iSeries

OPOS Developer's Guide

ingenico

iSeries OPOS Developers Guide
Part Number DL00461, Revision E

Released February 2007

Ingenico Inc.
6195 Shiloh Road, Suite D
Alpharetta, GA 30005
USA
Tel: 678.456.1200
Fax: 678.456.1201
www.ingenico-us.com

Ingenico Canada, Ltd.
79 Torbarrie Road
Toronto, Ontario
Canada M3L 1G5
Tel: 416.245.6700
Fax: 416.245.6701
www.ingenico.ca

Table of Contents

Revision History

Manual Revision	Application Revision	Changes
E	2.40	• The i3070 terminal is now included. All file names have been changed from i6xxx to iSeries to reflect the inclusion of the i3070 terminal. • The OPOS Config window has been changed to a window with multiple tabs and new settings. The Allow Bitmap Background on Signature Form check box was removed since it is no longer used. • Corrected error in *DUKPT Key Serial Number Format* on page 4-13: removed "0x" prefix from second number. • Added a new section: *Best Practices* on page 4-13.
D	2.30	• Added i6780 terminal. • Updated 1.2.1 *Contents*, added driver-only package. • Updated Figure 3 *OPOS Configuration Window*. • Added .NET description to 1.4 *Using the OPOS Controls*. • Added Enable CPEM Reader and Base Slot Number to 1.5 *Setting Up the System Registry*. • Corrected Remarks section of 2.4.1 *StoreFormOnDevice Method*. • Added remarks about physical keys to 2.4.2 *DisplayFormOnDevice Method*. Also added cross reference to Configure Key Masks parameter. • Added remarks about physical keys to 2.5.1 *DataEvent*. • Added the following new methods: 3.4.8 *ConvertSignatureToImageBuffer* 3.4.9 *EnableLiveCapture* 3.4.10 *StartLiveCapture* 3.4.11 *SetDeviceResolution* • Added a command to remarks section of 4.2.1 *DirectIO Method*. • Added 4.2.10 *Delete Receipt Contents Parameter*. • Added 4.5 *DUKPT Key Serial Number Format*.

Manual Revision	Application Revision	Changes
C	2.20	• From section 1.2.4 *OPOS Configuration*, updated the screenshot, removed bullets about IBM 485 Device and Signature Format list box. Added description of the Use CPEM Reader check box. • From section 1.5 *Setting Up the System Registry*, removed the following: 485DeviceID line, SecureMSR/Ing6xxx line, SecurePINPad/Ing6xxx line, and notes about SecureMSR and SecurePINPad. • To section 2.1.3 *Form Types*, bitmaps may now be saved as .gif files. • Changes to section 2.1.5 *File System Maintenance*: changed the message command and added a second approach. • Removed all properties and methods that were added in the previous release, except for **QueryRadioButtonState** Method. • Changed **QueryKeyPadText** to **QueryKeyPadBoardText**. • Removed **QueryKeyBoardText** property. • In section 2.4.2 *DisplayFormOnDevice Method*, added mention of check boxes and radio buttons. • In section 2.4.1 *StoreFormOnDevice Method*, described what to do with alphanumeric keyboard and numeric keypad forms. • In section 2.4.2 *DisplayFormOnDevice Method*, added mention of check boxes and radio buttons. • Removed mention of **StoreFormOnDeviceEx** method and **DisplayFormOnDeviceEx** method. • Removed Chapter 4, SecurePINPad Control. • Added a new Chapter 4, Special Features of the i6000 Series Terminal
B	2.20	Added the following specific properties to Chapter 2: • **PromptMAC1** Property W • **PromptMAC2** Property W • **FormatSpecifier** Property W • **FormatSpecMAC** Property W • **FormFileMAC** Property W • **BitmapMAC1** Property W • **BitmapMAC2** Property W Added the following specific methods to Chapter 2: • **QueryRadioButtonState** Method • **QueryCheckBoxState** Method
A	2.10	Initial release.

CHAPTER 1

Installation

1.1 OPOS Overview

Object linking and embedding for retail point of sale (OPOS) is an object-based programming environment for the development of point of sale (POS) applications. OPOS allows developers to run their applications across a wide range of POS terminals and peripherals. This provides increased flexibility and reduces the effort required to ensure cross-platform operations for POS applications developed using the OPOS standard. OPOS reduces the cost of development usually associated with developing applications for proprietary hardware peripherals.

Epson, NCR, and ICL Retail Systems, in conjunction with Microsoft, developed the OPOS specification. The OPOS specification defines a set of POS device interfaces based on Microsoft's object linking and embedding (OLE) architecture. The OLE control architecture allows development in environments such as Visual Basic and Visual C++, to create POS application software in a device independent manner.

In the past, developers had to create device-specific communication layers in the POS application for each peripheral (see Figure 1, *Architectural Overview,* on page 1).

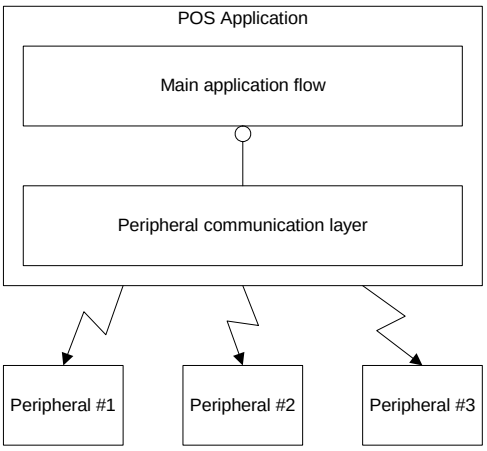


Figure 1 Architectural Overview

To add a new peripheral or install another model of the same type of peripheral, the communication layer had to be modified to support the new device. This usually meant that a new version of the entire application had to be created.

Using OLE control architecture, POS applications communicate with a defined interface (see Figure 2, *OLE Architectural Overview*). This interface remains the same even though a peripheral change occurred.

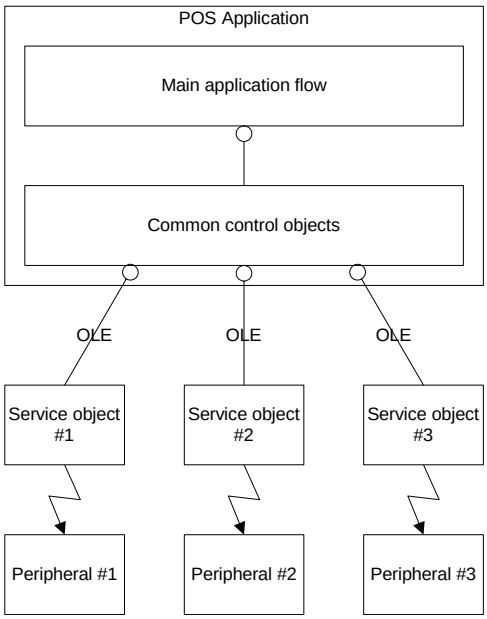


Figure 2 OLE Architectural Overview

Because the application interface to the common control object does not change, when a new peripheral is added or a peripheral is changed, the application does not have to change (assuming the application supports the device type to begin with). To add a new device, simply replace the service object and the device. The application will not know the difference.

1.2 Installation

Your OPOS Integration Kit contains all of the materials necessary for writing a Windows-based application for an iSeries (i3070/i6510/i6550/i6770/i6780) terminal.

1.2.1 Contents

There are two options available for OPOS users. There is the complete OPOS Integration Kit, and the driver-only package.

The **OPOS Integration Kit** contains the following:

- **IngenicoDeviceApplication** folder, which contains the UPOS Interface Application to be installed on the terminal.
- **IngenicoDocuments** folder, which contains:

  - This manual
  - User's Guides for the i6500, i6770, i6780
  - Installation & Operations Guides for the i3070, i6500, i6770, i6780
- **IngenicoUtilities** folder, which contains:
  - MLDT application for downloading files to a terminal
  - Form Designer application for designing terminal screens
- **OPOSTestDriveApplication** folder, which contains a sample Visual Basic project

The **OPOS Drivers** folder, which is available independent of the Integration Kit and is intended for customers desiring the bare minimum software to deploy OPOS on a system. This package can be installed silently and contains the following components, each of which can be installed independently:

- Windows Control Panel Configuration Utility
- Ingenico iSeries Service Object
- Ingenico implementation of the standard OPOS Controls
- Ingenico-specific controls, including Form and SigDisplay

For more information regarding silent installation and component selection, see the release notes in the OPOS Driver package.

1.2.2 Connecting and Powering the Terminal

For information on connecting and powering up your terminal, consult the *Installation and Operations Guide* included on your Integration Kit CD-ROM.

1.2.3 Software Installation

To install your OPOS Integration Kit using the attended installation:

1. Exit all open programs.
2. Open the **OPOS for the Ingenico iSeries.exe** file.
3. The Welcome window displays, which strongly recommends that you exit all programs before continuing. Click **Next**.
4. The License Agreement window displays. Read through the agreement and if you agree, click **Yes** to accept it.
5. The Choose Destination Location window displays. Click **Next** to accept the default directory, C:\Program Files\Ingenico\OPOS for the Ingenico iSeries.
6. The Setup Complete window displays. Click **Finish**.

1.2.4 OPOS Configuration

To establish communication parameters and system registry entries for the terminal, run the **OPOS - Ingenico iSeries Setup Program**:

1. Open **Start** > **Programs** > **Ingenico** > **OPOS for the Ingenico iSeries** > **Ingenico iSeries Setup**.

   The Ingenico iSeries OPOS Configuration window displays.

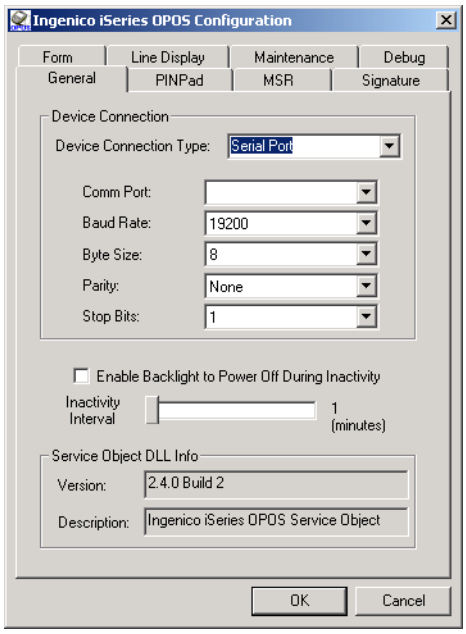


Figure 3 OPOS Configuration Window

2. On the General tab, In the **Device Connection Type** list box, select the appropriate option:

   - If using **TCP/IP Port**, specify an IP address and a port number for that IP address. The TCP/IP setting is typically used in conjunction with a "terminal server" device. The terminal server usually occupies one IP address and provides multiple RS-232 or RS-485 serial ports.

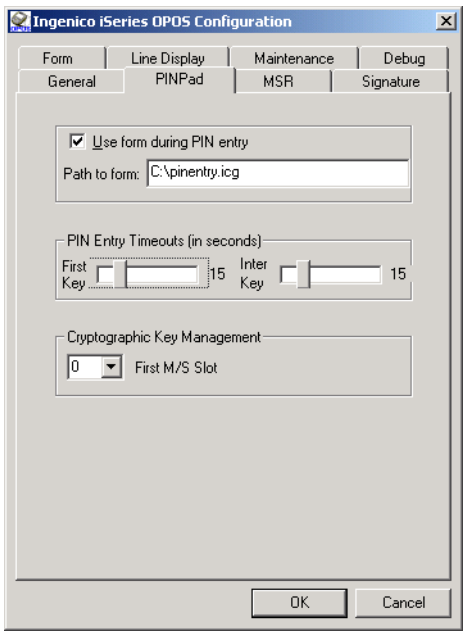
     Note: The i3070 PIN pad does not support TCP/IP connections.

   - If using **Serial Port**, fill in the fields as follows:
     - In the **Comm Port** box, select the appropriate communications port.
     - In the **Baud Rate** box, select the desired baud rate.
     - In the **Byte Size** box, select **8**.
     - In the **Parity** box, select **None**.
     - In the **Stop Bits** box, select **1**.
   - If using **USB**, select **USB Device** from the drop down list.

     Note: The i3070 PIN pad does not support USB connections.

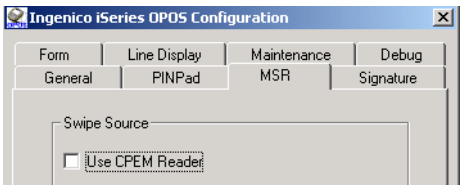
   - Select the **Enable Backlight to Power Off During Inactivity** check box if desired. Use the slider to select the amount of idle time to wait before powering off backlight (between 1 and 10 minutes).

3. Click the PINPad tab.



- Click the **Use Form During PIN Entry** check box to specify the form that you would like your iSeries terminal to display during a PIN entry session initiated by **EnablePINEntry**(). This form will be sent to the terminal when **ClaimDevice**() is called on the PINPad control. If you clear the check box, OPOS will delete any existing PIN entry forms from the terminal when **ClaimDevice**() is called.
- **PIN Entry Timeouts**: Use the sliders to specify the timeout interval, in seconds. The First Key slider is the timeout before the first key is pressed during PIN entry. The Inter Key slider is the timeout for any two consecutive key presses during PIN entry.
- Under Cryptographic Key Management, use the **First M/S Slot** list box to specify the minimum acceptable value for the *TransactionHost* parameter of the PINPad control's **BeginEFTTransaction**() method. By setting this parameter to 1, e^N-Touch 1000 customers can migrate to the iSeries platform without any changes to their *TransactionHost* parameter.

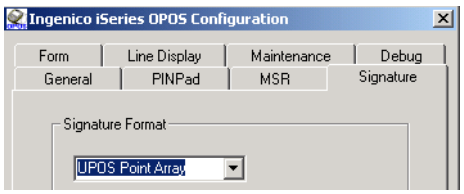
4. Click the MSR tab.



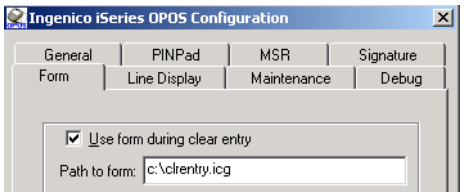
- Select the **Use CPEM Reader** check box if you have installed Ingenico's contactless payment expansion module on your terminal. This option allows you to receive swipe events from contactless payment cards. For more information on contactless payment card usage, see Chapter 4, *Special Features of the iSeries Terminals*. The contactless payment expansion module, available from your Ingenico representative, connects to your terminal's Aux port.

  Note: This does not apply to the i3070 PIN pad, since the CPEM is not available for this terminal.

5. Click the Signature tab.

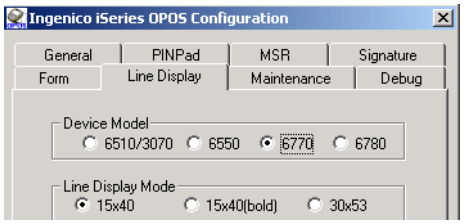


6. Click the Form tab.



- Select the **Use Form During Clear Entry** check box, and then specify the form you would like your iSeries terminal to display during a buffered entry session initiated by **QueryKeyPadBoardText**(). This form will be sent to the terminal when **Claim**() is called on the Form control. If you clear the check box, OPOS will delete any existing clear entry forms from the terminal when **Claim**() is called.

7. Click the LineDisplay tab.



- Select the appropriate **Device Model**: **i3070** (keypad, no signature capture), **i6510** (keypad, no signature capture), **i6550** (keypad, signature capture), **i6770** (signature capture with color and no keypad), or **i6780** (signature capture with color and keypad). This parameter is used to determine which screen modes are available for the line display.

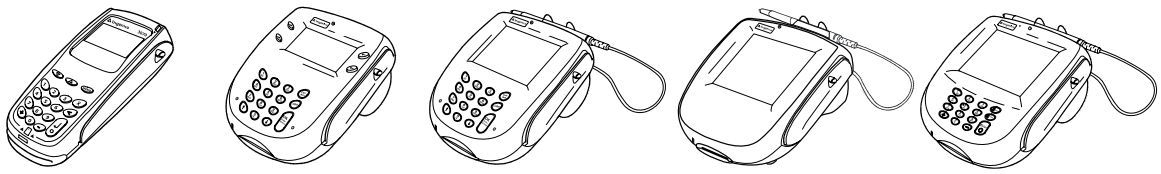
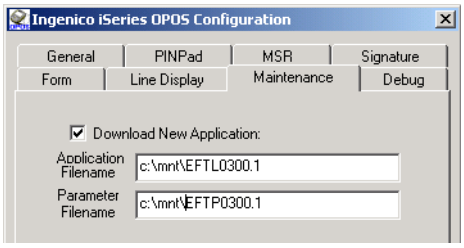


Figure 4 Device Models

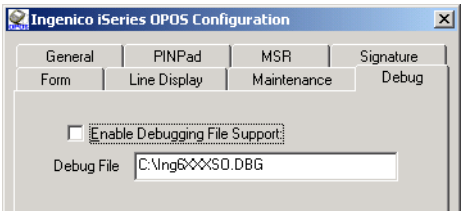
- Select the desired **Line Display Mode** to control the number of rows (first digit) and columns (second digit) that are available on the screen when the line display control is in use. For more information on screen modes, refer to the Unified POS Specification, available from http://www.nrf-arts.org/.

8. Click the Maintenance tab.



- To download a new application to the device, select the **Download New Application** check box, then specify the entire path for the application filename and/or parameter filename. At runtime, when the **Claim**() method is invoked, OPOS will determine whether the current application and/or associated parameters match the contents of the specified file(s). If they match, no download is performed. If they do not match, the IBM EFT protocol is used to download the new files to the device. Note that an IBM EFT application download can take one minute or longer. Therefore, be sure to set any **Claim**() timeouts accordingly.

9. Click the Debug tab.



- Select the **Enable Debugging File Support** check box to turn on debugging support for the iSeries terminal. In the **Debug File** box, specify the path for the debugging file. All OPOS errors will be logged to this file. To turn debugging support off, clear the check box.

1.3 Designing and Testing Forms

To design and test forms for the graphical display screens on the iSeries terminal, use the Form Designer application. For instructions on using the Form Designer application, access the program's online help.

1.4 Using the OPOS Controls

In order to use the ActiveX controls listed in the following tables, you must first bring them into your development environment. This can be done in either Visual Basic, Visual C++, or .NET environment.

- To insert an OPOS control into a Visual Basic project:
  - — Right-click the **VB Controls Toolbox**, select **Components**, and then select the **Controls** tab.
  - — Select the controls you would like to insert from the following table.
- To insert an OPOS control into a Visual C++ dialog:
  - — Right-click the dialog, and then select **Insert ActiveX Control**.
  - — Select the controls you would like to insert from the following table.
- To insert an OPOS Control into a Visual Studio .NET solution:

— Select **Tools** > **Add/Remove Toolbox Items** > **COM Components** tab.

— Select the controls you would like to insert (see following table). The selected controls become available in the Toolbox toolbar.

— Click on the toolbar to select the desired type.

— Click in your project's form to insert the control.

Table 1.1 OPOS Controls

Control	Description
OPOSLineDisplay	This OPOS control allows displaying and manipulation of text on the terminal's display. The size of the text displayed is controlled by the registry setting: HKEY_LOCAL_MACHINE\Software\OLEforRetail\ServiceOPOS\LineDisplay\Ing6XXX\DisplayMode Valid choices for the i3070: 4x16, 4x16BOLD, and 8x21. Valid choices for the i6510: 4x16, 4x16BOLD, and 8x21. Valid choices for the i6550: 10x30, 10x30BOLD, 20x40. Valid choices for the i6770: 15x40, 15x40BOLD, 30x53. Valid choices for the i6780: 14x40, 14x40BOLD, 29x53. Another way to set the line display is using the Ingenico iSeries OPOS Configuration window (open **Start > Programs > Ingenico > Ingenico iSeries Toolkit > OPOS Ingenico iSeries Setup**). ***Note:*** When using the OPOS controls and Visual C++, if the LineDisplay control is added to a project, errors will occur when the project is compiled. The LineDisplay Common Control Object defines CreateWindow() and DestroyWindow() methods that conflict with member functions defined in the control's base class (CWnd). To avoid these errors, edit the source files (.cpp and .h) created by adding the LineDisplay control and modify the names of the two methods. In the project, reference these LineDisplay methods with the modified names.
OPOSSigCap	(i6550, i6770, i6780 only) This is an OPOS control that allows an application to retrieve a signature from the terminal. This control is provided for users who wish to migrate from other OPOS SigCap applications to the terminal. New application development should use the OPOS form control, OPOSIVICMForm. The behavior of this control is highly dependent on the string parameter that is passed to the **BeginCapture**() method. • If the string is not NULL, an attempt is made to match this string with the name of a registry value located under the OPOS SignatureCapture hive. If the match fails, the control returns OPOS_E_NOEXIST. • If the string is NULL, a default form is provided during signature capture. If the match succeeds, the control uses the matching registry value's type to determine the next step. • If the value is of type REG_DWORD, this value is interpreted as the number of the form to display during signature capture. This form must have been previously stored by the Form control with a form number that matches this value. • If the value is of type REG_SZ, the value is interpreted as the path to a file which contains the desired form information. This form is sent to the device and the signature capture process is begun. • If the value is of type REG_BINARY, it is interpreted as raw binary form data to be sent directly to the device prior to capturing the signature.

Table 1.1 OPOS Controls (Continued)

Control	Description
ENFormSigDisplay	This control is used to render a signature on your PC or electronic cash register. This control will display an OPOS signature returned from the SigCap control or the form control, IVICMForm. This control will accept signatures in any of the three OPOS formats and either of the two Ingenico terminal native formats. To use the ENFormSigDisplay, call one of the following two methods to set the binary option: • ***SetOPOSBCNIBBLESignatureData(PointArray)*** – If the property *BinaryConversion* is set to OPOS_BC_NIBBLE, this function can be called with the data contained in the property *PointArray*. **THIS METHOD WILL ONLY WORK IF THE OPOS BINARY CONVERSION IS SET TO OPOS_BC_NIBBLE.** • ***SetSignatureData(RawDataVariant)*** – The data in the property *RawData* or *PointArray* must first be placed in a variant data structure. The variant containing the signature data is passed in to this function. The ENFormSigDisplay has other properties for Border, PenWidth, Background, etc. This ActiveX control is printable from containers that support printing.
OPOSIVICMForm (Ingenico Form OPOS Extension)	This is a form control used to store and display forms created with the Form Designer. For the i6550/i6770/i6780 touch screen terminal, this control provides button events to the application, provides survey responses, and is an exact superset of the OPOSSigCap control.
OPOSMSR	This is an OPOS control that allows an application to get magnetic stripe information from a credit or debit card via the magnetic stripe reader (MSR).
OPOSPINPad	This is an OPOS control that allows an application to manage keys, compute MAC values, get an encrypted PIN block from the secure PIN entry screen.

For instructions on how to use and format these controls, see the following chapters.

1.5 Setting Up the System Registry

OPOS relies on the system registry for proper operation. Each OPOS device has associated registry entries. The *OPOS Application Developer's Guide*, included in this software development kit, describes the necessary registry entries.

All OPOS registry settings are located under:

```
HKEY_LOCAL_MACHINE\SOFTWARE\OLEforRetail\ServiceOPOS
```

and are further subdivided by OPOS class type. The following is a listing of Ingenico's registry settings for each device.

All OPOS controls will have the following two entries that describe the associated device and service object version.

```
Description                 "Ingenico iSeries"
Version                     "Version versionNumber"
```

The SignatureCapture key contains configuration entries for the iSeries OPOS controls as a whole:

```
SignatureCapture\Ing6XXX            OPOS.SigCap.SO.Ing6XXX

    OverlaySigFormOnBackground      0x00000001
    Port                            "COM1"
    BaudRate                        0x00004b00
    ByteSize                        0x00000008
    Parity                          0x00000000
    StopBits                        0x00000000
    ConnectType                     "Serial Port"
    IPAddress                       "10.1.1.5"
    IPPort                          0x00001f41
    UseDebugFile                    0x00000001
    DebugFile                       "C:\\Ing6XXXSO Com1.DBG"
    NewAppFileName                  "C:\temp\eftl0600"
                                    (optional)
    NewParmFileName                 "C:\temp\eftp0600"
                                    (optional)

    IBM EFT Executable              "C:\Program Files\Ingenico\OPOS for
                                    the Ingenico 6XXX\ibmeftdl.exe"
                                    (optional)

IVICMForm\Ing6XXX                   OPOS.IVICMForm.SO.Ing6XXX

    ClearEntryForm                  "c:\clrentry.icg"
    UseClearEntryForm               0x00000000

MSR\Ing6XXX                         OPOS.MSR.SO.Ing6XXX

Enable CPEM Reader                  0x00000000

PINPad\Ing6XXX                      OPOS.PINPad.SO.Ing6XXX
```

```
    First Key Timeout           0x0000000F
    Inter Key Timeout           0x0000000F
    FormFile                    "c:\pinentry.icg"
    UseFormFile                 0x00000000

Base Slot Number                0x00000000

LineDisplay\Ing6XXX             OPOS.LineDisplay.SO.Ing6XXX

    DeviceWindows               “10”
    DeviceModel                 "6550"
    DisplayMode                 “10x30”
```

The detail of the **SignatureCapture** entry in the System Registry holds all the connection detail for these terminals. The registry entries for each device are modified through the control panel applet for the terminal. These settings are accessed when you call:

- SigCap.Open(“Ing6XXX”)
- MSR.Open(“Ing6XXX”)
- PINPad.Open(“Ing6XXX”)
- LineDisplay.Open(“Ing6XXX”)
- IVICMForm.Open(“Ing6XXX”)

To access multiple devices from a single host, simply duplicate the above registry entries with a new device name. For example, a hotel may have registry entries for Ing6XXX-CheckIn and Ing6XXX-CheckOut. The hotel’s registry entries would be as follows:

SignatureCapture\Ing6XXX-CheckIn
IVICMForm\Ing6XXX-CheckIn
MSR\Ing6XXXIng6XXX-CheckIn
PINPad\Ing6XXX-CheckIn
LineDisplay\Ing6XXX-CheckIn

SignatureCapture\Ing6XXX-CheckOut
IVICMForm\Ing6XXX-CheckOut
MSR\Ing6XXX-CheckOut
PINPad\Ing6XXX-CheckOut
LineDisplay\Ing6XXX-CheckOut

With these registry settings, you can now uniquely access multiple devices from one host device. To do this, specify the new name in the open member function (SigCap.Open(“Ing6XXX-CheckIn”) or MSR.Open(“Ing6XXX.Check-Out”).

CHAPTER 2

Form Control

2.1 General Information

Ingenico's form control, IVICMForm.ocx, is not a standard OPOS control, but rather an extension to the OPOS specifications that was written to allow form-based display and control on the iSeries terminal. The Ingenico form control's OLE programmatic ID is **OPOS.IVICMForm**. With respect to common properties and methods, Ingenico form control follows the version 1.3 specifications for OPOS controls exactly.

The registry entries for the Ingenico form control are located at:

```
HKEY_LOCAL_MACHINE\Software\OLEforRetail\ServiceOPOS\IVICMForm\Ing6XXX
```

2.1.1 Model

The general model of Ingenico form control:

- Consists of a form download and display mechanism for rendering and controlling forms on the iSeries terminal.
- Is directly tied to the Form Designer, a PC-based form generation tool with a graphical user interface that is included on the OPOS Software Development Kit CD-ROM. This tool allows you to generate forms and save them on the disk drive for download and future editing.
- Consists of a subset of OPOS Line Display primitives for the display of dynamic text on top of forms. This functionality is necessary because the OPOS Line Display does not allow multiple fonts during display.

2.1.2 Device Sharing

The Ingenico form control is an exclusive-use device. Its device sharing rules are:

- The application must claim the device before enabling it.
- The application must claim and enable the device before accessing properties or calling methods that update the device.
- For precise usage prerequisites, see section 2.2 *Summary* on page 2-3.

2.1.3 Form Types

Developers can create non-data entry forms for the iSeries terminal, such as select payment type screens, signature capture screens, line item displays, and advertising screens.

The Form Designer automatically saves a form generated for iSeries terminals as an Ingenico Custom Graphic (.icg or “text” file). The bitmap on the form must be saved separately using the .bmp or .gif extension.

2.1.4 Specific Events

DataEvents are fired under the following circumstances:

- When a user activates a button on the currently displayed form.
- When signature data is ready for retrieval.
- When you call **QuerySignatureBoxData** and signature data is successfully retrieved.

2.1.5 File System Maintenance

In order to prevent iSeries terminals from becoming fragmented and running out of file memory, it is important to delete all existing forms before storing new or updated forms on the device. There are currently two methods to achieve this. The first way is to send the device a five-byte message via the DirectIO command DIO_SEND_RAW_DATA, as follows:

```
\x05\x05\x93\xFF\x6C
```

and wait for a successful response.

To ensure these bytes are transmitted correctly to the OPOS Control, be sure to set the control's BinaryConversion property to either OPOS_BC_NIBBLE or OPOS_BC_DECIMAL.

For more information regarding the BinaryConversion property, consult the OPOS Application Developer's Guide.

The second, and preferred, approach, is to have OPOS send this message on your behalf using the DirectIO command DIO_DELETE_ALL_FORMS. For information regarding DirectIO Commands, see section 4.2 *Direct I/O Usage* on page 4.5.

2.2 Summary

This section contains precise usage prerequisites for each property and method.

2.2.1 Properties

Table 2.1 Common Properties

Name	Type Access	Initialized After
AutoDisable	Boolean R/W	Open
BinaryConversion	Long R/W	Open
CheckHealthText	String R	Open
Claim	Boolean R	Open
DataCount	Long R	Open
DataEventEnabled	Boolean R/W	Open
DeviceEnabled	Boolean R/W	Open, Claim
FreezeEvents	Boolean R/W	Open
OutputID	Long R	Open, Claim
ResultCode	Long R	--
ResultCodeExtended	Long R	Open
State	Long R	--
ControlObjectDescription	String R	--
ControlObjectVersion	Long R	--
ServiceObjectDescription	String R	Open
ServiceObjectVersion	Long R	Open
DeviceDescription	String R	Open
DeviceName	String R	Open

Table 2.2 Specific Properties

Name	Type Access	Initialized After
CharacterSet	Long R/W	Open, Claim, Enable
CharacterSetList	String R	Open
FontHeight	Long R	Open
FontWidth	Long R	Open
FontStyle	Long R/W	Open, Claim, Enable
FontTypefaceList	String R	Open
FontTypeface	Long R/W	Open, Claim, Enable
DeviceRows	Long R	Open
DeviceColumns	Long R	Open
MaximumX	Long R	Open
MaximumY	Long R	Open
RawData	String R	Open, Claim, Enable
TotalPoints	Long R	Open, Claim, Enable
PointArray	String R	Open, Claim, Enable
KeyPadBoardPrompt1	String R/W	Open, Claim, Enable
KeyPadBoardText	String R	Open, Claim, Enable

2.2.2 Methods

Table 2.3 Common Methods

Name	May Use After
Open	--
Close	Open
Claim	Open
Release	Open, Claim
CheckHealth	Open, Claim, Enable
ClearInput	Open, Claim, Enable
ClearOutput	Open, Claim, Enable
DirectIO	Open

Table 2.4 Specific Methods

Name	May Use After
StoreFormOnDevice	Open, Claim, Enable
DisplayFormOnDevice	Open, Claim, Enable
DisplayTextAt	Open, Claim, Enable
DisplayText	Open, Claim, Enable
QuerySignatureBoxData	Open, Claim, Enable
QueryKeyPadBoardText	Open, Claim, Enable
QueryRadioButtonState	Open, Claim, Enable

Note: Those migrating from the e^{N}-Touch 1000 may notice that **QueryScriptBoxData** is not applicable to iSeries terminals. This is because it does not use script (or *initial*) boxes.

2.2.3 Events

Table 2.5 Events

Name	May Occur After
DataEvent	Open, Claim, Enable
DirectIOEvent	Open
ErrorEvent	Open
StatusUpdateEvent	Open, Claim, Enable

2.3 Properties

2.3.1 CharacterSet Property R/W

Syntax **LONG CharacterSet;**

Remarks **CharacterSet** contains the character set for displaying characters.

Possible ranges or values:

Value	Definition
Range 101 - 199	A device-specific character set that does not match a code page, nor the ASCII or Windows ANSI character sets.
Range 400 - 990	Windows Code page; matches one of the standard values.
FORM_CS_ASCII	The ASCII character set, supporting the ASCII characters between 20-hex and 7F-hex. The value of this constant is 998.
FORM_CS_WINDOWS	The Windows ANSI character set. The value of this constant is 999. This is exactly equivalent to the Windows code page 1252.
Range 1000 and higher	Windows code page; matches one of the standard values.

This property is initialized to an appropriate value when the device is first enabled following the **Open** method.

Return When this property is set, one of the following values is placed in the **ResultCode** property:

Value	Meaning
OPOS_SUCCESS	The property was set successfully.
Other Values	See 2.3.14 *ResultCode Property R*.

See Also 2.3.2 *CharacterSetList Property R*

2.3.2 CharacterSetList Property R

Syntax **BSTR CharacterSetList**;

Remarks **CharacterSetList** is a string of character set numbers.

This property is initialized by the **Open** method. The string consists of ASCII numeric set numbers separated by commas.

For example, if the string is 101,850,999, then the device supports a device-specific character set, code page 850, and the Windows ANSI character set.

See Also 2.3.1 *CharacterSet Property R/W*

2.3.3 FontHeight Property R

Syntax **LONG FontHeight**;

Remarks **FontHeight** returns the height of the currently selected font, in pixels.

This property is initialized by the **Open** method.

See Also 2.3.4 *FontWidth Property R*
2.3.5 *FontStyle Property R/W*

2.3.4 FontWidth Property R

Syntax **LONG FontWidth**;

Remarks **FontWidth** returns the width of the currently selected font, in pixels.

This property is initialized by the **Open** method.

See Also 2.3.3 *FontHeight Property R*
2.3.5 *FontStyle Property R/W*

2.3.5 FontStyle Property R/W

Syntax **LONG FontStyle**;

Remarks **FontStyle** sets/returns the style of the currently selected font.

This property is initialized by the **Open** method. The style can be any combination (bitwise OR) of the following: underline and reverse. The numeric values are:

- Underline is 0x02
- Reverse is 0x04

See Also 2.3.3 *FontHeight Property R*
2.3.4 *FontWidth Property R*

2.3.6 FontTypefaceList Property R

Syntax **BSTR FontTypefaceList**;

Remarks **FontTypefaceList** specifies the typefaces (e.g., bold) and/or fonts (e.g., Arial) that are supported by the device.

This property is initialized by the **Open** method. The string consists of font or typeface names separated by commas.

8x16, 8x16BOLD, 6x8 are the default typefaces supported by the device.

See Also 2.3.8 *DeviceRows Property R*

2.3.7 FontTypeface Property R/W

Syntax **LONG FontTypeface;**

Remarks **FontTypeface** is an index into the comma-separated list of typefaces allowed by this device.

With 8x16, 8x16BOLD, 6x8 as the FontTypefaceList, values of the identifiers are set according to the following table.

FontTypeface Identifier	Associated Font
0	8x16
1	8x16 BOLD
2	6x8

This property is initialized by the **Open** method.

2.3.8 DeviceRows Property R

Syntax **LONG DeviceRows;**

Remarks **DeviceRows** contains the number of rows on the terminal's display screen. This value is dependent on the current **FontTypeface** and the dimensions of the pixel display, which are:

- 128 x 64 for the i3070 terminal
- 128 x 64 for the i6510 terminal
- 240 x 160 for the i6550 terminal
- 320 x 240 for the i6770 terminal
- 320 x 234 for the i6780 terminal

This property is initialized by the **Open** method.

2.3.9 DeviceColumns Property R

Syntax **LONG DeviceColumns;**

Remarks **DeviceColumns** contains the number of columns on the terminal's display screen. This value is dependent on the current FontTypeface and the dimensions of the pixel display, which are:

- 128 x 64 for the i3070 terminal
- 128 x 64 for the i6510 terminal
- 240 x 160 for the i6550 terminal
- 320 x 240 for the i6770 terminal
- 320 x 234 for the i6780 terminal

This property is initialized by the **Open** method.

2.3.10 MaximumX Property R

Syntax **LONG MaximumX;**

Remarks **MaximumX** contains the maximum horizontal coordinate of the terminal's display screen.

This property is initialized by the **Open** method.

2.3.11 MaximumY Property R

Syntax **LONG MaximumY;**

Remarks **MaximumY** contains the maximum vertical coordinate of the terminal's display screen.

This property is initialized by the **Open** method.

2.3.12 PointArray Property R

Syntax **BSTR PointArray;**

Remarks **PointArray** contains the signature captured from the device. It consists of an array of xy-coordinate points with the number of array entries specified in **TotalPoints**. Each point is represented by four characters: *x* (low 8 bits), *x* (high 8 bits), *y* (low 8 bits), *y* (high 8 bits). The format of this data depends upon the value of the **BinaryConversion** property (for more information, refer to the Unified POS Specification, available from http://www.nrf-arts.org/).

A special point value is (0xFFFF, 0xFFFF) which indicates the end of a line (that is, a pen lift). Almost all signatures are comprised of more than one line.

- If the **RealTimeDataEnabled** property is FALSE, then **PointArray** contains the entire captured signature.
- If the **RealTimeDataEnabled** property is TRUE, then **PointArray** contains at least one point of the signature. The actual number of points delivered at one time is implementation dependent. The points from multiple **DataEvents** are logically concatenated to form the entire signature, such that the last point from a **DataEvent** is followed immediately by the first point of the next **DataEvent**.

The point representation definition is the same regardless of whether the signature is presented as a single **PointArray**, or as a series of real time **PointArray**s.

Reconstruction of the signature using the points is accomplished by beginning a line from the first point in the signature to the second point, then to the third, and so on. When an end-of-line point is encountered, the drawing of the line ends, and the next line is drawn beginning with the next point. An end-of-line point is assumed (but need not be present in **PointArray**) at the end of the signature.

This property can be set by:

- The control just before delivering the **DataEvent**
- The **EndCapture** method

See Also 2.3.13 *RawData Property R*

2.3.13 RawData Property R

Syntax **BSTR RawData;**

Remarks **RawData** contains the signature captured from an iSeries terminal in a device-specific format. The format of this data depends upon the value of the **BinaryConversion** property (for more information, refer to the Unified POS Specification, available from http://www.nrf-arts.org/).

This data is often in a compressed form to minimize signature storage requirements. Reconstruction of the signature from this data requires device-specific processing.

This property can be set by:

- The control just before delivering the **DataEvent**
- The **EndCapture** method

See Also 2.3.12 *PointArray Property R*
2.3.15 *TotalPoints Property R*

2.3.14 ResultCode Property R

Syntax **LONG ResultCode;**

Remarks Each method and writable property sets its own **ResultCode**. **ResultCode** is always readable.

Before the **Open** method is called, it returns the value OPOS_E_CLOSED.

It is conceivable that more than one of the result codes in the following list could be validated for a particular failure. The order of error reporting precedence for such scenarios is the following:

- OPOS_E_CLAIMED
- OPOS_E_NOTCLAIMED
- OPOS_E_DISABLED

The result code values are:

Value	Meaning
OPOS_SUCCESS	Successful operation.
OPOS_E_CLOSED	Attempt was made to access a closed device.

OPOS_E_CLAIMED	Attempt was made to access a device that is claimed by another process. The other process must release the device before this access may be made. For exclusive-use devices, the application will also need to claim the device before the access is legal.
OPOS_E_NOTCLAIMED	Attempt was made to access an exclusive-use device that must be claimed before the method or property set action can be used. If the device is already claimed by another process, then the status OPOS_E_CLAIMED is returned instead.
OPOS_E_NOSERVICE	The control cannot communicate with the Service Object. Most likely, a setup or configuration error must be corrected.
OPOS_E_DISABLED	Cannot perform operation while device is disabled.

2.3.15 TotalPoints Property R

Syntax **LONG TotalPoints;**

Remarks Contains the number of signature points in **PointArray**.

If **RealTimeDataEnabled** is TRUE, then **TotalPoints** is set to zero to indicate that all of the partial signatures have been provided to the application by the control.

This property is set by the control just before delivering the **DataEvent** or by the **EndCapture** method. It includes the line drawing terminators (see 2.3.12 *PointArray Property R*).

2.3.16 KeyPadBoardPrompt1 Property R/W

Syntax **BSTR KeyPadBoardPrompt1;**

Remarks **KeyPadBoardPrompt** contains the onscreen prompt that is displayed when the user begins a clear entry session. A clear entry session is initiated by a call to **QueryKeyPadBoardText**().

Note: **KeyPadBoardPrompt2** exists in the application, but cannot be used by an iSeries terminal because it only permits one string when specifying a prompt for PIN entry.

2.3.17 KeyPadBoardText Property R

Syntax **BSTR KeyPadBoardText;**

Remarks **KeyPadBoardText** contains the text entered by the user during a clear entry session. The form control will fire a **DataEvent** with a status of zero (0) for successfully entered text, or a status of negative one (-1) for a user cancellation.

2.4 Methods

2.4.1 StoreFormOnDevice Method

Syntax **LONG StoreFormOnDevice**(**LONG** *formNumber*, **LPCTSTR** *formFile*)

Parameter	Description
formNumber	A numeric identifier for referencing this form in the future with **DisplayForm**(), and for identifying this form during **DataEvents**. *formNumber* must be a numeric identifier in the range of 1 to 255. A value of zero will use the form number defined in the *formFile*.
formFile	A relative path that includes the form file name. Form files for use with the Ingenico form control are generated by the Form Designer.

Remarks **StoreFormOnDevice** writes a form and its corresponding background image to the device's FLASH memory for later instant recall. It is important to note that a form may be as simple as a bitmap advertisement screen, or as complex as a bitmap with corresponding sensitive areas that generate a button event upon user activation. A form may also contain radio buttons and signature boxes.

WARNING

In order to prevent the iSeries terminal from running out of file memory due to fragmentation, it is important to delete all existing forms before storing new or updated forms on the device. Be careful to only store a form if an update is necessary. For more details, see 2.1.5 *File System Maintenance* on page 2-2.

Return One of the following values is returned by the method and placed in the **ResultCode** property:

Value	Meaning
OPOS_SUCCESS	The method was successful.
OPOS_E_FAILURE	The method was not successful
Other Values	See 2.3.14 *ResultCode Property R*.

See Also 2.4.2 *DisplayFormOnDevice Method*

2.4.2 DisplayFormOnDevice Method

Syntax **LONG DisplayFormOnDevice(LONG** *formNumber*, **LPCTSTR** *formFile***)**

Parameter	Description
formNumber	A numeric identifier for referencing this form in the future with **DisplayForm**(), and for identifying this form during **DataEvents**. *formNumber* must be a numeric identifier in the range of 1 to 255. A value of zero will use the form number defined in the *formFile*.
formFile	A relative (to the application) path (plus extension) to a form file. Form files for use with the Ingenico form control are generated by the Form Designer.

Remarks The **OPOSIVICMForm** control fires a **DataEvent** when one of the action buttons on the form has been activated by the user. Other types of buttons, such as radio buttons and check boxes, do not result in a **DataEvent** being fired. The **Status** member of the **DataEvent** will contain the button's identifier. Because forms may contain radio buttons, check boxes, and signature boxes, it is the application's responsibility to query the status of the individual form members. This should be done after a **DataEvent** has been fired.

The form must be stored on the device and the correct form number must be specified. If a form is currently displayed on the device, subsequent behavior is subject to whether the new form is transparent. If it is transparent, the screen is not cleared and the new form is overlaid on top of the original form. This is often used with graphics. If it is not transparent, the screen is cleared and the new form takes its place. For more information regarding form transparency, see the Form Designer application’s online help.

If a form does not contain any buttons, the behavior of the terminal's physical keys are not affected. However, if a form contains one or more buttons, then the terminal's physical keys are automatically enabled, subject to the current keymask setting. Furthermore, **DataEvents** are enqueued for physical buttons just as they are for buttons on a form. For example, if the physical **Cancel** key is pressed and that key is not currently masked, a **DataEvent** will be enqueued, even if the form does not contain a button with an ID corresponding to **Cancel**. If in fact the form does contain a **Cancel** button, there is no way to distinguish whether the physical or virtual button was pressed.

Return One of the following values is returned by the method and placed in the **ResultCode** property:

Value	Meaning
OPOS_SUCCESS	The method was successful.
OPOS_E_FAILURE	The method was not successful.
Other Values	See 2.3.14 *ResultCode Property R*.

See Also 2.4.1 *StoreFormOnDevice Method*
2.4.5 *QuerySignatureBoxData Method*

2.4.6 *QueryKeyPadBoardText Method*

2.4.7 *QueryRadioButtonState Method*

2.5.1 *DataEvent*

4.2.8 *Configure Key Masks Parameter*

2.4.3 DisplayTextAt Method

Syntax **LONG DisplayTextAt (LONG** *Row*, **LONG** *Column*, **BSTR** *Data*, **LONG** *Attribute***)**

Parameter	**Description**
Row	The start row for the text.
Column	The start column for the text.
Data	The string of characters to display. The format of this data depends upon the value of the **BinaryConversion** property (for more information, refer to the Unified POS Specification, available from http://www.nrf-arts.org/).
Attribute	This parameter is included for compatibility with OPOS LineDisplay. This parameter is ignored.

Remarks The characters in *Data* are processed beginning at the screen/form location specified by the *Row* and *Column* parameters, and continuing in succeeding columns. When the limit of the screen has been reached, characters are continued on the next row beginning at Column 0. Special characters are allowed in *Data*, *CarriageReturn* (0x0D) returns the current *Column* to 0, and *LineFeed* (0x0A) increments the current *Row* and returns *Column* to 0.

Screen coordinates use the format (*Row*,*Column*) and start at (0,0) in the upper left hand corner and extend to the lower right hand corner (**DeviceRows**-1, **DeviceColumns**-1).

Return One of the following values is returned by the method and placed in the **ResultCode** property:

Value	**Meaning**
OPOS_SUCCESS	The method was successful.
OPOS_E_ILLEGAL	*Row* or *Column* are out of range
Other Values	See *2.3.14 ResultCode Property R*

See Also 2.3.8 *DeviceRows Property R*
2.3.9 *DeviceColumns Property R*
2.3.7 *FontTypeface Property R/W*

2.4.4 DisplayText Method

Syntax **LONG DisplayText (BSTR** *Data*, **LONG** *Attribute***)**

Parameter	Description
Data	The string of characters to display. The format of this data depends upon the value of the **BinaryConversion** property (for more information, refer to the Unified POS Specification, available from http://www.nrf-arts.org/).
Attribute	This parameter is included for compatibility with OPOS LineDisplay. This parameter is ignored.

Remarks This method is used to send text to a scrolling receipt element on a form. For more information on how to configure the behavior and appearance of scrolling receipts and other form elements, see the online help for the Ingenico Form Designer application.

The characters in *Data* are displayed at the current cursor position. The cursor's position starts at the origin, and advances as text is printed. The location of the origin depends on whether a scrolling receipt element in onscreen when **DisplayText** is called.

- If a scrolling receipt element is onscreen when **DisplayText** is called, the origin is located at the upper left corner of the *scrolling receipt element*.
- If there is no scrolling receipt element when **DisplayText** is called, the origin is located at the upper left corner of the *screen*.

Using the format (*Row*, *Column*), screen coordinates start at the origin (0, 0) and extend to the lower right corner (**DeviceRows**-1, **DeviceColumns**-1). The appearance of the text is dictated by the **FontStyle** and **FontTypeface** properties.

Special characters are allowed in *Data*:

- *CarriageReturn* (0x0D) returns the cursor to the leftmost column of the screen or scrolling receipt.
- *LineFeed* (0x0A) moves the cursor down one row before moving it to the leftmost column.

If the cursor reaches the bottom of the screen or the bottom of the scrolling receipt element, the text will automatically scroll upwards, and the first line of text will be lost.

Return One of the following values is returned by the method and placed in the **ResultCode** property:

Value	Meaning
OPOS_SUCCESS	The method was successful.
Other Values	See 2.3.14 *ResultCode Property R*.

See Also 2.3.8 *DeviceRows Property R*
2.3.9 *DeviceColumns Property R*
2.3.5 *FontStyle Property R/W*
2.3.7 *FontTypeface Property R/W*

2.4.5 QuerySignatureBoxData Method

Syntax **LONG QuerySignatureBoxData()**

Parameter	Description
none	This method call requires no parameters.

Remarks **QuerySignatureBoxData** may be called to terminate the displayed form or after a control button **DataEvent**. Once a **DataEvent** has been fired to the application, the stylus is disabled, and the form is no longer available for user modification. At this point, the status of form objects may be queried.

If **QuerySignatureBoxData** returns OPOS_SUCCESS, then the application may reliably read **RawData**, **TotalPoints**, and **PointArray** for the signature box contained on the form. **RawData**, **TotalPoints**, and **PointArray** will contain no data if the user has not signed in this signature box.

Return One of the following values is returned by the method and placed in the **ResultCode** property:

Value	Meaning
OPOS_SUCCESS	The method was successful.
OPOS_E_FAILURE	No data is available for the signature box.
Other Values	See 2.3.14 *ResultCode Property R.*

See Also 2.4.2 *DisplayFormOnDevice Method*
2.5.1 *DataEvent*

2.4.6 QueryKeyPadBoardText Method

Syntax **LONG QueryKeyPadBoardText ()**

This method call requires no parameters.

Remarks This method behaves differently for the i6510, i6550, and i6780, which have physical keypads, and the i6770, which does not.

- **i6780/i6550/i6510/i3070**: The **QueryKeyPadBoardText** method initiates a clear entry session, in which the user may enter unencrypted numeric data using the physical PIN pad. Once the user has submitted this data, the Form Control shall fire a **DataEvent** with a status of zero (0) for successfully entered text, or a status of negative one (-1) for a user cancellation. The resultant numeric data is stored in the **KeyPadBoardText** property.
- **i6770**: Since the i6770 device does not possess physical keys, a call to **QueryKeyPadBoardText** must be preceded by a call to display either a numeric keypad or an alphanumeric keyboard on the device. This is achieved via the **DisplayFormOnDevice** method, which is passed a number corresponding to a form containing the desired key-entry element. Once the user has submitted this data, the Form Control shall fire a **DataEvent** with a status of zero (0) for successfully entered text, or a status of negative one (-1) for a user cancellation. The resultant numeric or alphanumeric data is stored in the **KeyPadBoardText** property.

  The look and feel of the virtual keypad and keyboard is not subject to configuration. For more information about these and other form elements, refer to the online help in the Ingenico Form Designer application.

Return — One of the following values is returned by the method and placed in the **ResultCode** property:

Value	Meaning
OPOS_SUCCESS	The clear entry session was started successfully.
OPOS_E_ILLEGAL	There was an error during initiation of the clear entry session.
Other Values	See 2.3.14 *ResultCode Property R.*

See Also — 2.4.2 *DisplayFormOnDevice Method*
2.5.1 *DataEvent*

2.4.7 QueryRadioButtonState Method

Syntax — **LONG QueryRadioButtonState(**LONG *groupID,* **LONG** *controlID,* **LONG *** *controlState***)**

Parameter	Description
groupID	The group identifier for the button whose state is being sought. If this value is nonzero, the button is assumed to be a radio button. Otherwise, the button is assumed to be a check box.
controlID	An identifier that uniquely identifies the button whose state is being sought.
controlState	A pointer to a **LONG** which will hold the returned state of the button. A value of one (1) indicates the button is in the pressed state. A value of zero (0) indicates the button is not currently pressed.

Remarks — Applications should call **QueryRadioButtonState** after they have received a **DataEvent** indicating a successful button press. A call to **QueryRadioButtonState** causes the stylus to be disabled so that the form is no longer subject to user modification. A separate call to **QueryRadioButtonState** is needed for each check box on a form whose state is to be determined. On the other hand, the state of all radio buttons in a group can be inferred once the identity of the pressed button in the group is known.

Customers migrating from the e^{N}-Touch 1000 will notice a few minor changes with respect to radio button and check box behavior:

- All button types on the e^{N}-Touch 1000 were allocated independent sets of IDs; for example, a radio button could have the same ID as a checkbox or a regular button. For Ingenico iSeries devices, all buttons, be they check boxes, radio buttons, or regular buttons, must have a unique ID. The Ingenico Form Designer application automatically enforces these differences.
- On the e^{N}-Touch 1000, receipt of a DataEvent caused the stylus to be disabled. For i6500/i6700 devices, the stylus is not disabled until **QueryRadioButtonState** is called.

Radio buttons and check boxes are best used on a form in conjunction with regular buttons. For more information on designing forms with radio buttons and checkboxes, see the online help for the Ingenico Form Designer application.

Return — One of the following values is returned by the method and placed in the **ResultCode** property:

Value	Meaning
OPOS_SUCCESS	The method was successful.
OPOS_E_ILLEGAL	The device has not yet received a control button event.
Other Values	See 2.3.14 *ResultCode Property R*.

See Also
2.4.2 *DisplayFormOnDevice Method*
2.5.1 *DataEvent*

2.5 Events

2.5.1 DataEvent

Syntax **void DataEvent (LONG** *Status***);**

Remarks Fired to signal input data from the device to the application.

This event is fired in the following situations:

- When the user presses any one of a form's buttons
- When the user presses a physical button while a form with buttons is displayed
- When signature data is available

A value of zero in the status parameter indicates signature box data. Otherwise the *Status* parameter contains the button ID of the control button pressed on the currently active form.

See Also
2.4.5 *QuerySignatureBoxData Method*
2.4.6 *QueryKeyPadBoardText Method*
2.4.7 *QueryRadioButtonState Method*

2.5.2 ErrorEvent

Syntax **void ErrorEvent (LONG** *ResultCode*, **LONG** *ResultCodeExtended*, **LONG** *ErrorLocus***, LONG*** *pErrorResponse***);**

Parameter	Description
ResultCode	Result code causing the error event. For values, see section 2.3.14 *ResultCode Property R*.
ResultCodeExtended	Extended result code causing the error event. For values, see **ResultCodeExtended**.
ErrorLocus	Location of the error. See values below.
pErrorResponse	Pointer to the error event response. See values below.

The *ErrorLocus* parameter may be one of the following:

Value	Meaning
OPOS_EL_INPUT	Error occurred while gathering or processing event-driven input. No input data is available.
OPOS_EL_INPUT_DATA	Error occurred while gathering or processing event-driven input, and some previously buffered data is available. (Very unlikely – see **Remarks**.)

Remarks — Fired when an error is detected while trying to read signature box data. Input error events are not delivered until the **DataEventEnabled** property is **true**, to ensure proper application sequencing occurs.

See Also — Refer to the Unified POS Specification, available from http://www.nrf-arts.org/.

CHAPTER 3

SigDisplay Control

3.1 General Information

Ingenico wrote its own form control, **OPOSSigDisplay.ocx**, as an extension to the OPOS specifications in order to display or save signature capture data from the signature capture terminals: i6550, i6770, and i6780. With respect to common properties and methods, **OPOSSigDisplay** control follows the version 1.3 specifications for OPOS controls exactly.

3.2 Summary

This section contains precise usage prerequisites for each property.

3.2.1 Properties

Table 3.1 Specific Properties

Name	Type Access	Initialized Before
GetDrawBorder	Boolean R	Methods
SetDrawBorder	Boolean W	Methods
GetDrawBackground	Boolean R	Methods
SetDrawBackground	Boolean W	Methods
GetPenWidth[Printer]	Long R	Methods, Open
SetPenWidth[Printer]	Long W	Methods, Open
GetDisplayNumPoints	Boolean R	Methods
SetDisplayNumPoints	Boolean W	Methods
GetNumPointsInDisplay	Long R	Methods
SetNumPointsInDisplay	Long W	Methods

3.2.2 Methods

Table 3.2 Specific Methods

Name	May Use After
SetOPOSBCNIBBLESignatureData	Acquiring signature data as LPCSTR
SetOPOSBCNIBBLESignatureDataX	Acquiring signature data as LPCSTR
SetSignatureData	Creating VARIANT with signature data
SetSignatureDataX	Creating VARIANT with signature data
GetSignatureType	Methods
GetSignatureTypeString	Methods
WriteSignatureToFile	Methods
ConvertSignatureToImageBuffer	Methods
EnableLiveCapture	Methods
StartLiveCapture	Methods
SetDeviceResolution	Methods

3.3 Properties

3.3.1 Get DrawBorder

Syntax **BOOL GetDrawBorder;**

Remarks **GetDrawBorder** toggles the drawing of the border around the signature display window.

If **GetDrawBorder** is True (default value), then a border displays. If set to False, no border displays.

3.3.2 Set DrawBorder

Syntax **BOOL SetDrawBorder;**

Remarks **GetDrawBorder** toggles the drawing of the border around the signature display window.

If **GetDrawBorder** is True (default value), then a border displays. If set to False, no border displays.

3.3.3 Get DrawBackground

Syntax **BOOL GetDrawBackground;**

Remarks **GetDrawBackground** toggles the drawing (FillRect(...)) of the background before rendering the signature into the display window.

If **GetDrawBackground** is True (default value), the background drawing displays before the signature is rendered in the display window. If set to False, the signature is rendered first before the background drawing.

3.3.4 Set DrawBackground

Syntax **BOOL SetDrawBackground;**

Remarks **SetDrawBackground** toggles the drawing (FillRect(...)) of the background before rendering the signature into the display window.

If **SetDrawBackground** is True (default value), the background drawing displays before the signature is rendered in the display window. If set to False, the signature is rendered first before the background drawing.

3.3.5 Get PenWidth

Syntax **LONG GetPenWidth[Printer];**

Remarks **GetPenWidth** adjusts the thickness of the pen used to render the signature on the screen or printer device context.

- Default pen width on screen: 1
- Default pen width on printer: 2

3.3.6 Set PenWidth

Syntax **LONG SetPenWidth[Printer];**

Remarks **SetPenWidth** adjusts the thickness of the pen used to render the signature on the screen or printer device context.

- Default pen width on screen: 1
- Default pen width on printer: 2

3.3.7 Get DisplayNumPoints

Syntax **BOOL GetDisplayNumPoints;**

Remarks **GetDisplayNumPoints** toggles the textual display of the number of points contained in the signature. The default value is Off. This display is implemented in the display control of the signature window as:

```
if (m_bDisplayNumPoints)
        {
        CString csTmp;
        csTmp.Format("%ld", m_lTotalDisplayedPoints);
        pdc->TextOut(0, 0, csTmp);
        }
```

3.3.8 Set DisplayNumPoints

Syntax **BOOL SetDisplayNumPoints;**

Remarks **SetDisplayNumPoints** toggles the textual display of the number of points contained in the signature. The default value is Off. This display is implemented in the display control of the signature window as:

```
if (m_bDisplayNumPoints)
        {
        CString csTmp;
        csTmp.Format("%ld", m_lTotalDisplayedPoints);
        pdc->TextOut(0, 0, csTmp);
        }
```

3.4 Methods

3.4.1 SetOPOSBCNIBBLESignatureData

Syntax **void SetOPOSBCNIBBLESignatureData**(LPCSTR *lpszSigData*)

Remarks This method expects the signature data to be OPOS_BC_NIBBLE binary encoded. The format type of the encoded electronic signature data is autodetected by this function.

For example, the data format of the **PointArray** property from either an OPOS SigCap control or the Ingenico form control will always be OPOS_POINT_ARRAY and will conform to OPOS specifications. On the other hand, the data format of the **RawData** property will be the native data format of the hardware device and depending upon Control Panel configuration can be one of the following:

- CME_2BYTE_BINARY
- CME_3BYTE_ASCII
- CME_5BYTE_ASCII
- CME_4BYTE_RAW
- NCR_5991

```
enum enSignatureType
{
    SIG_NO_DATA,
    SIG_NOT_DEFINED,
    OPOS_POINT_ARRAY,
    CME_2BYTE_BINARY,
    CME_3BYTE_ASCII,
    CME_5BYTE_ASCII,
    CME_4BYTE_RAW,
    NCR_5991
};
```

3.4.2 SetOPOSBCNIBBLESignatureDataX

Syntax LONG **SetOPOSBCNIBBLESignatureDataX**(LPCTSTR *lpszSigData*, long *lSignatureType*)

Remarks This method expects the signature data to be OPOS_BC_NIBBLE binary encoded. This method is similar to the SetOPOSBCNIBBLESignatureData function, except the caller specifies the signature format type as a second parameter. Signature format type recognition is highly reliable, but if the caller knows the signature format type, it is recommended they make use of this function over the previous version of this control.

For example, the data format of the **PointArray** property from either an OPOS SigCap control or the Ingenico form control will always be OPOS_POINT_ARRAY and will conform to OPOS specifications. On the other hand, the data format of the **RawData** property will be the native data format of the hardware device and depending upon Control Panel configuration can be one of the following:

- CME_2BYTE_BINARY
- CME_3BYTE_ASCII
- CME_5BYTE_ASCII
- CME_4BYTE_RAW
- NCR_5991

```
enum enSignatureType
{
    SIG_NO_DATA,
    SIG_NOT_DEFINED,
    OPOS_POINT_ARRAY,
    CME_2BYTE_BINARY,
    CME_3BYTE_ASCII,
    CME_5BYTE_ASCII,
    CME_4BYTE_RAW,
    NCR_5991
};
```

Return One of the following values is returned by the method:

Value	Meaning
0	Success
Anything else	Failure

3.4.3 SetSignatureData

Syntax void **SetSignatureData** (const VARIANT& *varSigData*, long *lBinaryConversion*);

Remarks This method is designed to take a VARIANT argument containing the electronic signature data. This VARIANT can be any supported Ingenico signature type, and any supported OPOS binary conversion. The SetSignatureData method performs auto detection of the Ingenico or OPOS signature type.

When using these functions, it is necessary to provide the OPOS binary conversion to the signature display control so it can convert the data back into a raw data format. OPOS recognizes 3 binary conversion techniques that correspond to the BinaryConversion property on the OPOS SigCap or OPOS (Extension) Form control.

The standard OPOS binary conversion constants are:

```
// OPOS Defined Binary Conversion Constants
const LONG OPOS_BC_NONE          = 0;
const LONG OPOS_BC_NIBBLE        = 1;
const LONG OPOS_BC_DECIMAL     = 2;
```

3.4.4 SetSignatureDataX

Syntax LONG **SetSignatureDataX** (const VARIANT& *varSigData*, long *lBinaryConversion*, long *lSignatureType*)

Remarks This method is designed to take a VARIANT argument containing the electronic signature data. This VARIANT can be any supported Ingenico signature type, and any supported OPOS binary conversion. The SetSignatureDataX method allows the caller to specify the Ingenico or OPOS signature type, while the SetSignatureData method performs autodetection of the Ingenico or OPOS signature type.

When using these functions, it is necessary to provide the OPOS binary conversion to the signature display control can convert the data back into a raw data format. OPOS recognized 3 binary conversion techniques that will correspond to the BinaryConversion property on the OPOS SigCap or OPOS (Extension) Form control.

The standard OPOS binary conversion constants are:

```
// OPOS Defined Binary Conversion Constants
const LONG OPOS_BC_NONE          = 0;
const LONG OPOS_BC_NIBBLE        = 1;
const LONG OPOS_BC_DECIMAL     = 2;
```

Return One of the following values is returned by the method:

Value	Meaning
0	Success
Anything else	Failure

3.4.5 GetSignatureType

Syntax short **GetSignatureType**

Remarks Once **SetSignatureData[X]** or **SetOPOSBCNIBBLESignatureData[X]** have been called, **GetSignatureType** and **GetSignatureTypeString** will return the interrogated signature type to the caller.

Return values will be one of the following:

- OPOS_POINT_ARRAY
- CME_2BYTE_BINARY
- CME_3BYTE_ASCII
- CME_5BYTE_ASCII
- CME_4BYTE_RAW
- NCR_5991
- SIG_NOT_DEFINED

3.4.6 GetSignatureTypeString

Syntax BSTR **GetSignatureTypeString**

Remarks Once **SetSignatureData[X]** or **SetOPOSBCNIBBLESignatureData[X]** have been called, **GetSignatureType** and **GetSignatureTypeString** will return the interrogated signature type to the caller.

Return values will be one of the following:

- OPOS_POINT_ARRAY
- CME_2BYTE_BINARY
- CME_3BYTE_ASCII
- CME_5BYTE_ASCII
- CME_4BYTE_RAW
- NCR_5991
- UNKNOWN_SIGNATURE_TYPE

3.4.7 WriteSignatureToFile

Syntax LONG **WriteSignatureToFile**(LPCTSTR *lpszOutputFile*, long *lOutputFormat*, long *lOutputWidth*, long *lOutputHeight*, BOOL *bDrawBorder*)

Remarks This function can be used to render the captured signature data to either a monochrome TIFF or BMP file.

Parameter	Description
lpszOutputFile	Specifies a valid file name for the output graphic file.The lOutputFormat can be one of the following: • #define FF_TIFF 0 • #define FF_BMP 1
lOutputWidth	Specifies the width of the output image file.
lOutputHeight	Specifies the height of the output image file.
bDrawBorder	Specifies whether the output graphic has a one-pixel border around the enclosing rectangle.

3.4.8 ConvertSignatureToImageBuffer

Syntax BSTR **ConvertSignatureToImageBuffer**(long *lOutputFormat*, long *lOutputWidth*, long *lOutputHeight*, BOOL *bDrawBorder*)

Remarks This function can be used to render the captured signature data as a monochrome TIFF or BMP and store it in a buffer. This buffer is returned to the caller.

Parameter	Description
lOutputFormat	Can be one of the following: • #define FF_TIFF 0 • #define FF_BMP 1
lOutputWidth	Specifies the width of the image contained in the buffer.
lOutputHeight	Specifies the height of the image contained in the buffer.

bDrawBorder	Specifies whether the output graphic has a one-pixel border around the enclosing rectangle.

In a .NET environment, the buffer returned by this function is stored in a string object. By default .NET will encode this data into Unicode, which may cause some bytes to be changed. As a result, the signature may appear blurry or contain noise. The following code snippet in C# will convert the buffer data back to a Cp1252 encoding, which eliminates this problem.

```
string buffer = sigDisplay.ConvertSignatureToImageBuffer(FF_BMP,
nWid, nHgt, true);

// encoding we've been switched to
Encoding unicode = Encoding.Unicode ;

// target encoding
Encoding extAscii = Encoding.GetEncoding(1252) ;

byte[] unicodeBytes = unicode.GetBytes(buffer) ;

// this will now contain the desired image data
byte[] extAsciiBytes = Encoding.Convert(unicode, extAscii,
unicodeBytes) ;
```

3.4.9 EnableLiveCapture

Syntax long **EnableLiveCapture**(BOOL *bEnable*)

Remarks Call this function to toggle whether live capturing is currently enabled. If live capturing is enabled, then when a call is made to store any signature data, this data will be appended to any previously stored signature data. If live capturing is disabled, such a call will cause any previously stored signature data to be replaced.

A call to **SetDeviceResolution**() must be performed prior to enabling live capture. This is in order to specify the bounds of a signature area, which cannot be known otherwise since the signature is being stored in fragments.

Parameter	**Description**
bEnable:	Whether to enable (TRUE) or disable (FALSE) live capturing.

See also
3.4.1 *SetOPOSBCNIBBLESignatureData*
3.4.2 *SetOPOSBCNIBBLESignatureDataX*
3.4.3 *SetSignatureData*
3.4.4 *SetSignatureDataX*
3.4.10 *StartLiveCapture*

3.4.10 StartLiveCapture

Syntax void **StartLiveCapture**()

Remarks Destroys all previously stored signature data. If live capturing is enabled, all previously stored signature fragments are destroyed.

See also 3.4.1 *SetOPOSBCNIBBLESignatureData*
3.4.2 *SetOPOSBCNIBBLESignatureDataX*
3.4.3 *SetSignatureData*
3.4.4 *SetSignatureDataX*
3.4.9 *EnableLiveCapture*

3.4.11 SetDeviceResolution

Syntax long **SetDeviceResolution**(long *lMaxDeviceX*, long *lMaxDeviceY*)

Remarks Specifies the boundaries of a live signature, in pixels.

Parameter	**Description**
lMaxDeviceX:	Maximum x-coordinate of the bounding signature box.
lMaxDeviceY:	Maximum y-coordinate of the bounding signature box.

Return A value of 0 indicates success.
A value of -1 indicates one or more parameters were invalid.

CHAPTER 4

Special Features of the iSeries Terminals

This chapter explains how OPOS handles special features on the iSeries terminals. The following sections are included:

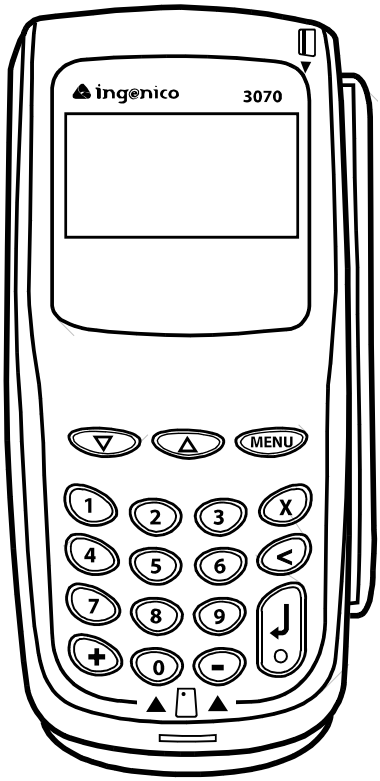
4.1 Function Key Tables (i3070, i6510, i6550, i6780 only)

The i3070, i6510, i6550, and i6780 terminals have physical keypads containing several function keys. (The i6770 does not have a keypad.) Function keys refer to keys other than the 0 through 9 keys. The following tables map these function keys to the KeyCodes that they return to the POS.

Note: Button labels for the two most popular configurations have been given. It is possible that your buttons may be labeled differently.

4.1.1 i3070 Key Table

Canadian Keymat



US Keymat

Table 4.1 KeyCodes for Ingenico 3070 Function Keys

Canadian Keys	US Keys	KeyCode (in HEX)	KeyCode (in ASCII)
CHQ	∇	0x40	'@'
SAV/EP	Δ	0x41	'A'
ALPHA	MENU	0x42	'B'
OK/Enter/Entree	↵ (Enter)	0x3A	':'
Can/Ann	X (cancel)	0x3B	';'
Corr	< (backspace)	0x3C	'<'
(blank)	+	0x3F	'?'
(blank)	-	0x3D	'='

4.1.2 i6000 Series Key Table

Figure 1 i6000 Series Sample Keymat

Table 4.2 KeyCodes for Ingenico 6510/6550/6780 Function Keys

Canadian Keys	US Keys	KeyCode (in HEX)	KeyCode (in ASCII)
↑	+	0x3F	'?'
↓	-	0x3C	'<'
OK/Enter/Entree	Enter O	0x3A	':'
Can/Ann	Cancel X	0x3B	';'
Corr	Clear <	0x3D	'='

The Ingenico 6510 supports all of the KeyCodes in Table 4.2, as well as KeyCodes for the four screen-addressable keys surrounding its display screen (see Table 4.3).

Since the Ingenico 6550 and 6780 terminals have touchscreen capabilities, it is possible to display forms with virtual buttons while the physical function keys are active. If a virtual button has an ID equal to one of the KeyCodes in Table 4.2, then the virtual button and its physical counterpart become indistinguishable, and pressing one causes the other to be changed to the pressed state as well.

4.1.3 i6510 Additional Keys

The i6510 has four additional keys, screen-addressable keys, that are not found on the other i6000 series terminals.

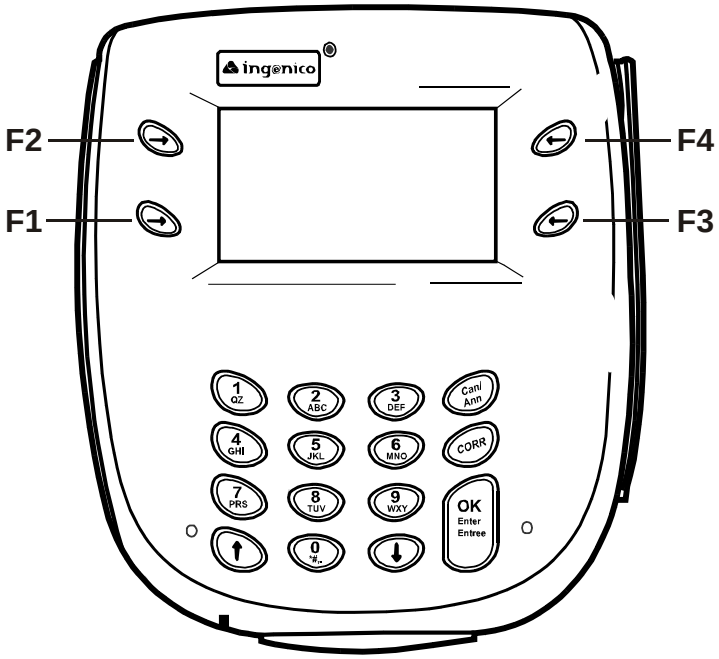


Figure 2 Ingenico 6510's Screen Addressable Keys

Table 4.3 KeyCodes for Ingenico 6510's Four Screen-Addressable Keys

Button Position	Identifier	KeyCode (in HEX)	KeyCode (in ASCII)
Lower-left corner	F1	0x40	'@'
Upper-left corner	F2	0x41	'A'
Lower-right corner	F3	0x42	'B'
Upper-right corner	F4	0x43	'C'

In the interest of security, physical numeric buttons are disabled. Consequently, virtual buttons may not possess KeyCodes in the range 0x30 - 0x39, which correspond to the ASCII values '0' through '9'.

Note: Physical numeric buttons are not disabled when a buffered clear-entry session is initiated via the **QueryKeyPadBoardText** method (for more information, see section 2.4.6 *QueryKeyPadBoardText Method* on page 2.16)

4.2 Direct I/O Usage

Several features of the Ingenico iSeries terminals that are not covered by the OPOS specification have been exposed by the **DirectIO** method. It doesn't matter which Control Object is used to send the following Direct I/O commands, because each underlying service object supports Direct I/O usage equally.

4.2.1 DirectIO Method

Syntax **LONG DirectIO**(LONG *Command*, LONG* *pData*, BSTR* *pString*)

Parameter	Description
Command	A numeric index corresponding to the desired operation.
pData	Supplementary data, dependent on command index. See below.
pString	*Input*: unused *Output*: When applicable, a string containing the terminal's response to the most recently received command.

Remarks The **DirectIO** method is capable of handling the following commands, as listed by DirectIO.h in your Developer's Documentation installation directory:

```
const LONG DIO_SEND_RAW_DATA              = 0;
const LONG DIO_CLEAR_SCREEN               = 1;
const LONG DIO_DELETE_ALL_FORMS           = 2;
const LONG DIO_RESET_TERMINAL             = 3;
const LONG DIO_ENABLE_KEY_BEEP            = 4;
const LONG DIO_DISABLE_KEY_BEEP           = 5;
const LONG DIO_CONFIGURE_KEY_MASK         = 6;
const LONG DIO_SET_FORMAT_SPECIFIER       = 7;
const LONG DIO_DELETE_RECEIPT_CONTENTS    = 8;
```

Each command is explained in the following sections.

To have OPOS perform the desired command immediately, set the *Command* parameter to the appropriate index. Specific instructions regarding each command are provided in the following sections.

When the terminal has successfully responded to a command sent by **DirectIO**, a **DirectIOEvent** is fired. Unless otherwise indicated, the *pString* parameter of this event will contain the terminal's response to that command, encoded subject to the current value of the **BinaryConversion** property. To guarantee the correctness of *pString*, this property must have a value of OPOS_BC_NIBBLE or OPOS_BC_DECIMAL when the event is fired.

Return One of the following values is returned by the **DirectIO** method and placed in the **ResultCode** property:

Value	Meaning
OPOS_SUCCESS	The method was successful.
OPOS_E_FAILURE	The method was not successful.
Other Values	See 2.3.14 *ResultCode Property R* on page 2.10

4.2.2 Send Raw Data Parameter

Parameter	Value
Command	`DIO_SEND_RAW_DATA`
pData	Ignored
pString	*Input:* An encoded sequence of bytes to send to the terminal.
	Output: The terminal's encoded response string.

Remarks Use this Direct I/O command to send a raw stream of bytes to the terminal (for full context, see 4.2.1 *DirectIO Method* on page 4.5). Ensure the **BinaryConversion** property of your control is set to OPOS_BC_NIBBLE or OPOS_BC_DECIMAL, and the contents of *pString* are encoded appropriately. See the OPOS specification for details regarding the **BinaryConversion** property.

4.2.3 Clear Screen Parameter

Parameter	Value
Command	`DIO_CLEAR_SCREEN`
pData	Ignored
pString	*Input:* ignored
	Output: The terminal's encoded response string.

Remarks Use this Direct I/O command to clear the contents of your terminal display (for full context, see 4.2.1 *DirectIO Method* on page 4.5). Your OPOS Controls will be unaware of this change in display state.

4.2.4 Delete All Forms Parameter

Parameter	Value
Command	`DIO_DELETE_ALL_FORMS`
pData	Ignored
pString	*Input:* ignored
	Output: The terminal's encoded response string.

Remarks Use this Direct I/O command to delete all forms currently stored on the terminal (for full context, see 4.2.1 *DirectIO Method* on page 4.5).

4.2.5 Reset Terminal Parameter

Parameter	Value
Command	`DIO_RESET_TERMINAL`
pData	Ignored
pString	Ignored

Remarks Use this Direct I/O command to reset the terminal (for full context, see 4.2.1 *DirectIO Method* on page 4.5). There is no response to this command.

4.2.6 Enable Key Beeps Parameter

Parameter	Value
Command	`DIO_ENABLE_KEY_BEEPS`
pData	Ignored
pString	*Input:* ignored
	Output: The terminal's encoded response string.

Remarks Use this Direct I/O command to enable key beeps (for full context, see 4.2.1 *DirectIO Method* on page 4.5). By default, all physical keys beep when pressed, so this command only has an effect when key beeps have been previously disabled via DIO_DISABLE_KEY_BEEPS.

4.2.7 Disable Key Beeps Parameter

Parameter	Value
Command	`DIO_DISABLE_KEY_BEEPS`
pData	Ignored
pString	*Input:* ignored
	Output: The terminal's encoded response string.

Remarks Use this Direct I/O command to disable key beeps (for full context, see 4.2.1 *DirectIO Method* on page 4.5). Key beeps can be re-enabled via the DIO_ENABLE_KEY_BEEPS command.

4.2.8 Configure Key Masks Parameter

Parameter	Value
Command	`DIO_CONFIGURE_KEY_MASK`
pData	Address of a number representing a logical ORing of the physical function keys to turn on.
pString	*Input:* ignored
	Output: The terminal's encoded response string.

Remarks Use this Direct I/O command to specify which physical function keys to enable on your terminal (for full context, see 4.2.1 *DirectIO Method* on page 4.5). The flags below can be ORed together to specify any combination of keys to enable. All function keys are enabled by default.

```
const LONG DIO_KM_ENABLE_NONE                    = 0x0000 ;
const LONG DIO_KM_ENABLE_ENTER                   = 0x0001 ;
const LONG DIO_KM_ENABLE_CANCEL                  = 0x0002 ;
const LONG DIO_KM_ENABLE_00                      = 0x0004 ;
const LONG DIO_KM_ENABLE_CLEAR                   = 0x0008 ;
const LONG DIO_KM_ENABLE_UP                      = 0x0020 ;
const LONG DIO_KM_ENABLE_F1                      = 0x0040 ;
const LONG DIO_KM_ENABLE_F2                      = 0x0080 ;
```

```
const LONG DIO_KM_ENABLE_F3                    = 0x0100 ;
const LONG DIO_KM_ENABLE_F4                    = 0x0200 ;
const LONG DIO_KM_ENABLE_ALL                   = 0xFFFF ;
```

A keymask can also be specified as an attribute of a form. When a form that possesses keymask information is displayed, the form's keymask overrides all keymask settings sent via the **DirectIO** method. When the form is no longer displayed on the terminal, the keymask setting will revert back to its previous setting. For more details on specifying keymasks in forms, see the Form Designer application's online help.

Example The following code snippet enables the <Enter> and <Clear> keys, and disable all others:

```
WORD nSomeStr = 0 ;
BSTR bstr = ::SysAllocString(&nsomeStr) ;
LONG nKeys = DIO_KM_ENABLE_ENTER | DIO_KM_ENABLE_CLEAR
;
LONG nCommand = DIO_CONFIGURE_KEY_MASK ;
m_Form.DirectIO(nCommand, &nKeys, &bstr) ;
```

4.2.9 Set Format Specifier Parameter

Parameter	Value
Command	`DIO_SET_FORMAT_SPECIFIER`
pData	Ignored
pString	*Input:* The format specifier string
	Output: None

Remarks Use this Direct I/O command to assign the format specifier (FS) string to use during clear entry (for full context, see 4.2.1 *DirectIO Method* on page 4.3). By default, the format specifier is an empty string, and numbers are displayed on the screen with no additional formatting.

An FS string allows you to customize the display of key data entered by the user during the clear entry process. FS strings contain display attributes that tell the terminal how to display the data on the screen. This section explains the display attributes and provides examples on how to use the attributes in an FS string.

There are two kinds of display attributes: general and specific.

- General attributes apply to the entire data entry process.
- Specific attributes apply to one or more display positions used by the data entry process. Display attributes are separated within the FS string by the percent sign (%).

These attributes are explained in the following sections.

FS strings are always parsed starting from the right, since it is assumed that the text will always be aligned to the right and shift to the left.

4.2.9.1 General Attributes

General attributes follow the format:

```
%

General attribute

Data
```

The general attributes are:

m – The *m* attribute specifies the minimum number of characters to be entered by the user. The value following this attribute is interpreted as the minimum number of characters. The default value is zero (0). The range for this attribute is 0 – MAX. If you press ENTER before typing the minimum number of characters, it has no effect. If the number specified cannot be accommodated, the *m* attribute is adjusted internally.

Note: MAX is the maximum number of display positions available within the data entry field.

M – The *M* attribute specifies the maximum number of characters to be entered by the user. The value following this attribute is interpreted as the maximum number of characters. The default value is MAX. The range for this attribute is 0 – MAX. If a user presses characters after the maximum number has been reached, those characters will not be displayed on the screen or recorded. If the number specified cannot be accommodated, the *M* attribute is adjusted internally.

p – The *p* attribute password-protects and changes the appearance of all characters entered by the user. The ASCII character following this attribute is interpreted as the password character and is displayed on the screen in place of the characters entered by the user. For example, when the user enters a PIN, the *p* attribute can specify that asterisks appear on the screen instead of numbers.

4.2.9.2 Specific Attributes

```
Specific attributes follow the format:

%

Specific attribute

Display string
```

The display string appears on the screen. The specific attributes are:

f – Fixed characters. The *f* attribute defines the corresponding positions to be displayed at all times. The *f* attribute cannot be modified during the data entry process.

h – Hidden characters. The *h* attribute causes the specific display positions to show only when the first one from the right is passed by the shifting text (text being entered by user). From that moment on, these positions are fixed and cannot be modified for the rest of the data entry process.

o – Overwriting characters. The *o* attribute defines the corresponding positions to be displayed at the beginning of the data entry process, but allows shifting text to overwrite them.

s – Shifting characters. The *s* attribute defines the corresponding positions to be displayed at the beginning of the data entry process, and then shifted one position at a time when the first one from the right is passed by shifting text.

The use of "%" as a constant in the format specifier is restricted to using it as a fixed character. It must be at the rightmost position in the specifier, e.g. `"%m0%M2%f %f%`

4.2.9.3 Examples of Format Specifiers

Table 4.4 Examples of Format Specifiers

Format Specifier Key	Pressed	Display
"%m2%M4"		
	1	1
	2	12
	3	123
	4	
	5	1234
		1234
"%m2%M6%h,%o 0%f.%o00"		0.00
	1	0.01
	2	0.12
	3	1.23
	4	12.34
	5	123.45
	6	1,234.56
	7	1,234.56
"%m0%M6%o %h,%o %s$%o0%f.%o00"		$0.00
	5	$0.05
	1	$0.51
	2	$5.12
	3	$51.23
	9	$512.39
	7	$5,123.97
	6	$5,123.97

Table 4.4 Examples of Format Specifiers (Continued)

Format Specifier Key	Pressed	Display
"%m10%M10%h(%o %s %h) %o %h-%o "		
	4	4
	1	41
	6	416
	2	4162
	4	4-1624
	5	41-6245
	6	416-2456
	7	4) 162-4567
	0	41) 624-5670
	0	(416) 245-6700
	1	(416) 245-6700
"%m9%M9%o %f/%o %f/%o "		" / / "
	1	" / / 1"
	2	" / / 12"
	3	" / /123"
	4	" / 1/234"
	5	" / 12/345"
	6	" /123/456"
	7	" 1/234/567"
	8	" 12/345/678"
	9	"123/456/789"
	1	"123/456/789"
"%m0%M2%f %f%"		%
	1	1 %
	5	15 %

4.2.10 Delete Receipt Contents Parameter

Parameter	Description
Command	DIO_DELETE_RECEIPT_CONTENTS
pData	Address of a number representing what scrolling receipt contents to delete.
pString	Input: Ignored Output: The terminal's encoded response string.

Use this Direct I/O Command to delete a row of text from a scrolling receipt area on the current form. The eight most significant bits of the number whose address is *pData* are interpreted as the index of the target row, where an index of 1 indicates the topmost row. The remaining twenty-four bits are reserved for future expansion.

4.3 Contactless Card Payment (i6510, i6550, i6770, i6780)

If you have installed Ingenico's Contactless Payment Expansion Module (CPEM) on your terminal(s), OPOS will support reading data from contactless payment cards. Writing data to a contactless card is not currently supported. The CPEM is available for the i6510, i6550, i6770, and i6780. For more information on purchasing a CPEM, contact your Ingenico representative.

OPOS clients can use the existing OPOS MSR control object to receive contactless card data. The behavior of the MSR control during device enable (i.e., when the **SetDeviceEnabled** property is set to TRUE) is subject to the current OPOS configuration as specified in the Control Panel Applet. To use the contactless payment reader, be sure to select the **Use CPEM Reader** check box in the OPOS-Ingenico Setup Program (for details, see 1.2.4 *OPOS Configuration* on page 1.4).

If you want to use the CPEM and OPOS fails to successfully enable the CPEM reader, it will fire an **ErrorEvent** to the application. MSR operation is unaffected by this failure.

When either the MSR or the CPEM reader receives card data, **DataEvents** are fired as usual by the MSR control. In addition, if OPOS is configured to use the CPEM reader, a **DirectIOEvent** is fired to indicate the source of the card data. The *pData* parameter contains the source information, and shall contain one of the following values defined in DirectIO.h:

```
const LONG DIO_CARD_SOURCE_MSR                = 0 ;
const LONG DIO_CARD_SOURCE_CPEM               = 1 ;
```

4.4 Migration from the e^{N}-Touch 1000

If you have an existing application that uses OPOS for the e^{N}-Touch 1000, and want to port your application to the Ingenico iSeries terminal family, follow these steps:

1. Ensure the UPOS Interface Application is installed on the target terminal. If it is, then when you power on the device, you will see the application title screen after a delay of a few seconds. If it is not, then you must install it (UPOS Interface Application).
2. Install OPOS on the host by running the file "OPOS for the Ingenico iSeries.exe".
3. On your desktop, select **Start** > **Settings** > **Control Panel**, and double click **Ingenico 6xxx Configuration**. The control panel application opens.

   Configure your device as needed. By default, OPOS assumes an RS-232 connection, with port settings of 19200,8,N,1.

   If your application is written in C++ and uses wrapper classes to interact with Ingenico's Control Objects, we recommend you remove and regenerate these wrapper classes. This is because the Control Objects in this Integration Kit support more recent versions of the OPOS specification.

4. Replace all device strings that are passed to the **Open**() method with the string "Ing6xxx". If "Ing6xxx" is not a suitable device string for your needs, you must modify the registry. The target keys are located at [HKLM\Software\OLEForRetail\ServiceOPOS*DeviceName*\Ing6xxx], where *DeviceName* refers to one of the six OPOS device classes supported by this Integration Kit. Simply rename Ing6xxx to whatever string suits your needs, and be sure to pass this same string to the **Open**() method.
5. If your application is written in C++, be sure it compiles with the most recent OPOS header files.

6. If you are using the Form control, you must port your existing .icf form files to the new .icg format. This can be done via the Ingenico Form Designer.

   Note: Some members of the 6xxx device family features lower screen resolutions than the e^{N}-Touch 1000, and so may require one or more elements to be resized. The screen resolutions are defined in the Form Designer online help file.

7. If you are using the PINPad control, you may need to modify the value of the *transactionHost* parameter to the **BeginEFTTransaction**() method. For the e^{N}-Touch 1000, this parameter is equal to *slotNumber+1,* where *slotNumber* is the index of the key to use for PIN encryption. For 6xxx terminals, this parameter is set equal to *slotNumber* to avoid confusion.

4.5 DUKPT Key Serial Number Format

During PIN transactions that use Derived Unique Key Per Transaction (DUKPT) key management, a key serial number (KSN) is returned from the terminal and stored in the **AdditionalSecurityInformation** property of the OPOS PINPad control. This property is a hex-formatted ASCII string 20 characters in length.

For example, if the KSN can be expressed in hexadecimal as 0xFFFF9876543210E000A, **AdditionalSecurityInformation** will report 'FFFF9876543210E000A'.

4.6 Best Practices

The following guidelines are provided to ensure maximum performance and efficiency when using OPOS with Ingenico's iSeries terminals.

When an iSeries terminal with the UPOS Interface Application installed is first powered on, a splash screen displays current connection settings and version information. If you are having trouble communicating with your iSeries terminal, compare the splash screen's connection settings with what is specified in the Ingenico OPOS Control Panel App. The splash screen will remain visible until the POS Application has begun sending data to the terminal, and then it will disappear.

Ingenico's OPOS Controls implement the OPOS Common Method **ClearInput**() to allow a user to clear incoming event queues. In some contexts, calling **ClearInput**() may result in data sent to the device, to, for example, disable the MSR. In order to minimize terminal traffic, Ingenico recommends only calling **ClearInput**() when there is data in the event queue, which can be determined by inspecting the control's **DataCount** property.

OPOS controls corresponding to input devices (PINPad, MSR, SignatureCapture, POSKeyboard, and Form) may poll the iSeries terminal when the **DeviceEnabled** property is set to TRUE. To minimize traffic and improve performance, Ingenico recommends setting **DeviceEnabled** to FALSE when that control is not in use.

Ingenico's OPOS Service Objects are designed to free all resources they consume when the **Close**() method is called. To minimize processor usage, Ingenico recommends calling **Close**() if the Service Object will not be used for an extended period of time, for example: overnight when no transactions are taking place. The **Open**() method can be subsequently called when transactions are resumed.

Customers moving to an i6780 terminal from an i6770 terminal must consider when porting their forms that the i6780 terminal's screen is 6 pixels shorter than the i6770's. Also, all line display screen modes result in one less row for the i6780 than the i6770. In addition, since the i6770 does not have a physical

keypad and the i6780 does have a keypad, you must consider that any or all physical keys may be active when a form is displayed, and so the POS application must consider a larger group of keys capable of generating data events than before.

Index

E

F

G

H

I

K

L

M

O

P

Q

R

S

T

U

V

W