HAND HELD PRODUCTS

a Welch Allyn affiliate

PadCom

Point-of-Sale Terminal Control Library

User's Guide

Disclaimer

Welch Allyn® Data Collection, Inc. (d/b/a Hand Held Products) reserves the right to make changes in specifications and other information contained in this document without prior notice, and the reader should in all cases consult Hand Held Products to determine whether any such changes have been made. The information in this publication does not represent a commitment on the part of Hand Held Products.

Hand Held Products shall not be liable for technical or editorial errors or omissions contained herein; nor for incidental or consequential damages resulting from the furnishing, performance, or use of this material.

Web Address: www.handheld.com

PenWare100 and PenWare1500 are registered trademarks of @pos.com.

Microsoft Visual C/C++, MS-DOS, Windows 95, Windows 98, Windows 2000, and Windows NT are either registered trademarks or trademarks of Microsoft Corporation in the United States and/or other countries.

Other product names mentioned in this document may be trademarks or registered trademarks of other companies and are the property of their respective owners.

Table of Contents

1 Introduction and Installation

Introducing Hand Held Products PadCom Library

PadCom library by Hand Held Products is a tool that allows developers to interface to Transaction Team units easily and quickly. Developers can use PadCom library to capture signatures, read MSR, perform PINpad transactions, display text or bitmaps, and write innovative graphical interfaces. By leveraging on Hand Held Products' 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 PadCom Library

Hand Held Products' PadCom library includes the following key features:

- Full interface to all Transaction Team devices.
- 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.
- Interrupt/message driven engine for quick response to hardware activities.
- Reliable implementation of RS-232 serial communications protocol.
- Automatic detection of connected Transaction Team unit and automatic configuration.

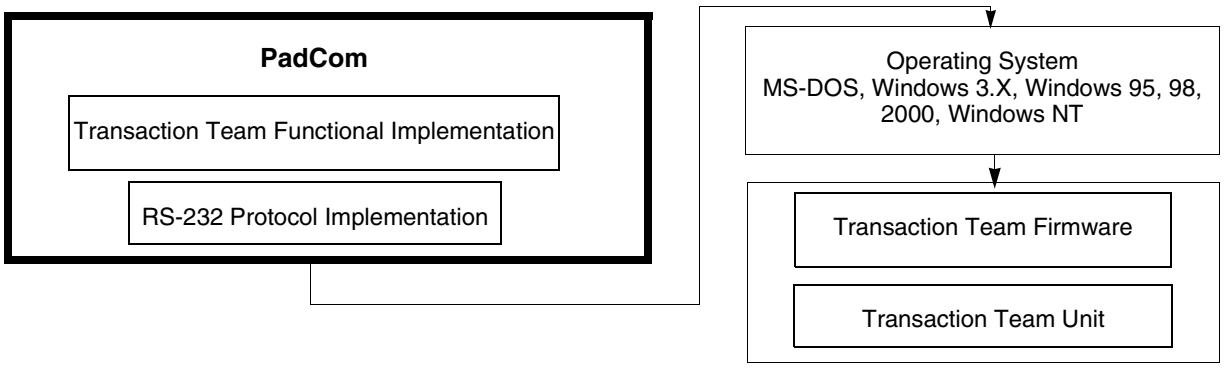
In addition, PadCom library has been designed to work seamlessly with Transaction Team SigKit[1], the signature processing library for a complete signature capture solution.

Hand Held Products Transaction Team Terminals Supported by PadCom

PadCom contains full support for the Transaction Team 1500 and the Transaction Team 3100 Series.

PadCom Interface

The following diagram represents the PadCom interface to Transaction Team units.



The diagram shows how PadCom communicates with the Transaction Team unit. The functional implementation of a Transaction Team device is implemented on top of the RS-232 protocol implementation. PadCom then communicates to the device over the supported operating systems platforms, which include MS-DOS®, Windows® 3.x, Windows® 95, Windows® 98, Windows® 2000, and Windows NT®. Although it is possible to use other RS-232 implementations, Hand Held Products recommends that you use the built-in driver. You do not have any direct access to internal firmware in the Transaction Team units.

1. *For more information about the SigKit library and other development tools, please contact your Hand Held Products sales representative.*

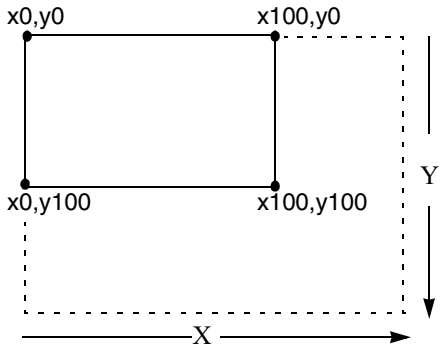
Installation

Hand Held Products' PadCom Library comes bundled as a set of static libraries with operating system dependent (MS-DOS®, WIN16, and WIN32®) and compiler dependent (Microsoft® and Borland®) solutions available for your use. PadCom is part of the Software Development Kit (SDK) which is available in 16-bit or 32-bit versions, and is included on the Transaction Team 1500 and Transaction Team 3100 Series software suite CDs. The SDK installation program is a simple to use InstallShield® application. It allows you to selectively install the desired components and sample programs. Please refer to the Startup Guide you received with your hardware for instructions on software installation. For further information on other components of the SDK, see the SDK Roadmap document which is available upon installation of the SDK.

Transaction Team Device Overview

The device's pad surface is composed of a large number of individual points or "pixels," as are video monitors, printers and other devices. Because the library is designed to maintain the integrity of the input points, it does NOT distort the image by adjusting for squareness. Therefore, when using points obtained, it is important to understand both the mapping of the pad surface and the concept of aspect ratios. In addition, the methods used to represent pen strokes must be considered.

The points on the pad are specified by using a pair of horizontal and vertical coordinates (referred to as X and Y coordinates, respectively). The origin of the coordinates is at the top/left corner and increases in a positive direction toward the bottom/right. Therefore, the coordinate pair "0,0" is at top/left corner; "0,100" is 100 points down, "100,0" is 100 points to the right; and "100,100" is both 100 points to the right and 100 points down. The actual coordinates of the bottom/right corner can be represented as **padWidth()-1, padHeight()-1**, or obtained by using the **padGetArea** function (page 3-10).



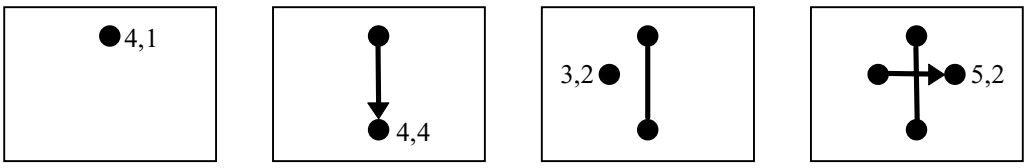
Aspect Ratio Considerations

Since the points returned by the library represent the actual points on the physical pad device, it is necessary to understand the concept of "aspect ratios." Basically, an aspect ratio refers to the difference between the width and height of the physical points. For example, a 1:1 ratio means that the width and height of a point are the same and that the point is perfectly square. Similarly, an aspect ratio of 2:1 means that a point is twice as wide as it is high. This is important because a shape that appears square when created at, for example, a 2:1 aspect ratio, will appear vertically stretched when viewed at an aspect ratio of 1:1. The aspect ratio of the pad surface is represented by the ratio between horizontal and vertical "dots-per-inch" and can be obtained by using the functions **padHorzDPI** (page 3-23), and **padVertDPI** (page 3-66), respectively. To maintain proper aspect ratios when drawing the points on the screen or other device, the library provides functions such as **padScaleDPI** (page 3-47) to facilitate aspect ratio adjustment.

Representing Pen Strokes

While a single point consists of a single horizontal and vertical coordinate, a pen stroke consists of all 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 in contact with the pad. Remaining points, referred to as "lineto" points, indicate that the pen is being dragged across the pad surface. Consequently, the **padGet** function (page 3-9) retrieves a "moveto/lineto" pen indicator as well as the corresponding coordinates.

Illustration of two pen strokes:



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

Using PadCom

PadCom provides an interface that can be fully customized to Transaction Team pads. An operating system-dependent set of libraries are provided which use the most popular compilers to support a wide variety of development needs. For example, for the Windows 3.x operating system, a separate set of three libraries is compiled using Borland C++ and Microsoft® Visual C++® compilers. In addition to this, various memory models and thread support is provided wherever applicable.

PadCom consists of statically linked libraries. This means that the application development needs to be carried out in the following stages:

Implementation

Before proceeding with the design specifications, it is recommended that you study the functionality supported by the chosen Transaction Team device. A complete list of functions exposed by PadCom is provided in the *Library Reference* beginning on page 2-1.

See *Sample Source Code* beginning on page 5-1 for an example of how to use the PadCom library and the exposed API functions. The same sample source code is also provided as a part of the sample application code installed with the SDK components. 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 (POS) applications.

Compiling and Linking

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 MS-DOS®

padcomds.lib padcomdm.lib and padcomdl.lib are available to support the small, medium and the large memory models.

For Microsoft® Windows® 3.x

padcomws.lib padcomwm.lib and padcomwl.lib are available to support the small, medium and the large memory models.

For Microsoft® Windows® 95/98/2000/NT

padcomw.lib and padcomwt.lib[1] are available.

Select the supported library and add it to the project. Add the common header file padcom.h or else the program will not compile correctly. Transaction Team libraries are not differentiated based on the type of compiler used. For example, padcomds.lib is available for both the Borland C++ and Microsoft® Visual C++® compilers. It is your responsibility to use the correct library. The automatic installation program puts these libraries in clearly marked folders.

Make sure the compiler settings comply with the memory model (or the thread option wherever applicable). Failure to do so can result in unpredictable program failures.

Troubleshooting

Application development always involves debugging your application code. Hand Held Products has put together a highly competent Technical Support Team to help you troubleshoot your applications quickly. However, there is some simple troubleshooting that you can do on your end before calling Technical Support at Hand Held Products.

Hand Held Products strongly recommends that you always run the sample test programs you installed while installing the library components. Call Hand Held Products 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, re-check the compiler settings, used libraries, and the sample code. Before you call Hand Held Products Technical Support, note 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 quickly diagnose and/or make recommendations.

Help Desk

If you need assistance installing or troubleshooting your software, please call your Distributor or the nearest Hand Held Products technical support office:

North America:

Telephone: (315) 685-2476 (8 a.m. to 6 p.m. EST)
Fax number: (315) 685-4960
E-mail: *support@handheld.com*

Europe:
Telephone-
European Ofc: Int+31 40 242 4486
U.K. Ofc: Int+44 1925 240055
E-mail: *support@handheld.com*

Asia:
Telephone: Int+852-2511-3050 *or* 2511-3132
E-mail: *support@handheld.com*

1. PadComwt.lib is available for the Microsoft compiler only.

2 Library Reference

Library Functions List by Transaction Team Pad Compatibility

A blank in the model column indicates that the function is not supported for that model.

Function	TT1500	TT3100 Series
padBinaryGetTable	X	X
padBinaryGetVar	X	X
padBox	X	X
padClear	X	X
padClearPixel		X
padComDate	X	X
padComVersion	X	X
padConnect	X	X
padConnectClearScreen	X	X
padDisplayObject		X
padDisplayTime		X
padEchoComm		X
padEraseTable	X	X
padFieldButton		X
padFieldSignature	X	X
padFlush	X	X
padFormDeleteFld		X
padFormSaveFld		X
padFrame	X	X
padGet	X	X
padGetAllMagCardTracks		X
padGetBaudRate	X	X
padGetArea	X	X
padGetBkColor		X
padGetCmdSetID		X
padGetColor		X
padGetColors		X
padGetConnectTimeout	X	X
padGetDefaultBaudRate		X
padGetDUKPTbinaryPIN		X
padGetDUKPTtextPIN		X
padGetFont	X	X
padGetFontSize		X
padGetInTimeout	X	X
padGetMagTrack		X
padGetMasterSessionBinaryPIN		X

Function	TT1500	TT3100 Series
padGetMasterSessionTextPIN		X
padGetMaxCardTracks		X
padGetMaxCardTrackSize		X
padGetModel	X	X
padGetNumVar	X	X
padGetOutTimeout	X	X
padGetPage	X	X
padGetPort	X	X
padGetPortAddr	X	X
padGetPortIrq	X	X
padGetPorts	X	X
padGetScanRate		X
padGetTableItem	X	X
padGetVersion	X	X
padHeight	X	X
padHideTime		X
padHorzDPI	X	X
padInkExport	X	X
padInvert		X
padIsKey		X
padIsLcd	X	X
padIsNewStroke	X	X
padIsaReset	X	X
padIsOn	X	X
padIsPenDown	X	X
padIsRecord	X	X
padLcdHeight	X	X
padLcdHorzDPI	X	X
padLcdVertDPI	X	X
padLcdWidth	X	X
padLightOff	X	
padLightOn	X	
padLine	X	X
padMemClear		X
padMemDelete		X
padMemDeleteVar	X	X
padMemFind		X
padMemGetChecksum		X
padMemGetFree		X
padMemGetVar	X	X
padMemLoadBitmap		X

Function	TT1500	TT3100 Series
padMemLoadBitmapFile		X
padMemLoadText		X
padMemReset		X
padMemSetVar	X	X
padName	X	X
padNewX	X	X
padNewY	X	X
padOff	X	X
padOldX	X	X
padOldY	X	X
padOn	X	X
padPassThroughHandshaking	X	X
padPassThroughOff		X
padPassThroughOn		X
padPassThroughResetCodes		X
padPassThroughSetOffCode		X
padPassThroughSetOnCode		X
padPortReclaim	X	X
padPortRelease	X	X
padPromptHexNumber		X
padPromptNum		X
padPromptNumber		X
padPromptReset		X
padPromptSignature		X
padPromptString		X
padPromptTimeout		X
padPutBits		X
padPutBmpFile		X
padPutLogo		X
padPutText	X	X
padReadByte	X	X
padRecord	X	X
padReset		X
padResetArea	X	X
padResetBaudRate		X
padResetConnectTimeout	X	X
padResetDefaultBaudRate		X
padResetInTimeout	X	X
padResetMagCard		X
padResetOutTimeout	X	X
padScale	X	X

Function	TT1500	TT3100 Series
padScaleDPI	X	X
padScaleTo	X	X
padScaleX	X	X
padScaleY	X	X
padSendByte	X	X
padSetArea	X	X
padSetAutoInking		X
padSetBaudRate		X
padSetBkColor		X
padSetColor		X
padSetCompress		X
padSetConnectTimeout	X	X
padSetDebug		X
padSetDefaultBaudRate		X
padSetFlowControl	X	
padSetFont		X
padSetInTimeout	X	X
padSetInkingArea		X
padSetLcdClearTimeout	X	
padSetLogo		X
padSetLogoBmpFile		X
padSetNumVar	X	X
padSetOutTimeout	X	X
padSetPadOffset	X	X
padSetPixel		X
padSetPort	X	X
padSetPortAddr	X	X
padSetPortHandle	X	X
padSetPortIrq	X	X
padSetPorts	X	X
padSetScanRate		X
padSetTime		X
padSetType	X	X
padStop	X	X
padSoundBell		X
padSoundEnable		X
padSoundSetFreq		X
padSoundTone		X
padToHIENGLISH	X	X
padToLOENGLISH	X	X
padToHIMETRIC	X	X

Function	TT1500	TT3100 Series
padToLOMETRIC	*X*	*X*
padType	*X*	*X*
padUpdate	*X*	*X*
padVertDPI	*X*	*X*
padWidth	*X*	*X*

Library Functions List by Category

The following lists the functions in groups of distinct functional categories. For a detailed description of the functions listed, refer to "Library Functions" on page 3-1.

Basic Operations:

padConnect	Attempts a connection to a pad (similar to padOn).
padConnectClearScreen	Connect and clear pad screen.
padGet	Retrieve the pen coordinate and status.
padIsaReset	Reset the extension card device
padOff	Turn the pad off.
padOn	Turn the pad on (similar to padConnect).
padRecord	Start recording pad data.
padStop	Stop recording pad data.
padUpdate	Update pad data.
padSetType	Sets a specific type of pad to be used.
padReset	Reset the pad.

Clipping Active Area:

padGetArea	Get the currently active clipping area.
padResetArea	Reset clipping to the full pad surface.
padSetArea	Sets the currently active clipping area.
padSetInkingArea	Sets the currently active inking area.

Data Collection Operations:

padBinaryGetTable	Get data from a table row.
padEraseTable	Erase all data in a binary table
padGetTableItem	Transfer data from a table row to a variable
padIsKey	Check if the DUPKT key is set.
padSetCompress	Set the compression number for storing the captured signature data.

Error Handling:

padError	Get the most recent error status.
padFlush	Flush any data waiting to be processed.
padIsError	Check if an error has occurred.
padIsPadError	Check if an error has occurred.
padGetPadError	Get the error code.
padSetDebug	Set the debug mode On or Off.

LCD Screen Operations:

padBox	Draws a box onto the LCD screen.
padClear	Clears the LCD screen.
padClearPixel	Clears a pixel on the LCD screen.
padFrame	Draws a frame onto the LCD screen.
padGetBkColor	Gets the current background color.
padDisplayObject	Draw a stored memory object.
padGetColor	Gets the current foreground color.
padGetColors	Gets the number of colors available.
padInvert	Inverts an area on the LCD screen.
padLine	Draws a line on the LCD screen.
padPutText	Draws text on the LCD screen.
padPutBits	Draws a bitmap on the LCD screen.
padPutBmpFile	Draws a Windows BMP file on the LCD screen.
padPutLogoh	Draws the logo image on the LCD screen.
padSetBkColor	Sets the current background color.
padSetColor	Sets the current foreground color.
padSetLcdClearTimeout	Sets the automatic clearing time out for the LCD.
padSetLogo	Sets the logo image.
padSetLogoBmpFile	Sets the logo image to a Windows BMP file.
padSetPixel	Sets a pixel on the LCD screen.
padSetFont	Set the current text font.
padGetFont	Get the current text font.
padSetAutoInking	Set the Auto-inking mode.
padGetFontSize	Get the dimensions of a font size.

LCD Display Specifications:

padIsLcd	Check if an LCD screen is available.
padLcdHeight	Get height of LCD screen area in pixels.

padLcdHorzDPI	Get horizontal resolution of the LCD screen.
padLcdVertDPI	Get vertical resolution of the LCD screen.
padLcdWidth	Get width of LCD screen area in pixels.

Memory Operations:

padMemAvailable	Check if memory is available.
padMemReset	Reset memory.
padMemGetFree	Get amount of free memory.
padMemGetChecksum	Perform a checksum of memory.
padMemClear	Clear memory content.
padMemDelete	Delete a memory item.
padMemFind	Find an item in memory.
padMemLoadText	Load a string into memory.
padMemLoadBitmap	Load bitmap into memory.
padMemLoadBitmapFile	Loads a bitmap file into memory.

Miscellaneous:

padLightOn	Turn on the light.
padLightOff	Turn off the light.

Model and Version Operations:

padComDate	Get the build date of the PadCom library
padComVersion	Get the version number of the PadCom library
padGetCmdSetID	Get the current command set ID.
padGetModel	Get the model number of the attached pad
padGetVersion	Get the model revision number

Pad Specifications:

padGetPage	Get pad dimension information.
padHeight	Get height of pad area in pixels.
padHorzDPI	Get horizontal resolution of the pad.
padVertDPI	Get vertical resolution of the pad.
padWidth	Get width of pad area in pixels.

Port Configuration for DOS:

padGetPortAddr	Gets a COM port's address.
padGetPortIrq	Gets a COM port's interrupt number.
padSetPortAddr	Sets a COM port's address.

padSetPortIrq	Sets a COM port's interrupt number.

Port Operations:

padGetPort	Gets the current communications port.
padGetPorts	Gets the current number of ports available.
padPassThroughHandshaking	Enables/disables hardware handshaking (pass through).
padSetPort	Sets the preferred communications port.
padSetPortHandle	Sets the port handle.
padSetPorts	Sets the amount of ports available on the PC.

POS Services:

padGetMagTrack	Gets a track of data from a magnetic card.
padGetMaxCardTracks	Get the amount of tracks supported.
padGetMaxCardTrackSize	Get the maximum card track size in bytes.
padGetAllMagCardTracks	Read all tracks on a magnetic card.
padGetDUKPTbinaryPIN	Gets a PIN number from the user.
padGetDUKPTtextPIN	Gets a PIN number from the user.
padGetMasterSessionBinaryPIN	Gets a PIN number from the user.
padGetMasterSessionTextPIN	Gets a PIN number from the user.
padResetMagCard	Resets the magnetic card reader and clears the track buffer

Prompt Operations:

padPromptReset	Reset all prompts.
padPromptHexNumber	Display a hexadecimal number entry prompt.
padPromptNum	Display an integer number entry prompt.
padPromptNumber	Display a decimal number entry prompt.
padPromptSignatrue	Display a signature capture prompt.
padPromptString	Display an alphanumeric data entry prompt.
padPromptTimeout	Sets the timeout value for the prompt commands.

Scaling of Coordinates Received:

padScale	Scale an arbitrary value to a given fraction.
padScaleDPI	Scale to a desired DPI resolution.
padScaleTo	Scale to a desired rectangle.
padScaleX	Scale a horizontal coordinate to given DPI.
padScaleY	Scale a vertical coordinate to given DPI.
padToHIENGLISH	Scale to units based on 1000ths of inch.
padToLOENGLISH	Scale to units based on 100ths of inch.

padToHIMETRIC	Scale to units based on 100ths of a millimeter.
padToLOMETRIC	Scale to units based on 10ths of a millimeter.

Scan Rate Operations:

padSetConstant	Set a constant scan rate On or Off.
padGetConstant	Check if constant scan rate is On or Off.
padSetScanRate	Set a scan rate.
padGetScanRate	Get the current scan rate.

Sound Operations:

padSoundBell	Type of sound action to make.
padSoundEnable	Enable or disable the sound.
padSoundSetFreq	The frequency for the sound.
padSoundTone	Sound with a frequency and duration.

State Information:

padIsOn	Check if pad has been turned on.
padIsNewStroke	Check if beginning a new stroke.
padIsPenDown	Check if the pen is down.
padIsRecord	Check if currently recording pad data.
padOldX	Returns the previous horizontal pen coordinate.
padOldY	Returns the previous vertical pen coordinate.
padNewX	Returns the current horizontal pen.
padNewY	Returns the current vertical pen coordinate.
padName	Get the name of the attached pad.
padType	Get the type of the attached pad.

System and Communication Operations:

padSysIsAvailable	Check if the system is available.
padSysReset	Reset the system.
padCommIsAvailable	Check if the communication port available.
padCommReset	Reset the communications port.
padCanConfigComm	Check if the communications can be configured.
padConfigComm	Configure commincations as specified.
padTestComm	Test communications.
padSetCommTimeout	Set time-out for communication operation.
padEchoComm	Echo data back to the host.
padGetBaudRate	Returns the current baud rate.

padGetConnectTimeout	Gets the initial connection intput/output timeout value.
padGetDefaultBaudRate	Returns the current default baud rate.
padGetInTimeout	Gets the timeout value for input.
padGetOutTimeout	Gets the timeout value for output.
padPassThroughOff	Turns off passthorugh mode
padPassThroughOn	Turns on passthorugh mode
padPassThroughResetCodes	Resets the passthrough codes
padPassThroughSetOffCode	Sets the passthrough “ON” code
padPassThroughSetOnCode	Sets the passthrough “OFF” code
padReadByte	Reads a single byte of from the pad.
padResetBaudRate	Resets the baud rate to the default (9600).
padResetConnectTimeout	Resets the initial connection intput/output timeout value.
padResetDefaultBaudRate	Resets the default baud rate to PadCom’s initial value (9600).
padResetInTimeout	Resets the timeout value for input.
padResetOutTimeout	Resets the timeout value for output.
padSendByte	Sends a single byte to the pad.
padSetBaudRate	Sets the baud rate to a specified rate.
padSetConnectTimeout	Sets the initial connection intput/output timeout value.
padSetDefaultBaudRate	Sets the default baud rate to open a connection at.
padSetFlowControl	Enables/disables flow control.
padSetInTimeout	Sets the timeout value for input.
padSetOutTimeout	Sets the timeout value for output.
padSetPadOffset	Set’s the pad’s touch surface offset.

Time and Date Operations:

padTimeAvailable	Check if the time function is available.
padDisplayTime	Displays the time.
padHideTime	Hide the time display.
padSetTime	Set the current time.
padGetTime	Get the current time.

3 Library Functions

This section provides detailed information about each library function, arranged in alphabetic order. "Library Reference" on page 2-1 presents a brief overview of these functions based on functional category.

padBinaryGetTable

Gets data from a table row.

Syntax

```
WORD padBinaryGetTable(
char FAR * lpszDatabaseName,
BYTE FAR * lpuBuffer,
WORD wBufSize,
WORD wIndex
)
```

Parameter	Description
lpszDatabaseName	Pointer to a buffer containing raw signature data
lpuBuffer	Pointer to the buffer
wBufSize	Size of the buffer
wIndex	The index of the row transferred

Returns

Receive a row data from the binary table.

padBinaryGetVar

Retrieves binary data from the specified memory location.

Syntax

```
BOOL padBinaryGetVar(
char FAR * pcName,
WORD wNameLength,
char FAR * pcBinData,
WORD FAR * pwDataLength
)
```

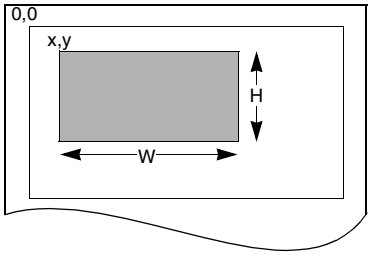
Parameter	Description
pcName	Variable name containing binary data
wNameLength	Name length
pcBinData	Pointer to a buffer to get binary data
pwDataLength	Buffer length

Returns

Returns TRUE if successful, FALSE otherwise.

padBox

Draws a solid box on the LCD screen starting at the coordinates specified by **X** and **Y** using the size specified by **Width** and **Height**. The box is drawn using the current foreground color. The illustration depicts the placement of an arbitrary box at location **x,y** with width of **W** and height **H.**



Syntax

BOOL padBox(
int X,
int Y,
int Width,
int Height
)

Parameter	Description
X	Horizontal coordinate to draw the box from
Y	Vertical coordinate to draw the box from
Width	Horizontal size of the box in pixels
Height	Vertical size of the box in pixels

Returns

Returns TRUE if successful, FALSE otherwise.

See Also

padClearPixel, padFrame, padInvert, padLine, padSetPixel

padClear

Clears the entire LCD screen if one is attached.

Syntax

void padClear(
void
)

See Also

padIsLcd, padPutText, padPutBits, padPutLogo

padClearPixel

Clears a pixel on the LCD screen by setting a pixel on the LCD screen at the location specified by **X** and **Y** to the current background color.

Syntax

BOOL padClearPixel(
int X, int Y
)

Parameter	Description
X	Horizontal coordinate of the pixel to set
Y	Vertical coordinate of the pixel to set

Returns

Returns TRUE if successful, FALSE otherwise.

See Also

padBox, padFrame, padInvert, padLine, padSetPixel

PadComDate

Returns a pointer to a string containing the date of the build of the PadCom library. The format of the returned string is in ASCII text format including the month, day, and year of the build. For example, “July 16, 1998.”

Syntax

char * PadComDate()

See Also

PadComVersion

PadComVersion

Returns a pointer to a string containing the version number of the build of the PadCom library. The format of the returned string is in ASCII text format and includes the major version, minor version, and revision number of the build. For example, “04.00.0008.”

Syntax

char * PadComVersion()

See Also

PadComDate

padConnect

Connects to the attached Transaction Team unit by checking for the existence of a pad and initializing communications. Normally this command searches all valid com ports for the existence of all supported Transaction Team pad types. This command must be executed before using any other commands in the library; the only exceptions to this are the commands **padSetPort**, **padSetPortAddr**, **padSetPortIrq**, and **padSetType**, which modify the behavior of **padConnect** and must be called prior to calling **padConnect**. This function is similar to **padOn** except that it will always attempt to connect to a pad even if a connection was already established.

Syntax

BOOL padConnect (
void
)

Returns

Returns TRUE if successful, FALSE otherwise.

See Also

padConnectClearScreen, padOn, padOff, padRecord, padSetPort, padSetPortAddr, padSetPortIrq, padSetType, padSetBaudRate, padGetBaudRate, padSetDefaultBaudRate, padGetDefaultBaudRate, padResetDefaultBaudRate, padResetBaudRate

padConnectClearScreen

padConnectClearScreen allows you to specify whether or not the screen will be cleared upon connection. The default is padConnectClearScreen set to TRUE. When set to TRUE, the LCD is cleared whenever a connection is made to the Transaction Team POS device using padConnect or padOn. If set to FALSE, the LCD is not cleared when a connection is made to a Transaction Team POS device. This command affects all subsequent calls to padOn and padConnect and does not itself make a connection to the Transaction Team POS device. To make a connection you must call padConnect or padOn.

Syntax

BOOL padConnectClearScreen (
BOOL enable
)

Parameter	Description
enable	Flag to enable/disable screen clear on Connect.

Returns

Returns TRUE if successful, FALSE otherwise.

See Also

padOn, padOff, padConnect

padDisplayObject

Draws a memory object onto the position specified by pptAt on the screen, if applicable. An object can be stored in non-volatile memory using **padMemLoadText**, **padMemLoadBitmap** or similar commands. Every memory object is associated with an **ObjectId** that identifies the object. The created memory object may be deleted or overwritten.

Non-volatile memory is typically used to store large and often used bitmaps and texts. Since the data resides on the unit, the data transfer between the COM-port and the unit is reduced resulting in much faster display times.

*Note: All memory objects may be displayed, even empty ones. To check if a specified memory object contains data, use **padMemFind**.*

Syntax

BOOL padDisplayObject(
WORD ObjectId,
padPOINT *pptAt
)

Parameter	Description
ObjectId	ID of the memory item to store
pptAt	Pointer to padPOINT specifying screen location

Returns

Returns TRUE if the function succeeds, FALSE otherwise.

See Also

padMemLoadText, padMemLoadBitmap, padMemDelete

padDisplayTime

This function displays the time on a Transaction Team 3100 Series or compatible terminal.

Syntax

BOOL padDisplayTime(
WORD wHorz,
WORD wVert,
BYTE nFontId
)

Parameter	Description
wHorz	The horizontal position for displaying the time
wVert	The vertical position for displaying the time
nFontId	The font used for displaying the time

Returns

Returns TRUE if the function succeeds, FALSE otherwise.

See Also

padHideTime, padSetTime, padGetTime, padSetFont

padEchoComm

Echoes data back to the host (the PC). The user sends a block of data specified by **SendDataBlock** (with the length of the data specified by **DataLength**) to the pad and the pad sends the data back to the host which is then placed in **RecvDataBlock**. This command can be used to verify the communication link between the host and the pad.

This command can optionally be used to synchronize the software with the state of the unit by waiting for the echo to return before continuing.

Syntax

```
int padEchoComm(
char *SendDataBlock,
char *RecvDataBlock,
WORD DataLength
)
```

Parameter	Description
SendDataBlock	Pointer to the block of data to send
RecvDataBlock	Pointer to the received data block
DataLength	Length of the sent data bytes

Returns

Returns TRUE if the function call succeeds, FALSE otherwise.

padEraseTable

Erases all data in a binary table.

Syntax

```
BOOL padEraseTable(
char FAR * lpszDatabaseName
)
```

Parameter	Description
lpszDatabaseName	Name of the database variable that stores the transactions

Returns

Returns TRUE if the function call succeeds, FALSE otherwise.

padFieldButton

Creates a button and displays it on the pad screen. Commands can be executed when the button is pressed and released.

Syntax

```
BOOL padFieldButton(
WORD wId,
WORD wStyle,
WORD wLeft,
WORD wTop,
WORD wWidth,
WORD wHeight,
BYTE uFont,
char FAR *pcText,
WORD wTextLen,
WORD wCmdPress,
char FAR * pcPressArgData,
WORD wPressArgDataLen,
WORD wCmdRelease,
char FAR * pcReleaseArgData,
WORD wReleaseArgDataLen
)
```

Parameter	Description
wId	Field identification number
wStyle	Field style attributes; 0 for defaults
wLeft	X-coordinate of upper point
wTop	Y-coordinate of upper point
wWidth	Width of the signature field
wHeight	Height of the signature field
uFont	Font to be used
pcText	Button text to be displayed
wTextLen	Text length
wCmdPress	Command to be executed when the button is pressed
pcPressArgData	Command data for press command
wPressArgDataLen	Data length for press command
wCmdRelease	Command to be executed when the button is released
pcReleaseArgData	Command data for release command
wReleaseArgDataLen	Data length for release command

Returns

Returns TRUE if the function call succeeds, FALSE otherwise.

padFieldSignature

Creates a signature field.

Syntax

```
BOOL padFieldSignature (
WORD wId,
WORD wStyle,
WORD wLeft,
WORD wTop,
WORD wWidth,
WORD wHeight,
WORD wMaxPoints,
WORD wEnterTime,
WORD wEnterCmd,
char FAR * pcData,
int nDataLen
)
```

Parameter	Description
wId	Field identification number
wStyle	Field style attributes. 0 for defaults
wLeft	X-coordinate of upper point
wTop	Y-coordinate of upper point
wWidth	Width of the signature field
wHeight	Height of the signature field
wMaxPoints	Max points that are allowed to be entered to the signature field
wEnterTime	Penup time out in seconds, which is used for AUTO ENTER
wEnterCmd	The command to be executed when ENTER button is pressed
pcData	Command data buffer
nDataLen	Length of the command data

Returns

Returns TRUE if the function call succeeds, FALSE otherwise.

padFlush

Removes any pending pad data from the input queue. This is not normally needed, however it may be useful in some situations.

Syntax

```
void padFlush(
void
)
```

padFormDeleteFld

Deletes the field given by wId.

Syntax

```
BOOL padFormDeleteFld(
WORD wId
)
```

Parameter	Description
wId	Field identification number

See Also

padFormSaveFld

padFormSaveFld

Saves the current signature data to the specified variable.

Syntax

```
BOOL padFormSaveFld(
WORD wId,
char FAR * pcName,
WORD wNameLen
)
```

Parameter	Description
wId	Field identification number
pcName	Variable name
wNameLen	Name length

See Also

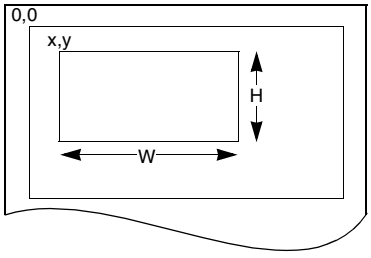
padBox, padClearPixel, padInvert, padLine, padSetPixel

padFrame

Draws a frame on the LCD screen.

Draws a frame (outline/empty box) on the LCD screen starting at the coordinates specified by **X** and **Y** using the size specified by **Width** and **Height**. The frame is drawn using the current foreground color. This function draws an empty frame as opposed to **padBox**, which draws a filled frame.

The illustration depicts the placement of an arbitrary frame at location **X, Y** with width of **W** and height **H.**



Syntax

```
BOOL padFrame(
int X,
int Y,
int Width,
int Height
)
```

Parameter	Description
X	Horizontal coordinate from which the frame is drawn
Y	Vertical coordinate from which the frame is drawn
Width	Horizontal size of the frame in pixels
Height	Vertical size of the frame in pixels

Returns

Returns TRUE if successful, FALSE otherwise.

See Also

padBox, padClearPixel, padInvert, padLine, padSetPixel

padGet

Receives new data from the pad by checking for and returning the last horizontal/vertical coordinates and pen status. A pen status of 1 indicates the continuation of a line (lineto(x,y)), whereas a 0 represents the beginning of a stroke (moveto(x,y)). NULL pointers may be passed as any of the parameters to suppress gathering of the associated data. Note that if no pen activity occurred since the last padUpdate, the function returns FALSE and does not alter the values pointed to.

Syntax

```
BOOL padGet (
int *XPosition,
int *YPosition,
int *Pen
)
```

Parameter	Description
XPosition	Pointer to an integer to hold the new horizontal coordinate of the pen
YPosition	Pointer to an integer to hold the new vertical coordinate of the pen
Pen	Pointer to an integer to hold the new pen status (0 = moveto, 1 = lineto)

Returns

Returns TRUE if new pen data was received, FALSE otherwise.

See Also

padUpdate, padRecord, padStop

padGetAllMagCardTracks

Returns data captured from all tracks on a magnetic card, if available. A card must have been swiped prior to executing this command, otherwise it returns zero. The TrackXSize parameters should contain the maximum buffer sizes for each track before the call, and after will contain actual track sizes after the call. The card reader must be reset using **padResetMagCard** before calling this command. Refer to page 5-1 for sample code that describes how to read the MSR.

Syntax

```
BOOL padGetAllMagCardTracks(
BYTE *Tracks,
char *Track1,
WORD *Track1Size,
char *Track2,
WORD *Track2Size,
char *Track3,
WORD *Track3Size
)
```

Parameter	Description
Tracks	Total tracks
Track1	Pointer to the track1
Track1Size	Pointer to the track1 size
Track2	Pointer to the track2
Track2Size	Pointer to the track2 size
Track3	Pointer to the track3
Track3Size	Pointer to the track3 size

Returns

Returns TRUE if the function call succeeds, FALSE otherwise.

See Also

padGetMaxCardTrackSize, padGetMaxCardTracks, padResetMagCard, padGetMagTrack

padGetArea

Finds the minimum and maximum coordinates of the active pad area by retrieving the current clipping area into the variables provided. NULL pointers may be passed as any or all parameters.

Syntax

```
BOOL padGetArea(
int *xLeft,
int *yTop,
int *xRight,
int *yBottom
)
```

Parameter	Description
xLeft	Pointer to an integer to hold the minimum horizontal coordinate
yTop	Pointer to an integer to hold the minimum vertical coordinate
xRight	Pointer to an integer to hold the maximum horizontal coordinate
yBottom	Pointer to an integer to hold the maximum vertical coordinate

Returns

Returns TRUE if area retrieved represents the full pad surface, FALSE otherwise.

See Also

padSetArea, padWidth, padHeight

padGetBaudRate

Used to find the current communications baud rate.

Syntax

```
DWORD padGetBaudRate(
void
)
```

Returns

Returns the current baud rate.

See Also

padResetBaudRate, padSetBaudRate

padGetBkColor

Gets the current background color.

Syntax

```
int padGetBkColor(
void
)
```

Returns

Returns the current background.

See Also

padGetColor, padSetBkColor, padSetColor

padGetCmdSetID

Gets the command set ID number. The command set ID number for a standard Transaction Team 3100 Series is “1234.” The command set ID number is a code that identifies the type of command set used and is not to be confused with the version of the command set. This command is for future purposes only.

Syntax

```
WORD padGetCmdSetID(
void
)
```

Returns

Returns the ID of the command set upon success, FALSE otherwise.

See Also

padGetVersion

padGetColor

Gets the current foreground color.

Syntax

```
int padGetColor(
void
)
```

Returns

Returns the current foreground color.

See Also

padGetBkColor, padGetColors, padSetBkColor, padSetColor

padGetColors

Gets the number of colors available.

Syntax

```
int padGetColors(
void
)
```

Returns

Returns the number of colors available.

See Also

padGetBkColor, padGetColor, padGetColors, padSetBkColor, padSetColor

padGetConnectTimeout

This function returns the time-out value used for the com port (in milliseconds) during the initial connection phase.

Syntax

```
unsigned int padGetConnectTimeout(
void
)
```

See Also

padSetInTimout, padResetInTimeout, padGetOutTimeout, padResetOutTimeout, padSetConnectTimeout, padResetConnectTimeout

padGetDefaultBaudRate

Gets the current default baud rate. See "padSetDefaultBaudRate" on page 3-52 for more information.

Syntax

```
DWORD padGetDefaultBaudRate(
void
)
```

Returns

Returns the current default baud rate.

See Also

padSetBaudRate, padGetBaudRate, padResetBaudRate, padSetDefaultBaudRate, padResetDefaultBaudRate

padGetDUKPTbinaryPIN

Displays a PIN entry prompt for collecting VISA standard encrypted pin numbers. This function is only for Transaction Team 3100 Series pads that support VISA standard DUKPT PIN entry prompts. A key must be injected into the unit before this function can be used. Please contact your hardware supplier or Hand Held Products for information on secure key injection.

Syntax

```
int padGetDUKPTbinaryPIN(
char FAR *pTitle,
char FAR *pAcctNum,
char FAR *pKeySerialNumber,
int *pKeySerialNumberLength,
char FAR *pBinaryPIN,
int iTimeout
)
```

Parameter	Description
pTitle	Specifies a string to display as the title of the PIN entry screen
pAcctNum	User's account number used to generate the encrypted PIN number
pKeySerialNumber	Generated key serial number used to decode the encrypted PIN number
pKeySerialNumberLength	Length of the generated key serial number
pBinaryPIN	The resulting binary PIN number returned as a byte array, not an actual string
iTimeout	Specifies the maximum number of seconds this function attempts to obtain PIN data from the PIN entry prompt. If the specified number of seconds elapse without user input, the PIN entry prompt is canceled and the display is cleared.

Returns

0 - Success
1 - Cancel
2 - Bad Parameters
3 - Pad Not On
4 - Command Error
5 - Response Error
6 - No PIN
7 - Not Supported

See Also

padGetDUKPTtextPIN, padGetMasterSessionBinaryPIN, padGetMasterSessionTextPIN

padGetDUKPTtextPIN

Displays a PIN entry prompt for collecting VISA standard encrypted pin numbers. This function is only for Transaction Team 3100 Series pads that support standard DUKPT PIN entry prompts. One DUKPT key must be injected into the unit before this function can be used. Please contact your hardware supplier or Hand Held Products for information on secure key injection.

Syntax

```
int padGetDUKPTtextPIN(
char FAR *pTitle,
char FAR *pAcctNum,
char FAR *pTextPIN,
int iTimeout
)
```

Parameter	Description
pTitle	Specifies a string to display as the title of the PIN entry screen
pAcctNum	The user's account number used to generate the encrypted PIN number
pTextPIN	The resulting standard ASCII/VISA standard type "71" text PIN block which includes both the encrypted PIN and the key serial number used to create the encrypted PIN.
iTimeout	Specifies the maximum number of seconds this function attempts to obtain PIN data from the PIN entry prompt. If the specified number of seconds elapse without user input, then the PIN entry prompt is canceled and the display is cleared.

Returns

0 - Success
1 - Cancel
2 - Bad Parameters
3 - Pad Not On
4 - Command Error
5 - Response Error
6 - No PIN
7 - Not Supported

See Also

padGetDUKPTbinaryPIN, padGetMasterSessionBinaryPIN, padGetMasterSessionTextPIN

padGetFont

Gets the current text font.

Syntax

int padGetFont(
void
)

Returns

Returns the current font number. The FontIDs correspond to the sizes specified in the table below:

Font ID	Horizontal Size	Vertical Size	Available for Transaction Team 3100 Series
0	8	8	YES
1	16	16	YES
2	6	8	YES
3	8	12	YES
4	12	16	YES
5	16	24	YES

See Also

padSetFont, padPutText

padGetFontSize

This command returns the horizontal and vertical size of the character dimensions used in the specified font. For example, on a Transaction Team 3100 Series, in font number 0 each character is 8 pixels wide and 8 pixels high. Calling this command with the Id of 0 will return 8x8 (in pptPixels).

Syntax

int padGetFontSize(
BYTE Id,
padPOINT *pptPixels
)

Parameter	Description
Id	ID of the specified font
pptPixels	Pointer to store the obtained font size

The following table describes the relationship between font ID and the font sizes:

Font ID	Horizontal Size	Vertical Size	Available for Transaction Team 3100 Series
0	8	8	YES
1	16	16	YES
2	6	8	YES
3	8	12	YES
4	12	16	YES
5	16	24	YES

Returns

Returns TRUE if function call succeeds and FALSE otherwise

See Also

padSetFont, padGetFont

padGetInTimeout

This function returns the current time-out value used for the com port's input (in milliseconds) from the connected pad. This is not related to the time-out used during the initial connection phase (see "padSetConnectTimeout" on page 3-52).

Syntax

```
unsigned int padGetInTimeout(
void
)
```

See Also

padSetInTimout, padResetInTimeout, padGetOutTimeout, padSetOutTimeout, padResetOutTimeout, padSetConnectTimeout, padGetConnectTimeout, padResetConnectTimeout

padGetMagTrack

This function retrieves a track from a magnetic card. It provides low level access to the magnetic card reader, if attached. When reading track 1, 2, or 3, the function returns the number of bytes actually read and the Buf parameter contains the data read, if any.

Before reading a track, you must reset the magnetic card reader. To reset the magnetic card reader you can call **padResetMagCard**, or you can call this function using 0 for all of the parameters. The return value is non-zero if a reader is attached, otherwise it is zero. Please refer to **padResetMagCard** (see page 3-46) for more information. Refer to page 5-1 for sample code that describes how to read the MSR.

Syntax

```
unsigned padGetMagTrack(
int Track,
char *Buf,
unsigned Size
)
```

Parameter	Description
Track	The number of the track to be read
Buf	Pointer to a character buffer to receive the track data
Size	Maximum number of characters to read into Buf

Returns

When Track is 0, returns non-zero if a reader is attached, zero otherwise. Other Track values return the number of bytes read from the desired track, if any.

See Also

padResetMagCard**,** padGetMaxCardTrackSize, padGetMaxCardTracks, padGetAllMagCardTracks padGetDUKPTbinaryPIN, padGetDUKPTtextPIN, padGetMasterSessionBinaryPIN, padGetMasterSessionTextPIN

padGetMasterSessionBinaryPIN

Displays a PIN entry prompt for collecting VISA standard encrypted pin numbers. This function is only for pads such as the TT 3100 Series that support standard Master/Session PIN entry prompts. 10 master keys must be injected into the unit before this function can be used. Please contact your hardware supplier or Hand Held Products for information on secure key injection.

Syntax

```
int padGetMasterSessionBinaryPIN(
char FAR *pTitle,
char FAR *pAcctNum,
char FAR *pSessionKey,
int pMasterKeyID,
char FAR *pBinaryPIN,
int iTimeout
)
```

Parameter	Description
pTitle	Specifies a string to display as the title of the PIN entry screen
pAcctNum	The user's account number used to generate the encrypted PIN number
pSessionKey	The session key used to generate the encrypted PIN number
pMasterKeyID	Used to identify which of the 10 (0-9) injected master keys to use to generate the encrypted PIN number
pBinaryPIN	Is the resulting binary PIN number returned as a byte array, not an actual string
iTimeout	Specifies the maximum number of seconds this function attempts to obtain PIN data from the PIN entry prompt. If the specified number of seconds elapse without user input, then the PIN entry prompt is canceled and the display is cleared.

Returns

0 - Success
1 - Cancel
2 - Bad Parameters
3 - Pad Not On
4 - Command Error
5 - Response Error
6 - No PIN
7 - Not Supported

See Also

padGetDUKPTbinaryPIN, padGetDUKPTtextPIN, padGetMasterSessionTextPIN

padGetMasterSessionTextPIN

Displays a PIN entry prompt for collecting VISA standard encrypted pin numbers. This function is only for pads such as the Transaction Team 3100 Series that support standard Master/Session PIN entry prompts. 10 master keys must be injected into the unit before this function can be used. Contact your hardware supplier or Hand Held Products for information on secure key injection.

Syntax

```
int padGetMasterSessionTextPIN(
char FAR *pTitle,
char FAR *pAcctNum,
char FAR *pSessionKey,
int pMasterKeyID,
char FAR *pTextPIN,
int iTimeout
)
```

Parameter	Description
pTitle	Specifies a string to display as the title of the PIN entry screen
pAcctNum	The user's account number used to generate the encrypted PIN number
pSessionKey	The session key used to generate the encrypted PIN number
pMasterKeyID	Used to identify which of the 10 (0-9) injected master keys to use to generate the encrypted PIN number.
pTextPIN	The resulting standard ASCII/VISA standard type "71" text PIN block.
iTimeout	Specifies the maximum number of seconds this function attempts to obtain PIN data from the PIN entry prompt. If the specified number of seconds elapse without user input, the PIN entry prompt is canceled and the display is cleared.

Returns

0 - Success
1 - Cancel
2 - Bad Parametersv
3 - Pad Not On
4 - Command Error
5 - Response Error
6 - No PIN
7 - Not Supported

See Also

padGetDUKPTbinaryPIN, padGetDUKPTtextPIN, padGetMasterSessionBinaryPIN

padGetMaxCardTracks

Retrieves the number of tracks supported by the magnetic card reader in Tracks. Refer to page 5-1 for sample code that describes how to read the MSR.

Syntax

```
BOOL padGetMaxCardTracks(
BYTE *Tracks
)
```

Parameter	Description
Tracks	Pointer to a BYTE data type that returns the number of supported tracks

Returns

Returns TRUE upon success, FALSE otherwise.

See Also

padGetMaxCardTrackSize, padGetAllMagCardTracks

padGetMaxCardTrackSize

Retrieves the maximum readable card track size supported by the magnetic card reader. Refer to page 5-1 for sample code that describes how to read the MSR.

Syntax

```
BOOL padGetMaxCardTrackSize(
WORD *TrackSize
)
```

Parameter	Description
TrackSize	Pointer to a WORD data type that returns the maximum readable track size

Returns

Returns TRUE upon success, FALSE otherwise.

See Also

padGetMaxCardTracks, padGetAllMagCardTracks

padGetModel

Retrieves the model ID. The ID is returned as a 16-bit integer in the WORD pointer Model. For a Transaction Team 3100 Series, the model ID returned is 3100, and for a Transaction Team 1500, the model ID returned is 1500.

Syntax

```
BOOL padGetModel(
WORD *Model
)
```

Parameter	Description
Model	Pointer to a WORD data type that returns the model ID

Returns

Returns TRUE upon success, FALSE otherwise.

See Also

padGetVersion

padGetNumVar

Gets the contents of a variable in num format

Syntax

```
BOOL padGetNumVar(
char FAR *lpszVarName,
WORD FAR *pwValue
)
```

Parameter	Description
lpszVarName	Name of the variable to which the numeric value is assigned
pwValue	Value

Returns

Returns TRUE upon success, FALSE otherwise.

padGetOutTimeout

This function returns the current time-out value used for the com port's output (in milliseconds) to the connected pad. This is not related to the time-out used during the initial connection phase (see "padSetConnectTimeout" on page 3-52).

Syntax

```
unsigned int padGetOutTimeout(
void
)
```

See Also

padSetOutTimout, padResetInTimeout, padGetOutTimeout, padSetInTimeout, padResetOutTimeout, padSetConnectTimeout, padGetConnectTimeout, padResetConnectTimeout

padGetPage

Retrieves the size and resolution of the Transaction Team POS device's touch sensitive surface. This function retrieves the total number of touch points available on the Transaction Team POS device's touch surface area in the locations pointed to by **Width** and **Height**. It retrieves the dots per inch resolution of the Transaction Team POS device's touch surface area in the locations pointed to by **HorzDPI** and **VertDPI**. NULL pointers may be passed for any parameter to suppress gathering the associated data. The coordinates retrieved are not display coordinates, they are touch surface coordinates. In general, touch surface coordinates are much higher in resolution than display coordinates. For example, the Transaction Team 3100 Series' touch surface contains 4096x4096 touch points while the TT3100 Series' display contains only 320x240 pixels, even though they are about the same size in inches.

Syntax

```
void padGetPage(
int *Width,
int *Height,
int *HorzDPI,
int *VertDPI
)
```

Parameter	Description
Width	Pointer to hold the total amount of horizontal touch sensitive points
Height	Pointer to hold the total amount of vertical touch sensitive points
HorzDPI	Pointer to hold the horizontal dots-per-inch of the touch sensitive surface
VertDPI	Pointer to hold the vertical dots-per-inch of the touch sensitive surface

See Also

padHeight, padWidth, padHorzDPI, padVertDPI

padGetPort

Returns the active port. If the pad is in "off" state, the port number returned represents the "default port" as mentioned in **padSetPort** (see page 3-59). If the pad is "on" (after a successful call to **padOn** or **padConnect** functions), the port number is the actual port in use.

Syntax

```
int padGetPort(
void
)
```

Returns

Returns an integer representing one of the following ports:
0 for none, 1 for COM1, 2 for COM2, 3 for COM3, or 4 for COM4.

See Also

padSetPort, padSetPorts, padGetPorts

padGetPortAddr

Returns the address of a port.

Note: This function is available in the MS-DOS version only!

Syntax

MS-DOS: int padGetPortAddr(
int Port
)

Parameter	Description
Port	1 = COM1 2 = COM2 3 = COM3 4 = COM4

Returns

Returns the address of the specified communications port.

See Also

padSetPortAddr, padGetPortIrq

padGetPortIrq

Returns the interrupt number associated with a port.

Note: This function is available in the MS-DOS version only!

Syntax

MS-DOS: int padGetPortIrq(
int Port
)

Parameter	Description
Port	Should be 1 for COM1, 2 for COM2, 3 for COM3 or 4 for COM4

Returns

Returns the interrupt number used by the specified communications port.

See Also

padSetPortIrq, padGetPortAddr

padGetPorts

Returns the current number of available ports. Use **padSetPorts** (see page 3-60) to change the number of ports available.

Syntax

int padGetPorts(
void
)

Returns

The return values are 1 for 1 port, 2 for 2 ports, etc.

See Also

padSetPort, padSetPorts

padGetScanRate

Gets the current scan rate. Scan rate is used to vary the number of points captured per second.

Syntax

```
int padGetScanRate(
void
)
```

Returns

Returns the current scan rate if succeeds, FALSE if function fails

See Also

padSetScanRate

padGetTableItem

Transfers data from a table row to a variable.

Syntax

```
WORD padGetTableItem(
char FAR * lpszDatabaseName,
BYTE FAR * lpuBuffer,
WORD wBufSize,
WORD wIndex
)
```

Parameter	Description
lpszDatabaseName	Name of the database variable that stores the transaction
lpuBuffer	Pointer to the buffer
wBufSize	Size of the buffer
wIndex	ID of the index into the table used as a binary

Returns

Total bytes read.

padGetTime

This function gets the current time on the Transaction Team 3100 Series or compatible POS terminal.

Syntax

```
BOOL padGetTime(
BYTE FAR * pHour,
BYTE FAR * pMin,
BYTE FAR * pSec
)
```

Parameter	Description
pHour	a pointer to a byte for receving the current hour
pMin	a pointer to a byte for receving the current minute
pSec	a pointer to a byte for receving the current second

Returns

Returns TRUE upon success, FALSE otherwise.

See Also

padDisplayTime, padSetTime, padHideTime

padGetVersion

Retrieves the model revision number of the firmware. The revision number is returned in the following format:

HIBYTE Major revision number as binary value

LOBYTE Minor revision number as binary value

This command allows a user to check the version number of the firmware (ROM chips in the pad). For example, version 1.0 is returned as 0100h, 2.15 is returned as 0215h, 3.1 is returned as 0310h. This command is useful if you are writing a program that makes use of commands not found on earlier versions of the firmware. With this command your program can check the version number of the firmware in the pad and only run if it's equal to or greater than the version number required.

Syntax

```
BOOL padGetVersion(
WORD *Version
)
```

Parameter	Description
Version	Pointer to WORD data type to store the result

Returns

Returns TRUE upon success, FALSE otherwise.

See Also

padGetModel

padHeight

Syntax

```
int padHeight(
void
)
```

Returns

Returns the total number of vertical points on the pad surface (i.e., 1024).

See Also

padWidth, padVertDPI

padHideTime

Stops the time from being displayed on a Transaction Team 3100 Series or compatible terminal.

Syntax

```
BOOL padHideTime(
void
)
```

Returns

Returns TRUE upon success, FALSE otherwise.

See Also

PadDisplayTime, padGetTime, padSetTime

padHorzDPI

Syntax

```
int padHorzDPI(
void
)
```

Returns

Returns the number of horizontal points per inch.

See Also

padWidth, padVertDPI

padInkExport

Converts the raw signature data into given format.

01 - INK_POINTS

02 - INK_TOKEN

05 - INK_PACKET

07 - INK_COTF

08 – INK_NOLOSS

Syntax

```
BOOL padInkExport(
char FAR * pcName,
WORD wNameLen,
WORD wFormat,
char FAR * pcFmtData,
WORD FAR * pwFmtLen
)
```

Parameter	Description
pcName	Variable name containing raw signature data
wNameLen	Name length
wFormat	Compression format
pcFmtData	Pointer to buffer to receive compressed data
pcFmtLen	Buffer length

Returns

Returns TRUE if successful, FALSE otherwise.

See Also

padFieldsignature,padFormSaveFld

padInvert

Inverts an area on the LCD screen by turning all black pixels white and all white pixels black. The area to be inverted is defined by the coordinates specified by **X** and **Y**, using the size specified by **Width** and **Height**.

Syntax

```
BOOL padInvert(
int X,
int Y,
int Width,
int Height
)
```

Parameter	Description
X	Horizontal coordinate from which to start inverting
Y	Vertical coordinate from which to start inverting
Width	Horizontal size in pixels of the area to invert
Height	Vertical size in pixels of the area to invert

Returns

Returns TRUE if successful, FALSE otherwise.

See Also

padBox, padClearPixel, padFrame, padLine, padSetPixel

padIsaReset

Checks to determine if an extension card device is attached .

Syntax

```
BOOL padIsaReset (
WORD FAR * status
)
```

Parameter	Description
status	The address to contain the ISA status

Returns

Returns TRUE if successful, FALSE otherwise.

See Also

padWidth, padVertDPI

padIsKey

The functions checks if the DUKPT key has been set. The DUKPT key is used to secure the current transactions.

Syntax

```
BOOL padIsKey(
BYTE *IsKey
)
```

Parameter	Description
IsKey	Pointer to a BYTE type data that returns TRUE if DUKPT is set, FALSE if it is not.

Returns

Returns TRUE upon success, FALSE otherwise.

padIsLcd

Syntax

```
BOOL padIsLcd(
void
)
```

Returns

Returns TRUE if an LCD screen is available, FALSE otherwise.

See Also

padClear, padPutText, padPutBits

padIsNewStroke

Checks if the last X,Y coordinate received is the start of a new pen stroke. Note that the status is only updated when padUpdate returns TRUE to indicate new data received.

Syntax

```
BOOL padIsNewStroke(
void
)
```

Returns

Returns TRUE if the last point represents a new stroke, FALSE otherwise.

See Also

padUpdate, padRecord, padGet, padNewX, padNewY

padIsOn

```
BOOL padIsOn(
void
)
```

Returns

Returns TRUE if the pad is on, FALSE otherwise.

See Also

padOn, padOff, padIsRecord

padIsPenDown

This function checks if the pen is down or in contact with the pad. Note that the status is only updated when padUpdate returns TRUE to indicate new data received.

Syntax

```
BOOL padIsPenDown(
void
)
```

Returns

Returns TRUE if the pen is down, FALSE otherwise.

See Also

padUpdate, padRecord, padGet, padIsNewStroke

padIsRecord

Syntax

```
BOOL padIsRecord(
void
)
```

Returns

Returns TRUE if the pad is currently sending data, FALSE otherwise.

See Also

padRecord, padStop, padIsOn

padLcdHeight

Syntax

```
int padLcdHeight(
void
)
```

Returns

Returns the total number of vertical points on the LCD.

See Also

padIsLcd, padLcdWidth, padLcdHorzDPI, padLcdVertDPI

padLcdHorzDPI

Syntax

```
int padLcdHorzDPI(
void
)
```

Returns

Returns the number of horizontal points per inch of the LCD.

See Also

padIsLcd, padLcdWidth, padLcdVertDPI

padLcdVertDPI

Syntax

int padLcdVertDPI(
void
)

Returns

Returns the number of vertical points per inch of the LCD.

See Also

padIsLcd, padLcdHeight, padLcdHorzDPI

padLcdWidth

Syntax

int padLcdWidth(
void
)

Returns

Returns the total number of horizontal points on the LCD.

See Also

padIsLcd, padLcdHeight, padLcdHorzDPI

padLightOff

Sends commands to the pad to turn off the red light.

Syntax

void padLightOff(
void
)

See Also

padLightOn

padLightOn

Sends commands to the pad to turn on the red light.

Syntax

void padLightOn (
void
)

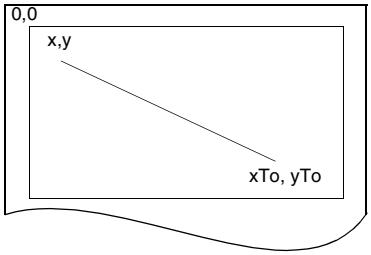
Turns on the light.

See Also

padLightOff

padLine

Draws a straight line on the screen starting at the location specified by **X** and **Y**, and ending at the location specified by **Xto** and **Yto**. The line is drawn using the current foreground color. The illustration depicts an arbitrary line drawn from a starting point **x,y** to **xTo,yTo.**



Syntax

BOOL pad Line(
int X,
int Y,
int Xto,
int Yto
)

Parameter	Description
X	Horizontal coordinate from which to draw the line
Y	Vertical coordinate from which to draw the line
Xto	Horizontal coordinate to draw the line to
Yto	Vertical coordinate to draw the line to

Returns

Returns TRUE if successful, FALSE otherwise.

See Also

padBox, padClearPixel, padFrame, padInvert, padSetPixel

padMemClear

Clears the entire contents of the Transaction Team POS device's on-board non-volatile memory. Non-volatile memory is typically used to store large and often used bitmaps and text objects. Since the data resides on the unit, the data transfer between the COM-port and the unit is reduced, resulting in much faster display times.

Syntax

BOOL padMemClear(
void
)

Returns

Returns TRUE if the non-volatile memory is cleared, FALSE otherwise.

See Also

padMemGetFree, padMemReset, padMemDelete, padMemLoadText, padMemLoadBitmap, padMemFind, padMemGetChecksum

padMemDelete

Deletes the specified item from the non-volatile memory if it exists.

Syntax

BOOL padMemDelete(
WORD Id
)

Parameter	Description
Id	ID of the memory item to be deleted

Note: All memory objects may be deleted, even empty ones. To check if a specified memory object contains data, use ***padMemFind*** *(see page 3-29).*

Returns

Returns TRUE if the function call succeeds, FALSE otherwise.

See Also

padMemGetFree, padMemReset, padMemClear, padMemLoadText, padMemLoadBitmap, padMemFind, padMemGetChecksum

padMemDeleteVar

Syntax

BOOL padMemDeleteVar(
char FAR *lpszVarName
)

Parameter	Description
lpszVarName	Name of the memory variable

Returns

Returns TRUE if the function call succeeds, FALSE otherwise.

padMemFind

This command allows you to check if a memory item, specified by **Id**, is stored in the unit's memory. Non-volatile memory is typically used to store large and often used bitmaps and texts. Since the data resides on the unit, the data transfer between the COM-port and the unit is reduced, resulting in much faster display times. The items are referred to in memory based on their IDs.

Syntax

BOOL padMemFind(
WORD Id,
BYTE *Stored
)

Parameter	Description
ID	ID of the stored memory item
Stored	Pointer to a BYTE that returns TRUE if the memory item is found, FALSE otherwise

Returns

Returns TRUE if the function succeeds, FALSE otherwise.

See Also

padMemReset, padMemClear, padMemDelete, padMemLoadText, padMemLoadBitmap, padMemFind, padMemGetChecksum.

padMemGetChecksum

Checksum is a pointer to a WORD data type that stores the checksum of all stored items. This is used to verify that memory contents have not been changed since the last call to **padMemGetChecksum**.

Syntax

```
BOOL padMemGetChecksum(
WORD *Checksum
)
```

Returns

Returns TRUE if the function succeeds, FALSE otherwise.

See Also

padMemReset, padMemGetFree, padMemClear, padMemDelete, padMemLoadText, padMemLoadBitmap, padMemFind,

padMemGetFree

FreeBytes is a pointer to WORD type data that returns the number of free non-volatile memory bytes available in the unit. Non-volatile memory is typically used to store large and often used bitmaps and texts. Since the data resides on the unit, the data transfer between the COM-port and the unit is reduced, resulting in much faster display times.

Syntax

```
BOOL padMemGetFree(
WORD *FreeBytes
)
```

Returns

Returns TRUE if the function succeeds, FALSE otherwise.

See Also

padMemReset, padMemClear, padMemDelete, padMemLoadText, padMemLoadBitmap, padMemFind, padMemGetChecksum

padMemGetVar

Retrieves the data stored at specified memory location.

Syntax

```
BOOL padMemGetVar(
char FAR *pcName,
WORD wNameLen,
char FAR *pcData,
WORD FAR *pwDataLen
)
```

Parameter	Description
pcName	Variable name containing the data
wNameLen	Name length
pcData	Pointer to a buffer where retrieved data is placed
pwDataLen	Buffer length

Returns

Returns TRUE if the function succeeds, FALSE otherwise.

See Also

padMemSetVar, padFormSaveFld

padMemLoadBitmap

Loads a raw unformatted bitmap into the non-volatile memory with the given **Id**. This function overwrites any item already assigned to **Id**. (Use "padMemFind" on page 3-29 to determine if the given **Id** is being used already.) This command does not accept BMP style bitmaps with header data; it only accepts raw bitmap data with no header. The maximum number of bitmaps that can be loaded depends on how much non-volatile memory is available (see "padMemGetFree" on page 3-30), and how large the bitmaps are.

On the Transaction Team 3100 Series the bitmap data should be stored using the standard 1 bit per pixel black and white scheme since the Transaction Team 3100 Series has a black and white display only. See *Supported Bitmap Format* beginning on page 4-1.

Syntax

```
BOOL padMemLoadBitmap(
WORD Id,
padPOINT *pptSize,
BYTE *Bits
)
```

Parameter	Description
Id	ID of the memory item
pptSize	Horizontal and vertical size of the bitmap in pixels. The maximum horizontal and vertical values allowable for the bitmap are identical to the values returned by **padLcdWidth** (equivalent to 320 for the Transaction Team 3100 Series) and **padLcdHeight** (equivalent to 240 for the Transaction Team 3100 Series).
Bits	Pointer to the raw bitmap data to be loaded. On the TT3100 Series the bitmap data should be stored using the standard 1 bit per pixel black and white scheme since the TT3100 Series has a black and white display only. See *Supported Bitmap Format* beginning on page 4-1.

Returns

Returns TRUE if the function succeeds, FALSE otherwise.

See Also

padMemReset, padMemClear, padMemDelete, padMemFind, padMemLoadText, padMemGetFree, padMemGetChecksum, padPutBits, padPutBmpFile, padDisplayObject, padMemLoadBitmapFile

padMemLoadBitmapFile

Loads a BMP-style bitmap file into the non-volatile memory with the given **Id**. This function overwrites any item already assigned to **Id**. (Use "padMemFind" on page 3-29 to determine if the given **Id** is being used already.) Only 1 bit per pixel black and white BMP formats are accepted. The maximum number of bitmaps that can be loaded depends on how much non-volatile memory is available (see "padMemGetFree" on page 3-30), and how large the bitmaps are.

Syntax

BOOL padMemLoadBitmapFile(
WORD Id,
char *FileName
)

Parameter	Description
Id	ID of the memory item
FileName	The name of the black and white BMP file to load

Returns

Returns TRUE if the function succeeds, FALSE otherwise.

See Also

padMemReset, padMemClear, padMemDelete, padMemFind, padMemLoadText, padMemGetFree, padMemGetChecksum, padPutBits, padPutBmpFile, padDisplayObject, padMemLoadBitmap

padMemLoadText

Loads an ASCII text string into the non-volatile memory with the given **Id**. The text can be arbitrarily long. However, Transaction Team units do not wrap around the texts. It is the responsibility of the programmer to ensure the text fits the screen and to perform the necessary wraparound.

Syntax

BOOL padMemLoadText(
WORD Id,
char *Text,
WORD TextLength
)

Parameter	Description
Id	ID of the memory item to store
Text	Pointer to the text string to store
TextLength	Length of the text string

Returns

Returns TRUE if the function succeeds, FALSE otherwise.

See Also

padMemReset, padMemClear, padMemDelete, padMemFind, padMemLoadBitmap, padMemGetFree, padMemGetChecksum

padMemReset

Resets the memory. This command doesn't clear the used memory, as opposed to **padMemGetFree** (see page 3-30) which clears all user memory.

Syntax

BOOL padMemReset(
void
)

Returns

Returns TRUE if the non-volatile memory is reset, FALSE otherwise.

See Also

padMemGetFree, padMemClear, padMemDelete, padMemLoadText, padMemLoadBitmap, padMemFind, padMemGetChecksum

padMemSetVar

Sets the specified varible with the data retrived by executing the given command.

Syntax

```
BOOL padMemSetVar(
char FAR * pcVarName,
WORD wNameLen,
WORD wCmd,
char FAR *pcCmdData,
WORD wDataLen
)
```

Parameter	Description
pcVarName	Variable name to which to save the data
wNameLen	Name length
wCmd	Command to be executed to get the data
pcCmdData	Command data
wDataLen	Length of the command data

Returns

Returns TRUE if the non-volatile memory is reset, FALSE otherwise.

See Also

padMemGetFree, padMemClear, padMemDelete, padMemLoadText, padMemLoadBitmap, padMemFind, padMemGetChecksum

padName

Returns a pointer to a string constant containing the name of the pad attached. For example, “PenWare3000” is returned if the attached pad is a Transaction Team 3100 Series, “PenWare1500” is returned if the pad attached is a Transaction Team 1500. This differs from **padType** in that **padType** returns a numeric value rather than a string.

```
const char *padName(
void
)
```

See Also

padType

padNewX

Syntax

```
int padNewX(
void
)
```

Returns

Returns the most recent horizontal coordinate received.

See Also

padGet, padOldX, padNewY

padNewY

Syntax

```
int padNewY(
void
)
```

Returns

Returns the most recent vertical coordinate received.

See Also

padGet, padOldY, padNewX

padOff

Stops any recording, turns off the pad and closes communication channels. The port is closed at the baud rate of the initial connection. The user must close Transaction Team devices used by calling this API. Failure to do so may result in failure to re-connect to the attached unit.

Syntax

```
void padOff(
void
)
```

See Also

padOn, padStop, padConnect, padSetBaudRate, padGetBaudRate, padSetDefaultBaudRate, padGetDefaultBaudRate, padResetDefaultBaudRate, padResetBaudRate

padOldX

Syntax

```
int padOldX(
void
)
```

Returns

Returns the previous horizontal coordinate received.

See Also

padOldY, padNewX

padOldY

Syntax

```
int padOldY(
void
)
```

Returns

Returns the previous vertical coordinate received.

See Also

padOldX, padNewY

padOn

Checks for the existence of a pad and initializes communications. Normally this command searches all valid COM-ports for the existence of all supported Transaction Team pad types. This command must be executed before using the library; the only exceptions to this are the commands **padSetPort**, **padSetPortAddr**, **padSetPortIrq**, and **padSetType**, which modify the behavior of **padOn** and must be called prior to calling **padOn**. This function is similar to **padConnect** except that it will never attempt to connect to a pad if a connection is was already established. The **padOn** command can only be issued once if it succeeds until a **padOff** command is issued. If you wish to reconnect to a pad, use the **padConnect** command instead.

Syntax

BOOL padOn(
void
)

Returns

Returns TRUE if successful, FALSE otherwise.

See Also

padConnect, padConnectClearScreen, padOff, padRecord, padSetPort, padSetPortAddr, padSetPortIrq, padSetType, padSetBaudRate, padGetBaudRate, padSetDefaultBaudRate, padGetDefaultBaudRate, padResetDefaultBaudRate, padResetBaudRate

padPassThroughHandshaking

Enables/disables hardware handshaking when in pass through mode.

Syntax

WORD padPassThroughHandshaking (
WORD aHandshaking
)

Parameter	Description
aHandshaking	Yes / No

Returns

Previous handshaking mode.

See Also

padPassThroughOn, padPassThroughSetOffCode, padPassThroughSetOnCode, padPassThroughResetCodes

padPassThroughOff

Turns off pass through mode. While in pass through mode, all pads in pass through mode connected together in a chain monitor the data passing through them. If any of the pads see the **padPassThroughOff** command and the correct code passed along with it, then that pad will disable pass through mode. If the code is incorrect for a pad seeing the command, then that pad will not process it. It simply passes the **padPassThroughOff** command through to its pass through port as if it was normal binary data. Whatever device is connected to its pass through port then will receive the p**adPassThroughOff** command. If there is another pad in pass through mode that is connected to its pass through port, and if the code is correct for that pad, then that pad will disable its pass through mode.

See **padPassThroughSetOffCode** and **padPassThroughResetCodes** for more information on using the correct code for turning off pass through mode. To enable pass through mode use **padPassThroughOn**.

Syntax

```
WORD padPassThroughOff(
char *code,
WORD codeLength
)
```

Parameter	Description
code	A buffer containing the binary code used to turn off pass through mode
codeLength	The length of the binary pass through code in bytes

Returns

Returns non-zero if successful, zero otherwise.

See Also

padPassThroughOn, padPassThroughSetOffCode, padPassThroughSetOnCode, padPassThroughResetCodes

padPassThroughOn

Turns on pass through mode. To enter pass through mode for a specific pad you must use the correct pass through code for that pad. If the code is not correct, then that pad will not enter pass through mode. After correctly enabling pass through mode, PadCom can send and receive data to and from a device connected to the pass through port of that pad, but it cannot send and receive data to or from that pad until the pass through mode of that pad is disabled (see **padPassThroughOff**).

The device connected to the pass through port of a pad must operate at the same baud rate as the pad or you will not be able to communicate with it. You can use **padSetBaudRate** (see page 3-50) before entering pass through mode to set the pad to the same baud rate as the device connected to its pass through port. When connecting a series of pads together using the pass through ports, it becomes a complex task to manage if all of the pads initially have different baud rates. PadCom does not readily support multiple devices with multiple baud rates; it is best to avoid this situation.

*Note: Before entering pass through mode it is advised that you set the pass through "OFF" code for that pad. Failure to do so may cause that pad to get stuck in pass through mode until powered down. If you send the **padPassThroughOff** command with an "OFF" code that doesn't match that pad's "OFF" code, then that pad will not process the command and will simply send it to any device connected to its pass through port.*

See **padPassThroughSetOnCode** and **padPassThroughResetCodes** for more information on using the correct code for turning on pass through mode. To disable pass through mode use **padPassThroughOff**.

Syntax

```
WORD padPassThroughOn(
char *code,
WORD codeLength
)
```

Parameter	Description
code	A buffer containing the binary code used to turn on pass through mode
codeLength	The length of the binary pass through code in bytes

Returns

Returns non-zero if successful, zero otherwise.

See Also

padPassThroughOff, padPassThroughSetOffCode, padPassThroughSetOnCode, padPassThroughResetCodes

padPassThroughResetCodes

This command resets the pass through “ON” and “OFF” codes to the factory defaults. The default codes are two bytes in length. The default “ON” code is AA 55 in hexadecimal (170 85 in decimal). The default “OFF” code is 55 AA in hexadecimal (85 170 in decimal). After using this command you must use these default values when enabling or disabling pass through for that pad.

Note: You cannot use this command to reset the codes of a pad that is currently in pass through mode. If a pad is in pass through mode it will ignore this command and send it to its pass through port.

See **padPassThroughSetOnCode** and **padPassThroughSetOffCode** for more information on using the correct code for turning on pass through mode.

Syntax

```
WORD padPassThroughResetCodes(
void
)
```

Returns

Returns non-zero if successful, zero otherwise.

See Also

padPassThroughOff, padPassThroughSetOffCode, padPassThroughSetOnCode, padPassThroughOn

padPassThroughSetOffCode

This command sets the pass through “OFF” code of a pad. The code can be from 1 to 4 bytes in length. After setting the “OFF” code you must use the same “OFF” code when disabling pass through mode for that pad or you will not be able to disable pass through mode for that pad (see **padPassThroughOff**). You can have a chain of pads connected in series using the pass through ports of each pad. Giving each pad a unique “OFF” code allows you to selectively disable pass through mode for a specific pad. If any two or more pads have the same “OFF” code then **padPassThroughOff** disables the first pad with that “OFF” code in the chain.

Note: You cannot use this command to set the “OFF” code of a pad that is currently in pass through mode. If a pad is in pass through mode it will ignore this command and send it to its pass through port.

Syntax

```
WORD padPassThroughSetOffCode(
char *code,
WORD codeLength
)
```

Parameter	Description
code	A buffer containing the binary code used to turn off pass through mode
codeLength	The length of the binary pass through code in bytes

Returns

Returns non-zero if successful, zero otherwise.

See Also

padPassThroughOff, padPassThroughResetCodes, padPassThroughSetOnCode, padPassThroughOn

padPassThroughSetOnCode

This command sets the pass through “ON” code of a pad. The code can be from 1 to 4 bytes in length. After setting the “ON” code you must use the same “ON” code when enabling pass through mode for that pad or you will not be able to enable pass through mode for that pad (see **padPassThroughOn**). If you have more than two pads connected together in a series using the pass through ports it is not nessesary for each pad to have a unique “ON” code, however, it is very useful for each pad to have a unique “OFF” code (see **padPassThroughSetOffCode**).

NOTE: You cannot use this command to set the "ON" code of a pad that is currently in pass through mode. If a pad is in pass through mode it will ignore this command and send it to its pass through port.

Syntax

```
WORD padPassThroughSetOnCode(
char *code,
WORD codeLength
)
```

Parameter	Description
code	A buffer containing the binary code used to turn on pass through mode
codeLength	The length of the binary pass through code in bytes

Returns

Returns non-zero if successful, zero otherwise.

See Also

padPassThroughOff, padPassThroughResetCodes, padPassThroughSetOffCode, padPassThroughOn

padPortReclaim

Reclaims the port.

Syntax

```
BOOL padPortReclaim ()
```

See Also

padPortRelease

padPortRelease

Releases the port.

Syntax

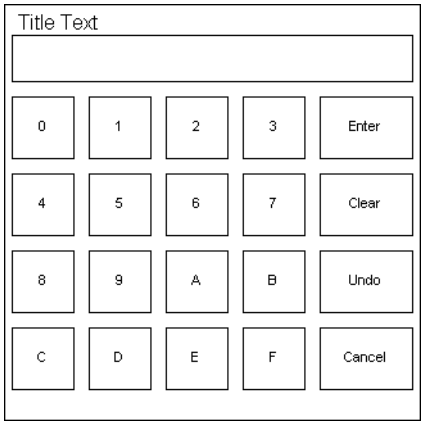
```
void padPortRelease ()
```

See Also

padPortReclaim

padPromptHexNumber

Displays a hexadecimal keypad and returns the entered hexadecimal number. The user is prompted to enter a hexadecimal value. The prompt includes “ENTER”, “CLEAR”, ”C”, “UNDO”, and “CANCEL” buttons. “ENTER” accepts the numeric entry. “CLEAR” and “C” both clear the numeric entry. “UNDO” removes the last digit entered. “CANCEL” exits the prompt. The numeric value can have up to 17 digits. The result is returned as a text string containing the digits entered. The illustration depicts the layout of a HEX number pad.



Syntax

BOOL padPromptHexNumber(
char *Title,
WORD MaxDigits,
char *Result
)

Parameter	Description
Title	Specifies the title of the displayed hexadecimal number pad. It’s a pointer to a null terminated character string.
MaxDigits	Maximum digits to be displayed
Result	Returns a variable length text string containing the number entered

Returns

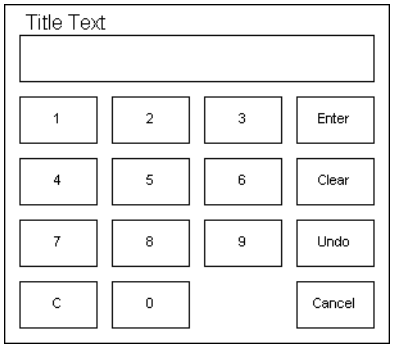
Returns TRUE upon success, FALSE otherwise.

See Also

padPromptNum, padPromptNumber, padPromptReset, padPromptSignature, padPromptString, padPromptTimeout

padPromptNum

This command displays a numeric keypad and returns the entered number. The user is prompted to enter a number whose value can range from 0 to 65,535. The prompt includes “ENTER”, “CLEAR”, ”C”, “UNDO”, and “CANCEL” buttons. “ENTER” accepts the numeric entry. “CLEAR” and “C” both clear the numeric entry. “UNDO” removes the last numeric digit entered. “CANCEL” exits the prompt. The illustration depicts the layout of a number pad.



Syntax

BOOL padPromptNum(
char *Title,
WORD *Result
)

Parameter	Description
Title	Specify the title of the displayed number pad. It’s a pointer to a null terminated character string.
Result	Returns a 16-bit value representing the entered number.

Returns

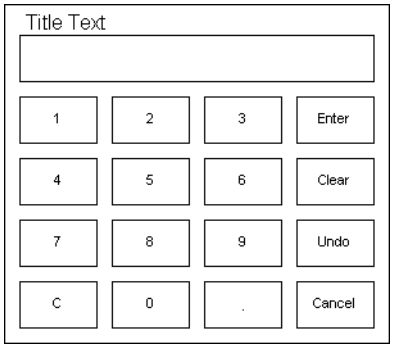
Returns TRUE upon success, FALSE otherwise.

See Also

padPromptHexNumber, padPromptNumber, padPromptReset, padPromptSignature, padPromptString, padPromptTimeout

padPromptNumber

Displays a numeric keypad and returns the entered number. The user is prompted to enter a numeric value. The prompt includes “ENTER”, “CLEAR”, ”C”, “UNDO”, and “CANCEL” buttons. “ENTER” accepts the numeric entry. “CLEAR” and “C” both clear the numeric entry. “UNDO” removes the last numeric digit entered. “CANCEL” exits the prompt. Unlike **padPromptNum**, the numeric value can be a decimal value and can have up to 17 digits, including the optional decimal point, and is not returned as a 16-bit value. Instead the result is returned as a text string containing the digits entered and a decimal point (if any). If **Decimal** equals 0, then no decimal point is allowed. If **Decimal** equals 2, then a total of two digits to the right of the decimal point are allowed; a value of 3 allows three digits to the right of the decimal point, and so on. The illustration depicts the layout of a number pad with the decimal point.



Syntax

BOOL padPromptNumber (
char *Title,
WORD MaxDigits,
WORD Decimal,
char *Result
)

Parameter	Description
Title	Specify the title of the displayed number pad. Its a pointer to a null terminated character string.
MaxDigits	Maximum digits to be displayed (including decimal point)
Decimal	The amount of digits allowed to the right of the decimal point
Result	Returns a variable length text string containing the number entered

Returns

Returns TRUE upon success, FALSE otherwise.

See Also

padPromptHexNumber, padPromptNum, padPromptReset, padPromptSignature, padPromptString, padPromptTimeout

padPromptReset

This command resets/cancels all prompts. This command causes the prompt to act as though the “CANCEL” button on the prompt has been pressed. If this command is called, and a prompt is currently being displayed (such as **padPromptNumber**), the prompt is immediately removed from the screen. This is used in cases where the prompt needs to be removed if the user has not acted on the prompt within a given amount of time. For example, suppose that **padPromptNumber** is called and a number prompt is now on the display. Three minutes pass and the user has not responded to the prompt. You can use **padPromptReset** to stop the current prompt and go on to something else.

Syntax

BOOL padPromptReset(
void
)

Returns

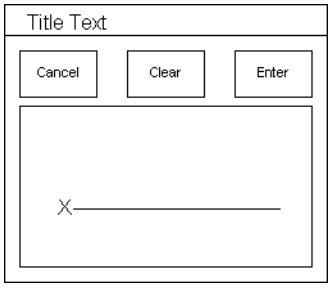
Returns TRUE upon success, FALSE otherwise.

See Also

padPromptHexNumber, padPromptNum, padPromptNumber, padPromptSignature, padPromptString, padPromptTimeout

padPromptSignature

Displays a signature capture prompt on the pad. You can set the compressed capture mode using the **padSetCompress** (see page 3-51) command. The illustration depicts the layout of the signature prompt. The data is returned in a special Transaction Team signature data packet. This packets needs to be post-processed to extract the signature data from the buffer. Please contact Hand Held Products for more information on extracting the signature data returned from this command.



Syntax

```
BOOL padPromptSignature(
char *Title,
WORD MaxPoints,
char *SigData,
WORD SigBufSize,
WORD *SigDataSize
)
```

Parameter	Description
Title	Specifies the title of the displayed prompt pad. Its a pointer to a null terminated character string.
MaxPoints	Maximum signature capture points (default = 1024)
SigData	A pointer to the buffer to hold the signature data
SigBufSize	Specifies the maximum size of the signature data buffer
SigDataSize	Returns a 16-bit value representing the size of the captured signature data

Returns

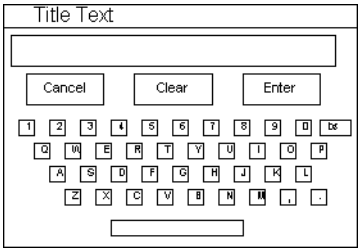
Returns TRUE upon success, FALSE otherwise.

See Also

padPromptHexNumber, padPromptNum, padPromptNumber, padPromptReset, padPromptString, padPromptTimeout

padPromptString

Displays an alphanumeric keypad and returns the entered alphanumeric data. This command prompts the user to enter alphanumeric data. The prompt includes “ENTER”, “CLEAR”, ”C”, “UNDO”, and “CANCEL” buttons. “ENTER” accepts the alphanumeric entry. “CLEAR” and “C” both clear the alphanumeric entry. “UNDO” removes the last character entered. “CANCEL” exits the prompt. The alphanumeric data can have up to 17 characters. The result is returned as a text string containing the characters entered. The illustration depicts the string input keypad.



Syntax

```
BOOL padPromptString(
char *Title,
WORD MaxDigits,
char *Result
)
```

Parameter	Description
Title	Specifies the title of the displayed alphanumeric entry pad. Its a pointer to a null terminated character string.
MaxDigits	Maximum alphanumeric characters to be displayed
Result	Returns a variable length text string containing the alphanumeric data entered

Returns

Returns TRUE upon success, FALSE otherwise.

See Also

padPromptHexNumber, padPromptNum, padPromptNumber, padPromptReset, padPromptSignature, padPromptTimeout

padPromptTimeout

This command is used to set the maximum amount of time a prompt (such as **padPromptSignature**) will remain on the display. The default is 600 seconds (10 minutes). After the specified amount of time elapses, the prompt is removed from the screen.

Syntax

```
void padPromptTimeout (
unsigned long Seconds
)
```

Parameter	Description
Seconds	Specifies the amount of time in seconds to keep the prompt on the display

See Also

padPromptHexNumber, padPromptNum, padPromptNumber, padPromptReset, padPromptSignature, padPromptString

padPutBits

Draws a bitmap on the LCD display by transferring the contents of a bitmap buffer to the LCD screen.

Note: This command does not display a BMP-type bitmap with header data. It only accepts raw bitmap data without a header. See Supported Bitmap Format beginning on page 4-1 for more information on using raw bitmaps.

Syntax

```
BOOL padPutBits(
int X,
int Y,
int Width,
int Height,
const void *Bits
)
```

Parameter	Description
X	Horizontal coordinate to place the bitmap
Y	Vertical coordinate to place the bitmap
Width	Horizontal size of the bitmap in pixels
Height	Vertical size of the bitmap in pixels
Bits	Pointer to the bitmap data

Returns

Returns TRUE if successful, FALSE otherwise.

See Also

padIsLcd, padClear, padPutLogo, padMemLoadBitmap, padPutBmpFile, padPutLogo, padSetLogoBmpFile

padPutBmpFile

This command transfers the contents of the Windows bitmap file to the LCD screen.

Note: This command accepts a Windows bitmap (BMP) file. It does not accept raw unformatted bitmap data.

Syntax

```
BOOL padPutBmpFile(
int X,
int Y,
const char *FileName
)
```

Parameter	Description
X	Horizontal coordinate to place the bitmap
Y	Vertical coordinate to place the bitmap
FileName	Name of Windows bitmap file

Returns

Returns TRUE if successful, FALSE otherwise.

See Also

padIsLcd, padClear, padSetLogoBmpFile, padMemLoadBitmap, padPutBits

padPutLogo

Draws the logo on the LCD screen by displaying the current logo image stored in the pad's non-volatile memory. The logo image can be set to any bitmap image the user desires (see "padSetLogo" on page 3-56 and "padSetLogoBmpFile" on page 3-56).

Syntax

```
BOOL padPutLogo(
void
)
```

See Also

padIsLcd, padSetLogo, padSetLogoBmpFile

padPutText

Draws the text pointed to by **String** onto the display. The upper left corner of the text is positioned at the coordinate specified by the giving X and Y parameters. Text is drawn using the current font. The parameter X specifies the horizontal position, and the parameter Y specifies the vertical position on the LCD screen.

Syntax

```
BOOL padPutText(
int X,
int Y,
const char *String
)
```

Parameter	Description
X	Horizontal coordinate to draw text
Y	Vertical coordinate to draw text
String	NULL terminated string of text to be displayed

See Also

padIsLcd, padClear, padSetFont

padReadByte

This command retrieves a single byte from the pad. This command can be used when the pad is in pass through mode to retrieve data from a device connected to the pass through port of the pad (if available). Using **padSetInTimeout** (see page 3-55) you can control how long this command will wait for the byte to be retrieved.

Syntax

```
BOOL padReadByte(
BYTE *b
)
```

Parameter	Description
b	Pointer to a memory location that will receive the byte

Returns

If successful it returns TRUE, otherwise it returns FALSE.

See Also

padSendByte, padPassThroughOn, padSetInTimeout

padRecord

Sends commands to the pad to start recording pad activities. When running DOS, this function does not perform any notifications and it is up to the application to use **padUpdate** (see page 3-66) repeatedly to collect incoming data. When using Windows, a notification message is sent to the Window specified by hWnd. Upon receiving the message, the window uses **padUpdate** to check for and receive new data. When using 16 bit Windows 3.x, the notification message is WM_COMMNOTIFY. When using 32 bit Windows 95/NT, you may specify any message with optional message parameters.

Syntax

DOS:
```
BOOL padRecord(
void
)
```
WIN16:
```
BOOL padRecord(
HWND hWnd
)
```
WIN32:
```
BOOL padRecord(
HWND hWnd,
UINT aMsg,
WPARAM p1,
LPARAM p2
)
```

Parameter	Description
hWnd	Handle to a window to receive notification messages
aMsg	A message to use for notification
p1	An optional message parameter
p2	An optional message parameter

Returns

Returns TRUE if successful, FALSE otherwise.

See Also

padStop, padUpdate, padGet

padReset

Resets the pad. This command should be used to reset all default values on the pad. At times, it can take a few seconds for the pad to be completely reset.

Syntax

BOOL padReset(
void
)

Returns

Returns TRUE upon success, FALSE otherwise.

See Also

padResetArea, padStop, padOff

padResetArea

Resets the minimum and maximum coordinates of the active pad area. This command is used to undo **padSetArea** (see page 3-49). It restores the usable area of the pad to the full pad surface.

Syntax

void padResetArea(
void
)

See Also

padGetArea, padSetArea

padResetBaudRate

Sets the communications baud rate to the default baud rate of 9600. Note that the command **padOff** (see page 3-34) resets the baud rate back to the baud rate at which the device was initially connected.

Syntax

BOOL padResetBaudRate(
void
)

Returns

Returns TRUE if successful, FALSE otherwise.

See Also

padSetBaudRate, padGetBaudRate, padSetDefaultBaudRate, padGetDefaultBaudRate, padResetDefaultBaudRate

padResetConnectTimeout

This function resets the time-out value used for the com port (both input and output) during the initial connecting phase. After a connection is established, the time-out values can be adjusted using **padSetInTimeout** and **padSetOutTimeout** (see page 3-55).

Syntax

unsigned int padResetConnectTimeout(
void
)

Returns

Returns the current connection time-out setting

See Also

padSetOutTimout, padGetInTimeout, padGetOutTimeout, padSetInTimeout, padResetOutTimeout, padSetConnectTimeout, padGetConnectTimeout

padResetDefaultBaudRate

Sets the default communications baud rate to the initial baud rate of 9600. See "padSetDefaultBaudRate" on page 3-52 for more information.

Syntax

BOOL padResetDefaultBaudRate(
void
)

Returns

Returns TRUE if successful, FALSE otherwise.

See Also

padSetBaudRate, padGetBaudRate, padSetDefaultBaudRate, padGetDefaultBaudRate, padResetBaudRate

padResetInTimeout

This function resets the time-out value used for the com port's input from the connected pad. This is not related to the time-out used during the initial connection phase (see "padSetConnectTimeout" on page 3-52).

Syntax

unsigned int padResetInTimeout(
void
)

Returns

Returns the default time-out setting

See Also

padSetOutTimout, padGetInTimeout, padGetOutTimeout, padSetInTimeout, padResetOutTimeout, padSetConnectTimeout, padGetConnectTimeout, padResetConnectTimeout

padResetMagCard

This function resets the magnetic card reader available on the Transaction Team 3100 Series. Commands such as **padGetMagTrack** and **padGetAllMagCardTracks** require that the magnetic card reader be reset before use. Resetting the magnetic card reader clears the card track data from the magnetic card reader's buffer and activates the card reader to allow capture of track data from another card. Only one card may be swiped at a time. Each time a new card is to be swiped, a call to **padResetMagCard** must be made before swiping the card. After that you can use **padGetMagTrack** (see page 3-15) or **padGetAllMagCardTracks** (see page 3-9) to read the card's captured track data.

Syntax

BOOL padResetMagCard(
void
)

Returns

Returns TRUE if a card reader is available, FALSE if no card reader is available.

See Also

padGetMagTrack, padGetMaxCardTrackSize, padGetMaxCardTracks, padGetAllMagCardTracks

padResetOutTimeout

This function resets the time-out value used for the com port's output to the connected pad. This is not related to the time-out used during the initial connection phase (see "padSetConnectTimeout" on page 3-52).

Syntax

```
unsigned int padResetOutTimeout(
void
)
```

Returns

Returns the default time-out setting

See Also

padSetOutTimout, padGetInTimeout, padGetOutTimeout, padSetInTimeout, padResetInTimeout, padSetConnectTimeout, padGetConnectTimeout, padResetConnectTimeout

padScale

This function uses integer math to perform scaling by a given fraction.

Syntax

```
int padScale(
int Value,
int Numerator,
int Denominator
)
```

Parameter	Description
Value	The value to be scaled
Numerator	The numerator of the fraction for the scale
Denominator	The denominator of the fraction for the scale

Returns

Returns the result of the scaling.

See Also

padScaleX, padScaleY, padScaleDPI, padScaleTo

padScaleDPI

Scales horizontal and vertical pad coordinates to a desired DPI resolution. This function is designed to provide proper aspect ratio scaling for accurate output. It converts the values pointed to by X and Y pointers. NULL pointers may be passed for either X or Y parameters to suppress scaling of the associated coordinate.

Syntax

```
void padScaleDPI(
int *X,
int *Y,
int HorzDPI,
int VertDPI
)
```

Parameter	Description
X	Pointer to an integer to hold the horizontal coordinate to be scaled
Y	Pointer to an integer to hold the vertical coordinate to be scaled
HorzDPI	The horizontal dots-per-inch resolution desired
VertDPI	The vertical dots-per-inch resolution desired

See Also

padScaleTo, padScaleX, padScaleY

padScaleTo

Scales horizontal and vertical pad coordinates based on a desired frame size. This function does not maintain the proper aspect ratio unless the desired output frame is proportional to the pad surface. It converts the values pointed to by X and Y pointers. NULL pointers may be passed for either X or Y parameters to suppress scaling of the associated coordinate.

Syntax

```
void padScaleTo(
int *X,
int *Y,
int Width,
int Height
)
```

Parameter	Description
X	Pointer to an integer to hold the horizontal coordinate to be scaled
Y	Pointer to an integer to hold the vertical coordinate to be scaled
Width	The horizontal size of the desired frame area
Height	The vertical size of the desired frame area

See Also

padScaleDPI, padScaleX, padScaleY

padScaleX

Scales a horizontal pad coordinate to a desired DPI resolution. This function is designed to provide proper aspect ratio scaling for accurate output.

Syntax

```
int padScaleX(
int X,
int HorzDPI
)
```

Parameter	Description
X	An integer specifying the horizontal coordinate to be scaled
HorzDPI	The horizontal dots-per-inch resolution of the horizontal coordinate

Returns

Returns the horizontal position scaled to the desired DPI.

See Also

padScaleY, padScaleDPI

padScaleY

Scales a vertical pad coordinate to a desired DPI resolution. This function is designed to provide proper aspect ratio scaling for accurate output.

Syntax

int padScaleY(int Y, int VertDPI)

Parameter	Description
Y	An integer specifying the vertical coordinate to be scaled
VertDPI	The vertical dots-per-inch resolution of the horizontal coordinate

Returns

Returns the vertical position scaled to the desired DPI.

See Also

padScaleX, padScaleDPI

padSendByte

This command sends a byte to the pad. This command can be used when the pad is in pass through mode to send data to a device connected to the pass through port of the pad (if available). Using **padSetOutTimeout** (see page 3-55) you can control how long this command will wait for the byte to be sent.

Syntax

BOOL padSendByte(
BYTE b
)

Parameter	Description
b	The byte to send to the pad

Returns

If successful it returns TRUE, otherwise it returns FALSE.

See Also

padReadByte, padPassThroughOn, padSetOutTimeout

padSetArea

This sets the minimum and maximum coordinates of the active pad area. This function causes clipping of points received from the Transaction Team POS device's touch sensitive surface. All points that are outside of the specified range are ignored. The coordinates given are not display coordinates, they are touch surface coordinates. In general, touch surface coordinates are much higher in resolution than display coordinates. For example the Transaction Team 3100 Series touch surface contains 903x1238 touch points while the Transaction Team 1500 display contains only 418x798 pixels, even though they are about the same size in inches. You can use **padGetPage** (see page 3-19) to obtain the size and resolution of the touch sensitive surface.

*Note: The **padRecord** (see page 3-44) function must be called before points are retrieved from the Transaction Team POS device's touch sensitive surface.*

Syntax

```
void padSetArea(
int xLeft,
int yTop,
int xRight,
int yBottom
)
```

Parameter	Description
xLeft	Minimum horizontal coordinate
yTop	Minimum vertical coordinate
xRight	Maximum horizontal coordinate
yBottom	Maximum vertical coordinate

See Also

padGetArea, padGetPage, padResetArea, padRecord

padSetAutoInking

Sets auto inking mode as specified by AutoInking (1 or 0). Auto-inking causes the Transaction Team POS device's screen to automatically display a pixel where the Transaction Team POS device's overlaying touch surface area is touched. The device's overlaying touch surface must be in record mode (see "padRecord" on page 3-44) for this to occur. If auto-inking is disabled, the screen doesn't automatically display a pixel when the pad's surface is touched, even when in record mode. If the value in AutoInking is set to 0, then auto-inking is disabled. If set to 1, auto-inking is enabled.

*Note: This function interacts with **padSetInkingArea** (see page 3-54).*

Syntax

```
BOOL padSetAutoInking(
BYTE AutoInking
)
```

Returns

Returns TRUE if successful, FALSE otherwise.

See Also

padGetBkColor, padGetColor, padGetColors, padSetColor, padSetInkingArea

padSetBaudRate

Sets the communications baud rate by attempting to set the communications baud rate to the amount specified by BaudRate. The valid values for BaudRate are 0, 1200, 2400, 4800, 9600, 19200, 38400. Setting the BaudRate to 0 causes the command to automatically try to connect at the highest possible baud rate.

Note: Using 0 may cause problems in DOS applications running under Windows.

The default baud rate is 9600 for PadCom. This command should be called only after **padOn** (see page 3-35) or **padConnect** (see page 3-3) is called, and a valid connection is already established, otherwise it will have no effect. Note that the command **padOff** resets the baud rate to the default baud rate of 9600.

Syntax

```
DWORD padSetBaudRate(
DWORD BaudRate
)
```

Parameter	Description
BaudRate	The baud rate for a communications link attempt

Returns

Returns the new baud rate if successful, FALSE otherwise.

See Also

padConnect, padGetBaudRate, padResetBaudRate, padOn

padSetBkColor

Sets the current background color. This value is either 1 or 0.

Syntax

```
BOOL padSetBkColor(
int Color
)
```

Parameter	Description
Color	The color for the background color

Returns

Returns TRUE if successful, FALSE otherwise.

See Also

padGetBkColor, padGetColor, padGetColors, padSetColor

padSetColor

Sets the current foreground color. This value can be 1 or 0.

Syntax

```
BOOL padSetColor(
int Color
)
```

Parameter	Description
Color	The color for the foreground

Returns

Returns TRUE if successful, FALSE otherwise.

See Also

padGetBkColor, padGetColor, padGetColors, padSetBkColor

padSetCompress

This command can be used to capture compressed data points directly from the pad. This reduces the number of collected data points and results in faster capture speeds, however, the signature obtained is compressed. The Compress flag must be set to TRUE to enable this mode. The default capture mode is uncompressed and remains so till this command is invoked.

Syntax

BOOL padSetCompress(
BOOL Compress
)

Parameter	Description
Compress	TRUE to enable compressed capture mode FALSE to collect uncompressed data points

Returns

Returns TRUE if successful, FALSE otherwise.

padSetConnectTimeout

This function sets the time-out value (in milliseconds) used for the com port during the initial connection phase. After establishing a connection, use **padSetInTimeout** and **padSetOutTimeout** (see page 3-55) to control the time-out settings of the com port.

Syntax

unsigned int padSetConnectTimeout(
unsigned int newTimeout
)

Returns

Returns the old time-out setting

See Also

padSetInTimout, padResetInTimeout, padGetOutTimeout, padResetOutTimeout, padGetConnectTimeout, padResetConnectTimeout

padSetDebug

Sets the system debug option on or off. If the system debug option is on, the LCD displays an error message when an error occurs.

Syntax

BOOL padSetDebug(
WORD Options
)

Parameter	Description
Options	Specifies the debug options: 0000h = no debug 0001h = display error messages

Returns

Returns TRUE upon success, FALSE otherwise.

See Also

padFlush

padSetDefaultBaudRate

Used to set the default baud rate. Ordinarily PadCom initially opens the com port at the default baud rate of 9600. The Transaction Team 3100 Series can communicate at baud rates higher than 9600. If, for example, a Transaction Team 3100 Series is connected and is configured to operate at 57600 baud, it will take PadCom longer to connect to it than it would if the Transaction Team 3100 Series was configured to operate at 9600 baud. This is because ordinarily PadCom searches

all com ports for all possible Transaction Team devices using all possible baud rates, starting with the default of 9600 baud. If a Transaction Team 3100 Series is connected to com port 2 and is configured to communicate at 57600 baud, first PadCom searches port 1 at the default baud rate for both the TT1500 and TT3100 Series (9600). If it fails, it searches for a Transaction Team 3100 Series at 57600, 19200, and then at 38400. PadCom then repeats this course of action for port 2, port 3, port 4, and so on, until a Transaction Team device is found. If the default baud rate is set to 57600, then it will be the first baud rate at which the port is opened, if a connection fails at 57600. Next it tries 9600, followed by 19200, then 38400. For the fastest possible connection, use this command in conjunction with **padSetPort** (see page 3-59) and **padSetType** (see page 3-61).

Syntax

DWORD padSetDefaultBaudRate(
DWORD BaudRate
)

Parameter	Description
BaudRate	Specifies the baud rate to use as the default baud rate. Valid values are 9600, 19200, 38400 and 57600.

Returns

Returns the old default baud rate, or zero if it fails.

See Also

padConnect, padGetDefaultBaudRate, padGetBaudRate, padSetBaudRate, padResetBaudRate, padResetDefaultBaudRate

padSetFlowControl

Enables/disables FlowControl for a connected Transaction Team device. This command only works when a connection is established with the Transaction Team device. Once this command has been successfully issued to the Transaction Team device you must call **padConnect** (or **padOff** followed by **padOn**) in order for the changes to take place. The default is flow control enabled.

Syntax

BOOL padSetFlowControl(
BOOL FlowControl
)

Parameter	Description
FlowControl	TRUE enables flow control, FALSE disables flow control

Returns

Returns TRUE if successful, FALSE otherwise.

See Also

padOn, padConnect, padOff

padSetFont

Sets the font used for text output to value specified in **Font**.

Syntax

int padSetFont(
int Font
)

Parameter	Description
Font	ID of the desired font

Available fonts:

Font Id	Horizontal Size	Vertical Size	Available for Transaction Team 3100 Series
0	8	8	Yes
1	16	16	Yes
2	6	8	Yes
3	8	12	Yes
4	12	16	Yes
5	16	24	Yes

Returns

Returns the previously selected font.

See Also

padGetFont, padPutText

padSetInkingArea

If auto inking is disabled for the entire surface of the pad (see **padSetAutoInking**) this function allows you to specify an area of the LCD where auto inking will occur. Pen stroke data is not altered so you will still receive pen strokes for strokes outside of the inking area. To limit inking and pen stroke data use **padSetArea**. To reset the inking area to the entire surface of the LCD pass 0,0,0,0 to this function as its **x**, **y**, **w**, and **h** parameter values.

The coordinates given are display coordinates, they are not touch surface coordinates. In general, touch surface coordinates are much higher in resolution than display coordinates. For example the 3100's touch surface contains 4096x4096 touch points while the 3100's display contains only 320x240 pixels, even though they are about the same size in inches. Use **padLcdHeight**, **padLcdWidth**, **padLcdHorzDPI**, and **padLcdVertDPI** to find the size and resolution of the LCD screen.

*Note: This function interacts with **padSetAutoInking**. If **padSetAutoInking** is set to true then the entire Transaction Team POS device's overlaying touch surface area will cause inking to occur on the entire LCD screen area. As a result, **padSetInkingArea** has no effect at all. In order for **padSetInkingArea** to have an effect, **padSetAutoInking** must be set to false and **padRecord** must be enabled.*

Syntax

```
BOOL padSetInkingArea(
int x,
int y,
int w,
int h
)
```

Parameter	Description
x	Horizontal starting position of the inking area
y	Vertical starting position of the inking area
w	Width of the inking area
h	Height of the inking area

See Also

padGetArea, padLcdHeight, padLcdWidth, padLcdHorzDPI, padLcdVertDPI, padRecord, padResetArea, padSetArea, padSetAutoInking

padSetInTimeout

This function sets the time-out value in milliseconds used for the com port's input from the connected pad. This is not related to the time-out used during the initial connection phase (see "padSetConnectTimeout" on page 3-52).

Syntax

unsigned int padSetInTimeout(
unsigned int newTimeout
)

Returns

Returns the old time-out setting.

See Also

padSetOutTimout, padResetInTimeout, padGetOutTimeout, padSetInTimeout, padResetOutTimeout, padSetConnectTimeout, padGetConnectTimeout, padResetConnectTimeout

padSetOutTimeout

This function sets the time-out value in milliseconds used for the com port's output to the connected pad. This is not related to the time-out used during the initial connection phase (see padSetConnectTimeout).

Syntax

unsigned int padSetOutTimeout(
unsigned int newTimeout
)

Returns

Returns the old time-out setting.

See Also

padSetInTimout, padResetInTimeout, padGetOutTimeout, padResetOutTimeout, padSetConnectTimeout, padGetConnectTimeout, padResetConnectTimeout

padSetLcdClearTimeout

Sets the LCD automatic clearing time-out value. When it's disabled, the LCD with not automatically clear itself. If it is set to 200, for example, and the device's touch sensitive surface is not touched for a total of 200 seconds, then the device's LCD automatically clears itself.

Syntax

BOOL padSetLcdClearTimeout(
BYTE Timeout
)

Parameter	Description
Timeout	The number of seconds to elapse before time-out is reached. A value of zero disables the automatic LCD clearing feature.

Returns

Returns TRUE if successful, FALSE otherwise.

See Also

padIsLcd, padClear

padSetLogo

Sets the LCD screen logo to the contents of the bitmap image Bits. The logo is not actually displayed until padPutLogo is called. To reset the logo to the default image, use NULL as the Bits parameter. The logo must have 240 pixels across and 128 pixels down.

Note: This command does not accept a Windows bitmap (BMP) with a header. It only accepts raw bitmap data without a header. See Supported Bitmap Format beginning on page 4-1.

Syntax

```
BOOL padSetLogo(
const void *Bits
)
```

Parameter	Description
Bits	Pointer to the bitmap bits

Returns

Returns TRUE if successful, FALSE otherwise.

See Also

padIsLcd, padPutLogo, padSetLogoBmpFile

padSetLogoBmpFile

Sets the LCD screen logo to the contents of the Windows Bitmap file named **FileName**. The logo is not actually displayed until **padPutLogo** is called (see page 3-43). The logo must have 240 pixels across and 128 pixels down.

Note: This command only accepts a Windows Bitmap file (BMP), it does not accept unformatted raw bitmap data without a header. For more information on acceptable bitmap formats, please refer to Appendix A on page 108.

Syntax

```
BOOL padSetLogoBmpFile(
const char *FileName
)
```

Parameter	Description
FileName	Name of Windows bitmap file

Returns

Returns TRUE if successful, FALSE otherwise.

See Also

padIsLcd, padClear, padSetLogo

padSetNumVar

Sets a variable with given numeric data.

Syntax

BOOL padSetNumVar(
char FAR *lpszVarName,
WORD wValue
)

Parameter	Description
lpszVarName	Name of the variable to which the numeric value is assigned
wValue	Value

Returns

Returns TRUE if successful, FALSE otherwise.

See Also

padGetNumVar

padSetPadMode

Sets the Transaction Team 1500's operational mode. When in native mode, the device does not emulate any other device. When in PenWare 100 emulation mode, the device's touch surface behaves as if it had the DPI resolution of a PenWare 100.

Mode 0 = native mode
Mode 1 = PenWare 100 emulation

Syntax

BOOL padSetPadMode(
BYTE Mode
)

Parameter	Description
Mode	Specifies the mode the Transaction Team device should be set to

Returns

Returns TRUE if successful, FALSE otherwise.

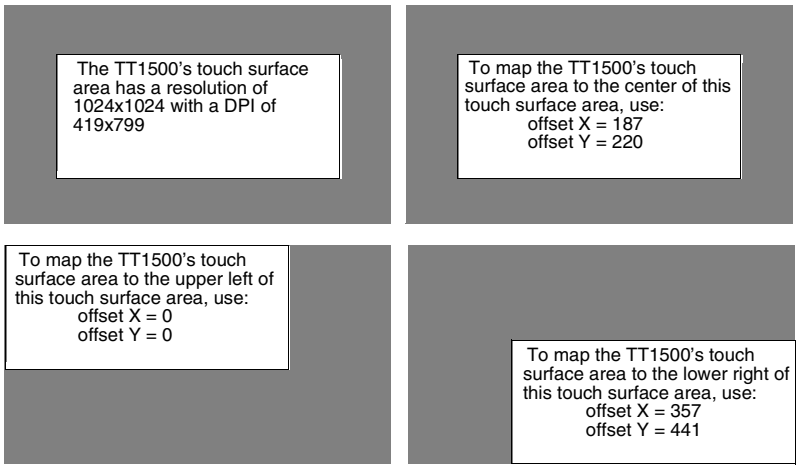
See Also

padSetPadOffset, padSetClearButton

padSetPadOffset

This command sets the offset value of the horizontal and vertical points retrieved from the Transaction Team device's touch surface area when mapping the device's smaller surface area to a larger surface area when in PenWare100 emulation mode. For example, say a touch surface area is approximately 9.5x5.5 centimeters in size. The Transaction Team 1500's touch surface area is approximately 7x3 centimeters in size. To center the smaller Transaction Team 1500's touch surface area within the larger touch surface area you would use an offset of 187x220. To place the Transaction Team 1500's touch surface area at the lower right corner of a larger touch surface area you would use an offset of 357x441. To place the Transaction Team 1500's touch surface area at the upper left corner of a larger touch surface area you would use an offset of 0x0, meaning no offset is used. This command only works if the Transaction Team device is in emulation mode. If the device is not in emulation mode this command has no effect. To put the device into emulation mode use the **padSetPadMode** command (see page 3-57).

If the touch surface area has a resolution of 1024x1024 with a DPI of 273x455:



Syntax

```
BOOL padSetPadOffset(
int OffsetX,
int OffsetY
)
```

Parameter	Description
OffsetX	Horizontal offset measure in the PenWare100's resolution
OffsetY	Vertical offset measure in the PenWare100's resolution

Returns

Returns TRUE if successful, FALSE otherwise.

See Also

padSetPadMode, padSetClearButton

padSetPixel

Sets a pixel on the LCD screen at the location specified by X and Y to the current foreground color.

Syntax

```
BOOL padSetPixel(
int X,
int Y
)
```

Parameter	Description
X	Horizontal coordinate of the pixel to set
Y	Vertical coordinate of the pixel to set

Returns

Returns TRUE if successful, FALSE otherwise.

See Also

padBox, padClearPixel, padFrame, padInvert, padLine

padSetPort

Sets the communications port. The next time **padOn** is executed, only the com port specified by **Port** is used. This function works only when the library is in an "off" state (i.e., before **padOn**() or after **padOff**()). This is not normally needed, however it may be useful in some situations.

Syntax

BOOL padSetPort(
int Port
)

Parameter	Description
Port	1 = COM1 2 = COM2 3 = COM3 4 = COM4 0 causes all the ports to be sequentially checked for a Transaction Team device.

Returns

Returns TRUE if the port specified is legal and library is "off", otherwise FALSE.

See Also

padGetPort, padIsOn, padSetPortAddr, padSetPortIrq, padSetType

padSetPortAddr

Sometimes it may be necessary to manually specify the address used when referring to a communications port. **padSetPortAddr** provides this ability. This function works only when the library is in an "off" state (i.e., before **padOn**() or after **padOff**()). This is not normally needed, however it may be useful in some situations.

Note: This function is available in the DOS version only!

Syntax

DOS: BOOL padSetPortAddr(
int Port,
int Address
)

Parameter	Description
Port	1 = COM1 2 = COM2 3 = COM3 4 = COM4
Address	The value to be used as the port address

Returns

Returns TRUE if the port specified is legal and library is "off", otherwise FALSE.

See Also

padSetPort, padSetPortIrq, padSetType

padSetPortHandle

Sets a handle to the port**.**

Syntax

Win32:	void padSetPortHandle(HANDLE newPortHandle)	Win16:	void padSetPortHandle(int newPortHandle)

Parameter	Description
newPortHandle	Handle to the port

See Also

padSetPort, padSetPortIrq, padSetType

padSetPortIrq

Sometimes it may be necessary to manually specify the interrupt request numbers used when referring to a communications port. **padSetPortIrq** provides this ability. This function works only when the library is in an “off” state (i.e., before **padOn**() or after **padOff**()). This is not normally needed, however it may be useful in some situations.

Note: This function is available in the DOS version only!

Syntax

DOS: int padSetPortIrq(
int Port,
int Irq
)

Parameter	Description
Port	1 = COM1 2 = COM2 3 = COM3 4 = COM4
Irq	The value to be used as the port interrupt number

Returns

Returns TRUE if the port specified is legal and library is “off”, otherwise FALSE.

See Also

padSetPort, padSetPortAddr, padSetType

padSetPorts

Sets the maximum number of communications ports available. The next time **padOn** or **padConnect** is executed, only the com ports specified by **MaxPorts** will be available for use. This function works only when the library is in an “off” state (i.e., before **padOn**/**padConnect** or after **padOff**).

Syntax

int padSetPorts(
int MaxPorts
)

Parameter	Description
MaxPorts	valid values start at 1 for 1 com port (the maximum allowed for Windows 3.1 is 9)

Returns

Returns the previous number of ports used.

See Also

padGetPort, padGetPorts, padIsOn, padSetPortAddr, padSetPortIrq, padSetType

padSetScanRate

Sets the current scan rate, specified by ScanRate. Values can be set from 26 to 199. Scan rate changes the number of signature points scanned per second.

Syntax

BOOL padSetScanRate(
int ScanRate
)

Returns

Returns TRUE upon success, FALSE otherwise.

See Also

padGetScanRate

padSetTime

This function sets the current time on the Transaction Team 3100 Series or compatible POS terminal.

Syntax

BOOL padSetTime(
BYTE cHour,
BYTE cMin,
BYTE cSec
)

Parameter	Description
cHour	Sets the hour
cMin	Sets the minute
cSec	Sets the second

Returns

Returns TRUE upon success, FALSE otherwise.

See Also

padDisplayTime, padGetTime, padHideTime

padSetType

Sets the type of pad to search the com ports for when using **padOn**. Normally **padOn** searches the com ports for all supported Transaction Team pad types. Setting **padSetType** to a valid Transaction Team pad type causes **padOn** to only search the com ports for that specific type of pad. The valid arguments are:

For a Transaction Team 1500: PENWARE1500

For a Transaction Team 3100 Series: PENWARE3000 or PW3000

For example “padSetType (PENWARE1500)” causes **padOn** to only search for a Transaction Team 1500.

*Note: You must call **padSetType** before calling **padOn** or it will have no effect.*

Syntax

BOOL padSetType (
int Type
)

Parameter	Description
Type	Used to specify the type of Transaction Team pad

Returns

Returns TRUE if the type of pad specified is valid and FALSE otherwise.

See Also

padOn, padSetPort, padSetPortAddr, padSetPortIrq

padSoundBell

This command allows you to play various different types of preset sounds stored in the Transaction Team 3100 Series devices. The parameter **BellType** has the following valid decimal setting (hexadecimal values are in parenthesis):

0	(0x00) - cancels any sound being played
1	(0x01) - plays the standard “bell” sound
16	(0x10) - plays an “alarm” sound
20	(0x14) - plays a “success” sound
24	(0x18) - plays a “fail” sound

Syntax

BOOL padSoundBell(
WORD BellType
)

Parameter	Description
BellType	Type of sound the device makes

Returns

Returns TRUE upon success, FALSE otherwise.

See Also

padSoundTone, padSoundSetFreq, padSoundEnable

padSoundEnable

This command enables or disables the speaker on a Transaction Team 3100 Series or compatible device.

Syntax

BOOL padSoundEnable(
BYTE Enable
)

Parameter	Description
Enable	Zero to disable; Non-zero to enable

Returns

Returns TRUE upon success, FALSE otherwise.

See Also

padSoundTone, padSoundBell, padSoundSetFreq

padSoundSetFreq

This command allows the user to set the internal speaker to play a frequency specified by the value in **Freq**. Setting the frequency to zero turns off the sound.

Syntax

BOOL padSoundSetFreq(
WORD Freq
)

Parameter	Description
Freq	The frequency value for the speaker

Returns

Returns TRUE upon success, FALSE otherwise.

See Also

padSoundTone, padSoundBell, padSoundEnable

padSoundTone

This command allows you to set the internal speaker to play a frequency specified by the value in **Freq** for the amount of "tempo beats" specified by **Duration**. 156 "temp beats" are equivalent to approximately 1000 milliseconds (1 second).

The following table shows the frequency values used to play musical notes:

	Octave 1	Octave 2	Octave 3	Octave 4
C	131	262	523	1047
C#	139	278	555	1111
D	147	294	587	1175
D#	156	312	623	1247
E	165	330	659	1319
F	175	349	698	1397
F#	186	371	741	1483
G	196	392	784	1568
G#	208	416	832	1664
A	220	440	880	1760
A#	234	467	934	1868
B	247	494	988	1976

Syntax

BOOL padSoundTone(
WORD Freq,
WORD Duration
)

Parameter	Description
Freq	The frequency value for the speaker
Duration	The amount of time to play the sound frequency

Returns

Returns TRUE upon success, FALSE otherwise.

See Also

padSoundBell, padSoundSetFreq, padSoundEnable

padStop

Stop receiving data from the pad. This function turns off real-time recording initiated with **padRecord** (see page 3-44), and sends commands to the pad to stop transmitting pad activities.

Syntax

```
void padStop(
void
)
```

See Also

padOff, padRecord

padToHIENGLISH

This function scales horizontal and vertical pad coordinates to units representing 0.001 inches. It is designed to provide proper aspect ratio scaling for accurate output. The function converts the values pointed to by X and Y pointers, and corresponds to the Windows mapping mode MM_HIENGLISH units. NULL pointers may be passed for either X or Y parameters to suppress scaling of the associated coordinate.

Syntax

```
void padToHIENGLISH(
int *X,
int *Y
)
```

Parameter	Description
X	Pointer to an integer to hold the horizontal coordinate to be scaled
Y	Pointer to an integer to hold the vertical coordinate to be scaled

See Also

padToLOENGLISH, padToLOMETRIC, padToHIMETRIC

padToLOENGLISH

This function scales horizontal and vertical pad coordinates to units representing 0.01 inches. It is designed to provide proper aspect ratio scaling for accurate output. The function converts the values pointed to by X and Y pointers, and corresponds to the Windows mapping mode MM_LOENGLISH units. NULL pointers may be passed for either X or Y parameters to suppress scaling of the associated coordinate.

Syntax

```
void padToLOENGLISH(
int *X,
int *Y
)
```

Parameter	Description
X	Pointer to an integer to hold the horizontal coordinate to be scaled
Y	Pointer to an integer to hold the vertical coordinate to be scaled

See Also

padToHIENGLISH, padToLOMETRIC, padToHIMETRIC

padToHIMETRIC

This function scales horizontal and vertical pad coordinates to units representing 0.01 millimeters. It is designed to provide proper aspect ratio scaling for accurate output, and converts the values pointed to by X and Y pointers. The function corresponds to the Windows mapping mode MM_HIMETRIC units. NULL pointers may be passed for either X or Y parameters to suppress scaling of the associated coordinate.

Syntax

```
void padToHIMETRIC(
int *X,
int *Y
)
```

Parameter	Description
X	Pointer to an integer to hold the horizontal coordinate to be scaled
Y	Pointer to an integer to hold the vertical coordinate to be scaled

See Also

padToLOMETRIC, padToLOENGLISH, padToHIENGLISH

padToLOMETRIC

This function scales horizontal and vertical pad coordinates to units representing 0.1 millimeters. It is designed to provide proper aspect ratio scaling for accurate output, and converts the values pointed to by X and Y pointers. The function corresponds to the Windows mapping mode MM_LOMETRIC units. NULL pointers may be passed for either X or Y parameters to suppress scaling of the associated coordinate.

Syntax

```
void padToLOMETRIC(
int *X,
int *Y
)
```

Parameter	Description
X	Pointer to an integer to hold the horizontal coordinate to be scaled
Y	Pointer to an integer to hold the vertical coordinate to be scaled

See Also

padToHIMETRIC, padToLOENGLISH, padToHIENGLISH

padType

Returns a value that identifies the type of pad attached. The numeric value returned by **padType** may be one of the following constant values defined in the file “PadCom.H”:

UNKNOWN	No recognizable device has been found
PW1500	A Transaction Team 1500 pad has been found
PW3000	A Transaction Team 3100 Series pad has been found

This command differs from **padName** in that it returns a numeric value rather than a string.

Syntax

enum padTypes padType(
void
)

See Also

padName

padUpdate

This function checks for and receives new data from the pad. It receives data packets from the pad and decodes them into meaningful information. All information received is stored until the next time this function is executed. Use functions such as **padGet** (see page 3-9) and **padIsPenDown** (see page 3-25) to access the information received. This function is generally used in response to a notification message as discussed in the **padRecord** (see page 3-44) reference.

Syntax

BOOL padUpdate(
void
)

Returns

Returns TRUE if new data was received, otherwise FALSE.

See Also

padRecord, padGet, padHardwareButton, padLeftButton, padRightButton, padMiddleButton

padVertDPI

Syntax

int padVertDPI(
void
)

Returns

Returns the number of vertical points per inch.

See Also

padHeight, padHorzDPI

padWidth

Syntax

intpadWidth(
void
)

Returns

Returns the total number of horizontal points on the pad surface (i.e., 1024).

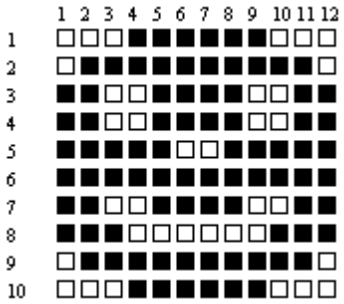
See Also

padHeight, padHorzDPI

4 Supported Bitmap Format

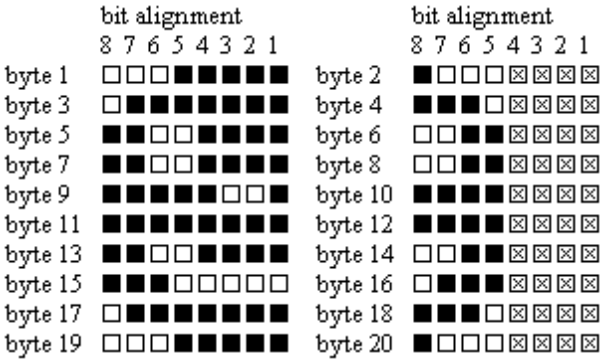
Many PadCom commands such as **padPutBmpFile** and **padSetLogoBmpFile** accept Windows style bitmap files. Other command such as **padPutBits** and **padMemLoadBitmap** do not. These commands accept raw black and white bitmap data. The raw black and white bitmap format used throughout PadCom is the standard raw bitmap format used for black and white bit mapped images. The bitmap data is byte aligned. The first byte represents the top left most portion of the image. The last byte represents the bottom right most portion of the image. The black and white format uses one bit per pixel. Bit 1 is the rightmost pixel of each byte and bit 8 is the leftmost pixel of each byte. A bit value of zero is black and a bit value of one is white.

The following illustrates a bit mapped image as it would appear on a black and white LCD screen. The image has a horizontal size of 12 pixels and a vertical size of 10 pixels.



□ represents a bit value of 1
■ represents a bit value of 0

The following illustrates the raw bit map data of the previous image. Notice that the image data is two bytes wide and 10 bytes tall even though the size of the actual image is 12 pixels (12 bits) wide and 10 pixels tall. Notice also that the right most pixels are not used. These unused bits are required for the data to be byte aligned.



□ represents a bit value of 1
■ represents a bit value of 0
☒ represents an unused bit

.

5 Sample Source Code

The following sample source code is for Microsoft compilers only. A copy of this sample code with the necessary makefiles, along with similar sample source code and makefiles for Borland compilers, are installed during the 16 or 32 bit SDK installation. 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.

PadCom Signature Capture 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.h
//Library:..\libs\padcomdm.lib
//Environment:
//DOS 3.3 or better
//
// Copyright (c) Hand Held Products. All rights reserved.
//
//=============================================================================
//

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

#include "PadCom.h"

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

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

//=======================
```

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

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

//===========================
// START REAL TIME RECORDING
//===========================
padRecord();

//=============
// COLLECT DATA
//=============
while( !kbhit( ) )
{
// CHECK IF THERE IS MORE DATA FROM THE PAD
if( padUpdate() && padGet( &x, &y, &p ) )
{
// SCALE THE POINTS TO ABOUT DOUBLE In SIZE
padScaleDPI( &x, &y, 160, 160  );

// DRAW THEM ON THE SCREEN
if( p )
_lineto( x, y );
else
```

```
_moveto( x, y );
}
}

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

//======================
// TURN PAD OFF-CLEANUP
//======================
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 );
}
```

PadCom Signature Capture Sample for Windows 3.x:

```
//
//=--------------------------------------------------------------------------
//Test.c
//
//A simple WIN16 Demo program
```

```
//
//Compiler:
//Microsoft Visual C++ 1.52
//Target: Windows Application
//Memory Model: Medium
//Include:..\..\..\padcom.h
//Library:..\libs\padcomwm.lib
//Environment:
//Windows 3.x
//
//Copyright (c) Hand Held Products. All rights reserved.
//
//=---------------------------------------------------------------------------
//

#include "windows.h"

#include "PadCom.h"

intPASCALWinMain( HINSTANCE, HINSTANCE, LPSTR, int );
LONG __export CALLBACKWndProc( HWND, UINT, WPARAM, LPARAM );

HINSTANCEtheInstance;
HWNDtheWnd;

// ---------------------------------------------------------------------------
// 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= "TESTAPP";
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, "TESTAPP", "SIGN YOUR NAME THEN PRESS ANY BUTTON",
WS_OVERLAPPED | WS_SYSMENU, 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
if( padOn() )
{
// Start recording...
padRecord( hWnd );
}
else
{
// Error
MessageBox( hWnd, "Unable to find writing pad", "TESTAPP", MB_OK );
PostQuitMessage( 1 );
}

break;
}

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

// If pen is down, get pen position info and draw it
if( padGet( &x, &y, &p ) )
{
RECT Rect;
HDC hDC;
```

```
// Setup DC to perform scaling based on client rect
hDC = GetDC( hWnd );
GetClientRect( hWnd, &Rect );
SetMapMode( hDC, MM_ANISOTROPIC );
SetWindowExt( hDC, padWidth(), padHeight() );
SetViewportExt( hDC, Rect.right, Rect.bottom );

// If not first point of new stroke, draw stroke
if( p )
{
MoveTo( hDC, padOldX(), padOldY() );
LineTo( hDC, x, y );
}

ReleaseDC( hWnd, hDC );
}

}

return 1;
}

case WM_DESTROY:
{
// Always turn things off!
padStop();
padOff();

PostQuitMessage( 0 );

break;
}

case WM_CHAR:
{
DestroyWindow( hWnd );
break;
}

default:
```

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

return 0;
}
```

PadCom Signature Capture Sample for Windows95/NT

```
//
//=--------------------------------------------------------------------------
// Test.c
//
// A simple WIN32 Demo program
//
// Compiler:
//Microsoft Visual C++ 4.0
//Target: Windows Application
//Include:..\..\..\padcom.h
//Library:..\libs\padcomw.lib
//Environment:
//Windows 95/NT
//
// Copyright (c) Hand Held Products. All rights reserved.
//
//=--------------------------------------------------------------------------
//

#include <windows.h>
#include "PadCom.h"

// USER DEFINED PADCOM MESSAGE
#define WM_PADNOTIFYWM_USER + 100

int CALLBACK WinMain( HINSTANCE, HINSTANCE, LPSTR, int );
LONG CALLBACK WndProc( HWND, UINT, WPARAM, LPARAM );

HINSTANCE   theInstance;
HWNDtheWnd;
```

```
// ------------------------------------------------------------------------
// 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= "TEST";
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, "TEST", "Sign on the pad and hit any key when done",
WS_OVERLAPPED | WS_SYSMENU, 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
if( padOn() )
{
// Start recording...
padRecord( hWnd, WM_PADNOTIFY, 0,0 );

}
else
{
// Error
MessageBox( hWnd, "Unable to find writing pad", "TESTAPP", MB_OK );
PostQuitMessage( 1 );
}
```

```
break;
}

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

// If pen is down, get pen position info and draw it
if( padGet( &x, &y, &p ) )
{
RECT Rect;
HDC hDC;

// Setup DC to perform scaling based on client rect
hDC = GetDC( hWnd );
GetClientRect( hWnd, &Rect );
SetMapMode( hDC, MM_ANISOTROPIC );
SetWindowExtEx( hDC, padWidth(), padHeight(), NULL );
SetViewportExtEx( hDC, Rect.right, Rect.bottom, NULL );

// If not first point of new stroke, draw stroke
if( p )
{
MoveToEx( hDC, padOldX(), padOldY(), NULL );
LineTo( hDC, x, y );
}

ReleaseDC( hWnd, hDC );
}

}

return 1;
}

case WM_DESTROY:
```

```
{
// Always turn things off!
padStop();
padOff();

PostQuitMessage( 0 );

break;
}

case WM_CHAR:
{
DestroyWindow( hWnd );
break;
}

default:
{
return DefWindowProc( hWnd, Msg, wParam, lParam );
}
}
return 0;
}
```

PadCom Sample Source Code for the Magnetic Stripe Reader (MSR)

The sample code for reading an MSR attached to a Transaction Team device is provided for various platforms below. The source code is only supplied for demo purposes and is NOT a part of the SDK installation. There are numerous ways to capture MSR from Transaction Team devices and the demo code simply shows one of these. Hand Held Products does not necessarily claim that this is the best way to read MSR data.

PadCom MSR Sample Code for DOS:

```
//
// ****************************************************************************
// ****************************************************************************
// **                                                                        **
// **  MSR.C                                                                 **
// **                                                                        **
// **  This is a sample program showing how to use the Magnetic Stripe Reader    **
// ** hardware available on some Transaction Team devices such as the Transaction Team 3100 Series
```

```
**
// **                                                                    **
// **  Platform: DOS                                                     **
// **                                                                    **
// **  Compiler: Microsoft Visual C++ 1.52                               **
// **                                                                    **
// **                                                                    **
// **                                                                    **
// **  (C) Copyright 1999-2000 Hand Held Products.                                **
// **                                                                    **
// ************************************************************************
// ************************************************************************
//


//
// ************************************************************************
// *                                                                      *
// * Required header files                                                *
// *                                                                      *
// ************************************************************************
//
#include <stdio.h>
#include <conio.h>
#include "PadCom.H"

//
// ************************************************************************
// *                                                                      *
// * Constant values                                                      *
// *                                                                      *
// ************************************************************************
//

//
// Transaction Team' magnetic stripe reader follows the MEGTEK standard.
// This standard has the following track sizes defined.
// We add 1 to include the appended NULL character at the end of each track.
//
```

```
#define TRACK1_MAX (74+1)
#define TRACK2_MAX (39+1)
#define TRACK3_MAX (106+1)

//
// ****************************************************************************
// *                                                                          *
// * main - the application entry point                                       *
// *                                                                          *
// ****************************************************************************
//
void main ( void )
{
char
track1 [TRACK1_MAX],
track2 [TRACK2_MAX],
track3 [TRACK3_MAX];
int
dataReadFromTrack1 = 0,
dataReadFromTrack2 = 0,
dataReadFromTrack3 = 0;

printf ( "\nExecuting Transaction Team Sample program MSR.EXE\n\n" );

//
// ******************************************************
// * Step 1: Turn on the Transaction Team device using padConnect.*
// ******************************************************
//
if ( ! padConnect() )
{
printf ( "ERROR: No Transaction Team device found\n\n" );
printf ( "\n\nPress any key\n" );
while ( !kbhit () );
return;
}

printf ( "Please swipe your Magnetic Card\n\n" );
```

```
printf ( "(Press ESC to cancel)\n\n" );

//
// ******************************************************
// * Step 2: prepare the MSR to read data.              *
// ******************************************************
//
if ( !padGetMagTrack ( 0,0,0 ) )
{
printf ( "ERROR: No Magnetic Stripe Reader found\n\n" );
printf ( "\n\nPress any key\n" );
while ( !kbhit () );
padOff ();
return;
}

//
// Initialize the contents of the buffers.
//
strcpy ( track1, "NO DATA READ" );
strcpy ( track2, "NO DATA READ" );
strcpy ( track3, "NO DATA READ" );

while ( 1 )
{
//
// If the ESC key is pressed then return.
//
if ( kbhit () )
{
if ( getch () == 27 )
{
padOff ();
return;
}
}

//
```

```
// ******************************************************
// * Step 3: Check if any of the tracks were read.       *
// ******************************************************
//
dataReadFromTrack1 = padGetMagTrack ( 1, track1, TRACK1_MAX );
dataReadFromTrack2 = padGetMagTrack ( 2, track2, TRACK2_MAX );
dataReadFromTrack3 = padGetMagTrack ( 3, track3, TRACK3_MAX );

if
(
dataReadFromTrack1 ||
dataReadFromTrack2 ||
dataReadFromTrack3
)
{
//
// At this point we know that at least 1 track was read
// successfully (the last track read, i.e., track 3).
//
// It is possible one or more of the other tracks were
// checked before any data was retrieved from the card.
//
// As soon as one track contains data all of the other
// tracks will as well.
//
// To make sure we get all of the data from all of the tracks
// we will read all of them again.
//

//
// ******************************************************
// * Step 4: Read all of the tracks                      *
// ******************************************************
// Note: step 4 is not needed if only one track is being read
//
dataReadFromTrack1 = padGetMagTrack ( 1, track1, TRACK1_MAX );
dataReadFromTrack2 = padGetMagTrack ( 2, track2, TRACK2_MAX );
dataReadFromTrack3 = padGetMagTrack ( 3, track3, TRACK3_MAX );
```

```
    //
    // print out the results
    //
    printf ( "Data received from Magnetic Card:\n\n");
    printf ( "Track 1 size: %d\n", dataReadFromTrack1 );
    printf ( "Track 1 data: %s\n\n", track1 );
    printf ( "Track 2 size: %d\n", dataReadFromTrack2 );
    printf ( "Track 2 data: %s\n\n", track2 );
    printf ( "Track 3 size: %d\n", dataReadFromTrack3 );
    printf ( "Track 3 data: %s\n\n", track3 );
    printf ( "\n\nPress any key\n" );
    while ( !kbhit () );

    //
    // Turn of the Transaction Team device and quit the test application
    //
    padOff ();
    return;
    }
    }
    }
```

PadCom MSR Sample Code for Win 3.x

```
//
// ***************************************************************************
// ***************************************************************************
// **                                                **
// **  MSR.C                                             **
// **                                                **
// **  This is a sample program showing how to use the Magnetic Stripe Reader    **
// **  hardware available on some Transaction Team devices such as the Transaction Team 3100 Series      **
// **                                                **
// **  Platform: Windows 3.1                                    **
// **                                                **
// **  Compiler: Microsoft Visual C++ 1.52                              **
// **                                                **
```

```
// **                                                    **
// **                                                    **
// **  (C) Copyright 1999-2000 Hand Held Products.                    **
// **                                                    **
// ************************************************************************
// ************************************************************************
//



//
// ************************************************************************
// *                                                    *
// * Required header files                                    *
// *                                                    *
// ************************************************************************
//
#include <windows.h>

#include <string.h>
#include <stdio.h>
#include "PadCom.H"

//
// ************************************************************************
// *                                                    *
// * Constant values                                        *
// *                                                    *
// ************************************************************************
//


//
// Transaction Team' magnetic stripe reader follows the MEGTEK standard.
// This standard has the following track sizes defined.
// We add 1 to include the appended NULL character at the end of each track.
//
```

```
#define TRACK1_MAX (74+1)
#define TRACK2_MAX (39+1)
#define TRACK3_MAX (106+1)

#define MSR_TIMER_ID    1
#define MSR_DATA_READ1


//
// ****************************************************************************
// *                                                          *
// * Function prototypes                                              *
// *                                                          *
// ****************************************************************************
//
int             PASCAL                  WinMain          ( HINSTANCE,HINSTANCE, LPSTR, int );
LONG __export CALLBACKWndProc        ( HWND,   UINT, WPARAM, LPARAM );
void CALLBACK                   TimerProc ( HWND,   UINT, UINT, DWORD );
int                                     ReadMSR         ( void );
int                                     ResetMSR( HWND );

//
// ****************************************************************************
// *                                                          *
// * Global variables                                                 *
// *                                                          *
// ****************************************************************************
//
HINSTANCE     theInstance;
HWND                    theWnd;
TIMERPROC     lpfnMyTimerProc;

char *appName         = "MSR";
char *appTitle          = "MSR.EXE - Please swipe your Magnetic Card";
```

```
char
        track1 [TRACK1_MAX],
        track2 [TRACK2_MAX],
        track3 [TRACK3_MAX];
int
        dataReadFromTrack1 = 0,
        dataReadFromTrack2 = 0,
        dataReadFromTrack3 = 0;

//
// ****************************************************************************
// *                                                          *
// * WinMain - the application entry point                                    *
// *                                                          *
// ****************************************************************************
//
int PASCAL WinMain
(
        HINSTANCEhInst,
        HINSTANCEhInstPrev,
        LPSTR     lpstrCmdLine,
        int             cmdShow
)
{
        MSG             msg;
        WNDCLASSwc;

        theInstance= hInst;

        //
        // Register the window class if this is the first instance.
        //
        if( !hInstPrev )
        {
                wc.lpszMenuName= NULL;
```

```
        wc.lpszClassName= appName;
        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;
    }

//
    // Attemp to create the main window
    //
    theWnd =CreateWindowEx
            (
                WS_EX_TOPMOST, appName, appTitle, WS_OVERLAPPED | WS_SYSMENU,
                CW_USEDEFAULT, CW_USEDEFAULT, 375, 225,NULL, NULL,hInst, NULL
            );

    //
    // If the window was not created then quit
    //
    if ( !theWnd )
    {
        return 0;
    }

    //
    // Show the main window
    //
    ShowWindow( theWnd, cmdShow );
```

```
        UpdateWindow( theWnd );

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

        return 0;
}



//
// ****************************************************************************
// *                                                                          *
// * WndProc - Message handler for the application                            *
// *                                                                          *
// ****************************************************************************
//
LONG __export CALLBACK WndProc(
        HWND hWnd, UINT Msg, WPARAM wParam, LPARAM lParam
)
{
        char msrData [1024];

        switch( Msg )
        {
            case WM_CREATE:
            {
                //
                // ******************************************************
```

```
        // * Step 1: Turn on the Transaction Team device using padConnect.*
        // ******************************************************
        //
        if( padConnect() )
        {
                if ( !ResetMSR ( hWnd ) )
                {
                        PostQuitMessage( 1 );
                }
        }
        else
        {
                //
                // Error
                //
                MessageBox( hWnd, "ERROR: No Transaction Team device found!", appName, MB_OK );
                PostQuitMessage( 1 );
        }

        break;
}

case WM_COMMAND:
{
        switch ( wParam )
        {
                case MSR_DATA_READ:
                {
                        //
                        // Display the data read from the Magnetic Stripe Reader
                        //
                        sprintf
                        (
                                msrData,
                                "Track 1 size: %d\n"
```

```
                                "Track 1 data: %s\n\n"
                                "Track 2 size: %d\n"
                                "Track 2 data: %s\n\n"
                                "Track 3 size: %d\n"
                                "Track 3 data: %s\n\n",

                                dataReadFromTrack1,
                                track1,
                                dataReadFromTrack2,
                                track2,
                                dataReadFromTrack3,
                                track3
                        );

                        MessageBox( hWnd, msrData, "MSR.EXE - Data received from the Magnetic
Card", MB_OK );

                        //
                        // Reset
                        //
                        if ( !ResetMSR ( hWnd ) )
                        {
                                PostQuitMessage( 1 );
                        }
                }
        }
        break;
}

case WM_DESTROY:
{
        //
        // Turn of the Transaction Team device
        //
        padOff();
```

```
                PostQuitMessage( 0 );

                break;
            }

            case WM_CHAR:
            {
                DestroyWindow( hWnd );
                break;
            }

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

        return 0;
}


//
// ****************************************************************************
// *                                                                          *
// * TimerProc - Message handler for timer used to read the MSR               *
// *                                                                          *
// ****************************************************************************
//
void CALLBACK  TimerProc( HWND hWnd, UINT Msg, UINT idTime, DWORD dwTime )
{
        KillTimer ( hWnd, idTime );

        switch( Msg )
        {
```

```
                case WM_TIMER:
                {
                        switch ( idTime )
                        {
                                case MSR_TIMER_ID:
                                {
                                        if ( ReadMSR () )
                                        {
                                                PostMessage ( hWnd, WM_COMMAND, MSR_DATA_READ, 0 );
                                                return;
                                        }
                                }
                        }
                }

                SetTimer(hWnd, MSR_TIMER_ID, 1000, lpfnMyTimerProc);
}

//
// ****************************************************************************
// *                                                                          *
// * ResetMSR - initializes the MSR and sets up a timer to pole for MSR data  *
// *                                                                          *
// ****************************************************************************
//
int ResetMSR ( HWND hWnd )
{
                //
                // ******************************************************
                // * Step 2: prepare the MSR to read data.              *
                // ******************************************************
                //
                if ( padGetMagTrack ( 0,0,0 ) )
                {
```

```
                //
                // Initialize the contents of the buffers.
                //
                strcpy ( track1, "NO DATA READ" );
                strcpy ( track2, "NO DATA READ" );
                strcpy ( track3, "NO DATA READ" );

                //
                // In Windows it is not good to use "do" or "while" loops.
                // Instead we will use a timer to pole the MSR for data
                //
                lpfnMyTimerProc = (TIMERPROC) MakeProcInstance( (FARPROC) TimerProc,
theInstance );
                SetTimer(hWnd, MSR_TIMER_ID, 1000, lpfnMyTimerProc);

                return 1;
        }
        {
                MessageBox( hWnd, "ERROR: Unable to initialize Magnetic Stripe Reader!",
appName, MB_OK );

                return 0;
        }
}



//
// ****************************************************************************
// *                                                          *
// * ReadMSR - reads the MSR data if any is available                    *
// *                                                          *
// ****************************************************************************
//
int ReadMSR ( void )
{
```

```
//
// ******************************************************
// * Step 3: Check if any of the tracks were read.      *
// ******************************************************
//
dataReadFromTrack1 = padGetMagTrack ( 1, track1, TRACK1_MAX );
dataReadFromTrack2 = padGetMagTrack ( 2, track2, TRACK2_MAX );
dataReadFromTrack3 = padGetMagTrack ( 3, track3, TRACK3_MAX );

if
(
      dataReadFromTrack1 ||
      dataReadFromTrack2 ||
      dataReadFromTrack3
)
{
      //
      // At this point we know that at least 1 track was read
      // successfully (the last track read, i.e., track 3).
      //
      // It is possible one or more of the other tracks were
      // checked before any data was retrieved from the card.
      //
      // As soon as one track contains data all of the other
      // tracks will as well.
      //
      // To make sure we get all of the data from all of the tracks
      // we will read all of them again.
      //

      //
      // ******************************************************
      // * Step 4: Read all of the tracks                     *
      // ******************************************************
      // Note: step 4 is not needed if only one track is being read
```

```
                //
                dataReadFromTrack1 = padGetMagTrack ( 1, track1, TRACK1_MAX );
                dataReadFromTrack2 = padGetMagTrack ( 2, track2, TRACK2_MAX );
                dataReadFromTrack3 = padGetMagTrack ( 3, track3, TRACK3_MAX );

                return 1;
            }
            return 0;
}
```

PadCom MSR Sample Code for Win 95/NT:

```
//
// ****************************************************************************
// ****************************************************************************
// **                                                    **
// **  MSR.C                                                **
// **                                                    **
// **  This is a sample program showing how to use the Magnetic Stripe Reader    **
// **  hardware available on some Transaction Team devices such as the Transaction Team 3100 Series      **
// **                                                    **
// **  Platform: Windows 95/NT                                    **
// **                                                    **
// **  Compiler: Microsoft Visual C++ 4.0                              **
// **                                                    **
// **                                                    **
// **                                                    **
// **  (C) Copyright 1999-2000 Hand Held Products.                          **
// **                                                    **
// ****************************************************************************
// ****************************************************************************
//



//
// ****************************************************************************
// *                                                     *
```

```
// * Required header files                                    *
// *                                                          *
// ************************************************************************
//
#include <windows.h>

#include <string.h>
#include <stdio.h>
#include "PadCom.H"

//
// ************************************************************************
// *                                                          *
// * Constant values                                          *
// *                                                          *
// ************************************************************************
//

//
// Transaction Team' magnetic stripe reader follows the MEGTEK standard.
// This standard has the following track sizes defined.
// We add 1 to include the appended NULL character at the end of each track.
//
#define TRACK1_MAX (74+1)
#define TRACK2_MAX (39+1)
#define TRACK3_MAX (106+1)

#define MSR_TIMER_ID    1
#define MSR_DATA_READ1



//
// ************************************************************************
// *                                                          *
// * Function prototypes                                      *
```

```
// *                                                    *
// ********************************************************************************
//
int         WINAPI              WinMain         ( HINSTANCE,HINSTANCE, LPSTR, int );
LRESULT                             WndProc         ( HWND,UINT, WPARAM, LPARAM );
void CALLBACK                   TimerProc ( HWND,   UINT, UINT, DWORD );
int                                 ReadMSR         ( void );
int                                 ResetMSR( HWND );

//
// ********************************************************************************
// *                                                    *
// * Global variables                                       *
// *                                                    *
// ********************************************************************************
//
HINSTANCE    theInstance;
HWND                theWnd;
TIMERPROC    lpfnMyTimerProc;

char *appName       = "MSR";
char *appTitle      = "MSR.EXE - Please swipe your Magnetic Card";

char
            track1 [TRACK1_MAX],
            track2 [TRACK2_MAX],
            track3 [TRACK3_MAX];
int
            dataReadFromTrack1 = 0,
            dataReadFromTrack2 = 0,
            dataReadFromTrack3 = 0;

//
// ********************************************************************************
// *                                                    *
```

```
// * WinMain - the application entry point                    *
// *                                                          *
// ***************************************************************************
//
int WINAPI WinMain
(
        HINSTANCEhInst,
        HINSTANCEhInstPrev,
        LPSTR     lpstrCmdLine,
        int               cmdShow
)
{
        MSG               msg;
        WNDCLASSwc;

        theInstance= hInst;

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

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

```
        }

//
        // Attemp to create the main window
        //
        theWnd =CreateWindowEx
                (
                    WS_EX_TOPMOST, appName, appTitle, WS_OVERLAPPED | WS_SYSMENU,
                    CW_USEDEFAULT, CW_USEDEFAULT, 375, 225,NULL, NULL,hInst, NULL
                );

        //
        // If the window was not created then quit
        //
        if ( !theWnd )
        {
            return 0;
        }

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

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

        return 0;
```

```
}



//
// ****************************************************************************
// *                                                                          *
// * WndProc - Message handler for the application                            *
// *                                                                          *
// ****************************************************************************
//
LRESULT WndProc(
        HWND hWnd, UINT Msg, WPARAM wParam, LPARAM lParam
)
{
        char msrData [1024];

        switch( Msg )
        {
            case WM_CREATE:
            {
                //
                // ******************************************************
                // * Step 1: Turn on the Transaction Team device using padConnect.*
                // ******************************************************
                //
                if( padConnect() )
                {
                    if ( !ResetMSR ( hWnd ) )
                    {
                        PostQuitMessage( 1 );
                    }
                }
                else
                {
```

```
            //
            // Error
            //
            MessageBox( hWnd, "ERROR: No Transaction Team device found!", appName, MB_OK );
            PostQuitMessage( 1 );
        }

        break;
}

case WM_COMMAND:
{
        switch ( wParam )
        {
            case MSR_DATA_READ:
            {
                //
                // Display the data read from the Magnetic Stripe Reader
                //
                sprintf
                (
                    msrData,
                    "Track 1 size: %d\n"
                    "Track 1 data: %s\n\n"
                    "Track 2 size: %d\n"
                    "Track 2 data: %s\n\n"
                    "Track 3 size: %d\n"
                    "Track 3 data: %s\n\n",

                    dataReadFromTrack1,
                    track1,
                    dataReadFromTrack2,
                    track2,
                    dataReadFromTrack3,
                    track3
```

```
                                        );

                                        MessageBox( hWnd, msrData, "MSR.EXE - Data received from the Magnetic
Card", MB_OK );

                                        //
                                        // Reset
                                        //
                                        if ( !ResetMSR ( hWnd ) )
                                        {
                                                PostQuitMessage( 1 );
                                        }
                                }
                        }
                        break;
                }

                case WM_DESTROY:
                {
                        //
                        // Turn of the Transaction Team device
                        //
                        padOff();

                        PostQuitMessage( 0 );

                        break;
                }

                case WM_CHAR:
                {
                        DestroyWindow( hWnd );
                        break;
                }
```

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

        return 0;
}



//
// ************************************************************************
// *                                                                      *
// * TimerProc - Message handler for timer used to read the MSR           *
// *                                                                      *
// ************************************************************************
//
void CALLBACK  TimerProc( HWND hWnd, UINT Msg, UINT idTime, DWORD dwTime )
{
        KillTimer ( hWnd, idTime );

        switch( Msg )
        {
                case WM_TIMER:
                {
                        switch ( idTime )
                        {
                                case MSR_TIMER_ID:
                                {
                                        if ( ReadMSR () )
                                        {
                                                PostMessage ( hWnd, WM_COMMAND, MSR_DATA_READ, 0 );
                                                return;
                                        }
                                }
```

```
                        }
                    }
                }

                SetTimer(hWnd, MSR_TIMER_ID, 1000, lpfnMyTimerProc);
}

//
// ****************************************************************************
// *                                                                          *
// * ResetMSR - initializes the MSR and sets up a timer to pole for MSR data  *
// *                                                                          *
// ****************************************************************************
//
int ResetMSR ( HWND hWnd )
{
                //
                // ******************************************************
                // * Step 2: prepare the MSR to read data.              *
                // ******************************************************
                //
                if ( padGetMagTrack ( 0,0,0 ) )
                {
                        //
                        // Initialize the contents of the buffers.
                        //
                        strcpy ( track1, "NO DATA READ" );
                        strcpy ( track2, "NO DATA READ" );
                        strcpy ( track3, "NO DATA READ" );

                        //
                        // In Windows it is not good to use "do" or "while" loops.
                        // Instead we will use a timer to pole the MSR for data
                        //
                        lpfnMyTimerProc = (TIMERPROC) MakeProcInstance( (FARPROC) TimerProc,
```

```
theInstance );
                SetTimer(hWnd, MSR_TIMER_ID, 1000, lpfnMyTimerProc);

                return 1;
        }
        {
                MessageBox( hWnd, "ERROR: Unable to initialize Magnetic Stripe Reader!",
appName, MB_OK );

                return 0;
        }
}


int ReadMSR ( void )
{
        //
        // ******************************************************
        // * Step 3: Check if any of the tracks were read.       *
        // ******************************************************
        //
        dataReadFromTrack1 = padGetMagTrack ( 1, track1, TRACK1_MAX );
        dataReadFromTrack2 = padGetMagTrack ( 2, track2, TRACK2_MAX );
        dataReadFromTrack3 = padGetMagTrack ( 3, track3, TRACK3_MAX );

        if
        (
                dataReadFromTrack1 ||
                dataReadFromTrack2 ||
                dataReadFromTrack3
        )
        {
                //
                // At this point we know that at least 1 track was read
                // successfully (the last track read, i.e., track 3).
```

```
            //
            // It is possible one or more of the other tracks were
            // checked before any data was retrieved from the card.
            //
            // As soon as one track contains data all of the other
            // tracks will as well.
            //
            // To make sure we get all of the data from all of the tracks
            // we will read all of them again.
            //

            //
            // ********************************************************
            // * Step 4: Read all of the tracks                       *
            // ********************************************************
            // Note: step 4 is not needed if only one track is being read
            //
            dataReadFromTrack1 = padGetMagTrack ( 1, track1, TRACK1_MAX );
            dataReadFromTrack2 = padGetMagTrack ( 2, track2, TRACK2_MAX );
            dataReadFromTrack3 = padGetMagTrack ( 3, track3, TRACK3_MAX );

            return 1;
        }
        return 0;
}
```

4619 Jordan Road
P.O. Box 187
Skaneateles Falls, New York 13153-0187