCvtScale

---

**sigCvtScale**

**BOOL sigCvtScale ( SIG aSig )**

Permanently convert scale.

<b>Parameter</b>	<b>Description</b>
aSig	Signature to be converted.

**Description**

Converts each point in the signature using the current scaling factors as set by sigSetScale and similar functions. Removes any duplicate points that may result from the resolution being reduced.

**Returns**

Returns TRUE if successful, FALSE otherwise.

**See Also**

sigSetScale, sigCvtDPI

---

**sigDelete**

**void sigDelete( SIG aSig )**

Delete a signature object.

<b>Parameter</b>	<b>Description</b>
aSig	Signature to be deleted.

**Description**

Frees memory by deleting the given signature object. A signature object can not be used after it has been deleted.

**See Also**

sigNew, sigDup

---

---

## sigDestroyMetaFile

**Win:** void sigDestroyMetaFile( HMETAFILE aMetaFile )

**DOS:** Not available for DOS

Destroy a metafile object.

Parameter	Description
aMetaFile	Handle to the MetaFile object.

**Description**  
Destroys a metafile object previously created by sigCreateMetaFile.

**See Also**  
sigCreateMetaFile

---

## sigDisplay

**Win:** unsigned sigDisplay( HDC aDC, SIG aSig, int atX, int atY )

**DOS:** unsigned sigDisplay( SIG aSig, int atX, int atY )

Display a signature at its actual size.

Parameter	Description
aDC	Handle to a Windows Display Context.
aSig	Signature to be displayed.
atX	Horizontal coordinate to display image.
atY	Vertical coordinate to display image.

**Description**  
Displays a signature at its actual size at a given location. Points are temporarily converted to match the display device dots-per-inch resolution.

**Returns**  
Returns the number of line segments actually drawn.

**See Also**  
sigDraw, sigRender, sigDisplayLast, sigDisplayLine, sigDisplayPoint

---

## sigDisplayDot

**Win:** unsigned sigDisplayDot( HDC aDC, SIG aSig, unsigned anIndex , int atX, int atY )

**DOS:** unsigned sigDisplayDot( SIG aSig, unsigned anIndex, int atX, int atY )

Display a point of a signature at its actual size.

Parameter	Description
-----------	-------------

aDC	Handle to a Windows Display Context.
aSig	Signature to be displayed.
anIndex	Index of the point to be drawn.
atX	Horizontal coordinate to display image.
atY	Vertical coordinate to display image.

**Description**

Displays a single point of a signature as a dot at its actual size at a given location. The points is temporarily converted to match the display device dots-per-inch resolution.

**Returns**

Returns the number of points actually drawn.

**See Also**

sigDrawDot, sigRenderDot, sigDisplayDots

---

**sigDisplayDots**

**Win:** unsigned sigDisplayDots( HDC aDC, SIG aSig, int atX, int atY )

**DOS:** unsigned sigDisplayDots( SIG aSig, int atX, int atY )

Display all the points of signature at its actual size.

<b>Parameter</b>	<b>Description</b>
aDC	Handle to a Windows Display Context.
aSig	Signature to be displayed.
atX	Horizontal coordinate to display image.
atY	Vertical coordinate to display image.

**Description**

Displays a signature as dots at its actual size at a given location. Points are temporarily converted to match the display device dots-per-inch resolution.

**Returns**

Returns the number of points actually drawn.

**See Also**

sigDrawDots, sigRenderDots, sigDisplayDot

---

**sigDisplayLast**

**Win:** unsigned sigDisplayLast( HDC aDC, SIG aSig, int atX, int atY )

**DOS:** unsigned sigDisplayLast( SIG aSig, int atX, int atY )

Display the last line of a signature at its actual size.

<b>Parameter</b>	<b>Description</b>
aDC	Handle to a Windows Display Context.
aSig	Signature to be displayed.

atX                    Horizontal coordinate to display image.  
atY                    Vertical coordinate to display image.

**Description**

Displays a the last line of a signature at its actual size at a given location. Points are temporarily converted to match the display device dots-per-inch resolution. This function is used to “update” a displayed signature after adding a new point, without having to redraw the entire signature.

**Returns**

Returns the number of points in the signature if successful, zero otherwise.

**See Also**

sigDraw, sigRender, sigDisplay, sigDisplayLine, sigDisplayPoint

---

**sigDisplayLine**

**Win:**    unsigned sigDisplayLine( HDC aDC, SIG aSig,, unsigned anIndex, int atX, int atY )

**DOS:**    unsigned sigDisplayLine( SIG aSig, unsigned anIndex, int atX, int atY )

Display a line segment of a signature at its actual size.

<b>Parameter</b>	<b>Description</b>
aDC	Handle to a Windows Display Context.
aSig	Signature to be displayed.
anIndex	Index of the first point of the line.
atX	Horizontal coordinate to display image.
atY	Vertical coordinate to display image.

**Description**

Displays a line segment of a signature at its actual size at a given location. Points are temporarily converted to match the display device dots-per-inch resolution.

**Returns**

Returns an index to the next line if successful, zero otherwise.

**See Also**

sigDraw, sigRender, sigDisplay, sigDisplayLast, sigDisplayPoint

---

**sigDisplayStroke**

**Win:**    unsigned sigDisplayStroke( HDC aDC, SIG aSig, unsigned anIndex, int atX, int atY )

**DOS:**    unsigned sigDisplayStroke( SIG aSig, unsigned anIndex, int atX, int atY )

Display a complete stroke of a signature at its actual size.

<b>Parameter</b>	<b>Description</b>
aDC	Handle to a Windows Display Context.
aSig	Signature to be displayed.
anIndex	Index of the first point of the stroke.

atX                    Horizontal coordinate to display image.  
atY                    Vertical coordinate to display image.

**Description**

Displays a complete stroke of a signature at its actual size at a given location. Points are temporarily converted to match the display device dots-per-inch resolution.

**Returns**

Returns an index to the next stroke if successful, zero otherwise.

**See Also**

sigDraw, sigRender, sigDisplay, sigDisplayLast, sigDisplayLine

---

**sigDraw**

**Win:**    unsigned sigDraw( HDC aDC, SIG aSig, int atX, int atY )

**DOS:**    unsigned sigDraw( SIG aSig, int atX, int atY )

Draw a signature without converting size.

<b>Parameter</b>	<b>Description</b>
aDC	Handle to a Windows Display Context.
aSig	Signature to be displayed.
atX	Horizontal coordinate to display image.
atY	Vertical coordinate to display image.

**Description**

Draws a signature at a given location. Points are not scaled to match the output device nor is the aspect ratio maintained.

**Returns**

Returns the number of line segments actually drawn.

**See Also**

sigDisplay, sigRender, sigDrawLast, sigDrawLine, sigDrawStroke

---

**sigDrawDot**

**Win:**    unsigned sigDrawDot( HDC aDC, SIG aSig, unsigned anIndex, int atX, int atY )

**DOS:**    unsigned sigDrawDot( SIG aSig, unsigned anIndex, int atX, int atY )

Draw a point of a signature without converting size.

<b>Parameter</b>	<b>Description</b>
aDC	Handle to a Windows Display Context.
aSig	Signature to be displayed.
atX	Horizontal coordinate to display image.
atY	Vertical coordinate to display image.

**Description**

Draws a single point of a signature relative to a given location. The point is not scaled to match the output device nor is the aspect ratio maintained.

**Returns**

Returns the number of points actually drawn.

**See Also**

sigDisplayDot, sigRenderDot, sigDrawDots

---

**sigDrawDots**

**Win:** unsigned sigDrawDots( HDC aDC, SIG aSig, int atX, int atY )

**DOS:** unsigned sigDrawDots( SIG aSig, int atX, int atY )

Draw all points of a signature without converting size.

<b>Parameter</b>	<b>Description</b>
aDC	Handle to a Windows Display Context.
aSig	Signature to be displayed.
anIndex	Index of the point to be drawn.
atX	Horizontal coordinate to display image.
atY	Vertical coordinate to display image.

**Description**

Draws all points of a signature as dots at a given location. The points are not scaled to match the output device nor is the aspect ratio maintained.

**Returns**

Returns the number of points actually drawn.

**See Also**

sigDisplayDots, sigRenderDots, sigDrawDot

---

**sigDrawLast**

**Win:** unsigned sigDrawLast( HDC aDC, SIG aSig, int atX, int atY )

**DOS:** unsigned sigDrawLast( SIG aSig, int atX, int atY )

Draw the last line of a signature without converting size.

<b>Parameter</b>	<b>Description</b>
aDC	Handle to a Windows Display Context.
aSig	Signature to be displayed.
atX	Horizontal coordinate to display image.
atY	Vertical coordinate to display image.

**Description**

Draws the last line of a signature at a given location. Points are not scaled to match the output device nor is the aspect ratio maintained. This function is used to “update” a displayed signature after adding a new point, without having to redraw the entire signature.

**Returns**

Returns the number of points in the signature if successful, zero otherwise.

**See Also**

sigDisplay, sigRender, sigDraw, sigDrawLine, sigDrawStroke

---

**sigDrawLine**

**Win:** `unsigned sigDrawLine( HDC aDC, SIG aSig, unsigned anIndex, int atX, int atY )`

**DOS:** `unsigned sigDrawLine( SIG aSig, unsigned anIndex, int atX, int atY )`

Draw a line segment of a signature without converting size.

<b>Parameter</b>	<b>Description</b>
aDC	Handle to a Windows Display Context.
aSig	Signature to be displayed.
anIndex	Index of the first point of the line.
atX	Horizontal coordinate to display image.
atY	Vertical coordinate to display image.

**Description**

Draws a line segment of a signature at a given location. Points are not scaled to match the output device nor is the aspect ratio maintained.

**Returns**

Returns an index to the next line if successful, zero otherwise.

**See Also**

sigDisplay, sigRender, sigDraw, sigDrawLast, sigDrawStroke

---

**sigDrawStroke**

**Win:** `unsigned sigDrawStroke( HDC aDC, SIG aSig, unsigned anIndex, int atX, int atY )`

**DOS:** `unsigned sigDrawStroke( SIG aSig, unsigned anIndex, int atX, int atY )`

Draw a complete stroke of a signature without converting size.

<b>Parameter</b>	<b>Description</b>
aDC	Handle to a Windows Display Context.
aSig	Signature to be displayed.
anIndex	Index of the first point of the stroke.
atX	Horizontal coordinate to display image.
atY	Vertical coordinate to display image.

**Description**

Draws a complete stroke of a signature at a given location. Points are not scaled to match the output device nor is the aspect ratio maintained.

**Returns**

Returns an index to the next stroke if successful, zero otherwise.

**See Also**

sigDisplay, sigRender, sigDraw, sigDrawLine, sigDrawLast

---

**sigDup**

**SIG sigDup( SIG aSig )**

Duplicates a signature.

<b>Parameter</b>	<b>Description</b>
aSig	Signature to be duplicated.

**Description**

Create a new signature object that is an exact duplicate of another. Once finished with the signature, the programmer should use *sigDelete* to release memory used.

**Returns**

Returns the duplicate signature object if successful, NULL otherwise.

**See Also**

sigDelete, sigNew

---

**sigEmpty**

**BOOL sigEmpty( aSig )**

Empty a signature.

<b>Parameter</b>	<b>Description</b>
aSig	Signature to be emptied.

**Description**

Empty a signature object by removing all points.

**Returns**

Returns TRUE if successful, FALSE otherwise.

**See Also**

sigDelete, sigUndoPoint, sigUndoStroke

---

## sigFirstLine

**unsigned sigFirstLine( SIG aSig )**

Get the first line.

<b>Parameter</b>	<b>Description</b>
aSig	Signature to be checked.

**Returns**  
Returns an index to the first line of a signature.

**See Also**  
sigNextLine, sigPrevLine, sigLastLine

---

## sigFirstPoint

**unsigned sigFirstPoint( SIG aSig )**

Get the first point.

<b>Parameter</b>	<b>Description</b>
aSig	Signature to be checked.

**Returns**  
Returns an index to the first point of a signature.

**See Also**  
sigNextPoint, sigPrevPoint, sigLastPoint

---

## sigFirstStroke

**unsigned sigFirstStroke( SIG aSig )**

Get first stroke.

<b>Parameter</b>	<b>Description</b>
aSig	Signature to be checked.

**Returns**  
Returns an index to the first stroke of a signature.

**See Also**  
sigNextStroke, sigPrevStroke, sigLastStroke

## sigFitHeight

**BOOL sigFitHeight( SIG aSig, int aHeight )**

Scale to fit height.

Parameter	Description
aSig	Signature to be scaled.
aHeight	Desired height.

### Description

Set a signature scaling factors to fit the given height.

### Returns

Returns TRUE if successful, FALSE otherwise.

### See Also

sigFitSize, sigFitWidth

---

## sigFitSize

**BOOL sigFitSize( SIG aSig, int aWidth, int aHeight )**

Scale to fit size.

Parameter	Description
aSig	Signature to be scaled.
aWidth	Desired width.
aHeight	Desired height.

### Description

Set a signature scaling factors to fit the given size. This function differs from *sigStretchSize* in that it does not alter the aspect ratio. Note that the size given is assumed to have a 1:1 aspect ratio.

### Returns

Returns TRUE if successful, FALSE otherwise.

### See Also

sigFitHeight, sigFitWidth, sigStretchSize

---

## sigFitWidth

**BOOL sigFitWidth( SIG aSig, aWidth )**

Scale to fit width.

Parameter	Description
-----------	-------------

---

aSig           Signature to be scaled.  
aWidth         Desired width.

**Description**

Set a signature scaling factors to fit the given width.

**Returns**

Returns TRUE if successful, FALSE otherwise.

**See Also**

sigFitHeight, sigFitSize

---

**sigFlipHorz**

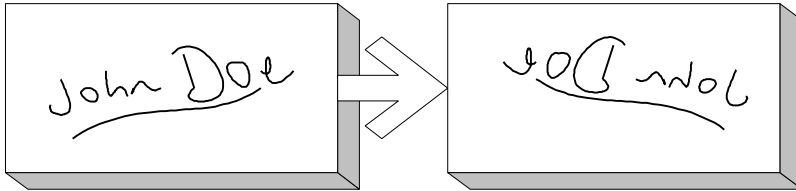
**BOOL sigFlipHorz( SIG aSig )**

Flip image horizontally.

Parameter	Description
aSig	Signature to be flipped.

**Description**

Flip the signature horizontally to produce a mirrored image. This function can be undone by simply executing it again to flip the image back to its original orientation.

**Returns**

Returns TRUE if successful, FALSE otherwise.

**See Also**

sigFlipVert

---

**sigFlipVert**

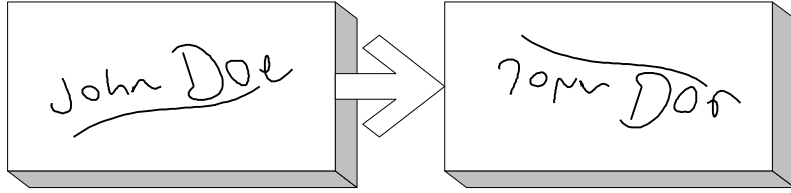
**BOOL sigFlipVert( SIG aSig )**

Flip image vertically.

Parameter	Description
aSig	Signature to be flipped.

**Description**

Flip the signature vertically to produce an upside down image. This function can be undone by simply executing it again to flip the image back to its original orientation.

**Returns**

Returns TRUE if successful, FALSE otherwise.

**See Also**

sigFlipHorz

---

**sigGet**

**BOOL sigGet( SIG aSig, unsigned anIndex, int \*anX, int \*aY, int \*aP )**

Get an unscaled point.

Parameter	Description
aSig	Signature.
anIndex	Index of the point to be retrieved.
anX	Optional pointer to location to hold the horizontal coordinate.
aY	Optional pointer to location to hold the vertical coordinate.
aP	Optional pointer to location to hold the pen type (0=moveto, 1=lineto).

**Description**

Retrieves a given point from a signature. The point is not adjusted to the current scaling factors. If an “optional pointer” is NULL, then retrieving of the associated value is suppressed.

**Returns**

Returns TRUE if successful, FALSE otherwise.

**See Also**

sigGetPoint, sigAdd

---

**sigGetBkColor**

**Win:** COLORREF sigGetBkColor( SIG aSig )

**DOS:** int sigGetBkColor( SIG aSig )

Gets the background color of a signature.

Parameter	Description
aSig	Signature object.

**Description**

Returns the background color of a signature, if any. This function will return the constant SIG\_NOCOLOR if none is specified.

**Returns**

Returns the background color.

**See Also**

sigSetBkColor, sigGetColor

---

**sigGetBounds**

**BOOL sigGetBounds( SIG aSig, int \*L, int \*T, int \*R, int \*B )**

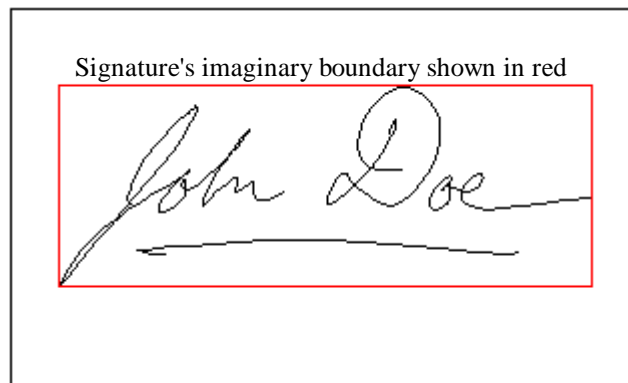
Get boundaries.

Parameter	Description
aSig	Signature to be checked.
L	Optional pointer to a location to store the leftmost horizontal coordinate
T	Optional pointer to a location to store the topmost vertical coordinate
R	Optional pointer to a location to store the rightmost horizontal coordinate
B	Optional pointer to a location to store the bottommost vertical coordinate

**Description**

Retrieves the boundaries of the signature image. The points are adjusted to the current scaling factors. If an “optional pointer” is NULL, then retrieval of the associated value is skipped.

NOTE: This function does not retrieve the bounding page the signature is located on. For example, the page of a signature object could be 0, 0, 1024, 1024, while the boundaries of the signature contained within the page could be 24, 295, 823, 576. The signature's boundaries in a signature object depend on where the signature is located on its page and how large the signature is. The drawing below illustrates this. The black box defines the signature object's page and the red box outlines the boundaries of the signature containing within the page.

**Returns**

Returns TRUE if successful, FALSE otherwise.

**See Also**

sigGetPage, sigShrinkWrap

---

---

## sigGetColor

**Win:** COLORREF sigGetColor( SIG aSig )

**DOS:** int sigGetColor( SIG aSig )

Gets the pen color of a signature.

Parameter	Description
aSig	Signature object.

### Description

Returns the pen color of a signature, if any. This function will return the constant SIG\_NOCOLOR if none is specified.

### Returns

Returns the pen color.

### See Also

sigSetColor, sigGetBkColor, sigGetPen

---

## sigGetPage

**void sigGetPage( SIG aSig, int \*aWidth, int \*aHeight, int \*aHorzDPI, int \*aVertDPI )**

Get page specifications.

Parameter	Description
aSig	Signature to be checked.
aWidth	Optional pointer to location to store the horizontal size in dots.
aHeight	Optional pointer to location to store the vertical size in dots.
aHorzDPI	Optional pointer to location to store the horizontal dots-per-inch.
aVertDPI	Optional pointer to location to store the vertical dots-per-inch.

### Description

Retrieves the page size of a signature object as defined when the signature object was first created using *sigNew*. If an optional pointer is NULL, then retrieval of the associated value is skipped. Note that the sizes returned due not reflect any scaling options. If you need scaled sizes, use *sigWidth* and *sigHeight* instead.

### Returns

Returns TRUE if successful, FALSE otherwise.

### See Also

sigHeight, sigHorzDPI, sigVertDPI, sigWidth, sigNew

---

## sigGetPen

**int sigGetPen( SIG aSig )**

Gets the pen size of a signature.

<b>Parameter</b>	<b>Description</b>
aSig	Signature object.

**Description**

Returns the pen size of a signature, if any. Note that pen sizes are measured in thousandths of a inch. This function will return the constant SIG\_NOPEN if none is specified.

**Returns**

Returns the pen size in thousandths of an inch.

**See Also**

sigSetPen,, sigGetColor

---

**sigGetPoint****BOOL sigGetPoint( SIG aSig, unsigned anIndex, int \*anX, int \*aY, int \*aP )**

Get scaled point.

<b>Parameter</b>	<b>Description</b>
aSig	Signature.
anIndex	Index of the point to be retrieved.
anX	Pointer to location to hold the scaled horizontal coordinate.
aY	Pointer to location to hold the scaled vertical coordinate.
aP	Pointer to location to hold the pen type (0=moveto, 1=lineto).

**Description**

Retrieves a given point from a signature. The point is adjusted to the current scaling factors. If an “optional pointer” is NULL, then retrieving of the associated value is suppressed.

**Returns**

Returns TRUE if successful, FALSE otherwise.

**See Also**

sigGet, sigAdd, sigSetScale

---

**sigGetScale****BOOL sigGetScale( SIG aSig, int \*xNum, int \*xDenom, int \*yNum, int \*yDenom )**

Get the scaling factors.

<b>Parameter</b>	<b>Description</b>
aSig	Signature to be checked.
xNum	Pointer to location to hold the horizontal scaling numerator.

---

xDenom        Pointer to location to hold the horizontal scaling denominator.  
yNum          Pointer to location to hold the vertical scaling numerator.  
yDenom        Pointer to location to hold the vertical scaling denominator.

**Description**

Retrieves the scaling factors of a signature.

**Returns**

Returns TRUE if successful, FALSE otherwise.

**See Also**

sigSetScale

---

**sigHeight**

**int sigHeight( SIG aSig )**

Get height.

<b>Parameter</b>	<b>Description</b>
aSig	Signature to be checked.

**Returns**

Returns the height, in dots, of a signature object's frame area.

**See Also**

sigWidth, sigGetPage

---

**sigHorzDPI**

**int sigHorzDPI( SIG aSig )**

Get horizontal dpi resolution.

<b>Parameter</b>	<b>Description</b>
aSig	Signature to be checked.

**Returns**

Returns the horizontal dot-per-inch resolution of a signature.

**See Also**

sigVertDPI, sigGetPage

---

**sigIsEmpty**

**BOOL sigIsEmpty( SIG aSig )**

---

Check if empty.

<b>Parameter</b>	<b>Description</b>
aSig	Signature to be checked.

**Returns**  
Returns TRUE if the signature is empty, FALSE otherwise.

**See Also**  
sigEmpty

---

## **sigIsEnhanced**

**BOOL sigIsEnhanced( SIG aSig )**

Checks if enhanced.

<b>Parameter</b>	<b>Description</b>
aSig	Signature to be checked.

**Returns**  
Returns TRUE if the signature is has been marked as “enhanced”, FALSE otherwise.

**See Also**  
sigSetEnhanced, sigIsReduced

---

## **sigIsReduced**

**BOOL sigReduced( SIG aSig )**

Checks if reduced.

<b>Parameter</b>	<b>Description</b>
aSig	Signature to be checked.

**Returns**  
Returns TRUE if the signature is has been marked as “enhanced”, FALSE otherwise.

**See Also**  
sigSetReduced, sigIsEnhanced

---

## **sigIsScaled**

**BOOL sigScaled( SIG aSig )**

Checks if scaled.

---

<b>Parameter</b>	<b>Description</b>
aSig	Signature to be checked.

**Returns**  
Returns TRUE if the signature has been scaled, FALSE otherwise.

**See Also**  
sigSetScale, sigGetScale

---

## sigIsVoid

**BOOL sigIsVoid( SIG aSig )**

Checks if void.

<b>Parameter</b>	<b>Description</b>
aSig	Signature to be checked.

**Returns**  
Returns TRUE if the signature has marked “void”, FALSE otherwise.

**See Also**  
sigVoid

---

## sigLastLine

**unsigned sigLastLine( SIG aSig )**

Get last line.

<b>Parameter</b>	<b>Description</b>
aSig	Signature to be checked.

**Returns**  
Returns an index to the last line of a signature.

**See Also**  
sigFirstLine, sigNextLine, sigPrevLine

---

## sigLastPoint

**unsigned sigLastPoint( SIG aSig )**

Get last point.

---

<b>Parameter</b>	<b>Description</b>
aSig	Signature to be checked.

**Returns**  
Returns an index to the last point of a signature.

**See Also**  
sigFirstPoint, sigNextPoint, sigPrevPoint

---

## sigLastStroke

**unsigned sigLastStroke( SIG aSig )**

Get last stroke.

<b>Parameter</b>	<b>Description</b>
aSig	Signature to be checked.

**Returns**  
Returns an index to the last stroke of a signature.

**See Also**  
sigFirstStroke, sigNextStroke, sigPrevStroke

---

## sigLoad

**BOOL sigLoad( SIG aSig, const char \*aFileName )**

Loads a signature.

<b>Parameter</b>	<b>Description</b>
aSig	Signature to be loaded.
aFileName	Name of the file to load.

### **Description**

Loads a signature image into *aSig* from a the file named *aFileName*.

**Returns**  
Returns TRUE if successful, FALSE otherwise.

**See Also**  
sigSave, sigRead

---

## sigMemError

**WORD sigMemError( void )**

---

Returns the last signature memory object error reported by sigWriteMem, sigReadMem and sigWriteCmpMem, and sigWriteNlcMem.

**Description**

Loads a signature image into *aSig* from a the file named *aFileName*.

**Returns**

This function returns the following values as specified in **SIGKIT.H**:

<b>MEM_OK</b>	no error occurred.
<b>MEM_POINTER_NULL</b>	the memory object was NULL.
<b>MEM_SIG_NULL</b>	that SIG object was NULL.
<b>MEM_SIG_FORMAT_BAD</b>	the data was unreadable.
<b>MEM_SIG_CANT_ADD</b>	unable to add points the SIG object.
<b>MEM_TOO_SMALL</b>	the memory object given was too small.
<b>MEM_ALLOC_ERROR</b>	unable to allocate memory.

**See Also**

sigWriteMem, sigReadMem and sigWriteCmpMem

---

**sigNew**

**SIG sigNew( int aWidth, int aHeight, int aHorzDPI, int aVertDPI )**

Create a new signature.

<b>Parameter</b>	<b>Description</b>
aWidth	Horizontal size, in dots, of the signature frame.
aHeight	Vertical size, in dots, of the signature frame.
aHorzDPI	Horizontal dots-per-inch resolution.
aVertDPI	Vertical dots-per-inch resolution.

**Description**

Creates a new and empty signature object with the given specifications. Once finished with the signature, the programmer should sigDelete to release memory used

**Returns**

Returns the new signature object if successful, NULL otherwise.

**See Also**

sigDelete, sigDup, sigEmpty

---

**sigNextLine**

**unsigned sigNextLine( SIG aSig, unsigned anIndex )**

Get next line.

<b>Parameter</b>	<b>Description</b>
aSig	Signature to be checked.
anIndex	An index to the point from which to start checking.

**Returns**  
Returns an index to a next line.

**See Also**  
sigFirstLine, sigPrevLine, sigLastLine

---

## **sigNextPoint**

**unsigned sigNextPoint( SIG aSig, unsigned anIndex )**

Get next point.

<b>Parameter</b>	<b>Description</b>
aSig	Signature to be checked.
anIndex	An index to the point from which to start checking.

**Returns**  
Returns an index to a next point.

**See Also**  
sigFirstPoint, sigPrevPoint, sigLastPoint

---

## **sigNextStroke**

**unsigned sigNextStroke( SIG aSig, unsigned anIndex )**

Get next stroke.

<b>Parameter</b>	<b>Description</b>
aSig	Signature to be checked.
anIndex	An index to the point from to start checking.

**Returns**  
Returns an index to a next stroke.

**See Also**  
sigFirstStroke, sigPrevStroke, sigLastStroke

---

## **sigNumPoints**

**unsigned sigNumPoints( SIG aSig )**

Get number of points.

---

<b>Parameter</b>	<b>Description</b>
aSig	Signature to be checked.

**Returns**  
Returns the total number of points in a signature.

**See Also**  
sigNumLines, sigNumStrokes

---

## sigNumLines

**unsigned sigNumLines( SIG aSig )**

Get number of lines.

<b>Parameter</b>	<b>Description</b>
aSig	Signature to be checked.

**Returns**  
Returns the total number of lines in a signature.

**See Also**  
sigNumPoints, sigNumStrokes

---

## sigNumStrokes

**unsigned sigNumStrokes( SIG aSig )**

Get number of strokes.

<b>Parameter</b>	<b>Description</b>
aSig	Signature to be checked.

**Returns**  
Return the total number of complete strokes in a signature.

**See Also**  
sigNumPoints, sigNumLines

---

## sigOffset

**BOOL sigOffset( SIG aSig, int xDist, int yDist )**

Offset a signature.

---

<b>Parameter</b>	<b>Description</b>
aSig	Signature to be offset.
xDist	Number of horizontal points to offset image.
yDist	Number of vertical points to offset image.

**Description**

Offset a signature image within its frame. If the offsets are large, the rendering may result in a truncated signature display.

**Returns**

Returns TRUE if successful, FALSE otherwise.

**See Also**

sigCenter, sigAlignTop, sigAlignBottom, sigAlignLeft, sigAlignRight

---

**sigPrevLine**

**unsigned sigPrevLine( SIG aSig, unsigned anIndex )**

Get previous line.

<b>Parameter</b>	<b>Description</b>
aSig	Signature to be checked.
anIndex	An index to the point from which to start checking.

**Returns**

Returns an index to a previous line.

**See Also**

sigFirstLine, sigNextLine, sigLastLine

---

**sigPrevPoint**

**unsigned sigPrevPoint( SIG aSig, unsigned anIndex )**

Get previous point.

<b>Parameter</b>	<b>Description</b>
aSig	Signature to be checked.
anIndex	An index to the point from which to start checking.

**Returns**

Returns an index to a previous point.

**See Also**

sigFirstPoint, sigNextPoint, sigLastPoint

## sigPrevStroke

**unsigned sigPrevStroke( SIG aSig, unsigned anIndex )**

Get previous stroke.

Parameter	Description
aSig	Signature to be checked.
anIndex	An index to the point from which to start checking.

### Returns

Returns an index to a previous stroke.

### See Also

sigFirstStroke, sigNextStroke, sigLastStroke

---

## sigRead

**BOOL sigRead( SIG aSig, FILE aFile )**

Read signature data. The user should use sigReadWIN if using windows.

Parameter	Description
aSig	Signature to be read.
aFile	Handle to a file stream.

### Description

Read signature data from an input file stream. The file must be open for reading and positioned at the start of the signature data to be retrieved.

### Returns

Returns TRUE if successful, FALSE otherwise.

### See Also

sigWrite, sigLoad, sigWriteWIN, sigReadWIN

---

## sigReadMem

**BOOL sigReadMem ( SIG aSig, BYTE FAR \*mem, WORD memSize )**

Reads points from SIG, NLC, and CMP format memory objects into a SIG data type.

Parameter	Description
aSig	SIG object to fill
mem	Memory to fill from
memSize	Size of the bytes in the memory containing the signature data

**Description**

Reads signature data into the signature object specified by **aSig** from a memory object pointed to by the byte pointer **mem**. The parameter **memSize** gives the size in bytes of the signature data stored in the memory object. Data formats read include SIG, NLC, and CMP (the CMP format is obsolete, please use NLC instead). This includes memory objects created using **sigWriteMem**, **sigWriteNlcMem**, **sigWriteCmpMem**, and memory objects created using the @pos.com SigBox ActiveX control.

**NOTE:** The user is responsible for creating the signature object **aSig** before using this function.

**Returns**

Returns TRUE if successful, FALSE otherwise (see sigMemError for a detailed error message).

**See Also**

sigLoad, sigMemError, sigRead, sigReadSigMem, sigReadNlcMem, sigReadCmpMem, sigWrite, sigWriteMem, sigWriteNlcMem

---

**sigReadCmpMem**

**BOOL sigReadCmpMem ( SIG aSig, BYTE FAR \*mem, WORD memSize )**

Reads points from CMP format memory into a SIG data type

<b>Parameter</b>	<b>Description</b>
aSig	SIG object to fill
mem	Memory to fill from
memSize	Size of the bytes in the memory containing the signature data

**Returns**

Returns TRUE if success, FALSE otherwise.

**See Also**

sigReadMem, sigReadSigMem, sigReadNlcMem

---

**sigReadNlcMem**

**BOOL sigReadNlcMem( SIG aSig, BYTE FAR \*mem, WORD memSize )**

Read the NLC format from memory into SIG type data structure.

<b>Parameter</b>	<b>Description</b>
aSig	SIG object to fill
mem	Memory to fill from
memSize	Size of the bytes in the memory containing the signature data

**Returns**

Returns TRUE if success, FALSE otherwise.

**See Also**

sigReadMem, sigReadSigMem, sigReadCmpMem

---

**sigReadSigMem****BOOL sigReadSigMem ( SIG aSig, BYTE FAR \*mem, WORD memSize )**

Reads points from SIG format memory into a SIG data type

<b>Parameter</b>	<b>Description</b>
aSig	SIG object to fill
mem	Memory to fill from
memSize	Size of the bytes in the memory object containing the signature data

**Returns**

Returns TRUE if success, FALSE otherwise.

**See Also**

sigReadMem, sigReadCmpMem, sigReadNlcMem

---

**sigReadNLC****BOOL sigReadNLC( SIG aSig, char \*nlcFile )****NOTE: This function is available only in libraries compiled with Microsoft compilers. Libraries compiled with Borland compilers do not support this function.**

Reads an NLC formatted compressed file into SIG object.

<b>Parameter</b>	<b>Description</b>
aSig	Signature object.
nlcFile	The name of the NLC formatted compressed file.

**Description**

Reads a NLC compression format file into SIG object. The file can then be saved as a SIG file of any one of the other supported conversion file formats. NLC compression is a format developed by @pos.com.

**Returns**

Returns TRUE if successful, FALSE otherwise.

**See Also**

sigSaveAsNLC, sigSave, sigRead

## sigReadWIN

**BOOL sigReadWIN( SIG aSig, HFILE aFile )**

Read signature data. This procedure call should be used while executing under windows.

<b>Parameter</b>	<b>Description</b>
aSig	Signature to be read.
aFile	Handle to a file stream. This is returned from the Windows OpenFile function call.

### **Description**

Read signature data from an input file stream. The file must be open for reading and positioned at the start of the signature data to be retrieved.

### **Returns**

Returns TRUE if successful, FALSE otherwise.

### **See Also**

sigWrite, sigLoad, sigRead, sigWriteWIN

---

## sigRemove

**unsigned sigRemove( SIG aSig, unsigned anIndex, unsigned aCount )**

Remove points.

<b>Parameter</b>	<b>Description</b>
aSig	Signature to be modified.
anIndex	Index to the first point to be removed.
aCount	Number of points to remove.

### **Description**

Remove *aCount* number of points from a signature starting at *anIndex*.

### **Returns**

Returns the number of points actually removed.

### **See Also**

sigAdd, sigEmpty

---

## sigRender

**Win: unsigned sigRender( HDC aDC, SIG aSig, int L, int T, int R, int B )**

**DOS: unsigned sigRender( SIG aSig, int xLeft, int L, int T, int R, int B )**

Draw a signature within a frame.

---

<b>Parameter</b>	<b>Description</b>
aDC	Handle to a Windows Display Context.
aSig	Signature to be drawn.
L	Horizontal coordinate of the top-left corner of the frame.
T	Vertical coordinate of the top-left corner of the frame.
R	Horizontal coordinate of the bottom-right corner of the frame.
B	Vertical coordinate of the bottom-right corner of the frame.

**Description**

Draws the signature within the frame defined by coordinates L,T and R,B. The signature is scaled to fit and the aspect ratio is maintained. Note that the frame defined assumed to have a 1:1 aspect ratio.

**Returns**

Returns the number of line segments actually drawn.

**See Also**

sigDraw, sigDisplay, sigRenderLast

---

**sigRenderDot**

**Win:** unsigned sigRenderDot( HDC aDC, SIG aSig, unsigned anIndex, int L, int T, int R, int B )  
**DOS:** unsigned sigRenderDot( SIG aSig, unsigned anIndex, int L, int T, int R, int B )

Draw a point of signature within a frame.

<b>Parameter</b>	<b>Description</b>
aDC	Handle to a Windows Display Context.
aSig	Signature to be drawn.
anIndex	Index of the point to be drawn.
L	Horizontal coordinate of the top-left corner of the frame.
T	Vertical coordinate of the top-left corner of the frame.
R	Horizontal coordinate of the bottom-right corner of the frame.
B	Vertical coordinate of the bottom-right corner of the frame.

**Description**

Draws a single point of a signature as a dot within the frame defined by coordinates L,T and R,B. The signature is scaled to fit and the aspect ratio is maintained. Note that the frame defined assumed to have a 1:1 aspect ratio.

**Returns**

Returns the number of points actually drawn.

**See Also**

sigDrawDot, sigDisplayDot, sigRenderDots

---

**sigRenderDots**

**Win:** `unsigned sigRenderDots( HDC aDC, SIG aSig, int L, int T, int R, int B )`

**DOS:** `unsigned sigRenderDots( SIG aSig, int L, int T, int R, int B )`

Draw all points signature within a frame as dots.

<b>Parameter</b>	<b>Description</b>
aDC	Handle to a Windows Display Context.
aSig	Signature to be drawn.
L	Horizontal coordinate of the top-left corner of the frame.
T	Vertical coordinate of the top-left corner of the frame.
R	Horizontal coordinate of the bottom-right corner of the frame.
B	Vertical coordinate of the bottom-right corner of the frame.

#### **Description**

Draws all points of a signature as dots within the frame defined by coordinates L,T and R,B. The signature is scaled to fit and the aspect ratio is maintained. Note that the frame defined assumed to have a 1:1 aspect ratio.

#### **Returns**

Returns the number of points actually drawn.

#### **See Also**

sigDrawDots, sigDisplayDots, sigRenderDot

---

## **sigRenderLast**

**Win:** `unsigned sigRenderLast( HDC aDC, SIG aSig, int L, int T, int R, int B )`

**DOS:** `unsigned sigRenderLast( SIG aSig, int L, int T, int R, int B )`

Draw the last line of a signature within a frame.

<b>Parameter</b>	<b>Description</b>
aDC	Handle to a Windows Display Context.
aSig	Signature to be drawn.
L	Horizontal coordinate of the top-left corner of the frame.
T	Vertical coordinate of the top-left corner of the frame.
R	Horizontal coordinate of the bottom-right corner of the frame.
B	Vertical coordinate of the bottom-right corner of the frame.

#### **Description**

Draws the last line of a signature within the frame defined by coordinates L,T and R,B. The signature is scaled to fit and the aspect ratio is maintained. Note that the frame defined assumed to have a 1:1 aspect ratio. This function is used to “update” a displayed signature after adding a new point, without having to redraw the entire signature.

#### **Returns**

Returns the number of points in the signature if successful, zero otherwise.

#### **See Also**

sigDrawLast, sigDisplayLast, sigRender

---

## sigRenderLine

**Win:** unsigned sigRenderLine( HDC aDC, SIG aSig, unsigned anIndex, int L, int T, int R, int B )  
**DOS:** unsigned sigRenderLine( SIG aSig, unsigned anIndex, int L, int T, int R, int B )

Draw a line of a signature within a frame.

Parameter	Description
aDC	Handle to a Windows Display Context.
aSig	Signature to be drawn.
anIndex	Index of the line to be drawn.
L	Horizontal coordinate of the top-left corner of the frame.
T	Vertical coordinate of the top-left corner of the frame.
R	Horizontal coordinate of the bottom-right corner of the frame.
B	Vertical coordinate of the bottom-right corner of the frame.

### Description

Draws the specified line of a signature within the frame defined by L,T and R,B. The signature is scaled to fit and the aspect ratio is maintained. Note that the frame defined assumed to have a 1:1 aspect ratio.

### Returns

Returns an index to the next line if successful, zero otherwise.

### See Also

sigDrawLine, sigDisplayLine, sigRender

---

## sigRenderStroke

**Win:** unsigned sigRenderStroke( HDC aDC, SIG aSig, unsigned anIndex, int L,int T, int R,int B )  
**DOS:** unsigned sigRenderStroke( SIG aSig, unsigned anIndex, int L, int T, int R, int B )

Draws a stroke of a signature within a frame.

Parameter	Description
aDC	Handle to a Windows Display Context.
aSig	Signature to be drawn.
anIndex	Index to the stroke to be drawn.
L	Horizontal coordinate of the top-left corner of the frame.
T	Vertical coordinate of the top-left corner of the frame.
R	Horizontal coordinate of the bottom-right corner of the frame.
B	Vertical coordinate of the bottom-right corner of the frame.

### Description

Draws the specified stroke of a signature within the frame defined by coordinates L,T and R,B. The signature is scaled to fit and the aspect ratio is maintained. Note that the frame defined assumed to have a 1:1 aspect ratio.

### Returns

Returns an index to the next stroke if successful, zero otherwise.

**See Also**

sigDrawStroke, sigDisplayStroke, sigRender

---

**sigResetScale**

**void sigResetScale( SIG aSig )**

Reset scaling.

<b>Parameter</b>	<b>Description</b>
aSig	Signature to be reset.

**Description**

Resets the scaling factors to 100% of the original size. Note that this function will not undo permanent scale conversions as performed by *sigCvtDPI* and *sigCvtScale*.

**See Also**

sigSetScale

---

**sigSave**

**BOOL sigSave( SIG aSig, const char \*aFileName )**

Save a signature.

<b>Parameter</b>	<b>Description</b>
aSig	Signature to be saved.
aFileName	Name of the file to be created/overwritten.

**Description**

Saves the signature image in *aSig* to a file named *aFileName*. The image can be reloaded at a later time using *sigLoad*.

**Returns**

Returns TRUE if successful, FALSE otherwise.

**See Also**

sigLoad, sigWrite

---

**sigSaveAs**

**BOOL sigSaveAs( SIG aSig, const char \*aFileName )**

Save a signature.

---

<b>Parameter</b>	<b>Description</b>
aSig	Signature to be saved.
aFileName	Name of the file to be created/overwritten.

**Description**

Saves the signature image in *aSig* to a file named *aFileName*. The image is saved according to the file extension, which must be one of the following:

ext	equivalent	description
.SIG	sigSave	@pos.com SIG vectored signature file
.PCX	sigSaveAsPCX	Z-Soft Paintbrush PCX bitmap
.BMP	sigSaveAsBMP	Windows BMP device dependent bitmap
.DIB	sigSaveAsDIB	Windows DIB device independent bitmap
.CGM	sigSaveAsCGM	Computer Graphic Metafile
.EPS	sigSaveAsEPS	Encapsulated PostScript
.PCL	sigSaveAsPCL	HP Printer Language codes
.TIF	sigSaveAsTIF	Tagged image file format bitmap
.TXT	sigSaveAsTXT	Text file dump
.WMF	sigSaveAsWMF	Placable Windows Metafile
.MF	sigSaveAsMF	Standard Windows Metafile
.NLC	sigSaveAsNLC	NLC formatted file using a compression mode of 3.

**WARNING:**

This function is provided for convenience to applications that need to export to all supported file types, such as a file conversion utility. Using this function will cause all file formats to be linked and will affect the size of the application accordingly!

**Returns**

Returns TRUE if successful, FALSE otherwise.

**Example:**

The following lines of code loads in a signature and save it as a Windows bitmap file.

```
SIG Sample = sigNew(0,0,0,0);  
sigLoad( Sample, "SAMPLE.SIG" );  
sigSaveAs( Sample, "SAMPLE.BMP" );  
sigDelete( Sample );
```

**See Also**

sigSave

---

**sigSaveAsBMP**

**BOOL sigSaveAsBMP( SIG aSig, const char \*aFileName )**

Save as Windows Bitmap file.

<b>Parameter</b>	<b>Description</b>
aSig	Signature to be saved.
aFileName	Name of the file to be created/overwritten.

**Description**

Saves the signature image in *aSig* to a Windows Bitmap file named *aFileName*.

**Returns**

Returns TRUE if successful, FALSE otherwise.

**Example**

See sigSaveAs for an example to uses a similar function.

**See Also**

sigSave, sigSaveAsDIB, sigSaveAsWMF, sigSetBMPTyp

---

**sigSaveAsCGM**

**BOOL sigSaveAsCGM( SIG aSig, const char \*aFileName )**

Save as CGM graphics file.

<b>Parameter</b>	<b>Description</b>
aSig	Signature to be saved.
aFileName	Name of the file to be created/overwritten.

**Description**

Saves the signature image in *aSig* to a CGM graphics format file named *aFileName*.

**Returns**

Returns TRUE if successful, FALSE otherwise.

**Example**

See sigSaveAs for an example to uses a similar function.

**See Also**

sigSave, sigSaveAs

---

**sigSaveAsDIB**

**BOOL sigSaveAsDIB( SIG aSig, const char \*aFileName )**

Save as Windows Device Independent Bitmap file.

<b>Parameter</b>	<b>Description</b>
aSig	Signature to be saved.
aFileName	Name of the file to be created/overwritten.

**Description**

Saves the signature image in *aSig* to a Windows Device Independent Bitmap file named *aFileName*.

**Returns**

Returns TRUE if successful, FALSE otherwise.

**Example**

The following function uses creates a Windows DIB file with a fixed 100 dpi resolution.

```
void DoSaveLowRes( SIG aSig, const char *aFileName )
{
    // create a new signature object
    SIG tmp = sigNew( 0,0,0,0 );
    // copy the given signature to the new sig
    sigCopy( tmp, aSig );
    // permanently change the new sig to 100 dpi resolution
    sigCvtDPI( tmp, 90,90 );
    // save the result as a Windows DIB file at 100 dpi
    sigSaveAsDIB( tmp, aFileName );
    // delete the temporary
    sigDelete( tmp );
}
```

**See Also**

sigSave, sigSaveAsBMP, sigSaveAsWMF

---

**sigSaveAsEPS**

**BOOL sigSaveAsEPS( SIG aSig, const char \*aFileName )**

Save as Enhanced PostScript graphic file.

<b>Parameter</b>	<b>Description</b>
aSig	Signature to be saved.
aFileName	Name of the file to be created/overwritten.

**Description**

Saves the signature image in *aSig* to an Enhanced PostScript file named *aFileName*.

**Returns**

Returns TRUE if successful, FALSE otherwise.

**Example**

See sigSaveAs for an example to uses a similar function.

**See Also**

sigSave, sigSaveAs

---

**sigSaveAsMF**

**Win:** **BOOL sigSaveAsMF( SIG aSig, const char \*aFileName )**  
**DOS:** **Not available for DOS**

Save as a Standard Windows Metafile.

<b>Parameter</b>	<b>Description</b>
aSig	Signature to be saved.

aFileName        Name of the file to be created/overwritten.

**Description**

Saves the signature image in *aSig* to a Standard Windows Metafile file named *aFileName*.

**Returns**

Returns TRUE if successful, FALSE otherwise.

**Example**

See sigSaveAs for an example to uses a similar function.

**See Also**

sigSave, sigSaveAsWMF, sigCreateMetaFile

---

**sigSaveAsNLC**

**BOOL sigSaveAsNLC( SIG aSig, const char \*aFileName, BYTE uMode )**

**NOTE: This function is available only in libraries compiled with Microsoft compilers. Libraries compiled with Borland compilers do not support this function.**

Save the SIG file as an NLC formatted compression file.

<b>Parameter</b>	<b>Description</b>
aSig	Signature to be saved.
aFileName	Name of the file to be created/overwritten.
uMode	Mode type for the NLC file. Can be either 3 or 4.

**Description**

Saves the signature image in *aSig* to the NLC compressed file named *aFileName*. The NLC file format is developed by @pos.com.

The *uMode* value is a trade-off between quality of the compressed data versus the compression ratio of the data. A *uMode* value of 3 provides very good quality of the compressed data but suffers from a lower compression ratio as compared with a *uMode* of 4 which provides better compression ratio but results in poorer quality of the compressed data.

**Returns**

Returns TRUE if successful, FALSE otherwise.

**See Also**

sigSave, sigReadNLC

---

**sigSaveAsPCL**

**BOOL sigSaveAsPCL( SIG aSig, const char \*aFileName )**

Save as HP LaserJet PCL printer file.

---

<b>Parameter</b>	<b>Description</b>
aSig	Signature to be saved.
aFileName	Name of the file to be created/overwritten.

**Description**

Saves the signature image in *aSig* to an HP LaserJet PCL printer file named *aFileName*.

**Returns**

Returns TRUE if successful, FALSE otherwise.

**Example**

See sigSaveAs for an example to uses a similar function.

**See Also**

sigSave, sigSaveAs

---

**sigSaveAsPCX**

**BOOL sigSaveAsPCX( SIG aSig, const char \*aFileName )**

Save as ZSoft PCX bitmap file.

<b>Parameter</b>	<b>Description</b>
aSig	Signature to be saved.
aFileName	Name of the file to be created/overwritten.

**Description**

Saves the signature image in *aSig* to a Zsoft PCX bitmap file named *aFileName*.

**Returns**

Returns TRUE if successful, FALSE otherwise.

**Example**

See sigSaveAs for an example to uses a similar function.

**See Also**

sigSave, sigSaveAs

---

**sigSaveAsPLS**

**BOOL sigSaveAsPLS( SIG aSig, const char \*aFileName )**

Save as Epson PLS printer file.

<b>Parameter</b>	<b>Description</b>
aSig	Signature to be saved.
aFileName	Name of the file to be created/overwritten.

**Description**

Saves the signature image in *aSig* to an Epson PLS printer file named *aFileName*.

**Returns**

Returns TRUE if successful, FALSE otherwise.

**Example**

See sigSaveAs for an example to uses a similar function.

**See Also**

sigSave, sigSaveAs

---

**sigSaveAsTIF**

**BOOL sigSaveAsTIF( SIG aSig, const char \*aFileName )**

Save as TIFF bitmap file.

Parameter	Description
aSig	Signature to be saved.
aFileName	Name of the file to be created/overwritten.

**Description**

Saves the signature image in *aSig* to a TIFF bitmap file named *aFileName*.

**Returns**

Returns TRUE if successful, FALSE otherwise.

**Example**

See sigSaveAs for an example to uses a similar function.

**See Also**

sigSave, sigSaveAs

---

**sigSaveAsTXT**

**BOOL sigSaveAsTXT( SIG aSig, const char \*aFileName )**

Save as text file.

Parameter	Description
aSig	Signature to be saved.
aFileName	Name of the file to be created/overwritten.

**Description**

Saves the signature image in *aSig* to a text file named *aFileName*.

**Returns**

Returns TRUE if successful, FALSE otherwise.

---

**Example**

See sigSaveAs for an example to uses a similar function.

**See Also**

sigSave, sigSaveAs

---

**sigSaveAsWMF**

**Win:** **BOOL sigSaveAsWMF( SIG aSig, const char \*aFileName )**

**DOS:** **Not available for DOS**

Save as a Placeable Windows Metafile.

<b>Parameter</b>	<b>Description</b>
aSig	Signature to be saved.
aFileName	Name of the file to be created/overwritten.

**Description**

Saves the signature image in *aSig* to a Placeable Windows Metafile file named *aFileName*.

**Returns**

Returns TRUE if successful, FALSE otherwise.

**Example**

See sigSaveAs for an example to uses a similar function.

**See Also**

sigSave, sigSaveAsMF

---

**sigSaveCmp**

**BOOL sigSaveCmp( SIG aSig, const char \*aFileName )**

Save a compressed signature.

<b>Parameter</b>	<b>Description</b>
aSig	Signature to be saved.
aFileName	Name of the file to be created/overwritten.

**Description**

Saves the signature image in *aSig* to a compressed file named *aFileName*. The image can be reloaded at a later time using sigLoad.

**Returns**

Returns TRUE if successful, FALSE otherwise.

**Example**

See sigSaveAs for an example to uses a similar function.

---

**See Also**

sigLoad, sigSave, sigWriteCmp

---

**sigScale****BOOL sigScale( SIG aSig, int aScale )**

Scale to a percent.

<b>Parameter</b>	<b>Description</b>
aSig	Signature to be scaled.
aScale	A whole number representing the scale.

**Description**

Scale the signature *aSig* to *aScale* percent of its original size. Note that scaling affects rendering and displaying of the signature only and does not alter the actual signature data.

**Returns**

Returns TRUE if successful, FALSE otherwise.

**See Also**

sigSetScale, sigScaleFrac, sigFitSize, sigStretchSize, sigCvtScale

---

**sigScaleFrac****BOOL sigScaleFrac( SIG aSig, int aNum, int aDenom )**

Scale a signature to a given fraction.

<b>Parameter</b>	<b>Description</b>
aSig	Signature to be scaled.
aNum	Numerator portion of fraction.
aDenom	Denominator portion of fraction.

**Description**

Scale the signature *aSig* to a fraction of its original size as defined by *aNum/aDenom*. Note that scaling affects rendering and displaying of the signature only and does not alter the actual signature data.

**Returns**

Returns TRUE if successful, FALSE otherwise.

**See Also**

sigScale, sigSetScale, sigFitSize, sigStretchSize, sigCvtScale

---

**sigScaleToDPI**

**BOOL sigScaleToDPI( SIG aSig, int aHorzDPI, int aVertDPI )**

Scale a signature to a fit a given dpi resolution.

<b>Parameter</b>	<b>Description</b>
aSig	Signature to be scaled.
aHorzDPI	Desired horizontal dots-per-inch resolution.
aVertDPI	Desired vertical dots-per-inch resolution.

**Description**

Scale the signature *aSig* to fit the dot-per-inch resolution as specified by *aHorzDPI* and *aVertDPI*. Note that scaling affects rendering and displaying of the signature only and does not alter the actual signature data.

**Returns**

Returns TRUE if successful, FALSE otherwise.

**See Also**

sigScale, sigSetScale, sigCvtDPI

---

**sigSetBkColor**

**Win:** COLORREF sigSetBkColor( SIG aSig, COLORREF aColor )

**DOS:** int sigSetBkColor( SIG aSig, int aColor )

Gets the background color of a signature.

<b>Parameter</b>	<b>Description</b>
aSig	Signature object.
aColor	Desired background color.

**Description**

Sets the background color of a signature. Using the constant SIG\_NOCOLOR specifies no color.

**Returns**

Returns the previous background color.

**See Also**

sigGetBkColor, sigSetColor

---

**sigSetBMPTyp**

**void sigSetEnhanced( int aBMPTyp )**

Allows specification of which version of the BMP format to use when saving signatures in BMP format.

<b>Parameter</b>	<b>Description</b>
aBMPTyp	BMP format

**Description**

Valid formats are **BMP2X** and **BMP3X** (found in **SIGKIT.H**). **BMP2X** specifies the Windows 2.x BMP format (also known as the OS/2 BMP format). **BMP3X** specifies the Windows 3.x BMP format. The default mode is the Windows 2.x BMP format.

**See Also**

sigSaveAsBmp

---

**sigSetColor**

**Win:** **COLORREF sigSetColor( SIG aSig, COLORREF aColor )**

**DOS:** **int sigSetColor( SIG aSig, int aColor )**

Gets the background color of a signature.

<b>Parameter</b>	<b>Description</b>
aSig	Signature object.
aColor	Desired pen color.

**Description**

Sets the pen color of a signature. Using the constant **SIG\_NOCOLOR** specifies no color.

**Returns**

Returns the previous pen color.

**See Also**

sigGetColor, sigSetBkColor, sigSetPen

---

**sigSetEnhanced**

**void sigSetEnhanced( SIG aSig )**

Mark as “enhanced”.

<b>Parameter</b>	<b>Description</b>
aSig	Signature to be marked.

**Description**

Mark a signature as having been “enhanced”. This flag is used for purposes distinguishing original signature data from data that has been enhanced.

**See Also**

sigIsEnhanced, sigSetReduced

---

**sigSetPage**

**BOOL sigSetPage( SIG aSig, int aWidth, int aHeight, int aHorzDPI, int aVertDPI )**

Set page specifications.

<b>Parameter</b>	<b>Description</b>
aSig	Signature object.
width	New horizontal size in dots.
height	New vertical size in dots.
aHorzDPI	New horizontal dots-per-inch.
aVertDPI	New vertical dots-per-inch.

**Description**

Sets the page specifications of a signature. If the signature is not empty, the image will be converted to the new specification. If an argument is zero, then the corresponding specification is not changed.

**Returns**

Returns TRUE if successful, FALSE otherwise.

**See Also**

sigGetPage, sigNew

---

**sigSetPen****int sigSetPen( SIG aSig, int aPenSize )**

Sets the pen size of a signature.

<b>Parameter</b>	<b>Description</b>
aSig	Signature object.
aPenSize	Size of pen in thousandths of an inch.

**Description**

Sets the size of the pen used to draw signature *aSig* to *aPenSize*. Note that pen sizes are measured in thousandths of an inch. Use the constant SIG\_NOPEN to specify no pen size.

**Returns**

Returns the previous pen size in thousandths of an inch.

**See Also**

sigGetPen, sigSetColor

---

**sigSetReduced****void sigSetReduced( SIG aSig )**

Mark as “reduced”.

<b>Parameter</b>	<b>Description</b>
------------------	--------------------

aSig                      Signature to be marked.

**Description**

Mark a signature as having been “reduced”. This flag is used for purposes distinguishing original signature data from data that has been reduced.

**See Also**

sigIsReduced, sigSetEnhanced

---

**sigSetScale**

**BOOL sigSetScale( SIG aSig, int xNum, int xDenom, int yNum, int yDenom )**

Set the scaling factors for a signature.

<b>Parameter</b>	<b>Description</b>
aSig	Signature to be scaled.
xNum	Horizontal scaling numerator.
xDenom	Horizontal scaling denominator.
yNum	Vertical scaling numerator.
yDenom	Vertical scaling denominator.

**Description**

Sets the scaling factors of the signature *aSig* to  $xNum/xDenom$  horizontally and  $yNum/yDenom$  vertically. Note that scaling affects rendering and displaying of the signature only and does not alter the actual signature data.

**Returns**

Returns TRUE if successful, FALSE otherwise.

**See Also**

sigScale, sigScaleFrac, sigFitSize, sigStretchSize, sigCvtScale

---

**sigShrinkWrap**

**BOOL sigShrinkWrap( SIG aSig )**

Shrinks a signature frame.

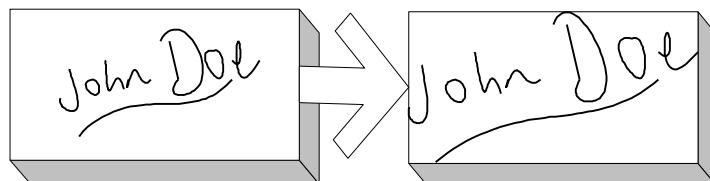
<b>Parameter</b>	<b>Description</b>
aSig	Signature to be shrink-wrapped.

**Description**

Shrinks a signatures frame area to the fit the image.

**Returns**

Returns TRUE if



successful, FALSE otherwise.

**See Also**

sigGetBounds, sigGetPage

---

## sigSmooth

**BOOL sigSmooth ( SIG aSig )**

Smooths a signature image.

Parameter	Description
aSig	Signature to be smoothed.

**Description**

Smooths a signature image. Note that successive smoothing produces varying results. This function alters the original signature image.

**Returns**

Returns TRUE if successful, FALSE otherwise.

**See Also**

sigThinTo, sigThinBy

---

## sigSmoothingFilter

**BOOL sigSmoothingFilter ( SIG aSig, int thinPercent, int avgFactor )**

Smoothens a signature. This one provided more control to the user to control various aspects of smoothing.

Parameter	Description
aSig	Signature to be smoothed.
thinPercent	Percentage value in integer that defined the smoothing factor
avgFactor	Average factor for smoothing

**Description**

The values of thinPercent must be  $0 < \text{thinPercent} < 100$

The value of avgFactor must be  $0 < \text{avgFactor} \leq 10$

Typical value for thinFactor is 50 and avgFactor is 8. The user is encouraged to finetune the smoothing by varying these values. thinFactor value is similar to the value in *sigThinBy* API. avgFactor uses exponential averaging of the current signature point.

**Returns**

Returns TRUE if successful, FALSE otherwise.

**See Also**

sigThinTo, sigThinBy

---

---

## sigStretchSize

**BOOL sigStretchSize( SIG aSig, int aWidth, int aHeight )**

Stretch a signature.

Parameter	Description
aSig	Signature to be stretched.
aWidth	Desired width of the frame area.
aHeight	Desired height of the frame area.

### Description

Stretch the frame size of signature *aSig* to match *aWidth* and *aHeight*. Proper aspect ratio will only be maintained if *aWidth* and *aHeight* are the same proportions as the actual signature frame area.

### Returns

Returns TRUE if successful, FALSE otherwise.

### See Also

sigFitSize, sigSetScale, sigCvtScale

---

## sigThinBy

**unsigned sigThinBy( SIG aSig, unsigned aPercent )**

Reduces the number of points by percent.

Parameter	Description
aSig	Signature to be modified.
aPercent	Percentage of reduction desired.

### Description

Attempts to remove *aPercent* of points from *aSig*. Thinning a signature removes points that are considered the least critical and may result in a significant loss of quality along with reduced memory requirements. Note that it may not be possible to reduce the number of points as desired. In this case, the function removes as many as it can.

### Returns

Returns the actual number of points removed.

### See Also

sigThinTo

---

## sigThinTo

**unsigned sigThinTo( SIG aSig, unsigned aNumPoints )**

Reduces the number of points to the number of points specified.

<b>Parameter</b>	<b>Description</b>
<i>aSig</i>	Signature to be modified.
<i>aNumPoints</i>	Number of points desired.

**Description**

Attempts to reduce the number of points contained in *aSig* to amount specified by *aNumPoints*. Thinning a signature removes points that are considered the least critical and may result in a significant loss of quality along with reduced memory requirements. Note that it may not be possible to reduce the number of points as desired. In this case, the function removes as many as it can.

**Returns**

Returns the actual number of points removed.

**See Also**

*sigThinBy*

---

**sigToHIENGLISH**

**BOOL sigToHIENGLISH( SIG *aSig* )**

Scale a signature to HIENGLISH units.

<b>Parameter</b>	<b>Description</b>
<i>aSig</i>	Signature to be scaled.

**Description**

Scale the signature *aSig* to HIENGLISH units. Each unit represents 0.001 inches. Note that scaling affects rendering and displaying of the signature only and does not alter the actual signature data.

**Returns**

Returns TRUE if successful, FALSE otherwise.

**See Also**

*sigScaleToDPI*, *sigCvtScale*

---

**sigToHIMETRIC**

**BOOL sigToHIMETRIC( SIG *aSig* )**

Scale a signature to HIMETRIC units.

<b>Parameter</b>	<b>Description</b>
<i>aSig</i>	Signature to be scaled.

**Description**

Scale the signature *aSig* to HIMETRIC units. Each unit represents 0.01 millimeters. Note that scaling affects rendering and displaying of the signature only and does not alter the actual signature data.

**Returns**

Returns TRUE if successful, FALSE otherwise.

**See Also**

sigScaleToDPI, sigCvtScale

---

**sigToLOENGLISH****BOOL sigToLOENGLISH( SIG aSig )**

Scale a signature to LOENGLISH units.

Parameter	Description
aSig	Signature to be scaled.

**Description**

Scale the signature *aSig* to LOENGLISH units. Each unit represents 0.01 inches. Note that scaling affects rendering and displaying of the signature only and does not alter the actual signature data.

**Returns**

Returns TRUE if successful, FALSE otherwise.

**See Also**

sigScaleToDPI, sigCvtScale

---

**sigToLOMETRIC****BOOL sigToLOMETRIC( SIG aSig )**

Scale a signature to LOMETRIC units.

Parameter	Description
aSig	Signature to be scaled.

**Description**

Scale the signature *aSig* to LOMETRIC units. Each unit represents 0.1 millimeters. Note that scaling affects rendering and displaying of the signature only and does not alter the actual signature data.

**Returns**

Returns TRUE if successful, FALSE otherwise.

**See Also**

sigScaleToDPI, sigCvtScale

---

## sigUndoPoint

**BOOL sigUndoPoint( SIG aSig )**

Undo last point.

Parameter	Description
aSig	Signature to be modified.

**Description**  
Removes the last point added to a signature.

**Returns**  
Returns TRUE if successful, FALSE otherwise.

**See Also**  
sigUndoStroke, sigEmpty

---

## sigUndoStroke

**BOOL sigUndoStroke( SIG aSig )**

Undo last stroke.

Parameter	Description
aSig	Signature to be modified.

**Description**  
Removes the last complete point added to a signature.

**Returns**  
Returns TRUE if successful, FALSE otherwise.

**See Also**  
sigUndoPoint, sigEmpty

---

## sigVertDPI

**int sigVertDPI( SIG aSig )**

Get vertical dpi resolution.

Parameter	Description
aSig	Signature to be checked.

**Returns**

Returns the vertical dots-per-inch resolution of a signature object.

**See Also**

sigHorzDPI, sigGetPage

---

**sigVoid**

**BOOL sigVoid( SIG aSig )**

Mark a signature as void.

<b>Parameter</b>	<b>Description</b>
aSig	Signature to be marked.

**Description**

Mark a signature as void. This function will permanently embedded an X within the signature image itself.

**Returns**

Returns TRUE if successful, FALSE otherwise.

**See Also**

sigIsVoid

---

**sigWidth**

**int sigWidth( SIG aSig )**

Get width.

<b>Parameter</b>	<b>Description</b>
aSig	Signature to be checked.

**Returns**

Returns the width, in dots, of a signature object's frame area.

**See Also**

sigHeight, sigGetPage

---

**sigWrite**

**BOOL sigWrite( SIG aSig, FILE aFile )**

Write signature data to a file stream. The user should use sigWriteWIN while executing in windows.

---

<b>Parameter</b>	<b>Description</b>
aSig	Signature to be written.
aFile	Handle to a file stream.

**Description**

Write signature data to an output file stream. The file must be open for writing. Writing will occur at the current file position.

**Returns**

Returns TRUE if successful, FALSE otherwise.

**See Also**

sigRead, sigSave, sigReadWIN, sigWriteWIN

---

**sigWriteCmp**

**BOOL sigWriteCmp( SIG aSig, FILE aFile )**

Write compressed signature data to a file stream. The user should use sigWriteCmpWIN function while executing under windows.

<b>Parameter</b>	<b>Description</b>
aSig	Signature to be written.
aFile	Handle to a file stream.

**Description**

Write compressed signature data to an output file stream. The file must be open for writing. Writing will occur at the current file position.

**Returns**

Returns TRUE if successful, FALSE otherwise.

**See Also**

sigRead, sigSaveCmp, sigWriteCmpWIN, sigReadWIN

---

**sigWriteCmpMem**

**WORD sigWriteCmpMem ( SIG aSig, BYTE \*mem, WORD memSize )**

Writes a compressed signature object to a memory object.

<b>Parameter</b>	<b>Description</b>
aSig	Signature object to be written.
mem	Pointer to the memory object to write the signature object to.

**Description**

DO NOT USE THIS COMMAND. This command is obsolete. Please use sigWriteNlcMem instead, which produces far better quality results.

---

Writes a compressed signature data object to a memory object pointed to by the byte pointer **mem**. The parameter **memSize** gives the maximum size of the specified memory object.

**NOTE:** The user is responsible for allocating the memory object before calling this command. Failure to pass the correct parameters for **mem** and **memSize** can result in a memory leak.

**Returns**

Returns the total amount of bytes written to the memory object if successful, returns FALSE otherwise (see sigMemError for a detailed error message).

**See Also**

sigMemError, sigRead, sigReadMem, sigReadWIN, sigSave, sigWrite, sigWriteMem, sigWriteNlcMem

---

## sigWriteCmpWIN

**BOOL sigWriteCmpWIN( SIG aSig, HFILE aFile )**

Write compressed signature data to a file stream. The user should use this function while executing under windows.

<b>Parameter</b>	<b>Description</b>
aSig	Signature to be written.
aFile	Handle to a file stream. This is returned from the Windows OpenFile function call.

**Description**

Write compressed signature data to an output file stream. The file must be open for writing. Writing will occur at the current file position.

**Returns**

Returns TRUE if successful, FALSE otherwise.

**See Also**

sigRead, sigSaveCmp, sigReadWIN

---

## sigWriteNlcMem

**WORD sigWriteNlcMem**  
(  
    **SIG**          **aSig,**  
    **BYTE FAR \***  **mem,**  
    **WORD**       **memSize,**  
    **BYTE**       **uCompMode**  
)

Writes a compressed signature object to a memory object.

<b>Parameter</b>	<b>Description</b>
------------------	--------------------

aSig	Signature object to be written.
mem	Pointer to the memory object to write the signature object to.
memSize	The size of the memory object.

**Description**

Writes an NLC formatted compressed signature data object to a memory object pointed to by the byte pointer **mem**. The parameter **memSize** gives the maximum size of the specified memory object.

**NOTE:** The user is responsible for allocating the memory object before calling this command. Failure to pass the correct parameters for **mem** and **memSize** can result in a memory leak.

**Returns**

Returns the total amount of bytes written to the memory object if successful, returns FALSE otherwise (see sigMemError for a detailed error message).

**See Also**

sigMemError, sigRead, sigReadMem, sigReadWIN, sigSave, sigWrite, sigWriteMem

---

**sigWriteWIN**

**BOOL sigWriteWIN( SIG aSig, HFILE aFile )**

Write signature data to a file stream. This function call should be used while executing in windows.

<b>Parameter</b>	<b>Description</b>
aSig	Signature to be written.
aFile	Handle to a file stream. This is returned from the Windows OpenFile function call.

**Description**

Write signature data to an output file stream. The file must be open for writing. Writing will occur at the current file position.

**Returns**

Returns TRUE if successful, FALSE otherwise.

**See Also**

sigRead, sigSave, sigWrite, sigReadWIN

---

**sigWriteMem**

**WORD sigWriteMem ( SIG aSig, BYTE \*mem, WORD memSize )**

Writes a signature object to a memory object.

<b>Parameter</b>	<b>Description</b>
aSig	Signature object to be written.
mem	Pointer to the memory object to write the signature object to.

**Description**

Writes a signature data object to a memory object pointed to by the byte pointer **mem**. The parameter **memSize** gives the maximum size of the specified memory object.

**NOTE:** The user is responsible for allocating the memory object before calling this command. Failure to pass the correct parameters for **mem** and **memSize** can result in a memory leak.

**Returns**

Returns the total amount of bytes written to the memory object if successful, returns FALSE otherwise (see sigMemError for a detailed error message).

**See Also**

sigMemError, sigRead, sigReadMem, sigReadWIN, sigSave, sigWrite, sigWriteCmpMem

## Appendix A

### Sample Source Code

The sample source code is supplied for Microsoft compilers only. A copy of the sample code can be found on the installation diskettes along with the necessary makefiles. Please bear in mind that these samples are intended for demo purposes only and do not necessarily perform useful tasks. However, they provide a basis on which you can build complex and meaningful Point Of Sale applications.

#### ***SigKit Sample for DOS:***

```
//
//=====
// Test.c
//
// A simple DOS Demo program using DOS Graphics Functions.
//
// Compiler:
//      Microsoft Visual C++ 1.52
//      Target      :      DOS Application
//      Memory Model :      Medium
//      Include     :      ..\..\..\..\padcom\padcom.h
//      Include     :      ..\..\..\..\sigkit.h
//      Library     :      ..\..\..\..\padcom\microsoft\dos\libs\padcomdm.lib
//      Library     :      ..\libs\sigkitdm.lib
//      Environment:
//      DOS 3.3 or better
//
// Copyright (c) 1994-1999, @pos.com. All rights reserved.
//=====
//

#include <graph.h>
#include <stdio.h>
#include <conio.h>

#include "PadCom.h"
#include "SigKit.h"

int GraphOn( void );
void GraphOff( void );

void main()
{
    int x, y, p;
    SIG aSig;

    //=====
    // SWITCH TO GRAPHICS MODE
    //=====
    if( !GraphOn() )
    {
        puts( "Unable to use graphics" );
        return;
    }

    //=====
    // TURN ON THE PAD-INITIALIZE
    //=====
    if( !padOn() )
    {
        GraphOff();
        puts( "Can't find writing pad!" );
        return;
    }

    //=====
    // CREATE SIG OBJECT
```

```
//=====
if( !(aSig = sigNew( padWidth(), padHeight(), padHorzDPI(), padVertDPI() )) )
{
    padOff();
    GraphOff();
    puts( "Unable to create signature!" );
    return;
}

//=====
// START RECRDING IN REAL-TIME
//=====
padRecord();

while( !kbhit() )
{
    // CHECK IF THERE IS MORE DATA
    if( padUpdate() && padGet( &x, &y, &p ) )
    {
        // ADD THIS TO SIG OBJECT
        sigAdd( aSig, x, y, p );

        // DRAW POINT
        sigRenderLast( aSig, 99,99,539,379 );
    }
}

//=====
// RESTORE VIDEO MODE
//=====
GraphOff();

//=====
// SAVE SIGNATURE DATA
//=====
puts( "SAVING SIGNATURE..." );
puts( "SAVING AS INFORITE SIG FILE: TEST.SIG..." );
if( !sigSave( aSig, "TEST.SIG" ) )
    puts( "ERROR SAVING AS SIG FILE!" );
puts( "SAVING AS PCX FILE: TEST.PCX..." );
if( !sigSaveAsPCX( aSig, "TEST.PCX" ) )
    puts( "ERROR SAVING AS PCX FILE!" );
puts( "DONE!" );

//=====
// DESTROY SIG OBJECT
//=====
sigDelete( aSig );

//=====
// TURN THE PAD OFF
//=====
padOff();
}

int GraphOn( void )
{
    char Msg[] = "PLEASE SIGN ON THE PAD, PRESS ANY KEY WHEN DONE";

    if( !_setvideomode( _MAXRESMODE ) )
        return 0;
    _setviewport( 0,0,639,479 );
    _settextposition( 25,15 );
    _outtext( Msg );

    return 1;
}

void GraphOff( void )
{
    _setvideomode( _DEFAULTMODE );
}
```

**SigKit Sample for Win3.x:**

```

//
//-----
// Test.c
//
// A simple WIN16 Demo program
//
// Compiler:
//     Microsoft Visual C++ 1.52
//     Target       :       Windows Application
//     Memory Model :       Medium
//     Include      :       ..\..\..\padcom\padcom.h
//     Include      :       ..\..\..\sigkit.h
//     Library      :       ..\..\..\padcom\microsoft\win16\libs\padcomwm.lib
//     Library      :       ..\..\..\libs\sigkitwm.lib
// Environment:
//     Windows 3.x
//
// Copyright (c) 1994-1999, @pos.com. All rights reserved.
//
//-----
//

#include "windows.h"
#include "PadCom.h"
#include "SigKit.h"

#define MSG(w,s)  MessageBox(w,s,theApp,MB_OK)

int PASCAL WinMain( HINSTANCE, HINSTANCE, LPSTR, int );
LONG __export CALLBACK WndProc( HWND, UINT, WPARAM, LPARAM );
BOOL SaveSig( SIG aSig );

static char      theApp[] = "TESTSIG";
static HINSTANCE theInstance;
static HWND      theWnd;

// STATIC SIGNATURE OBJECT IS USED HERE FOR SIMPLICITY (SEE NOTE ABOVE)
static SIG      theSig;

// -----
// WinMain
// -----
int PASCAL WinMain(
    HINSTANCE hInst, HINSTANCE hInstPrev, LPSTR lpstrCmdLine, int cmdShow
)
{
    MSG msg;
    WNDCLASS wc;

    theInstance = hInst;

    // Register the window class if this is the first instance.
    if( !hInstPrev )
    {
        wc.lpszMenuName      = NULL;
        wc.lpszClassName     = theApp;
        wc.hInstance        = hInst;
        wc.hIcon             = NULL;
        wc.hCursor           = NULL;
        wc.hbrBackground     = (HBRUSH)COLOR_WINDOW + 1;
        wc.style              = 0;
        wc.lpfnWndProc       = WndProc;
        wc.cbClsExtra        = 0;
        wc.cbWndExtra        = 0;

        if( !RegisterClass( &wc ) )
            return 0;
    }

    // Create the main window
    if( !(theWnd = CreateWindowEx(
        WS_EX_TOPMOST, theApp, "SIGN YOUR NAME THEN PRESS ANY KEY",

```

```
        WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, CW_USEDEFAULT, 375, 225,
        NULL, NULL, hInst, NULL )
    )
    return 0;

// Show main window
ShowWindow( theWnd, cmdShow );
UpdateWindow( theWnd );

// Main message loop
while( GetMessage( (LPMSG)&msg, NULL, 0, 0 ) )
{
    TranslateMessage( (LPMSG)&msg );
    DispatchMessage( (LPMSG)&msg );
}

return 0;
}

// -----
// WndProc
// -----
LONG __export CALLBACK WndProc(
    HWND hWnd, UINT Msg, WPARAM wParam, LPARAM lParam
)
{
    switch( Msg )
    {
        case WM_CREATE:
        {
            // Turn on the writing pad and start recording points
            if( !padOn() || !padRecord( hWnd ) )
            {
                MSG( hWnd, "Can't find writing pad" );
                return -1;
            }

            // Create a new and empty signature object
            theSig = sigNew( padWidth(), padHeight(), padHorzDPI(), padVertDPI() );
            if( !theSig )
            {
                MSG( hWnd, "Can't create signature object" );
                return -1;
            }

            break;
        }
        case WM_DESTROY:
        {
            // Always delete the signature object!
            sigDelete( theSig );

            // And things off!
            padOff();

            PostQuitMessage( 0 );

            break;
        }

        case WM_COMMNOTIFY:
        {
            // Process incoming pad data
            RECT Rect;
            HDC hDC;

            // Set the output DC and rectangle
            hDC = GetDC( hWnd );
            GetClientRect( hWnd, &Rect );

            // Check for any new data received
            while( padUpdate() )
            {
                int x, y, p;
            }
        }
    }
}
```

```
        // If pen is down, get point
        if( padGet( &x, &y, &p ) )
        {
            // Add the new point to the signature
            sigAdd( theSig, x, y, p );

            // Draw the last point within the specified rectangle
            sigRenderLast( hDC, theSig, Rect.left, Rect.top, Rect.right,
                          Rect.bottom );
        }
    }

    ReleaseDC( hWnd, hDC );

    return 1;
}

case WM_PAINT:
{
    // Redraw the signature
    PAINTSTRUCT ps;
    RECT Rect;
    HDC hDC;

    hDC = BeginPaint( hWnd, &ps );
    GetClientRect( hWnd, &Rect );

    // Draw the entire signature scaled to fit client rectangle
    sigRender( hDC, theSig, Rect.left, Rect.top, Rect.right, Rect.bottom );

    EndPaint( hWnd, &ps );

    return 1;
}

case WM_SIZE:
{
    // This allows the image to be scaled to fit the new window size
    InvalidateRect( hWnd, NULL, TRUE );
    break;
}

case WM_CHAR:
{
    // Save signature on any key pressed

    MSG( hWnd, "Key pressed, saving data..." );

    // Process and save the signature
    if( !SaveSig( theSig ) )
        MSG( hWnd, "Error saving signature!" );

    // Redraw the window
    InvalidateRect( hWnd, NULL, TRUE );

    break;
}

default:
{
    return DefWindowProc( hWnd, Msg, wParam, lParam );
}
}

return 0;
}

BOOL SaveSig( SIG aSig )
{
    if( sigIsEmpty( aSig ) )
        return FALSE;
    else

```

```
{
    SIG tmp;

    // create a temporary copy of the signature
    tmp = sigDup( aSig );
    if( !tmp )
        return FALSE;
    else
    {
        // now we have the freedom to manipulate the signature
        // as we wish without altering the original...
        // for example, sigShrinkWrap permanently changes the
        // height and width of a signature's "frame" area...
        sigShrinkWrap( tmp );
        sigSave( tmp, "TEST.SIG" );
        sigDelete( tmp );
    }
}

// Clear the signature
sigEmpty( theSig );

return TRUE;
}

/*EOF*/
```

### ***SigKit Sample for Win95/NT:***

```
//
//-----
// Test.c
//
// A simple WIN32 Demo program
//
// Compiler:
//      Microsoft Visual C++ 4.0
//      Target      :      Windows Application
//      Include     :      ..\..\..\padcom\padcom.h
//      Include     :      ..\..\..\sigkit.h
//      Library     :      ..\libs\sigkitw.lib
//      Library     :      ..\..\..\padcom\microsoft\win32\libs\padcomw.lib
//
// Environment:
//      Windows 95/NT
//
// Copyright (c) 1994-1999, @pos.com. All rights reserved.
//
//-----
//

#include "windows.h"
#include "PadCom.h"
#include "SigKit.h"

#define MSG(w,s)  MessageBox(w,s,theApp,MB_OK)

#define WM_PADNOTIFY  WM_USER + 100

int CALLBACK WinMain( HINSTANCE, HINSTANCE, LPSTR, int );
LONG CALLBACK WndProc( HWND, UINT, WPARAM, LPARAM );
BOOL SaveSig( SIG aSig );

static char      theApp[] = "TESTSIG";
static HINSTANCE theInstance;
static HWND      theWnd;

// STATIC SIGNATURE OBJECT IS USED HERE FOR SIMPLICITY
static SIG      theSig;

// -----
// WinMain
```

```
// -----  
int CALLBACK WinMain(  
    HINSTANCE hInst, HINSTANCE hInstPrev, LPSTR lpstrCmdLine, int cmdShow  
)  
{  
    MSG msg;  
    WNDCLASS wc;  
  
    theInstance = hInst;  
  
    // Register the window class if this is the first instance.  
    if( !hInstPrev )  
    {  
        wc.lpszMenuName      = NULL;  
        wc.lpszClassName    = theApp;  
        wc.hInstance        = hInst;  
        wc.hIcon            = NULL;  
        wc.hCursor          = NULL;  
        wc.hbrBackground    = (HBRUSH)(COLOR_WINDOW + 1);  
        wc.style            = 0;  
        wc.lpfnWndProc      = WndProc;  
        wc.cbClsExtra       = 0;  
        wc.cbWndExtra       = 0;  
  
        if( !RegisterClass( &wc ) )  
            return 0;  
    }  
  
    // Create the main window  
    if( !(theWnd = CreateWindowEx(  
        WS_EX_TOPMOST, theApp, "SIGN YOUR NAME THEN PRESS ANY KEY",  
        WS_OVERLAPPEDWINDOW, CW_USEDEFAULT, CW_USEDEFAULT, 375, 225,  
        NULL, NULL, hInst, NULL ) )  
    )  
        return 0;  
  
    // Show main window  
    ShowWindow( theWnd, cmdShow );  
    UpdateWindow( theWnd );  
  
    // Main message loop  
    while( GetMessage( (LPMSG)&msg, NULL, 0, 0 ) )  
    {  
        TranslateMessage( (LPMSG)&msg );  
        DispatchMessage( (LPMSG)&msg );  
    }  
  
    return 0;  
}  
  
// -----  
// WndProc  
// -----  
LONG CALLBACK WndProc(  
    HWND hWnd, UINT Msg, WPARAM wParam, LPARAM lParam  
)  
{  
    switch( Msg )  
    {  
        case WM_CREATE:  
        {  
            // Turn on the writing pad and start recording points  
            if( !padOn() || !padRecord( hWnd, WM_PADNOTIFY, 0,0 ) )  
            {  
                MSG( hWnd, "Can't find writing pad" );  
                return -1;  
            }  
  
            // Create a new and empty signature object  
            theSig = sigNew( padWidth(), padHeight(), padHorzDPI(), padVertDPI() );  
            if( !theSig )  
            {  
                MSG( hWnd, "Can't create signature object" );  
                return -1;  
            }  
        }  
    }  
}
```

```
        break;
    }
    case WM_DESTROY:
    {
        // Always delete the signature object!
        sigDelete( theSig );

        // And things off!
        padOff();

        PostQuitMessage( 0 );

        break;
    }

    case WM_PADNOTIFY:
    {
        // Process incoming pad data
        RECT Rect;
        HDC hDC;

        // Set the output DC and rectangle
        hDC = GetDC( hWnd );
        GetClientRect( hWnd, &Rect );

        // Check for any new data received
        while( padUpdate() )
        {
            int x, y, p;

            // If pen is down, get point
            if( padGet( &x, &y, &p ) )
            {

                // Add the new point to the signature
                sigAdd( theSig, x, y, p );

                // Draw the last point within the specified rectangle
                sigRenderLast( hDC, theSig, Rect.left, Rect.top, Rect.right,
                               Rect.bottom );
            }
        }

        ReleaseDC( hWnd, hDC );

        return 1;
    }

    case WM_PAINT:
    {
        // Redraw the signature
        PAINTSTRUCT ps;
        RECT Rect;
        HDC hDC;

        hDC = BeginPaint( hWnd, &ps );
        GetClientRect( hWnd, &Rect );

        // Draw the entire signature scaled to fit client rectangle
        sigRender( hDC, theSig, Rect.left, Rect.top, Rect.right, Rect.bottom );

        EndPaint( hWnd, &ps );

        return 1;
    }

    case WM_SIZE:
    {
        // This allows the image to be scaled to fit the new window size
        InvalidateRect( hWnd, NULL, TRUE );
        break;
    }
}
```

```
case WM_CHAR:
{
    // Save signature on any key pressed

    MSG( hWnd, "Key pressed, saving data..." );

    // Process and save the signature
    if( !SaveSig( theSig ) )
        MSG( hWnd, "Error saving signature!" );

    // Redraw the window
    InvalidateRect( hWnd, NULL, TRUE );

    break;
}

default:
{
    return DefWindowProc( hWnd, Msg, wParam, lParam );
}
}

return 0;
}

BOOL SaveSig( SIG aSig )
{
    if( sigIsEmpty( aSig ) )
        return FALSE;
    else
    {
        SIG tmp;

        // create a temporary copy of the signature
        tmp = sigDup( aSig );
        if( !tmp )
            return FALSE;
        else
        {
            // now we have the freedom to manipulate the signature
            // as we wish without altering the original...
            // for example, sigShrinkWrap permanently changes the
            // height and width of a signature's "frame" area...
            sigShrinkWrap( tmp );
            sigSave( tmp, "TEST.SIG" );
            sigDelete( tmp );
        }
    }

    // Clear the signature
    sigEmpty( theSig );

    return TRUE;
}

/*EOF*/
```