

# *posVisual*

Programmer Manual 98010

---



# User License

---

posVisual Copyright 1999, 2000 @pos.com

posVisual is a trademark of @pos.com

The software may only be used or copied in accordance with the terms of the license. It is against the law to copy the software on any medium except as specifically allowed in the license. No part of this manual may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or information storage and retrieval systems, for any purpose other than the individual licensees use, with out the express written permission of @pos.com .

Companies, names and data used herein are fictitious unless otherwise noted.

Visual Basic, Windows 95, Windows NT are trademarks of Microsoft Corporation. Other product names mentioned in this document may be trademarks or registered trademarks of other companies and are the property of their respective owners.

@pos.com makes no warranties, either express or implied, fitness for a particular purpose, title, and non-infringement, with regard to the SOFTWARE PRODUCT, and the provision of or failure to provide Support Services. This limited warranty gives you specific legal rights. You may have others, which vary from state/ jurisdiction to state/jurisdictions.

# Table of Contents

---

<b>USER LICENSE</b>	<b>III</b>
<b>TABLE OF CONTENTS</b>	<b>IV</b>
<b>INTRODUCTION 1</b>	<b>1</b>
<i>Functions</i> .....	1
<i>Benefits</i> .....	1
<i>Modes</i> .....	1
Script Mode .....	1
VB Mode .....	2
<i>Why use posVisual?</i> .....	2
<i>Other @pos.com Development Software</i> .....	2
<i>posVisual Users</i> .....	3
System Requirements .....	3
Installation of posVisual .....	3
The steps for installing of posVisual are the following:.....	3
Installed Files.....	4
<b>DEVELOPER ENVIRONMENT 2</b>	<b>6</b>
<i>Starting posVisual Projects</i> .....	6
To start building a new posVisual project:.....	6
To continue an existing posVisual Project: .....	6
<i>The posVisual Assistant</i> .....	8
Script Mode .....	8
<i>The Controls on posVisual Assistant</i> .....	8
About Button .....	8
Mode Indicator.....	8
Status Line .....	8
Check Box .....	8
Message Window .....	9
Clear Button.....	9
List All Labels Button .....	9
Put Script on Device Button.....	9
Modify Control Button .....	9
<i>Controls for posVisual Forms</i> .....	9
<b>POSVISUAL FORMS 3</b>	<b>11</b>
<i>Adding posVisual Forms</i> .....	11
<i>Pad Form Events</i> .....	11
<i>posVisual Form Properties</i> .....	12
<b>PROGRAMMING MODES OVERVIEW 4</b>	<b>14</b>
<i>Comparison of posVisual Modes</i> .....	14
<i>Deciding Which posVisual Mode to Use</i> .....	14
Script Mode .....	14

<b>SCRIPT MODE 5</b>	<b>15</b>
<i>What are Scripts?</i> .....	15
<i>Operating a Script</i> .....	15
<i>Screen Creation Process in Script Mode</i> .....	16
<i>Script Mode Software Components</i> .....	16
Controls .....	17
Forms .....	17
Modules .....	17
References .....	17
Downloading the Designed Script to the iPOS Terminal .....	17
Operational Host Communication with Scripts .....	18
Host to Script Communication .....	18
Host-iPOS Terminal Command Traffic .....	19
<i>Transaction Events</i> .....	19
SCRIPT CONTROLS.....	21
posVisual Tool Box.....	21
Control Font Sizes .....	22
<i>Command Button</i> .....	22
Using the Control .....	22
<i>Image</i> .....	22
Using the Control .....	22
<i>Label</i> .....	23
Using the Control .....	23
<i>TextBox</i> .....	<a href="#">23</a>
<i>Line</i> .....	23
<i>ListBox</i> .....	23
Using the Control .....	24
<i>MSRControl</i> .....	24
EVENTS:.....	24
Using the Control .....	24
<i>SigControl</i> .....	24
Using the Control .....	25
<i>Shape</i> .....	25
<i>Timer</i> .....	25
Using the Control .....	25
SCRIPT COMMANDS.....	26
<i>Command Prefixes</i> .....	26
<i>Declaring Variables</i> .....	26
<i>Script Command Generation</i> .....	27
POSVISUAL OBJECTS, PROPERTIES AND METHODS (SCRIPT MODE) .....	29
TOOL .....	29
DISP .....	32
SCRIPT .....	35
SIG .....	39
SOUND.....	43
STOPWATCH.....	44
TXTBOX .....	45
UNIT.....	47
VAR.....	48
VAR.BIN.....	49
VAR.NUM.....	50
VAR.STR.....	53
<b>VB MODE 6</b>	<b>56</b>
<i>When to use VB mode</i> .....	56
<i>Developing applications in the VB mode</i> .....	56
<i>Distributing posVisual Applications</i> .....	56

<i>VB Mode Software Components</i> .....	57
<i>Class</i> .....	58
<i>Controls</i> .....	58
<i>Forms</i> .....	58
<i>Modules</i> .....	58
<i>References</i> .....	58
<b>VB MODE CONTROLS</b> .....	59
<i>posVisual Tool Box</i> .....	59
<i>Control Fonts</i> .....	59
<i>CheckBox</i> .....	60
Using the Control .....	60
See also .....	60
<i>Command Button</i> .....	60
Using the Control .....	61
<i>Image</i> .....	61
Using the Control .....	61
<i>Label</i> .....	62
Using the Control .....	62
See also .....	62
<i>Line</i> .....	62
<i>ListBox</i> .....	63
Using the Control .....	63
<i>MSRControl</i> .....	63
Using the Control .....	64
See also .....	64
<i>SigControl</i> .....	64
Using the Control .....	64
See also .....	64
<i>Shape</i> .....	64
<b>VB MODE COMMANDS</b> .....	66
<b>V.A.</b> .....	67
<b>MSR</b> .....	77
<b>VAEVENT CLASS MODULE</b> .....	79
<b>ERROR MESSAGES 7</b> .....	<b>81</b>
<i>VB Mode Error Messages</i> .....	81
<i>Script Mode Error Messages</i> .....	81

# Introduction

---

# 1

posVisual provides two important benefits to developers building @pos.com interactive Point of Sale (iPOS) applications. It speeds programming and eases the task of integrating @pos.com terminals into existing software applications.

## FUNCTIONS

Based on the popular Visual Basic programming environment, posVisual provides simple tools to integrate command buttons, signature capture boxes, graphics, MSR data capture, text boxes, list boxes and check boxes onto @pos.com iPOS terminal screens. posVisual enables developers to quickly create, modify and add new screens and capabilities to @pos.com iPOS terminals.

## BENEFITS

Designing the appearance of an iPOS terminal screen is as simple as using a drawing program. The graphical nature of the posVisual programming environment saves development time. Elements such as boxes, buttons, pictures and lines can be created easily with Visual Basic controls to compose ads and graphics. Controls added to Visual Basic by posVisual enable developers to create applications that capture signatures, read credit cards, and display forms for user input. With the ease of creating and programming display screens using posVisual, developers can build and update their systems in a timely manner.

Developers writing in C, C++ and other languages benefit especially when using posVisual to integrate @pos.com iPOS terminals with existing applications. With posVisual, developers can create applications that reside and run on the @pos.com iPOS terminals. Existing C and C++ programs trigger iPOS resident applications. When the iPOS application runs to completion, it can return information such as signature or magnetic stripe values to the triggering application. posVisual allows creation of @pos.com iPOS resident code that minimizes disruption to existing host code.

## MODES

posVisual offers two modes of programming: **Script Mode** and **VB Mode**. Both modes rely on the Visual Basic environment for code generation.

### ***SCRIPT MODE***

Programmers typically employ Script mode when iPOS terminals and programs must integrate into existing programs written in C, C++ or other languages. One common example of that is retail point of sale, where new interactive applications must integrate with proprietary retail sales applications and dated equipment. Script mode creates new applications that reside and execute on the iPOS terminals and are triggered by existing retail applications.

***VB MODE***

Programmers typically employ VB mode for building stand-alone Visual Basic applications that control the iPOS terminal or integrating with existing Visual Basic applications that reside and execute on a host such as a Windows 95/98/NT PC. A common Visual Basic stand-alone application is security sign-in.

The flexibility inherent in posVisual allows developers to place application logic where it's most appropriate, either on a host or on the iPOS terminal. Such flexibility allows @pos.com transaction applications to be created for practically any hardware and software configuration.

**WHY USE POSVISUAL?**

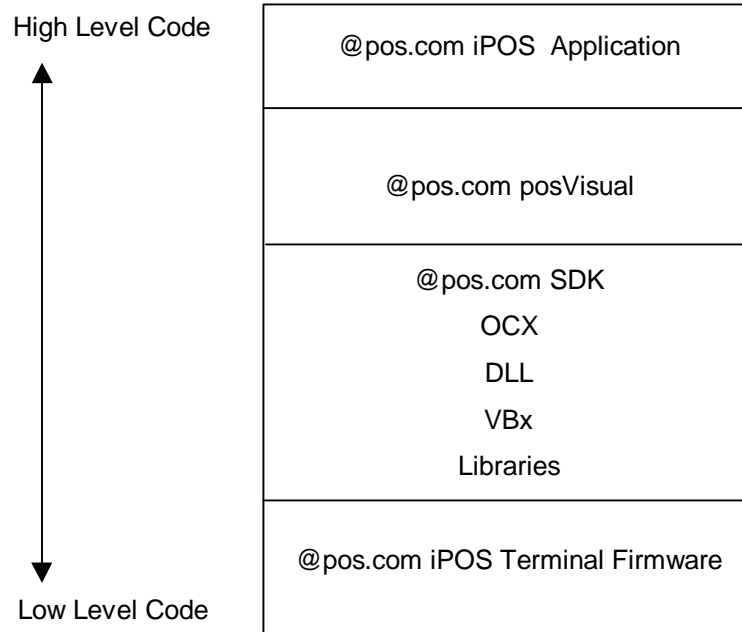
Use posVisual to efficiently write applications for @pos.com iPOS terminals. The most important uses for posVisual are the following:

- As a tool for creating iPOS resident scripts running on @pos.com iPOS terminals
- As a tool for integrating @pos.com iPOS terminals and transaction software into existing systems.
- As a tool to quickly modify projects and produce stable code for @pos.com transaction applications
- As a tool to quickly create iPOS proof of concept demonstrations

**OTHER @POS.COM DEVELOPMENT SOFTWARE.**

@pos.com Software Development Kit (SDK) is included with posVisual . The SDK contains other development components such as libraries for @pos.com iPOS terminal integration, objects for iPOS terminal control and key technologies such as compression and encryption. A small part of the SDK is required if integrating iPOS applications into existing retail environments (such as IBM 468X / 469X) or need additional features such as compression, encryption or debit key support. Additional drivers for OPOS, IBM POS and DOS as well as utilities for handling debit keys, compressing data, and encrypting data are also provided by the SDK.

posVisual is a shell that insulates developers from lower level coding that is required if using the @pos.com SDK directly. The following diagram shows the relationship between posVisual the SDK.



Developers of stand-alone Visual Basic PC applications may only need posVisual and not the SDK. Many developers use both the SDK and posVisual . The SDK supplies developers with special software components such as encryption routines and POS libraries while posVisual provides an environment for rapid application development.

## POSVISUAL USERS

The intended users are application developers and integrators building systems that contain @pos.com iPOS terminals or extending systems to include @pos.com terminals. Users of posVisual need familiarity with Visual Basic.

### **SYSTEM REQUIREMENTS**

The system requirements for posVisual are the following:

- Windows PC with 100 MHz or faster Pentium microprocessor.
- 16 MB of RAM (32 MB recommended) and 10 MB free hard disk space
- @pos.com iPOS terminal
- Visual Basic 5.0 with Service Pak 3

### **INSTALLATION OF POSVISUAL**

The installation procedure for posVisual is automated. The install program copies posVisual files to the hard disk then notifies Visual Basic of the location of the files.

### **THE STEPS FOR INSTALLING OF POSVISUAL ARE THE FOLLOWING:**

1. Insert the CD-ROM into a development PC containing Visual Basic 5.0.

2. Close all open programs.
3. Double-click on the CD-ROM icon. The CD-ROM window opens.
4. Run SETUP.EXE.
5. Follow the instruction provided by the installation wizard.

### **INSTALLED FILES**

The files installed on the developmental PC are listed in the following table:

<b>Type</b>	<b>Item</b>	<b>Description</b>	<b>Location</b>
Application	mxVisual.dll		Program Directory
DLLs	mxSptool.dll	Downloads scripts and supports graphics	System Directory
	mxScript.dll	Script commands	System Directory
	mxVAhost.dll	Host required DLL for host mode	System Directory
OCX	Sigbox.ocx	Host mode connectivity	System Directory
	VisualControls.ocx	MSR + Sig Controls	System Directory
Text Files	scriptdescriber.txt	Definitions for script commands	Program Directory
VB Mode Templates	MDIMain.frm (read only)	MDI form that holds all screens	Program\Templates\posVisual\VB
	posVisualHost.vbp	Project files template	Program\Templates\ posVisual \VB
	frmSplash.frm	Application specific splash screen	Program\Templates\ posVisual \VB
	padStart.frm	Initial Screen	Program\Templates\ posVisual \VB
	VEvents.cls	User can code these posVisual generated events	Program\Templates\ posVisual \VB
	modMain.bas	Main code module that connects classes	Program\Templates\ posVisual \VB
SCRIPT Mode Templates	posVisual Script.vbp	Template project for Script mode	Program\Templates\ posVisual \SCRIPT
	MainScript.bas	Creates the main TOOL class and user code area	Program\Templates\ posVisual \SCRIPT
	padStart.frm	First screen	Program\Templates\ posVisual \SCRIPT
FORM Template	posVisual.frn	Form used in both modes	Program\Templates\FORMS
<b>Type</b>	<b>Item</b>	<b>Description</b>	<b>Location</b>
Output Files	script.bin	Temp BIN file for Script loading, Script file	...\TEMP or C:\

Admin Program	Admin.exe	This file does the following: Creates the posVisual Assistant entry in the VB Addin.ini file. Copies the posVisual Templates to the VB Templates directory This program can be run at any time.	Program\Templates
---------------	-----------	--	-------------------

**Note:** All template files copy automatically to the VB program template directory.

# Developer Environment 2

---

posVisual leverages the powerful Visual Basic facilities to provide a programming environment for generating applications that incorporate @pos.com iPOS terminals.

**Note:** Familiarity with Visual Basic is recommended before attempting to use posVisual.

All Visual Basic applications, including posVisual applications, take the form of projects. A project is the master file for an application. It can contain three other files: a file containing forms, a file containing modules and a file containing classes. A project can be started in one of two modes: VB or Script.

## STARTING POSVISUAL PROJECTS

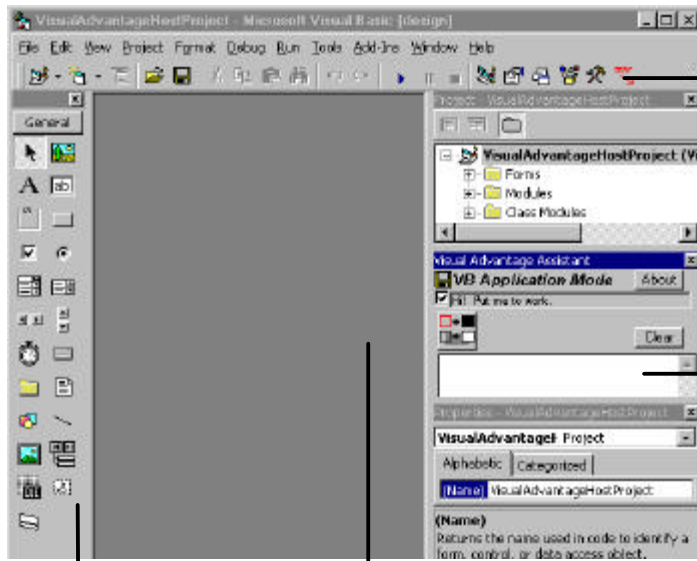
### *TO START BUILDING A NEW POSVISUAL PROJECT:*

1. In Visual Basic, open a blank project.
2. From the Add-Ins menu, select the Add-in Manager, click on the check box in front of @pos.com posVisual Assistant then click OK. The posVisual Assistant appears.
3. Select one of the two posVisual modes available from posVisual Message box: Visual Basic EXE Mode or Script Mode. Double-click the forms folder in the Project window. The Forms folder opens.
4. Double-click padStart form. The padStart form appears in the Designer window. All projects begin with the padStart form.

### *TO CONTINUE AN EXISTING POSVISUAL PROJECT:*

1. In Visual Basic click on the File menu and select Open. The Open message box appears. The Open message box appears.
2. Select the file from the list and click OK. The file opens.
3. From the Add-Ins menu click on the check box in front of @pos.com posVisual Assistant. The file menu appears.
4. Click on the check box in front of @pos.com posVisual Assistant then select OK. The posVisual Assistant appears.

**Note:** It is not necessary to activate the posVisual Assistant when opening an existing VB mode project..



Visual Basic Toolbox

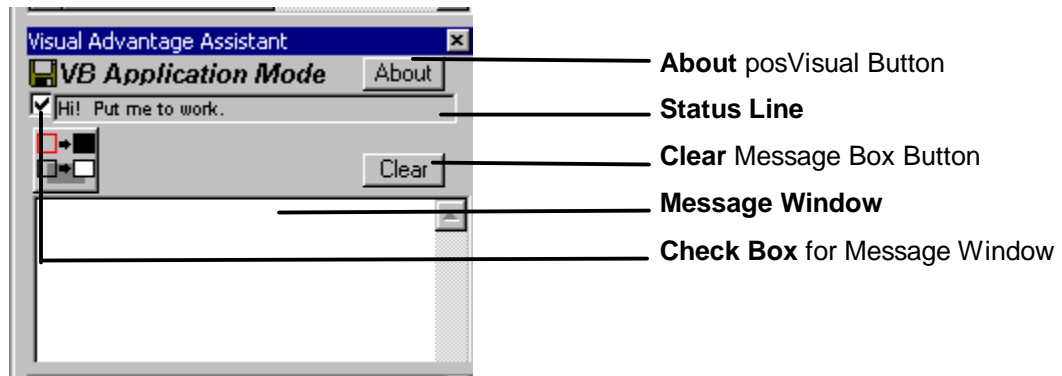
Form Designer

**Assistant Display button.** If the posVisual Assistant tool window has been closed, open it by clicking on the red button

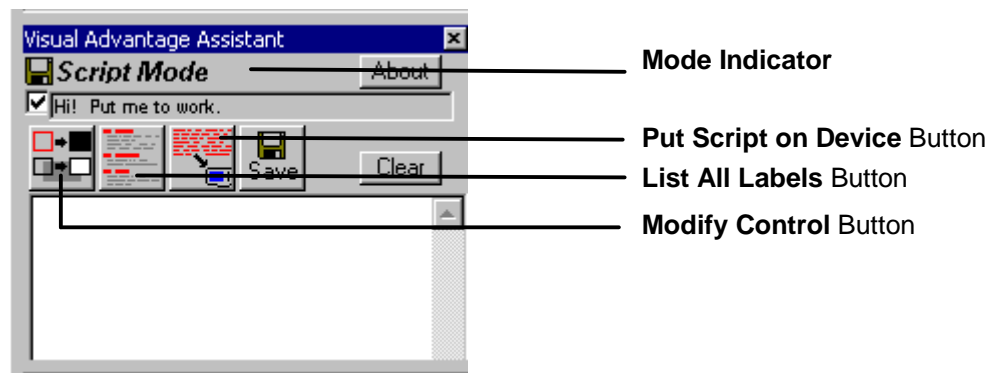
**posVisual Assistant tool window** is dockable and can be placed where convenient. When first activated posVisual appears in the upper left of the form designer.

## THE POSVISUAL ASSISTANT

The posVisual Assistant takes two different forms depending on the operating mode as shown below.



### SCRIPT MODE



## THE CONTROLS ON POSVISUAL ASSISTANT

The controls and indicators needed to manipulate posVisual Assistant capabilities are the following:

### **ABOUT BUTTON**

The About button works in both modes and when clicked, calls the About posVisual message box. The message box shows the version number of posVisual and presents a button that calls the Microsoft System Information message box.

### **MODE INDICATOR**

The Mode Indicator shows the mode (either Script or VB) that Visual Assistant is working in. (See Mode chapter)

### **STATUS LINE**

The Status line shows process steps that posVisual is taking as they are occurring.

### **CHECK BOX**

The check box toggles the status line and error sound on and off.

**MESSAGE WINDOW**

The Message window summarizes the results of processes, such as compilation errors or the successful loading of a script.

**CLEAR BUTTON**

The button erases the Message Window and Status line.

**LIST ALL LABELS BUTTON**

The List All Labels Button only appears in the Script mode. It lists user defined sub-routines and iPOS screens in the Message Window.

**PUT SCRIPT ON DEVICE BUTTON**

The Put Script on Device Button only appears in the Script mode. This button loads compiled scripts onto the iPOS terminal.

**MODIFY CONTROL BUTTON**

The Modify Control Button appears in both modes Script and VB mode. It changes the appearance of a limited number of controls from visible to transparent.

**CONTROLS FOR POSVISUAL FORMS**

**posVisual** adds two special controls to the Toolbox: Signature Capture (SigControl) and Magnetic Stripe Reader (MSRControl).



**Note:** Only a selected subset of Visual Basic controls will display and work on PenWare terminals. See Script Mode and VB Mode Chapters.

**Visual Basic Toolbox**

**Signature Capture Control (SigControl)** allows developers to put a signature capture box on iPOS terminals and handle signature information.

**Magnetic Stripe Reader Control (MSRControl)** activates magnetic stripe reader to capture credit or debit card information.

# posVisual Forms

## 3

The special forms that posVisual adds to Visual Basic behave similarly to other Visual Basic forms but with some important differences. All posVisual projects start with a padStart form. Additional forms can be added as needed.

### ADDING POSVISUAL FORMS

Do the following to add a posVisual form.

1. Starting from an open posVisual project, click on the Project menu. The Project menu appears.
2. Select Add Forms from the Project menu. The Add Forms message box appears.
3. Double-click on the posVisual form in the Add Forms window. The posVisual form appears in the Designer Window.
4. Name the new form. All posVisual forms need to start with the word **pad**, such as padSignature or padSwipe.

### PAD FORM EVENTS

Each posVisual mode has an associated code window. The code window for the VB mode form contains four subroutines while the code window for Script mode form contains two subroutines. The subroutines are listed in the following table:

VB Mode Subroutines	Script Mode Subroutines
ScreenInitialize()	ScreenInitialize()
AfterScreenDraw()	AfterScreenDraw()
GoBack()	
GoNext()	

These routines are executed automatically by posVisual. posVisual executes any code added to the routines. For example: if code is placed inside the AfterScreenDraw routine, it executes after all the controls on the iPOS screen are displayed. See the comments in the posVisual Form code window for detailed information.

## PINPAD FORM

This PinPad form only support for Script mode.This screen used to get the PIN from the user and saves the PIN in the PIN variable.User does not need to define the PIN variable ,it's added by the posVisual. This form code consist with three routines.

<b>Script Mode Subroutines</b>
ScreenInitialize()
cmdCancel_Click()
cmdEnter_Click()

These routines are executed automatically by posVisual. posVisual executes any code added to the routines. Coding need to save the PIN is automatically added by the posVisual. As an example: if Tool.SCRIPT.GotoScreen padScreen command added then it saves the PIN and go to padScreen.

### **Note:**

If the property Tag of fldPIN set to "dukpt", PinPad will use the dukpt encryption method. If the property is blank, PinPad will use Master Session encryption method.

## POSVISUAL FORM PROPERTIES

The posVisual pad form is designed to generate an optimum display on the @pos.com iPOS terminals. Most changes to properties of posVisual forms have no effect on the resulting screen display. Only five form properties modify the resulting screen. Of the five properties, only two should be changed during application development: the Caption property and the Name property. Do not change the other three properties, ScaleMode, WidthMode and HeightMode. Changes in these properties degrade screen display appearance and function.

# Programming Modes Overview 4

---

Applications written for @pos.com iPOS terminals are developed with posVisual in one of two modes: **Script** or **VB**. These applications, by nature, are event driven. posVisual creates event driven software for @pos.com iPOS terminals as scripts or Visual Basic executables. Both modes have unique advantages depending upon the iPOS terminal host and existing software.

## COMPARISON OF POSVISUAL MODES

Intelligence built into @pos.com iPOS terminals causes posVisual derived applications to fall naturally into two categories: applications consisting of internal scripts that control @pos.com iPOS terminal functions and applications in which external programs control @pos.com iPOS terminal functions. Both types of applications can be developed with posVisual. Applications residing on @pos.com iPOS terminals are written in posVisual Script mode. Applications residing on an external host are written in VB mode.

## DECIDING WHICH POSVISUAL MODE TO USE

Each of the two posVisual modes addresses an important class of applications for @pos.com iPOS terminals.

### ***SCRIPT MODE***

Developers integrating @pos.com iPOS terminals into legacy software and hardware systems should use the posVisual scripting mode. Sensitive systems such as financial transaction networks need to be thoroughly tested after they are modified. The less such code is disturbed, the more easily it can be readied for re-use. By inserting script activating “hooks” into legacy host software, iPOS functionality can be added to a transaction system with **light** modifications to existing software. Light coding changes mean efficient upgrades of legacy transaction systems.

# Script Mode

---

# 5

The Script mode generates @pos.com iPOS terminal-resident programs (scripts) that are activated from existing transaction software. Programmers creating new e-transaction systems, where the host controls all the iPOS functions, should use the VB mode while programmers upgrading legacy systems should use the Script mode.

## WHAT ARE SCRIPTS?

Scripts are compiled programs written in the Visual Basic environment using the posVisual Assistant. With the customized controls and commands resident in the Visual Assistant add-in, programmers construct semi-conventional Visual Basic programs that are compiled to machine code by a posVisual associated DLL.

Once compiled, developers use posVisual to download scripts into a @pos.com iPOS terminal attached to the com port of a host. The iPOS terminal holds the script in non-volatile memory. The speed and ease with which posVisual compiles and generates scripts, allows developers to rapidly iterate and optimize @pos.com iPOS terminal applications.

## OPERATING A SCRIPT

To activate a script, power down then power up the iPOS terminal by disconnecting and reconnecting the power cable. The firmware resident operating system then runs the script. The operating system and the script work semi-autonomously, communicating and receiving commands through the RS-232 and RS 485 ports of the iPOS terminal.

If a script contains a Sub Main(), the program starts execution with Sub Main, otherwise the program starts execution with padStart form.

## SCREEN CREATION PROCESS IN SCRIPT MODE

Screens on the iPOS terminals are created control-by-control. Although it is not necessary to understand how iPOS screens are created, knowledge about the process helps with troubleshooting. For example, when using the command “Tool.Display.Screen”, the following occurs in the listed order.

Steps	Explanation
Clear Screen	Freezes the operation of iPOS screen and deletes any objects on the screen
Process ScreenInitization event on forms	Executes “ScreenInitilization” code in a form.
Draw screen	Places objects on the screen in the following order: <ol style="list-style-type: none"> <li>1. Signature areas</li> <li>2. Magnetic Stripe Readers</li> <li>3. TextBoxes</li> <li>4. All other controls in reverse order of creation except for Command Buttons.</li> <li>5. Command Buttons</li> </ol>
Process AfterScreenDraw event on form	Executes “ AfterScreenDraw” code in a form
Process timer routine if part of code	<b>Note:</b> Use only one timer per form. The timer code only executes once
Wait for user input	Script stops to wait for user or host initiated event.

**Note:** The iPOS terminals ignores all host commands and events while in steps 1 - 4.

## SCRIPT MODE SOFTWARE COMPONENTS

Script mode uses certain components in projects. The posVisual Assistant automatically adds those components to projects, so this information is mostly for detailed understanding of posVisual Script mode.

The components needed to operate posVisual in the Script mode are the following:

**CONTROLS**

<b>Name</b>	<b>Description</b>	<b>Additional Notes</b>
VisualControls.ocx	Provides the SigArea and MSRArea controls	DO NOT MODIFY

**FORMS**

<b>Name</b>	<b>Description</b>	<b>Additional Notes</b>
padStart.frm	posVisual defaults to padStart on starting in script mode unless submain() is present in mainScript. The first screen must always be padStart.	Modifiable component. To add more forms to the project, see Chapter 3 - Visual Advantage Forms.

**MODULES**

<b>Name</b>	<b>Description</b>	<b>Additional Notes</b>
mainScript.bas	Module contains: posVisual internal function routines User routines A user modifiable command for com port selection A user modifiable command for specifying command prefixes	In a script, all execution stops at the end of a routine. To continue execution past the end of a routine, add a GoToRoutine command at the end of the code. Script mode in posVisual does not have subroutines. A routine normally halts execution at the end of code. Communication port is selected with a line in MainScript.bas Command prefixes are also defined in MainScript.bas

**REFERENCES**

<b>Name</b>	<b>Description</b>	<b>Additional Notes</b>
mxSptool.dll	Enables the downloading of scripts and graphics to the iPOS terminal.	DO NOT MODIFY
mxScript.dll	Enables the "autolist" in Visual Basic that helps displays the posVisual commands.	DO NOT MODIFY

**DOWNLOADING THE DESIGNED SCRIPT TO THE IPOS TERMINAL**

Once the port number is entered, be certain the iPOS terminal is attached to that port then press the "Put Script on Device" button (See Chapter 2-The VA Assistant). The

status area indicates the progress of the download and the iPOS device display reads "Downloading". The Status indicator signifies when the download completes.

### **OPERATIONAL HOST COMMUNICATION WITH SCRIPTS**

For the host to communicate with a Script on an iPOS terminals, the Script.bas component of posVisual needs to be configured to use the port attached iPOS terminal. The example shows where to change the port in the **mainScript.bas** code.

```

'-----
' Start of required code for scripting
' *** THIS MUST BE THE FIRST DIM or DECLARE IN THIS
MODULE!!!!
Public Tool As New clsTool
'      ^
'      You can change this Command Indicator name.
'
Const PortNumber = 1 ←
'      ^
'      Change this to the port number
where
'      the POS device is connected
'
Const BaudRate = 57600 ←
'      ^
'      Change this to the baud rate
which the
'      POS device to be communicated
'
' End of required code for scripting
'-----

```

### **HOST TO SCRIPT COMMUNICATION**

With the script on the iPOS device, power up and run the script from the iPOS terminal. Once started the script functions independently, however, a host terminal is required to send and receive information.

Communications between iPOS terminals and hosts takes place using three commands:  
**GoToLabel** LabelName

Executes the code starting at the label parameter. **LabelName** can be a routine name.

**SetTextVar** VarName, StringValue

Allows the host to set a string variable. For example, the host might set the variable "Data: to "12/01/1999", to display a date on an attached iPOS terminal.

**Note:** All variables are case sensitive.

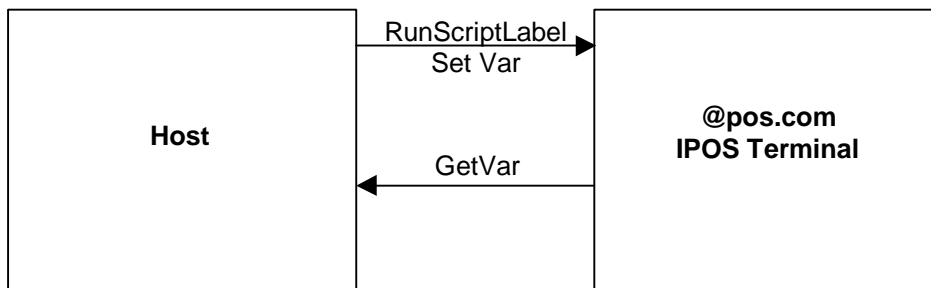
**SetVar** allows the operating host to set a string variable. For example, the host might set the variable "Data" to "12/01/1999", to display a date on an attached iPOS terminal.

GetTextVar (VarName) as String

Allows the host to retrieve a string value from the iPOS terminal for use in a host application. If script variable, "CredCardNum", holds credit card data from an MSR read event. The operational host can request the variable information from the script by issuing a "GetVar" command to the script.

Any text (String) variable in a script can be read or be written to by using SetVar or GetVar commands. These commands are located on the mxsptool.cll and used by the host machines. See the Sample Host application on the posVisual CD.

### Host-iPOS Terminal Command Traffic

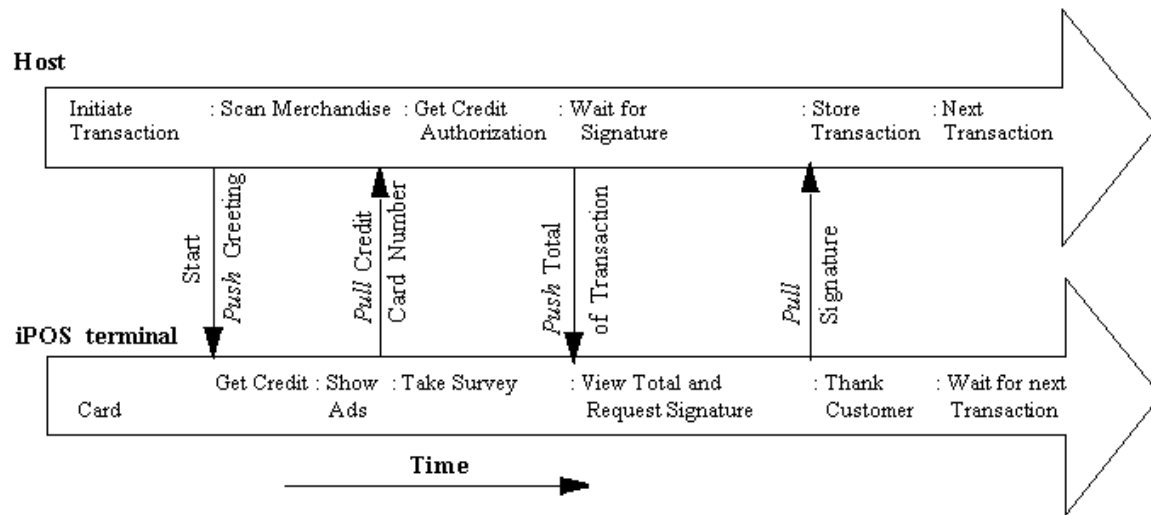


On start-up, the iPOS terminal runs the loaded script. The Script can be designed to operate quite independent from the host. Communication between host and iPOS terminal for a transaction can be as simple as:

1. Start transaction (RunScriptLabel)
2. Get credit card information (GetTextVar)
3. Set and display the purchase price (SetTextVar)
4. Get Signature (GetTextVar).

### TRANSACTION EVENTS

Communication between host and @pos.com iPOS terminals are simple in script mode. Light communications demands by @pos.com iPOS terminals of the host relieves the host for other tasks.



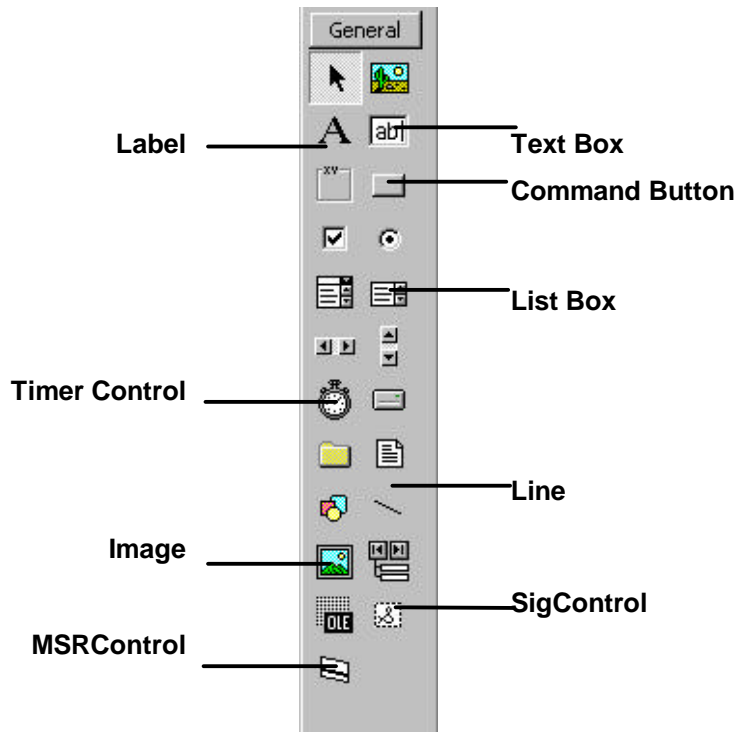
---

## Script Controls

posVisual scripts run in an environment unique to @pos.com iPOS terminals. The special demands of the @pos.com iPOS terminal environment requires a different set of controls than the Windows environment that is native for Visual Basic. Thus, while many of the controls function in posVisual as they do in Visual Basic alone, some do not.

### *POSVISUAL TOOL BOX*

The eight controls shown in the following diagram work in posVisual Script mode. The MSRControl and SigControl are specialized controls added to the tool box by posVisual. Descriptions of the controls follow below:



**CONTROL FONT SIZES**

Font characters displayed on @pos.com iPOS terminals come in six sizes. All characters in each font size have the same pixel dimensions. The posVisual font families are mapped onto the Courier New font families in the property boxes of posVisual controls. The following table shows the correspondence between the posVisual font sizes and Courier New font sizes.

<b>@pos.com iPOS Terminal Font Sizes in Pixels</b>	<b>Courier New Font Sizes in Points</b>
6 x 8	10 to 11 pt
8 x 8	12 pt
8 x 12	13 to 16 pt
12 x 16	17 to 22 pt
16 x 16	23 to 26 pt
16 x 24	27 to 30 pt

**COMMAND BUTTON**

posVisual command buttons operate the same as a Visual Basic command buttons.

<b>Property</b>	<b>How property is used</b>
Font-Size	Controls the size on the font displayed on the iPOS terminal
Caption	The text displayed inside the button
Style	property can be set to normal or hidden

**USING THE CONTROL**

Use posVisual buttons in the same way as a Visual Basic buttons.

**IMAGE**

The image sizes on @pos.com iPOS terminals depend on the number of pixels they contain. The images display with the same number of pixels as the images on the host monitor. The size of images can be adjusted once placed on a posVisual form but look better if adjusted in the creating application. Keep the all the image edges within the boundaries of the screen forms for best results.

<b>Property</b>	<b>How property is used.</b>
Properties are same as in Visual Basic	Images can not be larger than 320 x 240 pixels

**USING THE CONTROL**

All Script mode images must be 2-color, black-and-white bitmaps. Adjust the image colors with the picture property. Always use picture property to add or change the picture, do NOT cut & paste pictures.

**LABEL**

Label controls in Script mode work like Visual Basic label controls, but DO NOT wrap. Multiple lines of text require multiple labels.

<b>Property</b>	<b>How property is used</b>
Font-Size	Controls the size of the font on the pad

***USING THE CONTROL***

After putting code under the \_Click event for a label, posVisual automatically creates a hidden button in front of the label and assigns the \_Click event code to the hidden button.

**TEXTBOX**

Textbox controls in script mode work like a label. Also it works like a variable for script mode. Text property act likes name of the variable. Multiple lines of text require multiple labels.

<b>Property</b>	<b>How property is used</b>
BorderStyle	“0 - None” displays no border. “1 - Fixed Single” displays a border. This is the default setting.
Font-Size	Controls the size of the font on the pad

**LINE**

The line control displays a line on the iPOS terminal just as it appears on the developer host screen.

**LISTBOX**

Script mode ListBoxes behave very differently than Visual Basic ListBoxes. Script mode ListBoxes operate as a “serial marquee”. A line of text appears one word at a time in the ListBox window.

Create Script mode ListBoxes that are one text line in height. Any other height produces unpredictable results.

Property	How property is used
List	Place a line of text directly into this property so the box displays one line at a time.
Font-Size	Sets the size on the font on the pad

**USING THE CONTROL**

Enter the lines of text for display in the List property. It is important to manually adjust the height of the ListBox until only one line of text displays.

**MSRCONTROL**

Place the MSRControl anywhere on the form to give the MSR capabilities to iPOS terminals. Though the control is visible on the developer screen, it is invisible on the iPOS screen. When a form containing a MSR control activates the MSR also activates. Once activated, the MSR on the iPOS terminal can accept information from credit card swipes. If a good swipe occurs, the code under the **GoodSwipe** event is called.

**EVENTS:**

**GoodSwipe** - fires after a good swipe.

**BadSwipe** - fires after a bad swipe or after no swipe after timeout.

Property	How property is used
There are no configurable control properties	

**USING THE CONTROL**

Draw this control on the developer screen. The MSRControl is hidden on the iPOS terminal.

**SIGCONTROL**

The SigControl creates a signature capture area on the iPOS terminal.

Put code under the "SignTimer" event to handle timeouts.

Property	How property is used
BorderStyle	<p>“0 - None” displays no border</p> <p>“1 - Fixed Single” displays a border. This is the default setting.</p>
MaxSize	This property limits the number of points collected per signature. Once a user produces MaxSize pixels, the iPOS terminal stops taking signature data.
Timeout	This property sets the number of seconds, after the user lifts the pen from the iPOS terminal, until the “SignTimer” event fires.

### **USING THE CONTROL**

Use this control to draw a signature area. To save the signature into a string variable, use the **Tool.Sig.SaveVar** command. To clear a signature area, use the **Tool.Sig.Clear** command.

## SHAPE

The shape tool can only be used to draw rectangles on the device so be sure the control is set to the rectangle shape. After placing a Shape on a form, a dialog box appears asking if the object should be a “Box (Filled)” or a “Frame (Transparent)”.

Property	How property is used
FillStyle	<p>“0 - Solid” fills the rectangular area with the current fill color.</p> <p>“1 - Transparent” draws a transparent rectangle.</p>

## TIMER

Place the Timer control anywhere on a form to give timer capabilities to a screen. The Timer control is visible on the developer screen but hidden on the iPOS terminal. In Script mode the timer control does not act like a normal Visual Basic timer. It acts like a “countdown” timer. It waits for the number of seconds specified in the “interval” property, then executes the code under the timer event. The countdown occurs only once. Displaying a second screen after 15 seconds is an example of this timers’ usefulness.

Property	How property is used
Interval	<p>posVisual parameters for timer intervals are expressed in seconds.</p> <p>Parameters for timers in Visual Basic, are expressed in milliseconds.</p>

### **USING THE CONTROL**

posVisual script mode timers do not act like Visual Basic timers. Script mode timers are ‘countdown’ timers. Enter the countdown time i the Interval box of the timer’s properties.

Add the control on the form, then enter the interval to be timed. Double-click the control, add script commands to be executed after the interval elapses. Script mode timers are countdown timers. The timer control in Script mode does not act like a normal Visual

Basic timer. It acts as a countdown timer. To set the countdown period, enter the amount of time to be counted down in the Interval property. The timer waits for the number of seconds specified with the "Interval" property then executes the code under the timer event. A great use for the timer is to display a sequence of screens.

---

## Script Commands

posVisual code lines use either **statement** or **function** syntax. An example of statement syntax is:

```
Tool.Sound.Bell Alarm
```

while an example of function syntax is:

```
Tool.Sound.Bell(Alarm).
```

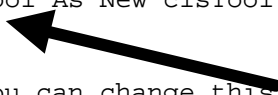
Use function syntax with parenthesis for sub commands such as a **Script.IfTrue** command.

### COMMAND PREFIXES

Every Script mode command begins with a prefix. The default prefix is **Tool**. The Script command prefix can be changed by substituting a new word in for **Tool** in the class declaration statement in the mainScript.bas file at the indicated location.

```
'-----
' Start of required code for scripting
' *** THIS MUST BE THE FIRST DIM or DECLARE IN THIS MODULE!!!!!!
Public Tool As New clsTool
'   ^
'   You can change this Command Indicator name.
'
Const PortNumber = 1
'           ^
'           Change this to the port number where
'           the POS device is connected
'
' End of required code for scripting
'-----

Sub main()
```



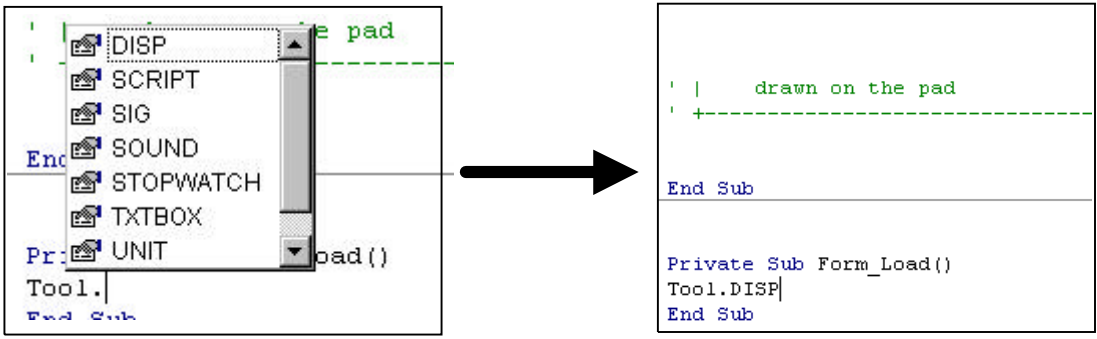
### DECLARING VARIABLES

In Script mode, there are four types of variables: String and three types of numeric. Variables are created with the **Var** command and sub-commands to **SetVar**.

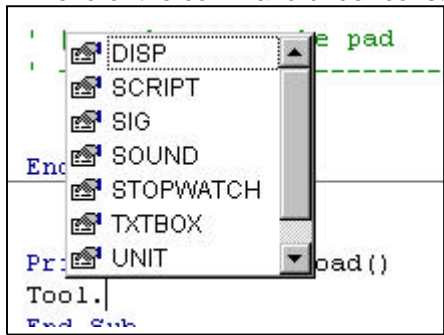
SCRIPT COMMAND GENERATION

posVisual uses the Visual Basic interface to provide command options to the developers as they construct Script commands. The illustration depicts a typical sequence:

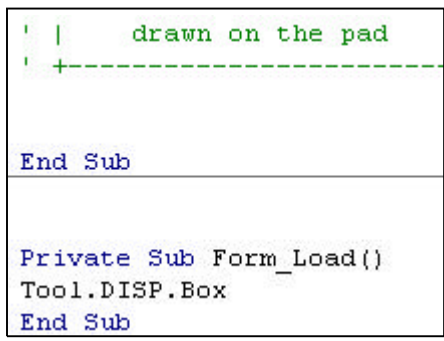
1. Type Tool. A scroll box with available command options appears.



2. Double click one of the command options. The selected command options appends to the end of the command under construction.



3. Repeat process until typing a period brings up no new sub-options.



4. Type a space after the last sub-option to make parameters easier to see. A parameter list for the command, if one exists, appears with the first parameter highlighted.

```
Private Sub Form_Load()  
  
Tool.DISP.Box  
    Box(Left As Integer, Top As Integer, Width As Integer, Height As Integer)  
  
End Sub
```

5. Type in the parameters until the parameter list is exhausted

```
' |   drawn on the pad  
' +-----+  
  
End Sub  
  
Private Sub Form_Load()  
Tool.DISP.Box 40, 20  
    Box(Left As Integer, Top As Integer, Width As Integer, Height As Integer)  
  
End Sub
```

---

## posVisual Objects, Properties and Methods (script mode)

The organization of the objects, properties and methods for posVisual are displayed in the chart below:

### TOOL

#### DISP

Box .....	31
Clear .....	31
DeleteAllControls.....	31
DrawLine .....	32
Frame .....	32
SetColor .....	32
SetFont .....	33
Text.....	33
Getpin .....	33
Mkeypin.....	34

#### SCRIPT

DoNothing .....	34
GotoRoutine .....	35
GotoScreen.....	35
GotoVar.....	36
IfFalse .....	36
IfTrue .....	37
Pause.....	38
StopScript .....	38

## SIG

Clear .....	38
Convert .....	39
GetCount.....	39
Height .....	39
InchesHigh .....	40
InchesWide .....	40
IsEmpty.....	40
Save.....	41
Width.....	41

## SOUND

Bell.....	42
Tone.....	42

## STOPWATCH

Continue.....	43
GetTimer.....	43
Pause.....	43
Start .....	44

## TXTBOX

IsEmpty.....	44
Load.....	45
Save.....	45
SendKeys.....	46

## UNIT

Model .....	46
Version.....	46

## VAR

DeleteVar .....	47
FindVar .....	47
SetVar .....	48

## BIN

SetVar .....	48
ToBase.....	48
ToHex .....	49

## NUM

Dec .....	49
Diff .....	50
GetVar .....	50
Inc.....	50
IsEqual .....	51
IsGreater .....	51
IsLess.....	51
SetVar .....	52
Sum .....	52

## STR

Concat.....	52
GetVar .....	53
IsEqual .....	53
Left.....	53
Right .....	54
SetVar .....	54

---

## DISP

The DISP commands relate to the visual aspects of the iPOS scripts. The DISP commands execute changes in the visual displays that occur during script operations.

DISP.BOX(LEFT, TOP, WIDTH, HEIGHT)

**Disp.Box** draws a filled box on the screen.

Parameters	Description	Values
Left	Integer, distance from left edge of screen	Range: 0 – 319
Top	Integer, distance from top edge of screen	Range: 0 – 239
Width	Integer, width of box	Range: 0 – 319
Height	Integer, height of box	Range: 0 – 239

**RETURN VALUES**

None

**SEE ALSO**

**Disp.Frame**

**EXAMPLE**

Tool.Disp.Box 10, 10, 200, 140

DISP.CLEAR

**Disp.Clear** makes all objects on the screen invisible

Parameters	Description	Values
None		

**RETURN VALUES**

None

**EXAMPLE**

Tool.Disp.Clear

DISP.DELETEALLCONTROLS

**Disp.DeleteControls** removes all objects from the screen.

Parameters	Description	Values
None		

**RETURN VALUES**

None

**EXAMPLE**

```
Tool.Disp.DeleteAllControls
```

DISP.DRAWLINE(X1, Y1, X2, Y2)

**Disp.DrawLine** draws a line on the screen.

Parameters	Description	Values
X1, Y1	Integers representing coordinates at the start of a line	X1: 0 to 319 Y1: 0 to 239
X2, Y2	Integers representing coordinates at the end of a line	X2: 0 to 319 Y2: 0 to 239

**RETURN VALUES**

None

**EXAMPLE**

```
Tool.Disp.DrawLine 20, 20, 200, 20
```

DISP.FRAME(LEFT, TOP, WIDTH, HEIGHT)

**Disp.Frame** draws unfilled rectangles on screen.

Parameters	Description	Values
Left	Integer, distance from left edge of screen	Range: 0 - 319
Top	Integer, distance from top edge of screen	Range: 0 - 239
Width	Integer, width of frame	Range: 0 - 319
Height	Integer, height of frame	Range: 0 - 239

**RETURN VALUES**

None

**SEE ALSO**

**Disp.Box**

**EXAMPLE**

```
Tool.Disp.Frame 10,10,50,50
```

This will draw a small frame on top left hand corner of the Padscreen

DISP.SETCOLOR(COLOR)

**Disp.SetColor** sets the current color.

Parameters	Description	Values
Color	PadColors	White or Black

**RETURN VALUES**

None

**EXAMPLE**

```
Tool.Disp.SetColor Black
```

## DISP.SETFONT(FONTTYPE)

**Disp.SetFont** sets the current font.

Parameters	Description	Values
FontType		

**RETURN VALUES**

None

**EXAMPLE**

```
Tool.Disp.SetFont Font6x8
```

## DISP.TEXT(LEFT, TOP, TEXT)

**Disp.Text** displays text beginning at coordinates Left(X), Top(Y).

Parameters	Description	Values
Left	Integer, distance from left edge of screen	0 - 319
Top	Integer, distance from top edge of screen	0 - 239
Text	String	

**RETURN VALUES**

None

**EXAMPLE**

```
Tool.Disp.Text 20, 40, "Hello"
```

## DISP.GETPIN (TITLE, ACCOUNT)

**Disp.Getpin** displays a DUKPT PIN entry prompt and returns a standard binary DUKPT PIN.

Parameters	Description	Values
Title	Optional, title message	String value to display as the title of the screen
Account	Required, account number as a string of ASCII digits.	

**RETURN VALUES**

This command encrypts a pin returned as a binary value. If an error occurs zero length result is returned. A canceled entry also returns a zero length result.

**EXAMPLE**

```
Tool.Var.SetVar "PIN", Tool.DISP.GetPin("Enter Pin", "764012345678909")
```

DISP.MKEYPIN (TITLE, ACCOUNT, SKEY, MKEYID)

**Disp.Mkeypin** displays a PIN entry prompt and returns a standard Masterkey PIN.

Parameters	Description	Values
Title	Optional, title message	String value to display as the title of the screen
Account	Required, account number as a string of ASCII digits.	9-19 ASCII digits
Skey	Transaction/Session key	16 ASCII hex digits
MkeyID	Stored master key ID	0-9

**RETURN VALUES**

This command encrypts a pin returned as a binary value.

**Example**

```
Tool.Var.SetVar "PIN", Tool.DISP.Mkeypin("Enter Pin", "764012345678909", "0123456789ABCDEF", "0")
```

---

## Script

These Script commands handle special logic that deals with scripts, such as navigation, and branching commands.

SCRIPT.DONOTHING

**Script.DoNothing** is a place holder. This command is mainly used with "If" statements when the developer wants the "Else" clause to do nothing.

Parameters	Description	Values
None		

**RETURN VALUES**

None

**SEE ALSO****Script.IfTrue, Script.IfFalse****EXAMPLE**

```
Tool.Script.IfTrue Tool.Object.IsEmpty(padScreen.SigArea1), _
    Tool.Sound.Bell(Alarm), Tool.Script.DoNothing()
```

This sample code checks SigArea1 to find out if the customer has signed in. If the user has not signed, then the iPOS terminal sounds an alarm, otherwise it does nothing.

**SCRIPT.GOTOROUTINE(ROUTINENAME) AS BOOLEAN**

**Script.GotoRoutine** executes RoutineName.

**Note:** This command is not a subroutine. It will not continue processing after the line that called it. Execution stops at the end of RoutineName unless the last line is a **Script.Goto** command.

In the posVisual Assistant, clicking on the "View Labels" button causes posVisual to list available routine names.

Parameters	Description	Values
RoutineName	Name of user defined routine.	Must be in the format of module or form routine. (case sensitive)

**RETURN VALUES**

Boolean - True if successful

**SEE ALSO****Script.GotoScreen****EXAMPLE**

```
Tool.Script.GotoRoutine CheckIdFormat
```

This command transfers script control to the **CheckIdFormat** routine in a module of the project.

**Note:** This command completely transfers execution of the script. Execution does not return automatically to the calling code.

**SCRIPT.GOTOSCREEN(SCREENNAME)**

**Script.GotoScreen** displays ScreenName. On the posVisual Assistant panel, click the "View Labels" button to get a list of available screen names to use.

Parameters	Description	Values
ScreenName	A name of a form.	case sensitive

**RETURN VALUES**

None

**SEE ALSO****Script.GotoRoutine****EXAMPLE**

```
Tool.Script.GotoScreen padGetMSR
```

**Note:** This command displays the **padGetMSR** form on the device.

**SCRIPT.GOTOVAR(VARNAME)**

**Script.GotoVar** is a typical goto statement. This command executes the routine VarName.

**Note:** This is not a subroutine, in other words, this will not continue processing after the line that called it. Program routines stop processing at routine end unless the routines are ended with **Script.Goto** commands.

In posVisual Assistant, clicking on the "View Labels" buttons produces a list of valid routine names.

**Note:** Insert routine and form names as the parameters.

Parameters	Description	Values
VarName	Variant: variable that contains the name of user defined subroutine.	Must be in the format of module or form subroutine or form name (case sensitive).

**RETURN VALUES**

Boolean - True if successful

**SEE ALSO****Script.GotoScreen, Script.GotoRoutine****EXAMPLE**

```
Tool.Var.Str.SetVar "NextScreenName", "padGetSig"
```

...

```
Tool.Script.GotoVar "NextScreenName"
```

**Note:** This example will display the **padGetSig** form on the device.

**SCRIPT.IFFALSE(EXPRESSION, THENCLAUSE, ELSECLAUSE)**

**Script.IfFalse** is a If Not True statement. If the expression is false, the ThenClause executes, otherwise the ElseClause executes.

**Note:** Every sub-command must be enclosed in parenthesis in function syntax fashion.

Parameters	Description	Values
Expression	A script command that returns a boolean value.	Must be a script command in a function syntax.
ThenClause	The command executed if the expression returns False.	Must be a script command in function syntax.
ElseClause	The command executed if the expression returns True.	Must be a script command in function syntax.

**RETURN VALUES**

None

**SEE ALSO****Script.IfTrue****EXAMPLE**

```
Tool.Script.IfFalse Tool.Object.IsEmpty(padScreen.SigArea1),
Tool.Sound.Bell(Alarm), Tool.Script.DoNothing()
```

If the Signature Area (SigArea1) on padScreen has not been signed, then an alarm sounds.

**SCRIPT.IFTRUE(EXPRESSION, THENCLAUSE, ELSECLAUSE)**

**Script.IfTrue** evaluates the Boolean Expression. If Expression is true, the ThenClause executes. If Expression is false, ElseClause executes.

**Note:** Every command used as a parameter must be enclosed in parenthesis in function syntax fashion.

Parameters	Description	Values
Expression	A Script command that returns a boolean value.	Must be a script command in a function syntax.
ThenClause	The command executed if the expression returns True.	Must be a script command in function.
ElseClause	The command executed if the expression returns False.	Must be a script command in function syntax.

**RETURN VALUES**

None

**SEE ALSO****Script.IfFalse****EXAMPLE**

```
Tool.Script.IfTrue Tool.Object.IsEmpty(padScreen.SigArea),
Tool.Sound.Bell(Alarm), Tool.Script.DoNothing()
```

## SCRIPT.PAUSE(SECONDS)

**Script.Pause** halts script execution for the indicated number of seconds.

**Note:** When a script is halted with **Script.Pause**, the command allows most actions to occur and events to be handled such as button clicks. Use for displaying messages and splash screens.

Parameters	Description	Values
Seconds	Integer, the number of seconds for delay.	1- 600

**RETURN VALUES**

None

**SEE ALSO**

**Script.StopScript**

**EXAMPLE**

```
Tool.Script.Pause 30
```

## SCRIPT.STOPSCRIPT

**Script.StopScript** completely halts script execution.

Parameters	Description	Values
None		

**RETURN VALUES**

None

**SEE ALSO**

**Script.Pause**

**EXAMPLE**

```
Tool.Script.StopScript
```

---

## Sig

The SIG commands are commands that work with signature data. In conjunction with these commands, use the **Sig.Save** command to save signatures into a binary variable.

## SIG.CLEAR(CONTROLNAME)

**Sig.Clear** clears the signature area.

Parameters	Description	Values
ControlName	The name of SigControl to be cleared.	

**RETURN VALUES**

None

**SEE ALSO**  
**TextBox.Clear**

**EXAMPLE**

```
Tool.Sig.Clear padStart.SigControll
```

**SIG.CONVERT(VARNAME1, FORMAT) AS VARNAME**

**Sig.Convert** changes signatures from one format to another.

Parameters	Description	Values
VarName1	String variable <u>containing</u> signature information for conversion to another format.	Signature variable
Format	Integer format for saving signature	Points = 1 Packet = 5 COTF = 7

**RETURN VALUES**

Variable containing string data for signature in new format.

**EXAMPLE**

```
Tool.Var.SetVar "SIGNATURE2" , Tool.Sig.Convert ("SIGNATURE1",  
CONF)
```

**SIG.GETCOUNT(VARNAME) AS INTEGER**

**Sig.GetCount** returns the number of points for a signature stored in VarName.

Parameters	Description	Values
VarName	String variable contains signature.	Signature variable

**RETURN VALUES**

Integer

**EXAMPLE**

```
Tool.Var.SetVar "Num" , Tool.Sig.GetCount("Sig")
```

**SIG.HEIGHT(VARNAME) AS INTEGER**

**Sig.Height** returns the signature height in pixels for a signature that is stored in the VarName.

Parameters	Description	Values
VarName	String, variable name that contains signature	Signature variable

**RETURN VALUES**

Integer

**SEE ALSO**  
**Script.Inches, Script.Width**

**EXAMPLE**

```
Tool.Var.SetVar "Height" , Tool.Sig.Height("Sig")
```

SIG.INCHESHIGH(VARNAME) AS INTEGER

**Sig.InchesHigh** returns the signature height in inches x 100 for signature that is stored in the VarName.

**Note:** This does not return fractions, so 2.32 inches returns as 232.

<b>Parameters</b>	<b>Description</b>	<b>Values</b>
VarName	String variable name that contains signature	Signature variable

**RETURN VALUES**

Integer

**SEE ALSO**  
**Sig.InchesWide, Sig.Width, Sig.Height**

**EXAMPLE**

```
Tool.Var.SetVar "InchesHigh" , Tool.Sig.InchesHigh("Sig")
```

SIG.INCHESWIDE(VARNAME) AS INTEGER

**Sig.InchesWide** returns the signature width in inches x 100 for signature that is stored in the VarName.

<b>Parameters</b>	<b>Description</b>	<b>Values</b>
VarName	String variable name that contains signature	Signature variable

**RETURN VALUES**

Integer

**SEE ALSO**  
**Sig.InchesHigh, Sig.Height, Sig.Width**

**EXAMPLE**

```
Tool.Var.SetVar "InchesWide" , Tool.Sig.InchesWide("Sig")
```

SIG.ISEMPTY(CONTROLNAME) AS BOOLEAN

**Sig.IsEmpty**.determines if ControlName holds a signature.

Parameters	Description	Values
ControlName	Name of SigControl	Control name in Visual Basic format

**RETURN VALUES**

Boolean - True if SigControl has no signature in it.

**EXAMPLE**

```
Tool.SCRIPT.IfTrue Tool.Sig.IsEmpty(SigControl1),
Tool.SOUND.Bell(Success), Tool.SOUND.Bell(Fail)
```

**SIG.SAVE (CONTROLNAME, VARNAME) AS BOOLEAN**

**Sig.Save** moves signature data in a control to a binary variable.

Parameters	Description	Values
ControlName	Name of SigControl	Control name in Visual Basic format
VarName	String variable stores signature	Signature variable

**RETURN VALUES**

Boolean - True if successful

**EXAMPLE**

```
Tool.Sig.Save padStart.SigControl1, "Sig1"
```

**SIG.WIDTH(VARNAME) AS INTEGER**

**Sig.Width** returns the width of a signature in pixels for a signature stored in VarName.

Parameters	Description	Values
VarName	String, variable contains signature	Signature variable

**RETURN VALUES**

Integer

**SEE ALSO**

**Sig.Height, Sig.InchesWide, InchesHigh**

**EXAMPLE**

```
Tool.Var.SetVar "Width" , Tool.Sig.Width("SIG")
```

---

## Sound

The Sound commands control all audio output for the iPOS terminals.

SOUND.BELL(BELLTYPE)

**Sound.Bell** causes an iPOS terminal to make the bell sound specified by BellType.

Parameters	Description	Values
BellType	BellTypes	Alarm Fail Normal_Bell Success

**RETURN VALUES**

None

**SEE ALSO**

**Sound.Tone**

**EXAMPLE**

Tool.Sound.Bell Alarm

SOUND.TONE(FREQHZ, TEMPOBEATS)

**Sound.Tone** causes an iPOS terminal to make the tone specified by the FreqHZ and TempoBeats.

Parameters	Description	Values
FreqHZ	Integer specifying the frequency of the tone generated.	
TempoBeats	Integer specifying the duration of the tone generated	

**RETURN VALUES**

None

**SEE ALSO**

**Sound.Bell**

**EXAMPLE**

Tool.Sound.Tone 220, 100

---

## Stopwatch

The **Stopwatch** commands control the timer. Use Start, Stop, Continue and GetTimer commands to manipulate the timer and use **Var.Num** commands to program decisions based on timer information.

### STOPWATCH.CONTINUE

**Stopwatch.Continue** re-activates the internal iPOS timer from the last **Pause** command.

Parameters	Description	Values
None		

**RETURN VALUES**

None

**SEE ALSO**

**Stopwatch.Pause**

**EXAMPLE**

```
Tool.Stopwatch.Continue
```

### STOPWATCH.GETTIME AS INTEGER

**Stopwatch.GetTime** returns the number of seconds in the elapsed since timer activation. Execute this command before pausing the timer.

Parameters	Description	Values
None		

**RETURN VALUES**

Integer

**SEE ALSO**

**Stopwatch.Start**

**EXAMPLE**

```
Tool.Var.SetVar "Seconds" , Tool.Stopwatch.GetTime
```

### STOPWATCH.PAUSE

**Stopwatch.Pause** halts the internal timer of an iPOS terminal.

Parameters	Description	Values
None		

**RETURN VALUES**

None

**SEE ALSO**  
**Stopwatch.Continue, Stopwatch.GetTime**

**EXAMPLE**

```
Tool.Stopwatch.Pause
```

## STOPWATCH.START

**Stopwatch.Start** initializes and starts the internal timer of an iPOS terminal. Use **GetTime** to get the number of seconds since the **Stopwatch.Start** command was executed. **Important:** Read the value of the timer before pausing.

Parameters	Description	Values
None		

**RETURN VALUES**

None

**SEE ALSO**  
**Stopwatch.Continue, Stopwatch.Time**

**EXAMPLE**

```
Tool.Stopwatch.Start
```

---

## TextBox

### TEXTBOX.ISEMPTY(CONTROLNAME) AS BOOLEAN

**TextBox.IsEmpty** determines if TextBox control contains data. Not the Text property of the TextBox control. Because of Text property acts as a variable, this method checks whether any values assign to the variable (ie. TextBox). By default TextBox controls contain "VAR not defined" as their values, so initially it is not empty.

Parameters	Description	Values
ControlName	Name of TextBox control	Must be a screen control name in the format of Form.Control

**RETURN VALUES**

Boolean - True if successful

**SEE ALSO**  
**Sig.IsEmpty**

**EXAMPLE**

```
Tool.TextBox.IsEmpty(padScreen1.txtResult)
```

TXTBOX.LOAD(CONTROLNAME, VARNAME) AS BOOLEAN

**TxtBox.Load** copies the contents of the VarName into ControlName.

Parameters	Description	Values
ControlName	Name of the TextBox control	Must be a screen control name in the format of form control.
VarName	variable name contains string value	String

**RETURN VALUES**

Boolean - True if successful

**SEE ALSO**

**TxtBox.Save**

**EXAMPLE**

```
Tool.TxtBox.Load PadScreen1.TextBox1, "CustName"
```

This example takes the value in the variable **CustName** and copies it into the **padScreen1** form textbox control **TextBox1**.

TXTBOX.SAVE(CONTROLNAME, VARNAME) AS BOOLEAN

**TxtBox.Save** copies the contents of the ControlName into the VarName.

Parameters	Description	Values
ControlName	Name of TextBox	Must be a screen control name in the format of Form.Control
VarName	variable name stores string value	String

**RETURN VALUES**

Boolean - True if successful

**SEE ALSO**

**TxtBox.Load**

**EXAMPLE**

```
Tool.TxtBox.Save padScreen1.txtCustName, "CustName"
```

This command copies the contents of the TextBox control **txtCustName** on the form **padScreen1** into **CustName**.

TXTBOX.SENDKEYS(CONTROLNAME, KEYS) AS BOOLEAN

**TextBox.SendKeys** sends text to a control then refreshes the control image. This command appends text to the contents of a Control.

Parameters	Description	Values
ControlName	Name of the TextBox control	Must be a screen control name in the format of form control
Keys	String	Characters to append

**RETURN VALUES**

Boolean - True if successful

**EXAMPLE**

```
Tool.TxtBox.SendKeys padScreen1.txtTotal, "110.99"
```

This command appends the text "110.99" to the end of the TextBox control, (txtTotal), on the form, (padScreen)1.

## Unit

The UNIT commands provide information about an iPOS terminal the script is running.

UNIT.MODEL AS INTEGER

**Unit.Model** returns the model number of the iPOS terminal running the script.

Parameters	Description	Values
None		

**RETURN VALUES**

Integer

**SEE ALSO**

**Unit.Version**

**EXAMPLE**

```
Tool.Var.SetVar "NumModel" , Tool.Unit.Model
```

UNIT.VERSION AS INTEGER

**Unit.Version** returns the version number of the firmware running on an iPOS terminal.

Parameters	Description	Values
None		

**RETURN VALUES**

Integer

**SEE ALSO**  
**Unit.Model**

**EXAMPLE**

```
Tool.Var.SetVar "NumVersion" , Tool.Unit.Version
```

---

## Var

The VAR commands handle variables. There are subcommands under the VAR command. These subcommands are Bin, Num, and Str. They control the following:

Command	Data Used
Var.Bin	Binary data
Var.Num	Numeric data (integers)
Var.Str	String data

### VAR.DELETEVAR(VARNAME)

**Var.DeleteVar** removes the variable name and frees its memory resources.

Parameters	Description	Values
VarName	The name of a variable	Case sensitive

**RETURN VALUES**

None

**EXAMPLE**

```
Tool.Var.DeleteVar "IsCardSwiped"
```

### VAR.FINDVAR(VARNAME) AS BOOLEAN

**Var.FindVar** indicates if VarName exists.

Parameters	Description	Values
VarName	The name of a variable	Case sensitive

**RETURN VALUES**

Boolean - True if the variable exists.

**EXAMPLE**

```
Tool.SCRIPT.IfTrue Tool.Var.FindVar("CustName"),  
Tool.SOUND.Bell(Success), Tool.SOUND.Bell(Fail)
```

**VAR.SETVAR(VARNAME, COMMAND) AS BOOLEAN**

**Var.SetVar** assigns the result of a command into VarName

Parameters	Description	Values
VarName	Name of a variable (case sensitive).	Case sensitive
Command	A command that returns a value.	Script command with function syntax

**RETURN VALUES**

Boolean - True if VarName was successfully set to the value that Command returned.

**EXAMPLE**

```
Tool.Var.SetVar"FullName", Tool.Var.Str.Concat("FirstName",
"Space", "LastName")
```

---

## Var.Bin

**Var.Bin** commands handle binary data.

**VAR.BIN.SETVAR(VARNAME, DATA) AS BOOLEAN**

**Var.BinSetVar** sets VarName to Data (binary).

Parameters	Description	Values
VarName	Name of a variable	Case sensitive
Data	Raw binary data	Binary data

**RETURN VALUES**

Boolean - True if successful.

**SEE ALSO**

**Var.Num.SetVar, Var.Str.SetVar**

**EXAMPLE**

```
Tool.Var.SetVar "IsSet" , Tool.Var.Bin.SetVar "Mode" , 2
```

This line of code sets the variable, "Mode", to 2.

**VAR.BIN.TOBASE64(VARNAME) AS STRING**

**Var.Bin.ToBase64** converts binary data in VarName to Base64 format.

Parameters	Description	Values
VarName	Name of variable containing binary data	Case sensitive

**RETURN VALUES**

String that contains data in Base64 format

**SEE ALSO**

**Var.Bin.ToHex**

**EXAMPLE**

```
Tool.Var.SetVar "Mode_Base64" , Tool.Var.Bin.ToBase64 ("Mode")
```

**VAR.BIN.TOHX(VARNAME) AS STRING**

**Var.Bin.ToHex** converts binary data in VarName to hexadecimal format.

Parameters	Description	Values
VarName	Name of variable that contains binary data	Case sensitive

**RETURN VALUES**

String that contains data in hexadecimal format

**SEE ALSO**

**Var.Bin.ToBase64**

**EXAMPLE**

```
Tool.Var.SetVar "ID_Hex" , Tool.Var.Bin.ToHex ("ID")
```

---

## Var.Num

VAR.NUM commands handle numeric information in the form of integers.

**VAR.NUM.DEC(VARNAME)**

**Var.Num.Dec** decrements the number in VarName by one.

Parameters	Description	Values
VarName	Name of variable that contains numeric data	Case sensitive

**RETURN VALUES**

None

**SEE ALSO**

**Var.Num.Inc**

**EXAMPLE**

```
Tool.Var.Num.Dec "Count"
```

**VAR.NUM.DIFF(VARNAME1, VARNAME2) AS INTEGER**

**Var.Num.Diff** subtracts the integer stored in VarName2 from the integer stored in VarName1 and returns the difference between the variables.

Parameters	Description	Values
VarName1	Name of variable that contains numeric value	Case sensitive
VarName2	Name of variable that contains numeric value	Case sensitive

**RETURN VALUES**

Integer

**EXAMPLE**

```
Tool.Var.SetVar "Difference" , Tool.Var.Num.Diff("Total",
"SubTotal")
```

**VAR.NUM.GETVAR(VARNAME) AS INTEGER**

**Var.Num.GetVar** returns the numeric value assigned to the variable.

Parameters	Description	Values
VarName	Name of variable that contains numeric value	Case sensitive

**RETURN VALUES**

Integers

**SEE ALSO**

**Var.Str.GetVar**

**EXAMPLE**

```
Tool.Var.SetVar "NumMode" , Tool.Var.Num.GetVar ("Mode")
```

**VAR.NUM.INC(VARNAME)**

**Var.Num.Inc** increments the integer stored in VarName by one.

Parameters	Description	Values
VarName	Name of variable that contains numeric value	Case sensitive

**RETURN VALUES**

None

**SEE ALSO**

**Var.Num.Dec**

**EXAMPLE**

```
Tool.Var.Num.Inc "Count"
```

**VAR.NUM.ISEQUAL(VARNAME, VALUE) AS BOOLEAN**

**Var.Num.IsEqual** determines if an integer stored in VarName is identical to Value.

Parameters	Description	Values
VarName	Name of variable that contains numeric data	Case sensitive
Value	An integer that is compared to VarName	Integer

**RETURN VALUES**

Boolean - True if the variable is equal to the value.

**SEE ALSO**

**Var.Str.IsEqual**

**EXAMPLE**

```
Tool.Var.Num.IsEqual("Mode", 2)
```

**VAR.NUM.ISGREATER(VARNAME, VALUE) AS BOOLEAN**

**Var.Num.IsGreater** indicates if data in VarName is greater than Value.

Parameters	Description	Values
VarName	Name of a variable that contains numeric data	Case sensitive
Value	An integer that is compared to VarName	Integer

**RETURN VALUES**

Boolean - True if the variable is greater than Value

**EXAMPLE**

```
Tool.Var.Num.IsGreater("Total", 10000)
```

**VAR.NUM.ISLESS(VARNAME, VALUE) AS BOOLEAN**

**Var.Num.IsLess** indicates if an integer in VarName is less than the integer Value.

Parameters	Description	Values
VarName	Name of a variable that contains numeric data	Case sensitive
Value	An integer that is compared to VarName	Integer

**RETURN VALUES**

Boolean - True if the variable is less than Value

**EXAMPLE**

```
Tool.Var.Num.IsLess("Total", 100)
```

**VAR.NUM.SETVAR(VARNAME, VALUE) AS BOOLEAN**

**Var.Num.SetVar** assigns Value to the VarName.

Parameters	Description	Values
VarName	Name of a variable that contains numeric data	Case sensitive
Value	An integer assigned to VarName	Integer

**RETURN VALUES**

Boolean - True if the Value is assigned to VarName.

**EXAMPLE**

```
Tool.Var.Num.SetVar "Mode", 4
```

**VAR.NUM.SUM(VARNAME1, VARNAME2) AS INTEGER**

**Var.Num.Sum** sums integers contained in VarName1 and VarName2.

Parameters	Description	Values
VarName1	Name of variable that contains numeric data	Case sensitive.
VarName2	Name of variable that contains numeric data	Case sensitive

**RETURN VALUES**

Integer - The sum of all parameters

**EXAMPLE**

```
Tool.Var.SetVar "Sum" , Tool.Var.Num.Sum( "LenFirstName" ,  
"LenLastName" )
```

---

## Var.Str

Var.Str commands handle string data.

**VAR.STR.CONCAT(VARNAME1, VARNAME2) AS STRING**

**Var.Str.Concat** concatenates the string contained in VarName2 to the end of the string contained in VarName1.

Parameters	Description	Values
VarName1	Name of variable that contains string data	Case sensitive
VarName2	Name of variable that contains string data	Case sensitive

**RETURN VALUES**

String

**EXAMPLE**

```
Var.Str.SetVar "Comma" , " , "
```

```
Tool.Var.SetVar "Name" , Tool.Var.Str.Concat("LastName" , "Comma" , "FirstName")
```

**VAR.STR.GETVAR(VARNAME) AS STRING**

**Var.Str.GetVar** returns the string stored in VarName.

Parameters	Description	Values
VarName1	Name of variable that contains string data	Case sensitive

**RETURN VALUES**

String

**EXAMPLE**

```
Tool.Var.SetVar "Name" , Tool.Var.Str.GetVar("LastName")
```

**VAR.STR.ISEQUAL(VARNAME, VALUE) AS BOOLEAN**

**Var.Str.IsEqual** determines if string data contained in VarName is identical to Value.

Parameters	Description	Values
VarName	Name of variable that contains string data to compare with Value	Case sensitive
Value	A string that is compared to VarName	

**RETURN VALUES**

Boolean - True if equal

**EXAMPLE**

```
Tool.Var.Str.IsEqual "LastName" , "Jones"
```

**VAR.STR.LEFT(VARNAME, COUNT) AS STRING**

**Var.Str.Left** returns left number of characters from string data contained in VarName.

Parameters	Description	Values
VarName	Name of variable that contains string data	Case sensitive
Count	Integer number of characters for extraction from the left edge of string data in VarName	

**RETURN VALUES**

String

**EXAMPLE**

```
Tool.Var.SetVar "Left" , Tool.Var.Str.Left("LastName" , 3)
```

VAR.STR.RIGHT(VARNAME, COUNT) AS STRING

**Var.Str.Right** returns right most specified number of characters from the string data contained in VarName.

Parameters	Description	Values
VarName	Name of variable that contains string data	Case sensitive
Count	Integer number of characters for extraction from the right edge of string data in VarName.	

**RETURN VALUES**

String

**EXAMPLE**

```
Tool.Var.SetVar "Right" , Tool.Var.Str.Right("Name", 2)
```

VAR.STR.SETVAR(VARNAME, VALUE) AS BOOLEAN

**Var.Str.SetVar** assigns string data to VarName.

Parameters	Description	Values
VarName	Name of variable created for storing string data	Case sensitive
Value	String data for storing in VarName	

**RETURN VALUES**

Boolean - True if successful

**EXAMPLE**

```
Tool.Var.Str.SetVar "IsCardSwiped", "Yes"
```

# VB Mode

---

# 6

posVisual VB mode generates PC-resident applications that control @pos.com iPOS terminals. Programmers creating electronic transaction systems should use the VB mode while programmers upgrading legacy systems should use the Script mode.

VB mode applications are programs in which the entire application resides on the host. That is opposed to Script mode applications in which screen generation and data collection is handled by a script or program residing on the iPOS terminal.

## WHEN TO USE VB MODE

posVisual VB mode is especially useful to developers who need to quickly develop @pos.com iPOS applications or prepare mock-ups and demos for project proposals.

posVisual supplies the tools to quickly implement new applications for @pos.com iPOS terminals. posVisual extends the Visual Basic programming environment to include special forms, controls and classes for creating @pos.com iPOS applications. With posVisual, the need to generate machine or C code is eliminated. All of the low level details are taken care of by the posVisual Assistant.

## DEVELOPING APPLICATIONS IN THE VB MODE

The visual programming environment of posVisual provides a high degree of parallelism between the host environment and iPOS environment. Objects appearing on host forms are mirrored on iPOS screens with the exception of those intended to be hidden on screen. Both platforms operate simultaneously from the same program. Most program responses triggered by actions at the host, closely match the program responses triggered by actions to the iPOS terminal.

Honing iPOS applications proceeds quickly. As soon as the host runs the posVisual application, the form appearing on the host screen appears on the iPOS terminal. The close coupling between host and iPOS terminals enables rapid iteration of application designs. The responsive posVisual interface allows cause-and-effect between application changes and application performance to remain clear.

Because developer hosts can run the VB mode application, it is possible to test many aspects of this application without having to hook an iPOS terminal to the host. To test a VB mode application, set the project into test mode by setting the global variable, TESTMODE, to "True." TESTMODE is found at the top of the ModMain module.

## COMMUNICATION

Communication with iPOS terminal can be configured by port number and baud rate. And also, maximum number of ports to be used can be changed accordingly. If port number specified beyond the range, COM1 will be selected as a default.

```

    SigBox.Ports = 4 ←
    ,           ^
    '           change this to define maximum number of ports to be
used.
    '
    '
    SigBox.Port = 0 ←
    ,           ^
    '           change this to connect to necessary port
    '           "0" is for auto select
    '
    '
    SigBox.BaudRate = Baud9600 ←
    ,           ^
    '           change this to select required baud rate
    '           if necessary
    '
    '
    V.A.start           'VA Required
    '---posVisual required code ABOVE

```

## DISTRIBUTING POSVISUAL APPLICATIONS

After completing a posVisual VB mode application, create an exe program using the normal Visual Basic "Make Exe" process.

To distribute applications to other machines, do one of the following:

1. Copy the "Distribution Setup folder" to the target machine. Run Setup.exe. This places all files normally required to run the program on the target machine. The executable file should now be able to run.

-or-

2. Use an Install creation application (such as Visual Basic's Application Setup Wizard, or InstallShield) to include all the necessary files in a Setup program for your executable file. Depending on the project, additional files can be required. The additional files that might be required are the following: **mxVAhost.dll** and **VisualControls.ocx**. Both files are found in the system directory.

## VB MODE SOFTWARE COMPONENTS

Several software components that are required for posVisual to function properly. Those components are the following:

## CLASS

Name	Description	Additional Notes
VAEvents.cls	This class allows code to be put behind certain VA events.	Code for this component can be modified in certain areas. See the code in the class module for more information.

## CONTROLS

Name	Description	Additional Notes
VisualControls.ocx	Provides the SigControl and MSRArea controls	DO NOT MODIFY. These function like normal OCX controls and must be listed in the controls windows.

## FORMS

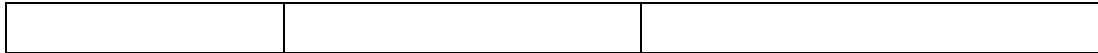
Name	Description	Additional Notes
frmSplash.frm	Application specific splash screen.	A modifiable graphical form for announcing an application or function or for displaying logos on the iPOS terminal. Modify the splash screen for the form under development.
padStart.frm	The first screen must always be padStart.	To add more forms, see the Forms chapter.
MDIMain.frm	This is the parent screen for all the displayed forms.	DO NOT MODIFY.

## MODULES

Name	Description	Additional Notes
modMain.bas	modMain.bas module connects posVisual classes.	Add functions and subroutines to this file in the areas designated inside the code.

## REFERENCES

Name	Description	Additional Notes
Sigbox.ocx	Sigbox.ocx provides connectivity between host and iPOS terminals. Sigbox.ocx also supplies additional “non-standard” commands.	DO NOT MODIFY. Note: Sigbox.ocx is a control, but is used as a reference. Normally the SigBox control is not needed for a posVisual project, but it needs to exist in the project as reference. SigBox is accessed indirectly through posVisual.
MxVAhost.dll	Controls the communication with the iPOS terminal	DO NOT MODIFY.



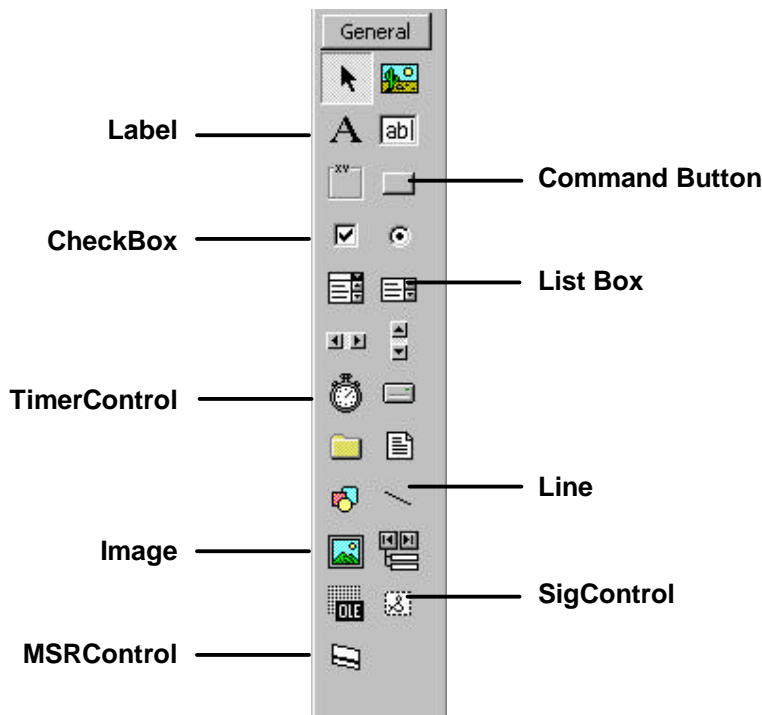
## VB Mode Controls

posVisual forms generate screen displays that run in a special environment - on @pos.com iPOS terminals. The special demands of the @pos.com iPOS environment requires a different set of controls than the Windows environment native to Visual Basic. Thus, while many of the controls function in posVisual as they do in Visual Basic alone, some do not.

### POSVISUAL TOOL BOX

Controls in the following diagram work in posVisual VB mode, some with modified capabilities. The MSRControl and SigControl are specialized controls supplied by posVisual.

Controls that can be used in VB mode are shown and described below:



### CONTROL FONTS

Font characters displayed on @pos.com iPOS terminals come in six sizes. All characters in each font size have the same pixel dimensions. The posVisual font families are mapped onto the Courier New font families in the properties boxes of posVisual controls. The correspondence between the posVisual font sizes and Courier New font sizes is shown in the table below.

iPOS Font Size in Pixels	posVisual Font Size in Points
6 x 8	10 pt
8 x 8	12 pt
8 x 12	13 pt
12 x 16	17 to 20 pt
16 x 16	23 to 26 pt
16 x 24	27 to 30 pt

CHECKBOX

The posVisual VB mode Check Box operates just as the Visual Basic Check Box. In VB mode the check box can also be used in an option group by setting the tag property.

Property	How property is used
Font-Size	Controls the size on the font on the pad. See the Fonts section in this chapter.
Value	Sets the default value when displayed, depending on the status of the check box..
Caption	Controls the text displayed beside the check box.
Tag	Setting Tag to “hide” (lowercase) causes this control to be hidden. Setting Tag to “groupname” allows a check box to be used as part of an option group. When using multiple check boxes on the form, set the Tag property of each to the same value. Then, checking one box causes all others to uncheck. Groupname can be any string except the words “groupname” and “hide”.

**USING THE CONTROL**

Touching the CheckBox image on the iPOS terminal checks and un-checks the control. Use the normal Visual Basic control properties for determining the status of the check box (checkbox1.value).

**SEE ALSO**

Commands: **V.A.PadFontSize**, **V.A.FontWidth**, **V.A.FontHeight**

COMMAND BUTTON

The posVisual command button functions the same way as the command button of the pure Visual Basic environment.

Property	How property is used
Font-Size	Controls the size on the font on the pad. See Control Fonts section in this chapter.
Caption	Controls the text displayed inside the button.

Value	Sets the default value when displayed. Setting to True activates the click event of the command button.
Tag	Setting Tag to "hide" (lowercase) makes the control invisible.
Style	Setting Style to "0 - Standard" displays a normal button. Setting Style to "1 - Graphical" displays a transparent button.

**USING THE CONTROL**

Use like a normal Visual Basic button. Put code under click\_Event to execute commands when clicked.

**IMAGE**

Image controls display a black-and-white bitmap graphics on iPOS screens.

Images in VB mode stretch to fit the size of the image control. Images in VB mode can be in any VB compatible format (BMP, WMF, etc.).

Static images in VB mode can be stored in the non-volatile iPOS terminal so they appear faster and don't have to be transmitted to the iPOS. To store an image, command posVisual to pre-load the form where the image resides. Use the Load "formname" in the "A\_PreLoadForms" event of the Class module "VAEvents."

**Note:** For best image quality set the image control to the exact size of the imported image. Stretching the image can drastically alter the quality of the image displayed on the iPOS terminal. Images should not be larger than 240 x 320 pixels and must not extend past the edges of the screen.

Property	How property is used.
Tag	Setting Tag to "hide" (lowercase) causes the control to be invisible.
BorderStyle	Setting BorderStyle to "0 - None" causes the Image control to display no border. Setting BorderStyle to "1 - Fixed Single" causes the Image control to display a border around the picture.
ToolTipText	Text entered into ToolTipText property box shows temporarily until the image displays - in Web fashion.
Stretch	The stretch property is not really used by posVisual. Setting the property to "True" allows posVisual to display a sized image. Leaving the property "False", causes Visual Basic to resize it back to the original image size when the program runs. If Stretch is "False", the size of the image cannot change.

**USING THE CONTROL**

Use the posVisual image control like a normal Visual Basic image control. Set stretch property to "True" for correct functionality.

## LABEL

The Label control in VB mode functions like a normal label and includes the capability to word wrap.

<b>Property</b>	<b>How property is used</b>
Tag	Setting Tag to "hide" (lowercase) makes this control invisible. Setting Tag to "clear" (lowercase) clears text from the label before it displays.
Font-Size	Font-Size sets the size on the font displayed on the form.
BorderStyle	Set BorderStyle to "0 - None" to make the label display with no border. Set BorderStyle to "1 - Fixed Single" to make the label display with a border.

### ***USING THE CONTROL***

Use posVisual labels just like Visual Basic labels.

### ***SEE ALSO***

Commands: **V.A.DisplayLabel**

## LINE

The Line control works the same in posVisual VB mode as in the pure Visual Basic environment.

## LISTBOX

The ListBox control in posVisual VB mode works just like a ListBox control in pure Visual Basic. Use for displaying a list of selectable items that user can pick by touching the iPOS terminal screen. A ListBox on the iPOS terminal adds a scroll bar and buttons when there are more lines in the ListBox than can be displayed all at once.

Property	How property is used
Tag	Setting Tag to "hide" (lowercase) hides this control.  Setting Tag to "scroll" places a scroll bar on the right side even if there is no need to scroll.  Setting Tag to "groupname" allows use of the ListBox as an option group. If using multiple ListBoxes on a form, set the tag properties of each ListBox to the same value. Then selecting an item on one ListBox, de-selects the item on all the others.
Font-Size	Sets the size on the font on the pad.
ToolTipText	After the ListBox displays on the iPOS terminal, the ToolTipText property displays the number of selections the ListBox holds. For example, if a ListBox draws on a screen and this property contains "7", this means that the ListBox displays 7 lines at a time.
Enabled	Setting this property to True, allows the posVisual code to accept a click event.

### **USING THE CONTROL**

Use like a normal Visual Basic ListBox.

## MSRCONTROL

Place the MSR Control anywhere on the form to activate the MSR. This control is visible on the host screen yet invisible on the iPOS display. Once activated, the MSR captures data from card swipes. If a good swipe occurs, posVisual fires the GoNext Event for the form. The GoBack event fires where the MSR times out. Such an event occurs where the iPOS didn't get a swipe within the time limit programmed for the swipe.

In VB Mode, do not put code under the MSR GoodSwipe or BadSwipe events. posVisual ignores code in those events while in VB mode. Instead, put code under the GoNext event for the current form for good swipe. (Use timeout = 0 for no timeout).

### EVENTS:

- GoNext - fires after a good swipe.
- GoBack - fires after no swipe before timeout.

Property	How property is used
Timeout	The number of seconds to wait before firing the GoBack event on the form. Property defaults to "0" for no timeout.

**USING THE CONTROL**

When using the MSRControl to design screens, the control is visible on the host development screen but does not show on the iPOS screen. After drawing this control on the form it will not appear on the device.

To obtain data from the MSR, call the **MSR.AssignTrackNames** method, then use the MSR object to get the properties from the MSR. See MSR object in this chapter.

Draw this control on the developer screen.

**SEE ALSO**

Commands:

## SIGCONTROL

The Signature Control displays signature data captured from the iPOS terminal as a series of dots within the signature area. Use the Line control to draw a signature line in the signature area.

Property	How property is used
BorderStyle	Setting BorderStyle to "0 - None" causes the SigControl to be drawn without a border.  Setting BorderStyle to "1 - Fixed Single" causes a border to be drawn.

**USING THE CONTROL**

Use this control to draw a signature area. To save the signature into a string variable, use the command: **V.A.Signature**. See VB Mode commands.

**SEE ALSO**

Commands: **V.A. Signature**, **V.A.Shrinkwrap**

## SHAPE

The Shape tool only displays rectangles on the iPOS terminal. After placing a Shape on a form, a dialog box appears asking if the object should be a "Box (Filled)" or a "Frame (Transparent)". For correct functionality be sure the shape is set to rectangle.

Property	How property is used
BorderStyle	Setting BorderStyle to "0 - None" causes the rectangle to be drawn without a border.  Setting BorderStyle to "1 - Fixed Single" causes the border to be drawn around the shape.
FillStyle	Setting FillStyle to "0 - Solid" fills the rectangular area with black.

	Setting FillStyle to "1-Transparent" draws a transparent rectangle.
FillColor	Setting FillColor to "0 -Solid" fills the rectangle with Black. Setting FillColor to non-zero fills the rectangle with White.

Note : If "FillStyle" sets to "1-Transparent", FillColor property has no control to make the shape filled with black which described above.

## TIMERCONTROL

The posVisual VB mode Timer Control does not operates like Visual Basic timer control.

The coding under the Timer1\_Timer() event only execute after the given interval reach after enabling the timer, and it executes only once. VA only recommend one timer control per form.

Property	How property is used
Interval	Controls the delay for the action follows the timer
Enabled	Activates/deactivates timer

### ***USING THE CONTROL***

posVisual VB mode timers do not act like Visual Basic timers. VB mode timers are 'countdown' timers. Enter the countdown time in the Interval box of the timer's properties.

---

## VB Mode Commands

All posVisual commands require a command prefix. This prefix depends on the posVisual mode. For VB Mode, there 3 command prefixes “V.A.”, “MSR” and “SigBox”. Use posVisual Command prefixes to bring up Visual Basic’s autocoding drop-down lists. They help complete commands and walk the developer through required parameters. This section describes all the posVisual commands that are available.

### V.A.

CurrentScreen.....	66
DisplayPrevious Screen.....	67
DisplayScreen .....	67
DrawLabel .....	67
DrawScreenHotSpot.....	68
ErrorDescription .....	69
FontHeight .....	69
FontWidth .....	70
InvertControl .....	70
MagOff .....	71
MagOn .....	71
PadFontSize .....	72
RedrawScreen .....	72
RefreshListBox .....	73
ScreenStackClear .....	73
ShowError .....	73
ShrinkWrap .....	74
Signature .....	74
Smoothing .....	75
Terminate .....	75

## MSR

AccountNumber.....76  
 AssignTracknames.....76  
 FirstName .....77  
 LastName .....77  
 Reset .....77  
 Track1, Track2, Track3.....78

## SigBox

SigBox .....79

posVisual relies on the methods and properties of the SigBox object to function. Most of the properties and methods of SigBox are accessed indirectly through the posVisual interface. For developers wanting to use the large number of methods and properties of SigBox directly, information about them can be found in the Software Developers Kit (SDK) documentation.

## V.A.

V.A. Commands relate specifically to an attached iPOS terminal. Some of V.A. commands are used by posVisual to initialize communication with the terminal.

### V.A.CURRENTSCREEN

Command Type: Object

**V.A.CurrentScreen** is a form method holding information about the currently displayed form. **V.A.CurrentScreen** gives access to all the properties and methods of a form.

Parameters	Description	Values
None		

**RETURN VALUES**

None

**SEE ALSO**

Visual Basic Manuals: Forms

**EXAMPLE**

```
Dim CurForm as Form

Set curForm = V.A.CurrentScreen
```

**V.A.DISPLAYPREVIOUSSCREEN**

Command Type: Method

**V.A.DisplayPreviousScreen** navigates back to the previous screen. If the current screen is the first screen, posVisual terminates normally.

Parameters	Description	Values
None		

**RETURN VALUES**

None

**SEE ALSO**

None

**EXAMPLE**

```
V.A.DisplayPreviousScreen
```

**V.A.DISPLAYSCREEN SCREENNAME**

Command Type: Method

**V.A.DisplayScreen** displays ScreenName.

Parameters	Description	Values
ScreenName	The name of the screen to display	For any posVisual form name to display on the device; That form name should start with "pad".

**RETURN VALUES**

None

**SEE ALSO**

**V.A. DisplayPreviousScreen, V.A.RedrawScreen**

**EXAMPLE**

```
V.A. DisplayScreen, padThankYou
```

**V.A.DRAWLABEL LBL [, CLEARSCREENUNDER LABEL]**

Command Type: Method

**V.A.DrawLabel Lbl** draws a label on the current screen. The label does **not** have to come from the current screen. This allows the developer to change the contents, then display the new contents on the screen.

Updating a label on the iPOS terminal screen is the one of the main uses for this command. For example, a label on the screen with a caption of “Please Swipe your Card.” can change to read “Thank you, Please wait”...**V.A.DrawLabel Lbl** changes the labels caption but does not display on the screen until a **V.A.DrawLabel** command executes or the **V.A.RedrawScreen** command is executed.

Error messages make a good application for the **DrawLabel** command. To do that:

1. Create a label on a screen containing a message for a user making an expected error.
2. Set the tag property of the label to “hide”. This keeps the label from displaying.
3. If the user makes the anticipated mistake, update the caption then have the program execute a **DrawLabel** command so that the label text appears on the iPOS terminal.

Parameters	Description	Values
Lbl	A label for display	Use full name of label. If the label resides on a different form prefix the label name with the label, form name. Example: <b>padScreen1.lblMessage</b>
<b>ClearScreenUnderLabel</b> (optional parameter)	Setting <b>ClearScreen UnderLabel</b> to “True” clears the screen under the label before drawing. This allows the label to overwrite other screen elements.	Optional parameter Boolean: True or False

**RETURN VALUES**

None

**SEE ALSO**

Controls: Labels

**EXAMPLE**

`V.A.DrawLabel lblMessage, True`

## V.A.DRAWSCREENHOTSPOT

Command Type: Method

**V.A.DrawScreenHotSpot** creates an invisible button that covers then entire screen. When the screen is touched anywhere, the hidden button activates and the **GoNext** command of the screen fires.

Parameters	Description	Values
None		

**RETURN VALUES**

None

**SEE ALSO**

Forms: **GoNext** event

**EXAMPLE**

V.A.DrawScreenHotSpot

**V.A.ERRORDESCRIPTION**

Command Type: Property

**V.A.ErrorDescription** displays error messages to the user. To display an error message, set **V.A.ErrorDescription** to the error message, then go to another screen with **V.A.DisplayScreen** or re-display the current screen with **V.A.RedrawScreen**. After the screen redraw, the error message appears along with an "OK" button. When the OK button is clicked, posVisual starts over at padStart.

**Note:** No variables lose information in the process.

Parameters	Description	Values
None		

**SEE ALSO**

**V.A.ShowError**

**EXAMPLE**

V.A.ErrorDescription = "Database connection lost."

V.A.RedrawScreen

**V.A.FONTHEIGHT PADFONT**

Command Type: Property

**V.A.FontHeight pad** returns the height of the padFont parameter in pixels.

Parameters	Description	Values
PadFont	A defined pad font size.	Font6x8 Font8x8 Font8x12 Font12x16 Font16x16 Font16x24

**RETURN VALUES**

Integer value

**SEE ALSO**

**V.A.FontWidth, V.A.PadFontSize**

**EXAMPLE**

FontYSize = V.A.FontHeight(Font8x8)

## V.A.FONTWIDTH PADFONT

Command Type: Property

**V.A.FontWidth.padFont** returns the width of the indicated font in pixels.

Parameters	Description	Values
PadFont	A defined pad font size	Font6x8 Font8x8 Font8x12 Font12x16 Font16x16 Font16x24

**RETURN VALUES**

Integer value

**SEE ALSO**

V.A.FontHeight, V.A.PadFontSize

**EXAMPLE**

FontXSize = V.A.FontWidth (Font16x16)

## V.A.INVERTCONTROL CONTROLNAME

Command Type: Method

**V.A.InvertControl** reverses or inverts the colors of ControlName on the screen.

Parameters	Description	Values
ControlName	Any existing control on a pad screen that has Left, Top, Width, and Height properties	Examples of the controls are: Labels, buttons, and images.

**RETURN VALUES**

None

**EXAMPLE**

V.A.InvertControl lblMessage

**V.A.MAGOFF**

Command Type: Method

**V.A.MagOff** turns off the Mag Card reader (MSR Reader).

Parameters	Description	Values
None		

**RETURN VALUES**

None

**SEE ALSO****V.A.MagOn**, CONTROLS: MSRcontrol**EXAMPLE**

V.A.MagOff

**V.A.MAGON**

Command Type: Method

**V.A.MagOn** turns on the Mag Card reader (MSR Reader). This method provides the same functionality as the MSRControl.To use **V.A.MagOn** put the command in the **ScreenInitialize** event on a form.

Parameters	Description	Values
None		

**RETURN VALUES**

None

**SEE ALSO****V.A.MagOff**, CONTROLS: MSRControl**EXAMPLE**

V.A.MagOn

**V.A.PADFONTSIZE**

Command Type: Method

**V.A.PadFontSize** returns the size of the font in the defined pad font size, **SigBoxTextFont**.

Use **V.A.PadFontSize** in connection with **FontWidth** and **FontHeight** to determine the screen width of a character in a control. To learn how wide the characters are in a label. Pass the result of **PadFontSize** to **PadWidth** (see example below)

Parameters	Description	Values
	Any object that has a Visual Basic Font.	

**RETURN VALUES**

None

**SEE ALSO**

V.A.FontWidth, V.A.FontHeight

**EXAMPLE**

This command returns the width of each character in the label. Each character has the same width so only one value is returned.

```
CharWidth = V.A.FontWidth(V.A.PadFontSize(lblMessage))
```

**V.A.REDRAWSCREEN**

Command Type: Method

**V.A.RedrawScreen** clears the current screen and redraws it.

Parameters	Description	Values
None		

**RETURN VALUES**

None

**SEE ALSO****V.A.DisplayScreen****EXAMPLE**

```
V.A.RedrawScreen
```

**V.A.REFRESHLISTBOX**

Command Type: Method

V.A.RefreshListBox redraws the list box control on POS terminal. You need this if you add new items to the list box or remove some.

Parameters	Description	Values
ListBoxControl	Valid List Control Object	

**RETURN VALUES**

None

**SEE ALSO**

Using the ListBox Control

**EXAMPLE**

V.A.RefreshListBox

**V.A.SCREENSTACKCLEAR**

Command Type: Method

V.A.ScreenStackClear clears stack of screens, which loaded to POS terminal. If you need to go back to a previous screen, you must clear screen stack.

Parameters	Description	Values
None		

**RETURN VALUES**

None

**SEE ALSO**

V.A.DisplayScreen

**EXAMPLE**

V.A.ScreenStackClear

**V.A.SHOWERROR ERRORMSG**

Command Type: Method

V.A.ShowError Error displays an error message on the iPOS screen with an "OK" Button. When this button is clicked, posVisual starts over at padStart.

Parameters	Description	Values
ErrorMsg	Text message to display	String Value

**RETURN VALUES**

None

**SEE ALSO**

V.A.ErrorDescription

**EXAMPLE**

V.A.ShowError "Cannot write to file"

**V.A.SHRINKWRAP**

Command Type: Property

Set **ShrinkWrap** to True to reduce the amount of memory storage a signature occupies. Setting **ShrinkWrap** to False causes the entire signature screen, 320 x 240 pixels, to save to memory. With **ShrinkWrap** set to "True", the extra pixels surrounding the signature are trimmed off before the save. A signature with dimensions of 50 x 200 pixels takes up more than **seven** times as much space in memory when **ShrinkWrap** is set to False than when it is set to True.

Parameters	Description	Values
= True	"True" shrink wraps the signature.	
= False	"False", is the default value.	

**RETURN VALUES**

None (Write only)

**SEE ALSO**

**V.A.Signature**

**EXAMPLE**

V.A.ShrinkWrap = True

**V.A.SIGNATURE**

Command Type: Property

**V.A.Signature** saves a signature into a string variable. This property is the recommended way to obtain signatures from POS terminals. Set the property to a string variable, then write the variable to a database or other file.

Parameters	Description	Values
None		

**RETURN VALUES**

Signature in the form of a string

**SEE ALSO**

**V.A.ShrinkWrap**

**EXAMPLE**

Dim Sig as String

Sig = V.A.Signature

### V.A.Smoothing

Command Type : Property

**V.A.Smoothing** Sets the level of smoothing applied to captured signatures. Smoothing is used to remove unwanted lines from the signature. Average smoothing is 5. More than 10 is not recommended because it causes loss of data in signature.

Parameters	Description	Values
None		

**RETURN VALUES**

None (Write only).

**SEE ALSO**

V.A.Signature

**EXAMPLE**

V.A.Smoothing = 5

### V.A.TERMINATE

Command Type: Method

**V.A.Terminate** ends the current application.

Parameters	Description	Values
None		

**RETURN VALUES**

None

**EXAMPLE**

V.A.Terminate

## MSR

The MSR object can be accessed at any point after a good swipe.

### MSR.ACCOUNTNUMBER

Command Type: Property

**MSR.AccountNumber** returns the account number that was read off of Track 1. This property does not work with all Mag Stripe cards. The remaining data on track1 can still be accessed using MSR.Track1 property.

Parameters	Description	Values
None		

**RETURN VALUES**

String

**EXAMPLE**

```
Dim AcctNum as String
```

```
AcctNum = MSR.AccountNumber
```

### MSR.ASSIGNTRACKNAMES

Command Type: Method

**MSR.AssignTrackNames** assigns all MSR information Track 1, Track 2 and Track 3 information: FirstName, LastName, and AccountNumber. Call **MSR.AssignTrackNames** before any property gets information. On a good swipe, this method gets called automatically.

Parameters	Description	Values
None		

**RETURN VALUES**

None

**EXAMPLE**

```
MSR.AssignTrackNames
```

**MSR.FIRSTNAME**

Command Type: Property

**MSR.FirstName** returns first name information from track 1. **MSR.FirstName** does not work with all Mag Stripe cards. The data on Track1 can still be accessed by using **MSR.Track1**.

Parameters	Description	Values
None		

**RETURN VALUES**

String

**EXAMPLE**

```
Dim FName as String
FName = MSR.FirstName
```

**MSR.LASTNAME**

Command Type: Property

**MSR.LastName** returns the last name information from Track 1. **LastName** does not work with all Mag Stripe cards. The data on Track1 can be accessed by using **MSR.Track1**.

Parameters	Description	Values
None		

**RETURN VALUES**

String

**EXAMPLE**

```
Dim FULLName as String
FULLName = Trim(MSR.FirstName) + " " + Trim(MSR.LastName)
```

**MSR.RESET**

Command Type: Method

**MSR.Reset** clears all MSR object properties (FirstName,Track1, etc).

Parameters	Description	Values
None		

**RETURN VALUES**

None

**EXAMPLE**

```
MSR.Reset
```

MSR.TRACK1

MSR.TRACK2

MSR.TRACK3

Command Type: Property

Returns the information from Track 1, Track 2 or Track 3.

Parameters	Description	Values
None		

**RETURN VALUES**

String

**EXAMPLE**

```
Dim TrackOne as String
TrackOne = MSR.Track1
```

---

## SigBox

Normally posVisual, V.A. Commands, and the MSR object provide all the necessary interface and controls needed to create iPOS terminal applications. If some facet of iPOS terminal function is beyond the properties and methods in the V.A. and MSR objects, use the methods and properties of the SigBox object.

The SigBox object contains over 140 properties and methods, and is not re-documented in detail here because of size considerations. Refer to those objects and properties in the @pos.com Software Developers Kit (SDK) included with posVisual.

---

## VAEvent Class Module

VA Events are modifiable subroutines in the VAEvent Module governing the start-up and shut-down of iPOS terminals.

**A\_ERRORBEFORESTART** EVENT

**A\_ErrorBeforeStart** event fires if posVisual fails to start or find the iPOS terminal.

## A\_PRELOADFORMS EVENT

**A\_PreLoadForms** stores code for pre-loading graphics into the non-volatile memory of iPOS terminals and fires upon start-up. Preloading images increases the speed with which the iPOS terminals operate. Fetching images from memory is faster than downloading them from a host.

### *EXAMPLE*

```
V.A.PreLoadImagesForms.Add padScreenWelcome
```

When a preload command executes, it loads **all** the images on the form into memory. In the example, the command PreLoadingImagesForms command loads the images on the form called **padScreenWelcome**.

**Note:** Images on a form cannot be selectively loaded. This event triggers before the splash screen is displayed.

## A\_PRESPLASHSCREEN EVENT

**A\_PreSplashScreen** fires before a splash screen is displayed on the iPOS screen.

## A\_PRETERMINATE EVENT

**A\_PreTerminate** fires just before normal termination of a posVisual application. With the **A\_PreTerminate** command, databases and files can be closed and objects freed up.

## A\_PREUNLOADSPLASHSCREEN EVENT

**A\_PreUnloadSplashScreen** fires just before the splash screen is removed. Place code in the event subroutine, to connect to databases, initialize variables, etc. With this line, status line displaying text information on the Splash screen, to reflect the activities of the code, can be implement

# Error Messages

# 7

## VB MODE ERROR MESSAGES

posVisual provides error messages that depend on the posVisual mode. In VB mode the posVisual error messages display in the same way as Visual Basic error messages.

Error/Warning	Occurred while	Description	Remedy
posVisual will not be able to display this control	Putting new controls on a posVisual form	This control is not a control that posVisual can display on the screen.	This is just a warning. In VB Mode many controls do not display on a iPOS terminal screen, such as RDO controls, Timer, etc.
Timeout occurred	Downloading Script	The device might have been disconnected while downloading.	Check the connection of the RS232 cable.

## SCRIPT MODE ERROR MESSAGES

Script mode error messages display in the posVisual Assistant message box.

Error/Warning	Occurred while	Description	Remedy
Port error occurred	Downloading Script	This message usually means that the iPOS terminal is not connected or VA is configured to use the wrong port.	Check the connection of the RS232 cable and verify the PORT setting in modMain.
Timeout occurred	Downloading Script	The iPOS terminal might have been disconnected while downloading.	Check the connections of the RS232 cable.
posVisual will not be able to display this control.	Putting new control on a VA form	This control is not a control that posVisual can display on the iPOS screen.	This is just a warning. See the documentation on “Script mode Controls” to see a list of valid controls.
Only posVisual screens can be added...	Adding a new form to your project	In script mode, only posVisual forms, padScreens, can be used.	When adding a new form, be sure to add only posVisual forms. (See “Forms”)
Only Modules and posVisual screens...	Adding a new component to your project	In script mode, only posVisual screens and normal modules can be added to a project.	See previous remedy.
padStart is a Fundamental component to...	Removing padStart form	The padStart form is a required form in script mode projects.	Re-attach the padStart form.

This form is not a valid Script Designer Form	Adding a form	The size of the form is incorrect. posVisual requires a form of a specific size.	If you must use this offending form, then change these properties for that form:  ScaleWidth = 320 ScaleHeight = 240  Be sure you have the ScriptDescriptor.txt file and it is not corrupted. Be sure it is located in the directory with mxVisual.dll
This is not a valid label. It is not the name of a valid label or routine	Trying to download script	The indicated parameter is not a label.	Use the “Show Labels” button to see a list of valid labels. Labels can only be screen or routine names.
ERROR!: Script command PARAMETER:... is not a valid command.	Trying to download script	The error message indicates a script command entered as a parameter to another command is incorrect.	Be sure that <i>every</i> Script mode command begins with the command prefix (See “Command prefix”).  For example, every command must begin with “TOOL”, even commands that are inside other commands, like:  TOOL.Script.IfTrue _ Tool.Var.Find(“NAME”), _ Tool.Script.Bell(Success), _ Tool.Script.Bell(Fail)
ERROR!:Script command PARAMETER:... an invalid parameter type was found in the describer file.	Trying to download script	The ScriptDescriptor.txt file might be corrupted.	Be sure the ScriptDescriptor.txt file is present, is <i>located</i> in the directory with mxVisual.dll and is not corrupted.
ERROR!: Script command:... is not a valid command	Trying to download script	The indicated command is not a valid script command.	Check the manual for the correct spelling and syntax of the indicated command.