

OLE for Retail POS

Application Programmer's Guide

Release 1.1 April 22, 1996

International Standard

Windows 95, Windows NT, or
other OLE compliant 32-bit
operating system

OLE for Retail POS Committee:

Microsoft
NCR
Epson
Post Software International

OLE for Retail POS

Application Programmer's Guide

Information in this document is subject to change without notice.

The latest revisions of the OLE for Retail POS documents are placed on the Microsoft Web site (www.microsoft.com), on the Industry Solutions / Retail / Technologies page. (This page is currently named [http://www.microsoft.com/industry/ret/retech.htm/.](http://www.microsoft.com/industry/ret/retech.htm/))

- © 1995-1996 NCR Corporation. All rights reserved.
- © 1995-1996 Microsoft Corporation. All rights reserved.
- © 1995-1996 Post Software International. All rights reserved.
- © 1995-1996 Seiko Epson Corporation. All rights reserved.

Table of Contents

OLE FOR RETAIL POS CONTROLS	7
<i>What Is “OLE for Retail POS?”</i>	<i>7</i>
<i>Who Should Read This Document</i>	<i>8</i>
GENERAL OLE FOR RETAIL POS CONTROL MODEL	8
<i>OPOS Definitions</i>	<i>9</i>
<i>How an Application Uses an OPOS Control</i>	<i>10</i>
<i>When Methods and Properties May Be Accessed</i>	<i>11</i>
STATUS, RESULT CODE, AND STATE MODEL	13
<i>Status Model</i>	<i>14</i>
<i>Result Code Model</i>	<i>14</i>
<i>State Model</i>	<i>15</i>
DEVICE SHARING MODEL	17
<i>Exclusive-Use Devices</i>	<i>17</i>
<i>Sharable Devices</i>	<i>17</i>
INPUT MODEL	19
OUTPUT MODEL	21
<i>Synchronous Output</i>	<i>21</i>
<i>Asynchronous Output</i>	<i>21</i>
OPOS CONTROL DESCRIPTIONS	22
COMMON PROPERTIES, METHODS, AND EVENTS.....	25
SUMMARY	25
GENERAL INFORMATION.....	27
PROPERTIES.....	28
METHODS.....	41
EVENTS.....	48
CASH DRAWER.....	53
SUMMARY	53
GENERAL INFORMATION.....	55
PROPERTIES.....	56
METHODS.....	57
EVENTS.....	59
COIN DISPENSER	61
SUMMARY	61
GENERAL INFORMATION.....	63
PROPERTIES.....	64
METHODS.....	66
EVENTS.....	67

HARD TOTALS.....	69
SUMMARY	69
GENERAL INFORMATION	73
PROPERTIES.....	77
METHODS.....	80
KEYLOCK.....	95
SUMMARY	95
GENERAL INFORMATION	97
PROPERTIES.....	98
METHODS.....	99
EVENTS.....	100
LINE DISPLAY	101
SUMMARY	101
GENERAL INFORMATION	104
PROPERTIES.....	107
METHODS.....	129
MICR - MAGNETIC INK CHARACTER RECOGNITION READER.....	141
SUMMARY	141
GENERAL INFORMATION	144
MICR CHARACTER SUBSTITUTION.....	147
PROPERTIES.....	148
METHODS.....	153
EVENTS.....	157
MSR - MAGNETIC STRIPE READER.....	161
SUMMARY	161
GENERAL INFORMATION	164
PROPERTIES.....	166
EVENTS.....	176
POS KEYBOARD.....	179
SUMMARY	179
GENERAL INFORMATION	181
PROPERTIES.....	182
EVENTS.....	182
POS PRINTER.....	185
SUMMARY	185
GENERAL INFORMATION	191
DATA CHARACTERS AND ESCAPE SEQUENCES	195
PROPERTIES.....	198
METHODS.....	240

EVENTS.....	268
SCALE.....	270
SUMMARY	270
GENERAL INFORMATION.....	272
PROPERTIES.....	273
METHODS.....	274
SCANNER (BAR CODE READER).....	277
SUMMARY	277
GENERAL INFORMATION.....	279
PROPERTIES.....	280
EVENTS.....	281
SIGNATURE CAPTURE.....	283
SUMMARY	283
GENERAL INFORMATION.....	285
PROPERTIES.....	287
METHODS.....	290
EVENTS.....	292
APPENDICES	295
CHANGE HISTORY	295
RELEASE 1.01	295
RELEASE 1.1	298
OPOS REGISTRY USAGE	301
OPOS APPLICATION HEADER FILES	305
OPOS.H : MAIN OPOS HEADER FILE.....	305
OPOSCASH.H : CASH DRAWER HEADER FILE	306
OPOSCOIN.H : COIN DISPENSER HEADER FILE.....	307
OPOSTOT.H : HARD TOTALS HEADER FILE.....	308
OPOSDISP.H : LINE DISPLAY HEADER FILE.....	309
OPOSLOCK.H : KEYLOCK HEADER FILE.....	311
OPOSMICR.H : MICR HEADER FILE	312
OPOSMSR.H : MSR HEADER FILE.....	313
OPOSKBD.H : POS KEYBOARD HEADER FILE.....	314
OPOSPTR.H : POS PRINTER HEADER FILE	315
OPOSSCAL.H : SCALE HEADER FILE	319
OPOSSCAN.H : BAR CODE SCANNER HEADER FILE.....	320
OPOSSIG.H : SIGNATURE CAPTURE HEADER FILE.....	320
TECHNICAL DETAILS.....	323

SYSTEM STRINGS (BSTR) 323
EVENT HANDLERS 328
END OF APPLICATION PROGRAMMER'S GUIDE 330

I N T R O D U C T I O N

OLE for Retail POS Controls

What Is “OLE for Retail POS?”

OLE for Retail POS provides an open device driver architecture that allows Point-of-Sale (“POS”)¹ hardware to be easily integrated into POS systems based on Microsoft Windows-95 and Microsoft Windows-NT.²

The goals of OLE for Retail POS (or “OPOS”) include:

- Defining an architecture for Win32-based POS device access.
- Defining a set of POS device interfaces sufficient to support a range of POS solutions.

Deliverables in this initial release of OPOS are:

- Application Programmer’s Guide – this document: For application developers and hardware providers.
- Control Programmer’s Guide: For hardware providers.
- Header files with OPOS constants.
- No complete software components: Hardware providers or third-party providers develop and distribute these components.

¹ POS may also refer to Point-of-Service – a somewhat broader category than Point-of-Sale.

² Other future operating systems that support OLE Controls may also support OLE for Retail POS, depending upon software support by the hardware manufacturers or third-party developers.

Who Should Read This Document

The Application Programmer's Guide is targeted to an application developer who requires access to POS-specific peripheral devices. It is also targeted for the system developer who will write an OPOS Control.

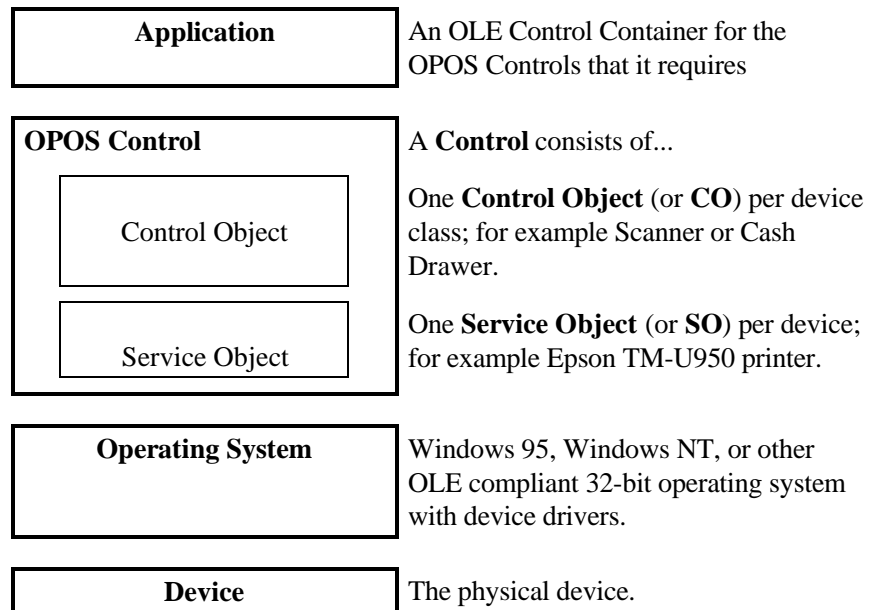
This guide assumes that the reader is familiar with the following:

- General characteristics of POS peripheral devices.
- OLE Control and OLE Automation terminology and architecture.
- Familiarity with an OLE Control Container development environment, such as Microsoft Visual Basic 4.0 (32-bit version) or Microsoft Visual C++ 4.0, will be useful.

General OLE for Retail POS Control Model

OLE for Retail POS Controls adhere to the OLE Control specifications. They expose properties, methods, and events to a containing Application. The controls are invisible at run time, and rely exclusively upon the containing application for requests through methods and sometimes properties. Responses are given to the application through method return values and parameters, properties, and events.

The OLE for Retail POS software is implemented using two layers, as shown in the following diagram:



OPOS Definitions

Device Class

A device class is a category of POS devices that share a consistent set of properties, methods, and events. Examples are Cash Drawer and POS Printer.

Some devices support more than one device class. For example, some POS Printers include a Cash Drawer kickout. Also, some Bar Code Scanners include an integrated Scale.

Control Object *or* CO

A Control Object exposes a set of properties, methods, and events to an application for its device class. This guide describes these APIs.

A CO is a standard OLE 32-bit Control that is invisible at runtime. The CO interfaces have been designed so that all implementations of a class' Control Object will be compatible. This allows the CO to be developed independently of the SO's for the same class – including development by different companies.

Service Object *or* SO

A Service Object is called by a Control Object to implement the OPOS-prescribed functionality for a specific device.

An SO is implemented as an OLE Automation server. It exposes a set of methods that are called by a CO. It can also call special methods exposed by the CO to cause events to be fired to the application.

A Service Object may include multiple sets of methods in order to support devices with multiple device classes.

A Service Object is typically implemented as a local in-proc server (in a DLL), although it may also be implemented as a local out-proc server (in a separate executable process). The latter implementation is probable when a device is shareable by more than one application, or when the device supports more than one device class.

OPOS Control *or* Control

An OPOS Control consists of a Control Object for a device class – which provides the application interface, plus a Service Object – which implements the APIs. The Service Object must support a device of the Control Object's class.

Usually, this guide will refer to “Control.” On occasion, we must distinguish between the actions performed by the Control Object and Service Object. Then the explicit layer is specified.

How an Application Uses an OPOS Control

The first action the application must take on the Control is to call its **Open** method. The parameter of this method selects a device name to associate with the Control. The **Open** method performs the following steps:

- Establishes a link to the device name.
- Initializes the properties **Claimed**, **DeviceEnabled**, **DataEventEnabled**, **FreezeEvents**, as well as descriptions and version numbers of the OPOS Control layers. Additional class-specific properties may also be initialized.

After the device is open, the application will often need to call the **Claim** method to gain exclusive access to the device. Many devices must be **Claimed** before the Control allows access to its methods and properties. Claiming the device ensures that other applications do not interfere with the use of the device. The application may **Release** the device when the device can be shared by other applications – for instance, at the end of a transaction.

Before using the device, the application must set the **DeviceEnabled** property to TRUE. This value brings the device to an operational state, while FALSE disables the device. For example, if a scanner Control is disabled, then the device will be physically disabled (when possible). Whether physically disabled or not, any input from the device will be discarded until the device is enabled.

After the application has finished using the device, the **Close** method should be called to release the device and associated resources. If the **DeviceEnabled** property is TRUE, then **Close** disables the device. If the **Claimed** property is TRUE, then **Close** releases the lock. Before exiting, an application should close all open OPOS Controls.

In summary, the application follows this general sequence:

- **Open** method: Call to link the Control Object to the Service Object.
- **Claim** method: Call to gain exclusive access to the device. Required for exclusive-use devices; optional for some sharable devices. (See “Device Sharing Model”, page 17 for more information).
- **DeviceEnabled** property: Set to TRUE to make the device operational. (For sharable devices, the device may be enabled without first **Claiming** it.)
- *Use the device.*
- **DeviceEnabled** property: Set to FALSE to disable the device.
- **Release** method: Call to release exclusive access to the device.
- **Close** method: Call to release the Service Object from the Control Object.

When Methods and Properties May Be Accessed

Methods

Before a successful **Open**, no other methods may be invoked. Doing so will do nothing but return a status of OPOS_E_CLOSED.

Exclusive-use devices require the application to call the **Claim** method and to set the **DeviceEnabled** property to TRUE before most other methods may be called.

Sharable devices require the application to set the **DeviceEnabled** property to TRUE before most other methods may be called.

The “Summary” section of each device class’ chapter should be consulted for the specific prerequisites for each method.

Properties

Before a successful **Open**, the values of most properties are not initialized. An attempt to set writable properties will be ignored.

The following properties are always initialized:

Property	Value
State	OPOS_S_CLOSED
ResultCode	OPOS_E_CLOSED
ControlObjectDescription	Control Object dependent string.
ControlObjectVersion	Control Object dependent number.

Capability properties are initialized after the **Open** is successfully called.

Exclusive use devices require the application to call the **Claim** method and to set the **DeviceEnabled** property to TRUE before some other properties are initialized or may be written.

Sharable devices require the application to set the **DeviceEnabled** property to TRUE before some other properties are initialized or may be written.

To determine when a property is initialized or writable, refer to the Summary section of each device class plus the property's Remarks section.

Setting writable properties before the prerequisites are met will cause the write to be ignored, and will set the **ResultCode** property to either OPOS_E_NOTCLAIMED or OPOS_E_DISABLED.

Reading an uninitialized property returns the following values, unless otherwise specified in the device class documentation:

Property Type	Value
<i>Boolean</i>	FALSE
<i>Long</i>	0
<i>String</i>	"[Error]" – include the brackets.

After properties have been initialized, subsequent claims and enables do not reinitialize the properties. They remain initialized until the **Close** method is called.

Status, Result Code, and State Model

The status, result code, and state models are built around several common properties, events, and methods, described in the following table, and are supported by additional class-specific components.

Name	Meaning
State	A property containing the current state of the Control: OPOS_S_CLOSED OPOS_S_IDLE OPOS_S_BUSY OPOS_S_ERROR
ResultCode	A property containing the status of the most recent method or the most recently changed writable property: OPOS_SUCCESS OPOS_E_CLOSED OPOS_E_CLAIMED OPOS_E_NOTCLAIMED OPOS_E_NOSERVICE OPOS_E_DISABLED OPOS_E_ILLEGAL OPOS_E_NOHARDWARE OPOS_E_OFFLINE OPOS_E_NOEXIST OPOS_E_EXISTS OPOS_E_FAILURE OPOS_E_TIMEOUT OPOS_E_BUSY OPOS_E_EXTENDED
ResultCodeExtended	A property containing the extended status of the most recent method or the most recently changed writable property. Value varies by ResultCode and by device class.
StatusUpdateEvent	An event fired when some class-specific state or status variable has changed.
ErrorEvent	An event fired when the State is changed to Error.

Status Model

The rules of the status model are as follows:

- The only aspect of the status model that is common to all device classes is the means of alerting the application, which is through the firing of the **StatusUpdateEvent**.
- Each device class specifies the status changes that cause it to fire the event. Examples of device class-specific status changes are:
 - ◆ A change in the cash drawer position (for example, a transition from open to closed).
 - ◆ A change in a POS printer sensor (for example, activation of a “form present” sensor, indicating that a slip has been inserted).

Result Code Model

The rules of the result code model are as follows:

- Every method returns a result code. This code is also placed into **ResultCode**.
- Setting a writable property causes a result code to be placed into **ResultCode**.
- The **ResultCode** OPOS_SUCCESS is assigned the value of zero. Non-zero values indicate an error or warning.
- Whenever possible, the Control selects one of the standard result codes (listed on page 36). If the result to be reported cannot be represented by one of these codes, then the Control sets **ResultCode** to OPOS_E_EXTENDED, and sets **ResultCodeExtended** to one of the values specified in the device class documentation.

If the Control sets **ResultCode** to a value other than OPOS_E_EXTENDED, then the Service Object may set the **ResultCodeExtended** property to an SO-specific value. If an application uses these values, it will, of course, need to add Service Object-specific code. (If the application needs to add such code, then the **ServiceObjectDescription**, **DeviceDescription**, or **DeviceName** property may be interrogated to determine the Service Object with which it is dealing.)

State Model

The rules of the state model are as follows:

- The Control's **State** is initially OPOS_S_CLOSED.
- The **State** is changed to OPOS_S_IDLE when the **Open** method is called and its result is OPOS_SUCCESS.
- The **State** is set to OPOS_S_BUSY when OPOS is processing output. The **State** is restored to OPOS_S_IDLE when these complete successfully.
- The **State** is changed to OPOS_S_ERROR when:
 - ◆ An asynchronous output encounters an error condition.
 - ◆ An error is encountered during the gathering or processing of event-driven input.

After OPOS changes the **State** property to OPOS_S_ERROR, it invokes **ErrorEvent**. The parameters to this event are the result code and extended result code, the locus of the error, and a pointer to the application's response to the error. The locus can indicate one of three error locations:

- ◆ Output – The error occurred while processing previously queued output.
- ◆ InputWithData – The error occurred while gathering or processing event-driven input. Some previously gathered input data is available for the application. When this error locus is given, then the application can continue to process input until a second **ErrorEvent** is received with the InputNoData locus, or it can clear the input.
- ◆ InputNoData – The error occurred while gathering or processing event-driven input, and either all previously gathered input data has been processed or there is no input data available.

When the application returns from the **ErrorEvent**, it may change the response parameter. The response values are:

- ◆ Retry – Use only if the locus is Output. Retry the asynchronous output and exit the error state. If an error occurs while retrying, then another **ErrorEvent** will be generated.
“Retry” is the default response when the locus is “Output.”
- ◆ Clear – Clear the asynchronous output or buffered input data and exit the error state.
“Clear” is the default response when the locus is “InputNoData.”

- ◆ Continue – Use only if the locus is InputWithData. This response acknowledges the error and directs the Control to continue processing. The Control remains in the error state, and will fire additional data events as directed by the **DataEventEnabled** property. When all input has been fired and the **DataEventEnabled** property is again set to TRUE, then another **ErrorEvent** is fired with locus “InputNoData.”
“Continue” is the default response when the locus is “InputNoData.”

The Control ensures that while the application is processing an **ErrorEvent**, it will not fire any other **ErrorEvents**.

Device Sharing Model

The OLE for Retail POS device sharing model supports devices that are to be used exclusively by one application³ at a time, as well as devices that may be partially or fully shared by multiple applications. (See “When Methods and Properties May Be Accessed”, page 11, for other details.)

Exclusive-Use Devices

The most common device type is called an “exclusive-use device.” An example is the POS printer. Due to physical or operational characteristics, this device can only be used by one application at a time. The application must call the **Claim** method to gain exclusive access to the device before most methods, properties, or events are legal. Until the device is claimed, calling methods or setting properties cause an OPOS_E_NOTCLAIMED error, and events are not fired to the application.

Should two closely cooperating applications want to treat an exclusive-use device in a shared manner, then one application may claim the device for a short sequence of operations, then release it so that the other application may use it.

When the **Claim** method is called again, settable device characteristics are restored to their condition at **Release**. Examples of restored characteristics are the line display’s brightness, the MSR’s tracks to read, and the printer’s characters per line. State characteristics are not restored, such as the printer’s sensor properties. Instead, these are updated to their current values.

Sharable Devices

Some devices are “sharable devices.” An example is the keylock. A sharable device allows multiple applications to call its methods and access its properties. Also, it may fire its events to all applications that have opened it. A sharable device may still limit access to some methods or properties to an application that has claimed it, or may fire some events only to this application.

Note One might argue that all devices should be defined as sharable to allow

³ This document assumes that an application consists of only one process. Multi-process applications are possible to create but uncommon. Technically, device sharing is performed on a process basis. However, with single-process applications we can view sharing as application-level.

maximum flexibility to applications. In practical use, this flexibility is unlikely to be useful. The downside is an implementation that may be significantly more complex and less likely to be accurate.

In the interest of a specification that is both sufficiently robust for application development, plus implementable by hardware manufacturers, this document defines most devices as exclusive-use, and defines as sharable only those devices that have a significant potential for simultaneous use by multiple applications.

Input Model

The OLE for Retail POS input model supports event-driven input. When the application is ready to receive input from the device, it sets the **DataEventEnabled** property to TRUE. Then, when input is received (usually as a result of a hardware interrupt), the Control generates a **DataEvent**. This event may include input status information through a numeric parameter. The Control places the input data plus other information as needed into device specific-specific properties just before the event is fired.

Just before firing this event, OPOS disables further data events by setting the **DataEventEnabled** property to FALSE. This causes buffering of further input data by OPOS while the application processes the current input and associated properties. When the application has finished the current input and is ready for more data, it reenables events by setting **DataEventEnabled** to TRUE.

If the input device is an exclusive-use device, the application must both claim and enable the device before the device begins reading input.

For sharable input devices, one or more applications must open and enable the device before the device begins reading input. An application must call the **Claim** method to request exclusive access to the device before OPOS will send data to it using the **DataEvent**. If event-driven input is received, but no application has claimed the device, then the input is buffered until an application **Claims** the device (and the **DataEventEnabled** property is TRUE). This behavior allows orderly sharing of the device between multiple applications, effectively passing the input focus between them.

If OPOS encounters an error while gathering or processing event-driven input, then it will fire an **ErrorEvent** to alert the application of the error condition. This event is not fired until the **DataEventEnabled** property is TRUE, so that orderly application sequencing occurs. Error events are fired with one or both of the following loci:

- ◆ **InputWithData (OPOS_EL_INPUT_DATA)** – Fired if the error occurred when some previously gathered input data has been buffered. This event gives the application the ability to immediately clear the input, or to optionally alert the user to the error and process the buffered input.

The latter case may be useful with a Scanner Control: The user can be immediately alerted to the error so that no further items are scanned until the error is resolved. Any previously scanned items can then be successfully processed before error recovery is performed.

- ◆ **InputNoData (OPOS_E_INPUT)** – Fired when an error has occurred and there is no data available. If some input data was buffered when the error occurred, then an **ErrorEvent** with the locus “InputWithData” was fired first, and this error event signals that all buffered data has been processed.

All input buffered by OPOS may be deleted by calling the **ClearInput** method.

The general event-driven input model does not specifically rule out the definition of device classes containing methods or properties that return input data directly. Some device classes will define such methods and properties in order to operate in a more intuitive or flexible manner.

Output Model

The OLE for Retail POS output model consists of two output types: synchronous and asynchronous. A device class may support one or both types, or neither type.

Synchronous Output

This type of output is preferred when device output can be performed quickly. Its merit is simplicity.

The application calls a class-specific method to perform output. The Control does not return until the output is completed.

Asynchronous Output

This type of output is preferred when device output requires slow hardware interactions. Its merit is perceived responsiveness, since the application can perform other work while the device is performing the output.

The application calls a class-specific method to start the output. The Control buffers the request, sets the **OutputID** property to an identifier for this request, and returns as soon as possible. When the device completes the request successfully, OPOS fires an **OutputCompleteEvent**. A parameter of this event contains the **OutputID** of the completed request.

If an error occurs while performing an asynchronous request, an **ErrorEvent** is fired. The application's event handler can either retry the outstanding output or clear it.

OPOS guarantees that asynchronous output is performed on a first-in first-out basis.

All output buffered by OPOS may be deleted by calling the **ClearOutput** method. **OutputCompleteEvents** will not be fired for cleared output. This method also stops any output that may be in progress (when possible).

OPOS Control Descriptions

Chapter 1 provides interface descriptions for the common properties, events, and methods.

The following chapters provide interface descriptions for the following OLE for Retail POS OLE Controls:

- Cash Drawer
- Coin Dispenser
- Line Display
- Hard Totals
- Keylock
- Magnetic Ink Character Recognition (MICR) Reader
- Magnetic Stripe Reader (MSR)
- POS Keyboard
- POS Printer
- Scale
- Scanner – Bar Code Reader
- Signature Capture

The parameter and return types specified in the descriptions are as follows:

Type	Meaning
BOOL	An integer with the legal values TRUE (non-zero) and FALSE (zero).
BSTR	A character string. Consists of a length component followed by the string and a terminating NUL (0) character. See “System Strings (BSTR)” (page 323) for more information.
BSTR*	A pointer to a character string.
LONG	An integer with a size of 32 bits.
LONG*	A pointer to an integer.

Appendix A provides a history of changes to this document.
Appendix B details the OPOS use of the system registry.

Appendix C contains the OPOS application header files.
Appendix D gives miscellaneous additional technical information.

CHAPTER 1

Common Properties, Methods, and Events

Summary

Properties

<i>Name</i>	<i>Type</i>	<i>Access</i>
CheckHealthText	String	R
Claimed	Boolean	R
DataEventEnabled	Boolean	R/W
DeviceEnabled	Boolean	R/W
FreezeEvents	Boolean	R/W
OutputID	Long	R
ResultCode	Long	R
ResultCodeExtended	Long	R
State	Long	R
ControlObjectDescription	String	R
ControlObjectVersion	Long	R
ServiceObjectDescription	String	R
ServiceObjectVersion	Long	R
DeviceDescription	String	R
DeviceName	String	R

Methods

Name

Open

Close

Claim

Release

CheckHealth

ClearInput

ClearOutput

DirectIO

Events

Name

DataEvent

DirectIOEvent

ErrorEvent

OutputCompleteEvent

StatusUpdateEvent

General Information

This section lists properties, events, and methods that are common to many of the subsequent device categories.

The summary section of each device class marks those common properties, events, and methods which do not apply to that class as “Not Supported.” These are not present in the class’ controls.

Properties

CheckHealthText Property

Syntax	BSTR CheckHealthText Property;
Remarks	<p>Holds the results of the most recent call to the CheckHealth method. The following examples illustrate some possible diagnoses:</p> <ul style="list-style-type: none">• “Internal HCheck: Successful”• “External HCheck: Not Responding”• “Interactive HCheck: Complete” <p>Before the first CheckHealth method call, its value is uninitialized.</p>
See Also	CheckHealth Method

Claimed Property

Syntax	BOOL Claimed;
Remarks	<p>If TRUE, the device is claimed for exclusive access. If FALSE, the device is released for sharing with other applications.</p> <p>Many devices must be claimed before the Control will allow access to many of its methods and properties, and before it will fire events to the application.</p> <p>The value of Claimed is initialized to FALSE by the Open method.</p>
See Also	“General OLE for Retail POS Control Model”; “Device Sharing Model”; Claim Method; Release Method

ControlObjectDescription Property

Syntax **BSTR ControlObjectDescription;**

Remarks String identifying the Control Object and the company that produced it.

The property identifies the Control Object. A sample returned string is:

 "POS Printer OLE Control, (C) 1995 Epson"

This property is always readable.

See Also **ControlObjectVersion** Property

ControlObjectVersion Property

Syntax **LONG ControlObjectVersion;**

Remarks Control Object version number.

This property holds the Control Object version number. Three version levels are specified, as follows:

Version Level	Description
Major	The “millions” place. A change to the OPOS major version level for a device class reflects significant interface enhancements, and may remove support for obsolete interfaces from previous major version levels.
Minor	The “thousands” place. A change to the OPOS minor version level for a device class reflects minor interface enhancements, and must provide a superset of previous interfaces at this major version level.
Build	The “units” place. Internal level provided by the Control Object developer. Updated when corrections are made to the CO implementation.

A sample version number is:

1002038

This value may be displayed as version “1.2.38”, and interpreted as major version 1, minor version 2, build 38 of the Control Object.

This property is always readable.

See Also **ControlObjectDescription** Property

DataEventEnabled Property R/W

Syntax	BOOL DataEventEnabled;				
Remarks	<p>When TRUE, a DataEvent will be fired as soon as input data is received. If changed to TRUE and some input data is already queued, then a DataEvent is fired immediately.</p> <p>When FALSE, input data is queued for later delivery to the application. Also, if an input error occurs, the ErrorEvent is not fired while DataEventEnabled is FALSE.</p> <p>This property is initialized to FALSE by the Open method.</p>				
Return	When this property is set, the following value is placed in the ResultCode property:				
	<table border="1"> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>OPOS_SUCCESS</td> <td>The property was set successfully.</td> </tr> </tbody> </table>	Value	Meaning	OPOS_SUCCESS	The property was set successfully.
Value	Meaning				
OPOS_SUCCESS	The property was set successfully.				
See Also	“Input Model”; DataEvent				

DeviceDescription Property

Syntax	BSTR DeviceDescription;
Remarks	<p>String identifying the device.</p> <p>The property identifies the device and any pertinent information about it. A sample returned string is:</p> <p style="padding-left: 40px;">“NCR 7192-0184 Printer, Japanese Version”</p> <p>This property is initialized by the Open method.</p>
See Also	DeviceName Property

DeviceEnabled Property R/W

Syntax **BOOL DeviceEnabled;**

Remarks When TRUE, the device has been placed in an operational state. If changed to TRUE, then the device is brought to an operational state.

When FALSE, the device has been disabled. If changed to FALSE, then the device is physically disabled when possible, any subsequent input will be discarded, and output operations are disallowed.

Changing this property usually does not physically affect output devices. For consistency, however, the application must set this property to TRUE before using output devices.

This property is initialized to FALSE by 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.
OPOS_E_NOTCLAIMED	An exclusive use device must be claimed before the device may be enabled.
<i>Other Values</i>	See ResultCode .

See Also “General OLE for Retail POS Control Model”

DeviceName Property

Syntax **BSTR DeviceName;**

Remarks Short string identifying the device.

The property identifies the device and any pertinent information about it. This is a short version of **DeviceDescription** and should be limited to 30 characters.

DeviceName will typically be used to identify the device in an application message box, where the full description is too verbose. A sample returned string is:

`"NCR 7192 Printer, Japanese"`

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

See Also **DeviceDescription** Property

FreezeEvents Property R/W

Syntax **BOOL FreezeEvents;**

Remarks When TRUE, the application has requested that the Control not fire events. Events will be held by the Control until events are unfrozen.

When FALSE, the application allows events to be fired. If some events have been held while events were frozen, then changing **FreezeEvents** to FALSE will cause these events to be fired.⁴

An application may choose to freeze events for a specific sequence of code where interruption by an event is not desirable.

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

Return When this property is set, the following value is placed in the **ResultCode** property:

Value	Meaning
OPOS_SUCCESS	The property was set successfully.

⁴ Actually, firing of events can also be deferred by the containing application. A control container may request controls to freeze event firing. For example, this feature is utilized by Visual Basic when modal dialog boxes are active. Therefore, events are fired when both **FreezeEvents** is FALSE and the container has not requested event freezing. Container-initiated event freezing is not referenced elsewhere in this document, since an Application will seldom if ever notice it and cannot directly control it.

OutputID Property

Syntax **LONG OutputID;**

Remarks Holds the identifier of the most recently started asynchronous output.

When a method successfully initiates an asynchronous output, the Control assigns an identifier to the request. When the output completes, the Control will fire an **OutputCompleteEvent** passing this output ID as a parameter.

The output ID numbers are assigned by the Control and are guaranteed to be unique among the set of outstanding asynchronous outputs. No other facts about the ID should be assumed.

See Also “Output Model”; **OutputCompleteEvent**

ResultCode Property

Syntax **LONG ResultCode;**

Remarks This property is set by each method. It is also set when a writable property is set. This property is always readable. Before the **Open** method is called, it returns the value OPOS_E_CLOSED.

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.
OPOS_E_ILLEGAL	Attempt was made to perform an illegal or unsupported operation with the device, or an invalid parameter value was used.
OPOS_E_NOHARDWARE	The device is not connected to the system or is not powered on.
OPOS_E_OFFLINE	The device is off-line.
OPOS_E_NOEXIST	The file name (or other specified value) does not exist.
OPOS_E_EXISTS	The file name (or other specified value) already exists.

OPOS_E_FAILURE	The device cannot perform the requested procedure, even though the device is connected to the system, powered on, and on-line.
OPOS_E_TIMEOUT	The Service Object timed out waiting for a response from the device, or the Control timed out waiting for a response from the Service Object.
OPOS_E_BUSY	Cannot perform operation while the State is OPOS_S_BUSY: Must wait until output is no longer in progress.
OPOS_E_EXTENDED	A class-specific error condition occurred. The error condition code is available in the ResultCodeExtended property.

See Also “Status, Result Code, and State Model”

ResultCodeExtended Property

Syntax **LONG ResultCodeExtended;**

Remarks When the **ResultCode** is set to OPOS_E_EXTENDED, this property is set to a class-specific value, matching one of the values given in the device class documentation.

When the **ResultCode** is set to any other value, this property may be set by the Service Object to an SO-specific value. These values are only meaningful if the application adds Service Object-specific code to handle them.

See Also **ResultCode** Property

ServiceObjectDescription Property

Syntax **BSTR ServiceObjectDescription;**

Remarks String identifying the Service Object supporting the device and the company that produced it.

A sample returned string is:

```
"TM-U950 Printer OPOS Service Driver, (C) 1995 Epson"
```

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

ServiceObjectVersion Property

Syntax **LONG ServiceObjectVersion;**

Remarks Service object version number.

This property holds the Service Object version number. Three version levels are specified, as follows:

Version Level	Description
Major	The “millions” place. A change to the OPOS major version level for a device class reflects significant interface enhancements, and may remove support for obsolete interfaces from previous major version levels.
Minor	The “thousands” place. A change to the OPOS minor version level for a device class reflects minor interface enhancements, and must provide a superset of previous interfaces at this major version level.
Build	The “units” place. Internal level provided by the Service Object developer. Updated when corrections are made to the SO implementation.

A sample version number is:

1002038

This value may be displayed as version “1.2.38”, and interpreted as major version 1, minor version 2, build 38 of the Service Object.

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

State Property

Syntax **LONG State;**

Remarks Contains the current state of the Control.

Value	Meaning
OPOS_S_CLOSED	The Control is closed.
OPOS_S_IDLE	The Control is in a good state and is not busy.
OPOS_S_BUSY	The Control is in a good state and is busy performing output.
OPOS_S_ERROR	An error has been reported, and the application must recover the Control to a good state before normal I/O can resume.

This property is always readable.

See Also “Status, Result Code, and State Model”

Methods

CheckHealth Method

Syntax **LONG CheckHealth (LONG Level);**

The *Level* parameter indicates the type of health check to be performed on the device. The following values may be specified:

Value	Meaning
OPOS_CH_INTERNAL	Perform a health check that does not physically change the device. The device is tested by internal tests to the extent possible.
OPOS_CH_EXTERNAL	Perform a more thorough test that may change the device. For example, a pattern may be printed on the printer.
OPOS_CH_INTERACTIVE	Perform an interactive test of the device. The supporting Service Object will typically display a modal dialog box to present test options and results.

Remarks Called to test the state of a device.

A text description of the results of this method is placed in the **CheckHealthText** property.

The **CheckHealth** method is always synchronous.

Return One of the following values are returned by the method, and also placed in the **ResultCode** property.

Value	Meaning
OPOS_SUCCESS	Indicates that the health checking procedure was initiated properly and, when possible to determine, indicates that the device is healthy. However, the health of many devices can only be determined by a visual inspection of the test results.
OPOS_E_ILLEGAL	The specified health check level is not supported by the Service Object.
OPOS_E_BUSY	Cannot perform while output is in progress.
<i>Other Values</i>	See ResultCode .

See Also “General OLE for Retail POS Control Model”; **CheckHealthText** Property

Claim Method

Syntax **LONG Claim (LONG Timeout);**

The *Timeout* parameter gives the maximum number of milliseconds to wait for exclusive access to be satisfied. If zero, the method attempts to claim the device, and returns immediately.

Remarks Call this method to request exclusive access to the device. Many devices require an application to claim them before they can be used.

When successful, the **Claimed** property is changed to TRUE.

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

Value	Meaning
OPOS_SUCCESS	Exclusive access has been granted. The Claimed property is now TRUE. Also returned if this application has already claimed the device.
OPOS_E_ILLEGAL	This device cannot be claimed for exclusive access.
OPOS_E_TIMEOUT	Another application has exclusive access to the device, and did not relinquish control before <i>Timeout</i> milliseconds expired.

See Also “Device Sharing Model”; **Release** Method

ClearInput Method

Syntax **LONG ClearInput ();**

Remarks Called to clear all device input that has been buffered.

Any data events or input error events that were pending – usually waiting for **DataEventEnabled** to be set to TRUE and **FreezeEvents** to be set to FALSE – are also cleared.

Return The following value is returned by the method and placed in the **ResultCode** property.

Value	Meaning
OPOS_SUCCESS	Input has been cleared.

See Also “Input Model”

ClearOutput Method

Syntax **LONG ClearOutput ();**

Remarks Called to clear all device output that has been buffered. Also, when possible, halts outputs that are in progress.

Any output error events that were pending – usually waiting for **FreezeEvents** to be set to FALSE – are also cleared.

Return The following value is returned by the method and placed in the **ResultCode** property.

Value	Meaning
OPOS_SUCCESS	Output has been cleared.

See Also “Output Model”

Close Method

Syntax **LONG Close ();**

Remarks Called to release the device and its resources.

 If the **DeviceEnabled** property is TRUE, then the device is first disabled.

 If the **Claimed** property is TRUE, then exclusive access to the device is first released.

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

Value	Meaning
OPOS_SUCCESS	Device has been disabled and closed.
<i>Other Values</i>	See ResultCode .

See Also “General OLE for Retail POS Control Model”; **Open** Method

DirectIO Method

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

Parameter	Description
<i>Command</i>	Command number. Specific values assigned by the Service Object.
<i>pData</i>	Pointer to additional numeric data. Specific values vary by <i>CommandNumber</i> and Service Object.
<i>pString</i>	Pointer to additional string data. Specific values vary by <i>CommandNumber</i> and Service Object.

Remarks Call to communicate directly with the Service Object.

This method provides a means for a Service Object to provide functionality to the application that is not otherwise supported by the standard Control Object for its device class. Depending upon the Service Object's definition of the command, this method may be asynchronous or synchronous.

Use of **DirectIO** will make an application non-portable. The application may, however, maintain portability by performing **DirectIO** calls within conditional code. This code may be based upon the value of the **ServiceObjectDescription**, **DeviceDescription**, or **DeviceName** property.

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

Value	Meaning
OPOS_SUCCESS	Direct I/O successful.
<i>Other Values</i>	See ResultCode .

See Also **DirectIOEvent**

Open Method

Syntax **LONG Open (BSTR DeviceName);**

The *DeviceName* parameter specifies the device name to open.

Remarks Call to open a device for subsequent I/O.

The device name specifies which of one or more devices supported by this Control Object should be used. The *DeviceName* must exist in the system registry for this device class. The relationship between the device name and physical devices is determined by entries within the operating system registry; these entries are maintained by a setup or configuration utility. (See the appendix “OPOS Registry Usage”, page 301.)

When the **Open** method is successful, it sets the properties **Claimed**, **DeviceEnabled**, **DataEventEnabled**, and **FreezeEvents**, as well as descriptions and version numbers of the OPOS software layers. Additional class-specific properties may also be initialized.

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

Value	Meaning
OPOS_SUCCESS	Open successful.
OPOS_E_ILLEGAL	The Control is already open.
OPOS_E_NOEXIST	The specified <i>DeviceName</i> was not found.
OPOS_E_NOSERVICE	Could not establish a connection to the corresponding Service Object.
<i>Other Values</i>	See ResultCode .

See Also “General OLE for Retail POS Control Model”; **Close Method**

Release Method

Syntax **LONG Release ();**

Remarks Call this method to release exclusive access to the device.
If the **DeviceEnabled** property is TRUE, then the device is first disabled.

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

Value	Meaning
OPOS_SUCCESS	Exclusive access has been released. The Claimed property is now FALSE.
OPOS_E_ILLEGAL	The application does not have exclusive access to the device.

See Also “Device Sharing Model”; **Claim** Method

Events

DataEvent Event

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

The *Status* parameter contains the input status. Its value is Control-dependent, and may describe the type or qualities of the input.

Remarks Fired to present input data from the device to the application. The **DataEventEnabled** property is changed to FALSE, so that no further data events will be generated until the application sets this property back to TRUE. The actual input data is placed in one or more device-specific properties.

If **DataEventEnabled** is FALSE at the time that data is received, then the data is queued in an internal OPOS buffer, the device-specific input data properties are not updated, and the event is not fired. (When this property is subsequently changed back to TRUE, the event will be fired immediately if input data is queued and **FreezeEvents** is FALSE.)

See Also “Input Model”; **DataEventEnabled** Property; **FreezeEvents** Property

DirectIOEvent Event

Syntax **void DirectIOEvent (LONG *EventNumber*, LONG* *pData*, BSTR* *pString*);**

Parameter	Description
<i>EventNumber</i>	Event number. Specific values are assigned by the Service Object.
<i>pData</i>	Pointer to additional numeric data. Specific values vary by <i>EventNumber</i> and the Service Object.
<i>pString</i>	Pointer to additional string data. Specific values vary by <i>EventNumber</i> and the Service Object.

Remarks Fired by a Service Object to communicate directly with the application.

This event provides a means for a Service Object to provide events to the application that are not otherwise supported by the Control Object.

See Also **DirectIO** Method

ErrorEvent Event

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

Parameter	Description
<i>ResultCode</i>	Result code causing the error event. See ResultCode for values.
<i>ResultCodeExtended</i>	Extended result code causing the error event. See ResultCodeExtended for values.
<i>ErrorLocus</i>	Location of the error. See values below.
<i>pErrorResponse</i>	Pointer to the error event response. See values below.

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

Value	Meaning
OPOS_EL_OUTPUT	Error occurred while processing asynchronous output.
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.

The contents at the location pointed to by the *pErrorResponse* parameter are preset to a default value, based on the *ErrorLocus*. The application may change them to one of the following:

Value	Meaning
OPOS_ER_RETRY	Use only when locus is OPOS_EL_OUTPUT. Retry the asynchronous output. The error state is exited. Default when locus is OPOS_EL_OUTPUT.
OPOS_ER_CLEAR	Clear the asynchronous output or buffered input data. The error state is exited. Default when locus is OPOS_EL_INPUT.
OPOS_ER_CONTINUEINPUT	Use only when locus is OPOS_EL_INPUT_DATA. Acknowledges the error and directs the Control to

continue processing. The Control remains in the error state and will fire additional **DataEvents** as directed by the **DataEventEnabled** property. When all input has been fired and the **DataEventEnabled** property is again set to TRUE, then another **ErrorEvent** is fired with locus OPOS_EL_INPUT.

Default when locus is OPOS_EL_INPUT_DATA.

Remarks Fired when an error is detected and the Control's **State** transitions into the error state.

Input error events are not fired until the **DataEventEnabled** property is TRUE, so that proper application sequencing occurs.

See Also "Status, Result Code, and State Model"

OutputCompleteEvent Event

Syntax void **OutputCompleteEvent** (**LONG** *OutputID*);

The *OutputID* parameter indicates the ID number of the asynchronous output request that is complete.

Remarks Fired when a previously started asynchronous output request completes successfully.

See Also "Output Model"

StatusUpdateEvent Event

Syntax **void StatusUpdateEvent (LONG *Data*);**

The *Data* parameter is for device class-specific data, describing the type of status change.

Remarks Fired when a Control needs to alert the application of a device status change.

Examples are a change in the cash drawer position (open vs. closed) or a change in a POS printer sensor (form present vs. absent).

See Also “Status, Result Code, and State Model”

CHAPTER 2

Cash Drawer

Summary

Properties

<i>Common</i>	<i>Type</i>	<i>Access</i>	<i>Initialized After</i>
CheckHealthText	String	R	Open
Claimed	Boolean	R	Open
DataEventEnabled	Boolean	R/W	<i>Not Supported</i>
DeviceEnabled	Boolean	R/W	Open
FreezeEvents	Boolean	R/W	Open
OutputID	Long	R	<i>Not Supported</i>
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
<i>Specific</i>			
CapStatus	Boolean	R	Open
DrawerOpened	Boolean	R	Open & Enable

Methods

<i>Common</i>	<i>May Use After</i>
Open	--
Close	Open
Claim	Open
Release	Open & Claim
CheckHealth	Open & Enable; <i>Note</i>
ClearInput	<i>Not Supported</i>
ClearOutput	<i>Not Supported</i>
DirectIO	Open
 <i>Specific</i>	
OpenDrawer	Open & Enable; <i>Note</i>
WaitForDrawerClose	Open & Enable; <i>Note</i>

Note: Also requires that no other application has claimed the cash drawer.

Events

<i>Name</i>	<i>May Occur After</i>
DataEvent	<i>Not Supported</i>
DirectIOEvent	Open
ErrorEvent	<i>Not Supported</i>
OutputCompleteEvent	<i>Not Supported</i>
StatusUpdateEvent	Open & Enable

General Information

The Cash Drawer Control's OLE programmatic ID is "OPOS.CashDrawer".

Capabilities

The Cash Drawer Control has the following capability:

- Supports a command to "open" the cash drawer.

The cash drawer may have the following additional capability:

- Drawer status reporting: Can determine whether the drawer is open or closed.

Device Sharing

The cash drawer is a sharable device. Its device sharing rules are:

- After opening and enabling the device, the application may access all properties and methods and will receive status update events.
- If more than one application has opened and enabled the device, all applications may access its properties and methods. Status update events are fired to all of the applications.
- If one application claims the cash drawer, then only that application may call the **OpenDrawer** and **WaitForDrawerClosed** methods. This feature provides a degree of security, such that these methods may effectively be restricted to the main POS application if that application claims the device at startup.
- See the "Summary" table for precise usage prerequisites.

Properties

CapStatus Property

Syntax **BOOL CapStatus;**

Remarks If TRUE, the drawer can report status.
 If FALSE, the drawer is not able to determine whether cash drawer is open or closed.

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

DrawerOpened Property

Syntax **BOOL DrawerOpened;**

Remarks If TRUE, the drawer is open.
 If FALSE, the drawer is closed.

 If the capability **CapStatus** is FALSE, then the device does not support status reporting, and **DrawerOpened** is always FALSE.

 This property is initialized and kept current while the device is enabled.

Methods

OpenDrawer Method

- Syntax** **LONG OpenDrawer ()**
- Remarks** Call to open the drawer.
- Return** One of the following values is returned by the method and placed in the **ResultCode** property:

Value	Meaning
OPOS_SUCCESS	The drawer was opened successfully.
<i>Other Values</i>	See ResultCode .

WaitForDrawerClose Method

Syntax **LONG WaitForDrawerClose (LONG *BeepTimeout*, LONG *BeepFrequency*, LONG *BeepDuration*, LONG *BeepDelay*);**

Parameter	Description
<i>BeepTimeout</i>	Number of milliseconds to wait before starting an alert beeper.
<i>BeepFrequency</i>	Audio frequency of the alert beeper in hertz.
<i>BeepDuration</i>	Number of milliseconds that the beep tone will be sounded.
<i>BeepDelay</i>	Number of milliseconds between the sounding of beeper tones.

Remarks Call to wait until the cash drawer is closed. If the drawer is still open after *BeepTimeout* milliseconds, then the system alert beeper is started.

Unless an error occurs, this method will not return to the application while the drawer is open. When the cashier closes the drawer, the beeper is turned off.

If the capability **CapStatus** is FALSE, then the device does not support status reporting, and this method will return immediately with a successful status.

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

Value	Meaning
OPOS_SUCCESS	The drawer was properly closed.
<i>Other Values</i>	See ResultCode .

Events

StatusUpdateEvent Event

Syntax **void StatusUpdateEvent (LONG Data);**

The *Data* parameter contains the updated drawer status.
If TRUE, the drawer is open. If FALSE, the drawer is closed.

Remarks Fired when the open status of the drawer changes.

If the capability **CapStatus** is FALSE, then the device does not support status reporting, and this event will never be fired.

CHAPTER 3

Coin Dispenser

Summary

Properties

<i>Common</i>	<i>Type</i>	<i>Access</i>	<i>Initialized After</i>
CheckHealthText	String	R	Open
Claimed	Boolean	R	Open
DataEventEnabled	Boolean	R/W	<i>Not Supported</i>
DeviceEnabled	Boolean	R/W	Open & Claim
FreezeEvents	Boolean	R/W	Open
OutputID	Long	R	<i>Not Supported</i>
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
<i>Specific</i>			
CapEmptySensor	Boolean	R	Open
CapJamSensor	Boolean	R	Open
CapNearEmptySensor	Boolean	R	Open
DispenserStatus	Long	R	Open, Claim, & Enable

Methods

<i>Common</i>	<i>May Use After</i>
Open	--
Close	Open
Claim	Open
Release	Open & Claim
CheckHealth	Open, Claim, & Enable
ClearInput	<i>Not Supported</i>
ClearOutput	<i>Not Supported</i>
DirectIO	Open
<i>Specific</i>	
DispenseChange	Open, Claim, & Enable

Events

<i>Name</i>	<i>May Occur After</i>
DataEvent	<i>Not Supported</i>
DirectIOEvent	Open
ErrorEvent	<i>Not Supported</i>
OutputCompleteEvent	<i>Not Supported</i>
StatusUpdateEvent	Open, Claim, & Enable

General Information

The Coin Dispenser Control's OLE programmatic ID is "OPOS.CoinDispenser".

Capabilities

The coin dispenser has the following capability:

- Supports a method that allows a specified amount of change to be dispensed from the device.

The coin dispenser may have the following additional capability:

- Coin dispenser status reporting, which indicates empty coin slot conditions, near empty coin slot conditions, and coin slot jamming conditions.

Model

The general model of a coin dispenser is:

- A coin dispenser consists of a number of coin slots which hold the coinage to be dispensed. The programmer using the Coin Dispenser Control is not concerned with controlling the individual slots of coinage, but rather calls a method with the amount of change to be dispensed. It is the responsibility of the coin dispenser device or the Control to dispense the proper amount of change from the various slots.

Device Sharing

The coin dispenser 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 some of the properties, dispensing change, or receiving status update events.
- See the "Summary" table for precise usage prerequisites.

Properties

CapEmptySensor Property

Syntax **BOOL CapEmptySensor;**

Remarks If TRUE, the coin dispenser can report an out-of-coinage condition; otherwise it is FALSE.

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

CapJamSensor Property

Syntax **BOOL CapJamSensor;**

Remarks If TRUE, the coin dispenser can report a mechanical jam or failure condition; otherwise it is FALSE.

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

CapNearEmptySensor Property

Syntax **BOOL CapNearEmptySensor;**

Remarks If TRUE, the coin dispenser can report when it is almost out of coinage; otherwise it is FALSE.

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

DispenserStatus Property

Syntax **LONG DispenserStatus;**

Remarks Holds the current status of the dispenser. It may be one of the following:

Value	Meaning
COIN_STATUS_OK	Ready to dispense coinage. This value is also set when the dispenser is unable to detect an error condition.
COIN_STATUS_EMPTY	Cannot dispense coinage because it is empty.
COIN_STATUS_NEAREMPTY	Can still dispense coinage, but it nearly empty.
COIN_STATUS_JAM	A mechanical fault has occurred.

This property is initialized and kept current while the device is enabled.

Methods

DispenseChange Method

- Syntax** **LONG DispenseChange (LONG Amount);**
- The *Amount* parameter contains the amount of change to be dispensed.
- Remarks** Call to dispense change. The value represented by the *Amount* parameter is a count of the currency units to dispense (such as cents or yen).
- Return** One of the following values is returned by the method and placed in the **ResultCode** property:

Value	Meaning
OPOS_SUCCESS	The specified change was dispensed successfully.
OPOS_E_ILLEGAL	An <i>Amount</i> parameter value of zero was specified, or the <i>Amount</i> parameter contained a negative value or a value greater than the device can dispense.
<i>Other Values</i>	See ResultCode .

Events

StatusUpdateEvent (Common)

Syntax **void StatusUpdateEvent (LONG Data);**

The *Data* parameter contains the coin dispenser status condition:

Value	Meaning
COIN_STATUS_OK	Ready to dispense coinage. This value is also set when the dispenser is unable to detect an error condition.
COIN_STATUS_EMPTY	Cannot dispense coinage because it is empty.
COIN_STATUS_NEAREMPTY	Can still dispense coinage, but is nearly empty.
COIN_STATUS_JAM	A mechanical fault has occurred.

Remarks Fired when a coin dispenser sensor indicates a status change.

The coin dispenser is only able to fire status event changes for the sensor types supported by the values described in the capabilities properties.

CHAPTER 4

Hard Totals

Summary

Properties

<i>Common</i>	<i>Type</i>	<i>Access</i>	<i>Initialized After</i>
CheckHealthText	String	R	Open
Claimed	Boolean	R	Open
DataEventEnabled	Boolean	R/W	<i>Not Supported</i>
DeviceEnabled	Boolean	R/W	Open
FreezeEvents	Boolean	R/W	Open
OutputID	Long	R	<i>Not Supported</i>
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

<i>Specific</i>	<i>Type</i>	<i>Access</i>	<i>Initialized After</i>
CapErrorDetection	Boolean	R	Open
CapSingleFile	Boolean	R	Open
CapTransactions	Boolean	R	Open
FreeData	Long	R	Open & Enable
TotalsSize	Long	R	Open & Enable
NumberOfFiles	Long	R	Open & Enable
TransactionInProgress	Boolean	R	Open

Methods

<i>Common</i>	<i>May Use After</i>
Open	--
Close	Open
Claim	Open
Release	Open & Claim
CheckHealth	Open & Enable; <i>Note 1</i>
ClearInput	<i>Not Supported</i>
ClearOutput	<i>Not Supported</i>
DirectIO	Open
 <i>Specific</i>	
ClaimFile	Open & Enable; <i>Note 2</i>
ReleaseFile	Open & Enable
Read	Open & Enable; <i>Note 2</i>
Write	Open & Enable; <i>Note 2</i>
SetAll	Open & Enable; <i>Note 2</i>
ValidateData	Open & Enable; <i>Note 2</i>
RecalculateValidationData	Open & Enable; <i>Note 2</i>
Create	Open & Enable; <i>Note 1</i>
Find	Open & Enable; <i>Note 1</i>
FindByIndex	Open & Enable; <i>Note 1</i>
Delete	Open & Enable; <i>Note 2</i>
Rename	Open & Enable; <i>Note 2</i>
BeginTrans	Open & Enable
CommitTrans	Open & Enable
Rollback	Open & Enable

Note 1: Also requires that no other application has claimed the hard totals device.

Note 2: Also requires that no other application has claimed the hard totals device or the file on which this method acts.

Events

<i>Name</i>	<i>May Occur After</i>
DataUpdateEvent	<i>Not Supported</i>
DirectIOEvent	Open
ErrorEvent	<i>Not Supported</i>
OutputCompleteEvent	<i>Not Supported</i>
StatusUpdateEvent	<i>Not Supported</i>

General Information

The Hard Totals Control's OLE programmatic ID is "OPOS.Totals".

Capabilities

The Hard Totals device has the following minimal set of capabilities:

- Supports at least one totals file with the name "" (the empty string) in an area of totals memory. Each totals file is read and written as if it were a sequence of byte data.
- Each totals file is created with a fixed size and may be deleted, initialized, and claimed for exclusive use.
- Totals memory is frequently a limited but secure resource - perhaps of only several thousand bytes of storage.

The Hard Totals device may have the following additional capabilities:

- Supports additional named totals files. They share some characteristics of a file system with only a root directory level. In addition to the minimal capabilities listed above, each totals file may also be renamed.
- Supports transactions, with begin and commit operations, plus rollback.
- Supports advanced error detection. This detection may be implemented through hardware or software.

Model

The following is the general model of the Hard Totals:

- A Hard Totals device is logically treated as a sequence of byte data, which the application subdivides into “totals files.” This is done by the **Create** method, which assigns a name, size, and error detection level to the totals file. Totals files have a fixed-length that is set at **Create** time.

At a minimum, a single totals file with the name “” (the empty string) can be created and manipulated. Optionally, additional totals files with arbitrary names may be created.

Totals files model many of the characteristics of a traditional file system. The intent, however, is not to provide a robust file system. Rather, totals files allow partitioning and ease of access into what is frequently a limited but secure resource. In order to reduce unnecessary overhead usage of this resource, directory hierarchies are not supported, file attributes are minimized, and files may not be dynamically resized.

- The following operations may be performed on a totals file:
 - ◆ **Read**: Read a series of data bytes.
 - ◆ **Write**: Write a series of data bytes.
 - ◆ **ClearAll**: Set all the data in a totals file to zero.
 - ◆ **Find**: Locate an existing totals file by name, and return a file handle and size.
 - ◆ **FindByIndex**: Used to enumerate all of the files in the Hard Totals area.
 - ◆ **Delete**: Delete a totals file by name.
 - ◆ **Rename**: Rename an existing totals file.
 - ◆ **ClaimFile**: Gain exclusive access to a specific file for use by the claiming application. A timeout value may be specified in case another application maintains access for a period a time.
The common **Claim** method may also be used to claim the entire Hard Totals device.
 - ◆ **ReleaseFile**: Releases exclusive access to the file.
- The **FreeData** property holds the current number of unassigned data bytes.
- The **TotalsSize** property holds the totals memory size.

- The **NumberOfFiles** property holds the number of totals files that exist in the hard totals device.
- Transaction operations are optionally supported. A transaction is defined as a series of data writes to be applied as an atomic operation to one or more Hard Totals files.

During a transaction, data writes will typically be maintained in memory until a commit or rollback. Also **FreeData** will typically be reduced during a transaction to ensure that the commit has temporary totals space to perform the commit as an atomic operation.

- ◆ **BeginTrans**: Marks the beginning of a transaction.
- ◆ **CommitTrans**: Ends the current transaction, and saves the updated data. Software and/or hardware methods are used to ensure that either the entire transaction is saved, or that none of the updates are applied.

This will typically require writing the transaction to temporary totals space, setting state information within the device indicating that a commit is in progress, writing the data to the totals files, and freeing the temporary totals space. If the commit is interrupted, perhaps due to a system power loss or reset, then when the Hard Totals service object is reloaded and initialized, it can complete the commit by copying data from the temporary space into the totals files. This ensures the integrity of related totals data.

- ◆ **Rollback**: Ends the current transaction, and discards the updates. This may be useful in case of user intervention to cancel an update. Also, if advanced error detection shows that some totals data cannot be read properly in preparation for an update, then the transaction may need to be aborted.
- ◆ **TransactionInProgress**: This property holds the current state of transactions.

The application should **Claim** the files used during a transaction so that no other Hard Totals Control claims a file before **CommitTrans**, causing the commit to fail, returning an already claimed status.

- Advanced error detection is optionally supported by the following:
 - ◆ A **Read** or a **Write** may report a validation error. Data is usually divided into validation blocks, over which sumchecks or CRCs are maintained. The size of validation data blocks is determined by the specific Service Object.

A validation error informs the application that one or more of the validation blocks containing the data to be read or written may be invalid due to a hardware error. (An error on a **Write** can occur when only a portion of a validation block must be changed. The validation block must be read and the block validated before the portion is changed.)

When a validation error is reported, it is recommended that the application read all of the data in the totals file. The application will want to determine which portions of data are invalid, and take action based on the results of the reads.
 - ◆ **RecalculateValidationData** may be called to cause recalculation of all validation data within a totals file. This may be called after recovery has been performed as in the previous paragraph.
 - ◆ **ValidateData** may be called to verify that all data within a totals file passes validation.
 - ◆ Data **Writes** automatically cause recalculation of validation data for the validation block or blocks in which the written data resides.
 - ◆ Since advanced error detection usually imposes a performance penalty, the application may choose to select this feature when each totals file is created.

Device Sharing

The hard totals device is sharable. Its device sharing rules are:

- After opening the device, most properties are readable.
- After opening and enabling the device, the application may access all properties and methods.
- If more than one application has opened and enabled the device, all applications may access its properties and methods.
- One application may claim the hard totals device. This restricts all other applications from reading, changing, or claiming any files on the device.
- One application may claim a hard totals file. This restricts all other applications from reading, changing, or claiming the file, and from claiming the hard totals device.

Properties

CapErrorDetection Property

- Syntax** **BOOL CapErrorDetection;**
- Remarks** If TRUE, then advanced error detection is supported;
otherwise it is FALSE.
- This property is initialized by the **Open** method.

CapSingleFile Property

- Syntax** **BOOL CapSingleFile;**
- Remarks** If TRUE, then only a single file, identified by the empty string (“”), is supported;
otherwise it is FALSE.
- This property is initialized by the **Open** method.

CapTransactions Property

- Syntax** **BOOL CapTransactions;**
- Remarks** If TRUE, then transactions are supported;
otherwise it is FALSE.
- This property is initialized by the **Open** method.

FreeData Property

Syntax **LONG FreeData;**

Remarks Holds the number of bytes of unallocated data in the Hard Totals device.

Its value is initialized to an appropriate value when the device is enabled and is updated as files are **Created** and **Deleted**. If creating a file requires some overhead to support the file information, then **FreeData** is reduced by this overhead amount. This guarantees that a new file of size **FreeData** may be created.

Data writes within a transaction may temporarily reduce **FreeData**, since some Hard Totals space may need to be allocated to prepare for the transaction commit. Therefore, the application should ensure that sufficient **FreeData** is maintained to allow its maximally sized transactions to be performed.

See Also **Create Method; Write Method**

NumberOfFiles Property

Syntax **LONG NumberOfFiles;**

Remarks Holds the number of totals file currently in the Hard Totals device.

This property is initialized and kept current while the device is enabled.

See Also **FreeData Property**

TotalsSize Property

Syntax **LONG TotalsSize;**

Remarks Holds the size of the Hard Totals area. This size is equal to the largest totals file that can be created if no other files exist.

This property is initialized when the device is enabled.

See Also **FreeData Property**

TransactionInProgress Property

Syntax **BOOL TransactionInProgress;**

Remarks If TRUE, then the application is within a transaction;
 otherwise it is FALSE.

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

See Also **BeginTrans** Method

Methods

BeginTrans Method

- Syntax** **LONG BeginTrans ();**
- Remarks** Marks the beginning of a series of Hard Totals writes that must either be applied as a group or not at all.
- 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	Transactions are not supported by this device.
<i>Other Values</i>	See ResultCode .

- See Also** **CommitTrans** Method; **Rollback** Method

Claim Method (Common)

Syntax **LONG Claim (LONG Timeout);**

The *Timeout* parameter gives the maximum number of milliseconds to wait for exclusive access to be satisfied. If zero, the method attempts to claim the device and returns immediately.

Remarks Call this method to request exclusive access to the device.

If any other application has claimed exclusive access to any of the hard totals files by using **ClaimFile**, then this **Claim** cannot be satisfied until those files are released by **ReleaseFile**.

When successful, the **Claimed** property is changed to TRUE.

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

Value	Meaning
OPOS_SUCCESS	Exclusive access has been granted. The Claimed property is now TRUE. Also returned if this application has already claimed the device.
OPOS_E_TIMEOUT	Another application has exclusive access to the device or one or more of its files and did not relinquish control before <i>Timeout</i> milliseconds expired.

See Also “Device Sharing Model”; **Release** Method; **ClaimFile** Method; **ReleaseFile** Method

ClaimFile Method

Syntax **LONG ClaimFile (LONG HTotalsFile, LONG Timeout);**

Parameter	Description
<i>HTotalsFile</i>	Handle to the totals file that is to be claimed.
<i>Timeout</i>	The time in milliseconds to wait for the file to become available.

Remarks Attempts to gain exclusive access to a specific file for use by the claiming application. Once granted, the application maintains exclusive access until it explicitly releases access or until the device is closed.

If any other applications have claimed exclusive access to this file by using **ClaimFile**, or if an application has claimed exclusive access to the entire totals area by using **Claim**, then this **ClaimFile** cannot be satisfied until those claims have been released.

All claims are released when the application calls the **Close** method.

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 handle is invalid.
OPOS_E_TIMEOUT	The <i>Timeout</i> value expired before another application released exclusive access of either the requested totals file or the entire totals area.

See Also **Claim** Method; **ReleaseFile** Method

CommitTrans Method

Syntax **LONG CommitTrans ();**

Remarks Ends the current transaction. All writes between the previous **BeginTrans** method and this method are saved to the Hard Totals areas.

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	Transactions are not supported by this device, or no transaction is in progress.
<i>Other Values</i>	See ResultCode .

See Also **BeginTrans** Method; **Rollback** Method

Create Method

Syntax **LONG Create** (**BSTR** *FileName*, **LONG*** *pHTotalsFile*, **LONG** *Size*, **BOOL** *ErrorDetection*);

Parameter	Description
<i>FileName</i>	The name to be assigned to the file. Must be no longer than 10 characters. All displayable characters – characters ≥ 20-hex – are valid.
<i>pHTotalsFile</i>	Pointer to the handle of the newly created totals file. Set by the method.
<i>Size</i>	The length of the file in bytes. Once created, the file size cannot be changed – totals files are fixed-length files.
<i>ErrorDetection</i>	The level of error detection desired for this file: If TRUE, then the Service Object will enable advanced error detection if supported. If FALSE, then higher performance access is required, so advanced error detection need not be enabled for this file.

Remarks Creates a totals file with the specified name, size, and error detection level. The data area is initialized to binary zeros.

If **CapSingleFile** is TRUE, then only one file may be created, and its name must be the empty string (“”). Otherwise, the number of totals files that may be created is limited only by the free space available in the Hard Totals area.

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_CLAIMED	Cannot create because the entire totals file area is claimed by another application.
OPOS_E_ILLEGAL	The <i>FileName</i> is too long or contains invalid characters.
OPOE_E_EXISTS	<i>FileName</i> already exists.
OPOS_E_EXTENDED	ResultCodeExtended = OPOS_ETOT_NOROOM: There is insufficient room in the totals area to create the file.
<i>Other Values</i>	See ResultCode .

See Also **Find** Method; **Delete** Method; **Rename** Method

Delete Method

Syntax **LONG Delete (BSTR *FileName*);**

The *FileName* parameter specifies the totals file to be deleted.

Remarks Delete the named file.

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_CLAIMED	Cannot delete because either the totals file or the entire totals area is claimed by another application.
OPOS_E_ILLEGAL	The <i>FileName</i> is too long or contains invalid characters.
OPOE_E_NOEXIST	<i>FileName</i> was not found.
<i>Other Values</i>	See ResultCode .

See Also **Create** Method; **Find** Method; **Rename** Method

Find Method

Syntax **LONG Find (BSTR *FileName*, LONG* *pHTotalsFile*, LONG* *pSize*);**

Parameter	Description
<i>FileName</i>	The totals file name to be located.
<i>pHTotalsFile</i>	Pointer to the handle of the totals file. Set by the method.
<i>pSize</i>	Pointer to the length of the file in bytes. Set by the method.

Remarks Locates an existing totals file.

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_CLAIMED	Cannot find because the entire totals file area is claimed by another application.
OPOS_E_ILLEGAL	The <i>FileName</i> is too long or contains invalid characters.
OPOE_E_NOEXIST	<i>FileName</i> was not found.
<i>Other Values</i>	See ResultCode .

See Also **Create Method**; **Delete Method**; **Rename Method**

FindByIndex Method

Syntax **LONG FindByIndex (LONG *Index*, BSTR* *pFileName*);**

Parameter	Description
<i>Index</i>	The index of the totals file name to be found.
<i>pFileName</i>	Pointer to the totals file name to be returned. Set by the method.

Remarks Returns the totals file name currently associated with the given index.

This method provides a means for enumerating all of the totals files currently defined. An *Index* of zero will return the file name at the first file position, with subsequent indices returning additional file names. The largest valid *Index* value is one less than **NumberOfFiles**.

The creation and deletion of files may change the relationship between indices and the file names, as the Control may compact or rearrange the data areas used to manage file names and attributes at these times. Therefore, the application may need to **Claim** the device to ensure that all file names are retrieved successfully.

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_CLAIMED	Cannot find because the entire totals file area is claimed by another application.
OPOS_E_ILLEGAL	The <i>Index</i> is greater than the largest file index that is currently defined.
<i>Other Values</i>	See ResultCode .

See Also **Create Method; Find Method**

Read Method

Syntax **LONG Read (LONG HTotalsFile, BSTR* pData, LONG Offset, LONG Count);**

Parameter	Description
<i>HTotalsFile</i>	Totals file handle returned from a Create or Find method.
<i>pData</i>	Pointer to the data buffer in which the totals data will be placed.
<i>Offset</i>	Starting offset for the data to be read.
<i>Count</i>	Number of bytes of data to read.

Remarks Read data from a totals file.

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_CLAIMED	Cannot read because either the totals file or the entire totals area is claimed by another application.
OPOS_E_ILLEGAL	The handle is invalid, or part of the data range is outside the bounds of the totals file.
OPOS_E_EXTENDED	ResultCodeExtended = OPOS_ETOT_VALIDATION: A validation error has occurred while reading data.
<i>Other Values</i>	See ResultCode .

See Also **Write Method**

RecalculateValidationData Method

Syntax **LONG RecalculateValidationData (LONG HTotalsFile);**

The *HTotalsFile* parameter contains the handle of a totals file.

Remarks Recalculates validation data for the specified totals file.

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_CLAIMED	Cannot recalculate because either the totals file or the entire totals area is claimed by another application.
OPOS_E_ILLEGAL	The handle is invalid, or advanced error detection is either not supported by the ServiceObject or by this file.
<i>Other Values</i>	See ResultCode .

Release Method (Common)

Syntax **LONG Release ();**

Remarks Call this method to release exclusive access to the device.

An application may own claims on both the Hard Totals device through **Claim** as well as individual files through **ClaimFile**. Calling **Release** only releases the claim on the Hard Totals device.

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

Value	Meaning
OPOS_SUCCESS	Exclusive access has been released. The Claimed property is now FALSE.
OPOS_E_ILLEGAL	The application does not have exclusive access to the device.

See Also “Device Sharing Model”; **Claim** Method; **ClaimFile** Method

ReleaseFile Method

Syntax **LONG ReleaseFile (LONG HTotalsFile);**

The *HTotalsFile* parameter contains the handle of the totals file to be released.

Remarks Releases exclusive access to a specific file.

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 handle is invalid, or the specified file is not claimed by this application.

See Also **Claim** Method; **ClaimFile** Method

Rename Method

Syntax **LONG Rename (LONG *HTotalsFile*, BSTR *FileName*);**

Parameter	Description
<i>HTotalsFile</i>	Handle of the totals file to be renamed.
<i>FileName</i>	The new name to be assigned to the file. The name must be no longer than 10 characters. All displayable characters – characters ≥ 20-hex – are valid.

Remarks Renames a totals file.

If **CapSingleFile** is TRUE, then this method will fail.

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_CLAIMED	Cannot rename because either the totals file or the entire totals area is claimed by another application.
OPOS_E_ILLEGAL	The file handle is invalid, the <i>FileName</i> is too long or contains invalid characters, or the CapSingleFile property is TRUE.
OPOE_E_EXISTS	<i>FileName</i> already exists.
<i>Other Values</i>	See ResultCode .

Rollback Method

Syntax	LONG Rollback ();								
Remarks	Ends the current transaction. All writes between the previous BeginTrans and this method are discarded; they are not saved to the Hard Totals areas.								
Return	One of the following values is returned by the method and placed in the ResultCode property:								
	<table> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>OPOS_SUCCESS</td> <td>The method was successful.</td> </tr> <tr> <td>OPOS_E_ILLEGAL</td> <td>Transactions are not supported by this device, or no transaction is in progress.</td> </tr> <tr> <td><i>Other Values</i></td> <td>See ResultCode.</td> </tr> </tbody> </table>	Value	Meaning	OPOS_SUCCESS	The method was successful.	OPOS_E_ILLEGAL	Transactions are not supported by this device, or no transaction is in progress.	<i>Other Values</i>	See ResultCode .
Value	Meaning								
OPOS_SUCCESS	The method was successful.								
OPOS_E_ILLEGAL	Transactions are not supported by this device, or no transaction is in progress.								
<i>Other Values</i>	See ResultCode .								
See Also	BeginTrans Method; CommitTrans Method								

SetAll Method

Syntax	LONG SetAll (LONG HTotalsFile, LONG Value);								
	<table> <thead> <tr> <th>Parameter</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td><i>HTotalsFile</i></td> <td>Handle of a totals file.</td> </tr> <tr> <td><i>Value</i></td> <td>Value to set is in the low byte.</td> </tr> </tbody> </table>	Parameter	Description	<i>HTotalsFile</i>	Handle of a totals file.	<i>Value</i>	Value to set is in the low byte.		
Parameter	Description								
<i>HTotalsFile</i>	Handle of a totals file.								
<i>Value</i>	Value to set is in the low byte.								
Remarks	Set all the data in a totals file to the specified value.								
Return	One of the following values is returned by the method and placed in the ResultCode property:								
	<table> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>OPOS_SUCCESS</td> <td>The method was successful.</td> </tr> <tr> <td>OPOS_E_CLAIMED</td> <td>Cannot set because either the totals file or the entire totals area is claimed by another application.</td> </tr> <tr> <td><i>Other Values</i></td> <td>See ResultCode.</td> </tr> </tbody> </table>	Value	Meaning	OPOS_SUCCESS	The method was successful.	OPOS_E_CLAIMED	Cannot set because either the totals file or the entire totals area is claimed by another application.	<i>Other Values</i>	See ResultCode .
Value	Meaning								
OPOS_SUCCESS	The method was successful.								
OPOS_E_CLAIMED	Cannot set because either the totals file or the entire totals area is claimed by another application.								
<i>Other Values</i>	See ResultCode .								

ValidateData Method

Syntax **LONG ValidateData (LONG HTotalsFile);**

The *HTotalsFile* parameter contains the handle of a totals file.

Remarks Verifies that all data in the specified totals file passes validation checks.

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_CLAIMED	Cannot validate because either the totals file or the entire totals area is claimed by another application.
OPOS_E_ILLEGAL	The handle is invalid, or advanced error detection is either not supported by the ServiceObject or by this file.
<i>Other Values</i>	See ResultCode .

Write Method

Syntax **LONG Write (LONG HTotalsFile, BSTR Data, LONG Offset, LONG Count);**

Parameter	Description
<i>HTotalsFile</i>	Totals file handle returned from a Create or Find method.
<i>Data</i>	Data buffer containing the totals data to be written.
<i>Offset</i>	Starting offset for the data to be written.
<i>Count</i>	Number of bytes of data to write.

Remarks Write data to a totals file.

If a transaction is in progress, then the write will be buffered until a **CommitTrans** or **Rollback** method is called.

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_CLAIMED	Cannot write because either the totals file or the entire totals area is claimed by another application.
OPOS_E_ILLEGAL	The handle is invalid, or part of all of the data range is outside the bounds of the totals file.
OPOS_E_EXTENDED	ResultCodeExtended = OPOS_ETOT_NOROOM: Cannot write because a transaction is in progress, and there is not enough free space to prepare for the transaction commit. ResultCodeExtended = OPOS_ETOT_VALIDATION: A validation error has occurred while reading data.
<i>Other Values</i>	See ResultCode .

See Also **Read** Method; **BeginTrans** Method; **CommitTrans** Method; **Rollback** Method; **FreeSpace** Property

CHAPTER 5

Keylock

Summary

Properties

<i>Common</i>	<i>Type</i>	<i>Access</i>	<i>Initialized After</i>
CheckHealthText	String	R	Open
Claimed	Boolean	R	Open
DataEventEnabled	Boolean	R/W	<i>Not Supported</i>
DeviceEnabled	Boolean	R/W	Open
FreezeEvents	Boolean	R/W	Open
OutputID	Long	R	<i>Not Supported</i>
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
<i>Specific</i>			
KeyPosition	Long	R	Open & Enable
PositionCount	Long	R	Open

Methods

<i>Common</i>	<i>May Use After</i>
Open	--
Close	Open
Claim	Open
Release	Open & Claim
CheckHealth	Open & Enable
ClearInput	<i>Not Supported</i>
ClearOutput	<i>Not Supported</i>
DirectIO	Open
 <i>Specific</i>	
WaitForKeylockChange	Open & Enable

Events

<i>Name</i>	<i>May Occur After</i>
DataEvent	<i>Not Supported</i>
DirectIOEvent	Open
ErrorEvent	<i>Not Supported</i>
OutputCompleteEvent	<i>Not Supported</i>
StatusUpdateEvent	Open & Enable

General Information

The Keylock Control's OLE programmatic ID is "OPOS.Keylock".

Capabilities

The keylock has the following minimal set of capabilities:

- Supports at least three keylock positions.
- Supports reporting of keylock position changes, either by hardware or software detection.

Model

The keylock defines three keylock positions as constants. It is assumed that the keylock supports locked, normal, and supervisor positions. The constants for these keylock positions and their values are as follows:

- LOCK_KP_LOCK 1
- LOCK_KP_NORM 2
- LOCK_KP_SUPR 3

The **KeyPosition** property holds the value of the keylock position where the values range from one (1) to the total number of keylock positions contained in the **PositionCount** property.

Device Sharing

The keylock is sharable. Its device sharing rules are:

- After opening and enabling the device, the application may access all properties and methods and will receive status update events.
- If more than one application has opened and enabled the device, each of these applications may access its properties and methods. Status update events are fired to all of these applications.
- The keylock may not be claimed for exclusive access. If an application calls **Claim**, the method always return OPOS_E_ILLEGAL.
- See the "Summary" table for precise usage prerequisites.

Properties

KeyPosition Property

Syntax **LONG KeyPosition;**

Remarks Holds a value which indicates the keylock position.

This value is set by the Control whenever the keylock position is changed. In addition to the application receiving the **StatusUpdateEvent**, this value is changed to reflect the new keylock position.

The **KeyPosition** property may hold one of the following values:

Value	Meaning
LOCK_KP_LOCK	Keylock is in the “locked” position. Value is one (1).
LOCK_KP_NORM	Keylock is in the “normal” position. Value is two (2).
LOCK_KP_SUPR	Keylock is in the “supervisor” position. Value is three (3).
<i>Other Values</i>	Keylock is in one of the auxiliary positions. This value may range from four (4) up to the total number of keylock positions indicated by the PositionCount property.

This property is initialized and kept current while the device is enabled.

PositionCount Property

Syntax **LONG PositionCount;**

Remarks Holds the total number of keylock positions.

Contains the total number of positions that are present on the keylock device.

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

Methods

WaitForKeylockChange Method

Syntax **LONG WaitForKeylockChange (LONG *KeyPosition*, LONG *Timeout*);**

Parameter	Description
<i>KeyPosition</i>	Requested keylock position. See values below.
<i>Timeout</i>	Maximum number of milliseconds to wait for the keylock before returning control back to the application.

The *KeyPosition* parameter may contain one of the following values:

Value	Meaning
LOCK_KP_ANY	Wait for any keylock position change. Value is zero (0).
LOCK_KP_LOCK	Wait for keylock position to be set to the “locked” position. Value is one (1).
LOCK_KP_NORM	Wait for keylock position to be set to the “normal” position. Value is two (2).
LOCK_KP_SUPR	Wait for keylock position to be set to the “supervisor” position. Value is three (3).
<i>Other Values</i>	Wait for keylock position to be set to one of the auxiliary positions. This value may range from four (4) up to the total number of keylock positions indicated by the PositionCount property.

Remarks Call to wait for a specified keylock position to be set.

If the keylock position specified by the *KeyPosition* parameter is the same as the current keylock position, then the method returns immediately.

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

Value	Meaning
OPOS_SUCCESS	The keylock is in the specified position. If <i>KeyPosition</i> is LOCK_KP_ANY, then the keylock position has changed.
OPOS_E_TIMEOUT	The timeout period expired before the requested keylock positioning occurred.

OPOS_E_ILLEGAL	An invalid parameter value was specified.
<i>Other Values</i>	See ResultCode .

Events

StatusUpdateEvent Event

Syntax **void StatusUpdateEvent (LONG Data);**

The *Data* parameter contains the updated keylock position. The following keylock position values may be set:

Value	Meaning
LOCK_KP_LOCK	Keylock is in the “locked” position. Value is one (1).
LOCK_KP_NORM	Keylock is in the “normal” position. Value is two (2).
LOCK_KP_SUPR	Keylock is in the “supervisor” position. Value is three (3).
<i>Other Values</i>	Keylock is in one of the auxiliary positions. This value may range from four (4) to the total number of keylock positions indicated by the PositionCount property.

Remarks Fired when the keylock position changes.

CHAPTER 6

Line Display

Summary

Properties

<i>Common</i>	<i>Type</i>	<i>Access</i>	<i>Initialized After</i>
CheckHealthText	String	R	Open
Claimed	Boolean	R	Open
DataEventEnabled	Boolean	R/W	<i>Not Supported</i>
DeviceEnabled	Boolean	R/W	Open & Claim
FreezeEvents	Boolean	R/W	Open
OutputID	Long	R	<i>Not Supported</i>
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

<i>Specific</i>	<i>Type</i>	<i>Access</i>	<i>Initialized After</i>
CapBlink	Long	R	Open
CapBrightness	Boolean	R	Open
CapCharacterSet	Long	R	Open
CapDescriptors	Boolean	R	Open
CapHMarquee	Boolean	R	Open
CapICharWait	Boolean	R	Open
CapVMarquee	Boolean	R	Open
DeviceWindows	Long	R	Open
DeviceRows	Long	R	Open
DeviceColumns	Long	R	Open
DeviceDescriptors	Long	R	Open
DeviceBrightness	Long	R/W	Open, Claim, & Enable
CharacterSet	Long	R/W	Open, Claim, & Enable
CharacterSetList	String	R	Open
CurrentWindow	Long	R/W	Open
Rows	Long	R	Open
Columns	Long	R	Open
CursorRow	Long	R/W	Open
CursorColumn	Long	R/W	Open
CursorUpdate	Boolean	R/W	Open
MarqueeType	Long	R/W	Open
MarqueeFormat	Long	R/W	Open
MarqueeUnitWait	Long	R/W	Open
MarqueeRepeatWait	Long	R/W	Open
InterCharacterWait	Long	R/W	Open

Methods*Common***Open****Close****Claim****Release****CheckHealth****ClearInput****ClearOutput****DirectIO***Specific***DisplayText****DisplayTextAt****ClearText****ScrollText****SetDescriptor****ClearDescriptors****CreateWindow****DestroyWindow****RefreshWindow***May Use After*

--

Open

Open

Open & Claim

Open, Claim, & Enable

*Not Supported**Not Supported*

Open

Open, Claim, & Enable

Open, Claim, & Enable

Open, Claim, & Enable

Open, Claim, & Enable

Open, Claim, & Enable

Open, Claim, & Enable

Open, Claim, & Enable

Open, Claim, & Enable

Open, Claim, & Enable

Events*Name***DataEvent****DirectIOEvent****ErrorEvent****OutputCompleteEvent****StatusUpdateEvent***May Occur After**Not Supported*

Open

*Not Supported**Not Supported**Not Supported*

General Information

The Line Display Control's OLE programmatic ID is "OPOS.LineDisplay".

Capabilities

The Line Display has the following capability:

- Supports text character display. The default mode (or perhaps only mode) of the display is character display output.

The line display may also have the following additional capabilities:

- Supports windowing with marquee-like scrolling of the window. The display may support vertical or horizontal marquees, or both.
- Supports a waiting period between displaying characters, for a teletype effect.
- Supports character-level or device-level blinking.
- Supports one or more descriptors. Descriptors are small indicators with a fixed label, and are typically used to indicate transaction states such as item, total, and change.
- Supports device brightness control, with one or more levels of device dimming. All devices support brightness levels of "normal" and "blank" (at least through software support), but some devices also support one or more levels of dimming.

The following capability is not addressed in this version of the OPOS specification:

- Support for graphical displays, where the line display is addressable by individual pixels or dots.

Model

The general model of a line display:

- Consists of one or more rows containing one or more columns of characters. The characters in the default character set will include at least one of the following, with a capability defining the character set:
 - ◆ The digits '0' through '9' plus space, minus ('-'), and period ('.').
 - ◆ The above set plus uppercase 'A' through 'Z.'
 - ◆ All ASCII characters from 0x20 through 0x7F, which includes space, digits, uppercase, lowercase, and some special characters.
- The rows and columns are numbered beginning with (0, 0) at the upper-left corner of the window.
- Window 0 is always defined as follows:
 - ◆ Its “viewport” — the portion of the display that is updated by the window — covers the entire display.
 - ◆ The size of the window matches the entire display.Therefore, window 0, which is also called the “device window”, maps directly onto the display.
- Additional windows may be created. A created window has the following characteristics:
 - ◆ Its viewport covers part or all of the display.
 - ◆ The window may either match the size of the viewport, or it may be larger than the viewport in either the horizontal or vertical direction. In the second case, marquee scrolling of the window can be set.
 - ◆ The window maintains its own values for rows and columns, current cursor row and column, cursor update flag, scroll type and format, and timers.
 - ◆ All viewports behave transparently. If two viewports overlap, then the last character displayed at a position by either of the windows will be visible.

Display Modes

- ***Immediate Mode***
In effect when **MarqueeType** is DISP_MT_NONE and **InterCharacterWait** is zero.

If the window is bigger than the viewport, then only those characters which map into the viewport will be seen.

- ***Teletype Mode***
In effect when **MarqueeType** is DISP_MT_NONE and **InterCharacterWait** is not zero.

DisplayText and **DisplayTextAt** requests are enqueued and processed in the order they are received. The **InterCharacterWait** timer specifies the time to wait between outputting each character. **InterCharacterWait** only applies to those characters within the viewport.

- ***Marquee Mode***
In effect when **MarqueeType** is not DISP_MT_NONE.

The window must be bigger than the viewport.

A marquee is typically initialized after entering **Marquee Init Mode** by setting **MarqueeType** to DISP_MT_INIT, then calling **ClearText**, **DisplayText**, and **DisplayTextAt** methods. Then, when **MarqueeType** is changed to an “on” value, **Marquee On Mode** is entered, and the marquee begins to be displayed in the viewport beginning at the start of the window (or end if the type is right or down).

When the mode is changed from Marquee On Mode to off, the marquee stops in place. A subsequent transition from back to Marquee On Mode continues from the current position.

When the mode is changed from Marquee On Mode to Marquee Init Mode, the marquee stops. Changes may be made to the window, then the window may be returned to Marquee On Mode to restart the marquee with the new data.

It is illegal to use **DisplayText**, **DisplayTextAt**, **ClearText**, **RefreshWindow**, and **ScrollText** unless in Marquee Init Mode or marquees are off.

Device Sharing

The line display 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 some properties or calling methods that update the device.
- See the “Summary” table for precise usage prerequisites.

Properties

CapBlink Property

Syntax **LONG CapBlink;**

Remarks Holds the character blink capability of the device. It may be one of the following:

Value	Meaning
DISP_CB_NOBLINK	Blinking is not supported. Value is 0.
DISP_CB_BLINKALL	Blinking is supported. The entire contents of the display are either blinking or in a steady state.
DISP_CB_BLINKEACH	Blinking is supported. Each character may be individually set to blink or to be in a steady state.

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

CapBrightness Property

Syntax **BOOL CapBrightness;**

Remarks If TRUE, the brightness control is supported; otherwise it is FALSE.

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

CapCharacterSet Property

Syntax **LONG CapCharacterSet;**

Remarks Holds the default character set capability. It may be one of the following:

Value	Meaning
DISP_CCS_NUMERIC	The default character set supports numeric data, plus space, minus, and period.
DISP_CCS_ALPHA	The default character set supports uppercase alphabetic plus numeric, space, minus, and period.
DISP_CCS_ASCII	The default character set supports all ASCII characters between 20-hex and 7F-hex.
DISP_CCS_KANA	The default character set supports partial code page 932, including ASCII characters 20-hex through 7F-hex and the Japanese Kana characters A1-hex through DF-hex, but excluding the Japanese Kanji characters.
DISP_CCS_KANJI	The default character set supports code page 932, including the Shift-JIS Kanji characters, Levels 1 and 2.

The default character set may contain a superset of these ranges. The initial **CharacterSet** property may be examined for additional information.

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

CapDescriptors Property

Syntax **BOOL CapDescriptors;**

Remarks If TRUE, then the display supports descriptors; otherwise it is FALSE.

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

CapHMarquee Property

Syntax **BOOL CapHMarquee;**

Remarks If TRUE, the display supports horizontal marquee windows;
otherwise it is FALSE.

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

CapICharWait Property

Syntax **BOOL CapICharWait;**

Remarks If TRUE, the display supports intercharacter wait;
otherwise it is FALSE.

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

CapVMarquee Property

Syntax **BOOL CapVMarquee;**

Remarks If TRUE, the display supports vertical marquee windows;
otherwise it is FALSE.

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

CharacterSet Property R/W**Syntax** LONG CharacterSet;**Remarks** Contains the character set for displaying characters.

It is one of the following ranges or values:

Value	Meaning
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	Code page; matches one of the standard values.
DISP_CS_ASCII	The ASCII character set, supporting the ASCII characters between 20-hex and 7F-hex. The value of this constant is 998.
DISP_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. This value is guaranteed to support at least the set of characters specified by the **CapCharacterSet** capability.

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.
<i>Other Values</i>	See ResultCode .

See Also **CharacterSetList** Property; **CapCharacterSet** Property

CharacterSetList Property

Syntax **BSTR CharacterSetList;**

Remarks 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 **CharacterSet** Property

Columns Property

Syntax **LONG Columns;**

Remarks Holds the number of columns for this window.

For window 0, **Columns** is the same as **DeviceColumns**.

For other windows, it may be less or greater than **DeviceColumns**.

This property is initialized to **DeviceColumns** by the **Open** method, and is updated when **CurrentWindow** is set and when **CreateWindow** or **DestroyWindow** are called.

See Also **Rows** Property

CurrentWindow Property R/W

Syntax **LONG CurrentWindow;**

Remarks Holds the current window to which text is displayed.

Several properties are associated with each window: **Rows**, **Columns**, **CursorRow**, **CursorColumn**, **CursorUpdate**, **MarqueeType**, **MarqueeUnitWait**, **MarqueeRepeatWait**, and **InterCharacterWait**.

When set, this property changes the current window and sets the associated properties to their values for this window.

Setting a window does not refresh its viewport. If this window and another window's viewports overlap, and the other window has changed the viewport, then **RefreshWindow** may be called to restore this window's viewport contents.

This property is initialized to zero – the device window – by the **Open** method, and is updated when **CreateWindow** or **DestroyWindow** are called.

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

Value	Meaning
OPOS_SUCCESS	The new current window was set successfully.
OPOS_E_ILLEGAL	The new current window value is not valid.

CursorColumn Property R/W

Syntax **LONG CursorColumn;**

Remarks Holds the column in the current window to which the next displayed character will be output.

Legal values range from (zero) through (**Columns**). (See **DisplayText** for a note on the interpretation of **CursorColumn = Columns**.)

This property is initialized to zero on the by the **Open** and **CreateWindow** methods, and is updated when **CurrentWindow** is set or **ClearText**, **DisplayTextAt**, or **DestroyWindow** is called. It is also updated when **DisplayText** is called if **CursorUpdate** is TRUE.

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

Value	Meaning
OPOS_SUCCESS	The cursor column was set successfully.
OPOS_E_ILLEGAL	An invalid cursor column value was used.

See Also **CursorRow** Property; **DisplayText** Method

CursorRow Property R/W

Syntax **LONG CursorRow;**

Remarks Holds the row in the current window to which the next displayed character will be output.

Legal values range from (zero) through (**Rows** - 1).

This property is initialized to zero by the **Open** and **CreateWindow** methods, and is updated when **CurrentWindow** is set or **ClearText**, **DisplayTextAt**, or **DestroyWindow** is called. It is also updated when **DisplayText** is called if **CursorUpdate** is TRUE.

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

Value	Meaning
OPOS_SUCCESS	The cursor row was set successfully.
OPOS_E_ILLEGAL	An invalid cursor row value was used.

See Also **CursorColumn** Property; **DisplayText** Method

CursorUpdate Property R/W

Syntax **BOOL CursorUpdate;**

Remarks If TRUE when characters are displayed by the **DisplayText** or **DisplayTextAt** method, then **CursorRow** and **CursorColumn** will be updated to point to the character beyond the last character output.

If FALSE when characters are displayed, then the cursor properties will not be updated.

This property is maintained fore each window. It initialized to TRUE by the **Open** and **CreateWindow** methods, and is updated when **CurrentWindow** is set or **DestroyWindow** is called.

Return When this property is set, the following value is placed in the **ResultCode** property:

Value	Meaning
OPOS_SUCCESS	The property was set successfully.

See Also **CursorRow** Property; **CursorColumn** Property

DeviceBrightness Property R/W

Syntax **LONG DeviceBrightness;**

Remarks Holds the device brightness value, expressed as a percentage between 0 and 100.

Any device can support 0% (blank) and 100% (full intensity). Blanking can, at a minimum, be supported by sending spaces to the device. If the capability **CapBrightness** is TRUE, then the device also supports one or more levels of dimming.

If a device does not support the specified brightness value, then the Service Object will choose an appropriate substitute.

This property is initialized to 100 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.
OPOS_E_ILLEGAL	An invalid property value was used: Not in the range 0 through 100.

DeviceColumns Property

Syntax **LONG DeviceColumns;**

Remarks Holds the number of columns on this device.

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

See Also **DeviceRows** Property

DeviceDescriptors Property

- Syntax** **LONG DeviceDescriptors;**
- Remarks** Holds the number of descriptors on this device.
- If the capability **CapDescriptors** is TRUE, then **DeviceDescriptors** is non-zero; otherwise it is zero.
- This property is initialized by the **Open** method.
- See Also** **SetDescriptor** Method; **ClearDescriptors** Method

DeviceRows Property

- Syntax** **LONG DeviceRows;**
- Remarks** Holds the number of rows on this device.
- This property is initialized by the **Open** method.
- See Also** **DeviceColumns** Property

DeviceWindows Property

- Syntax** **LONG DeviceWindows;**
- Remarks** Holds the maximum window number supported by this device. A value of zero indicates that only the device window is supported, and that no windows may be created.
- This property is initialized by the **Open** method.
- See Also** **CurrentWindow** Property

InterCharacterWait Property R/W

Syntax **LONG InterCharacterWait;**

Remarks Holds the wait time between displaying each character with the **DisplayText** and **DisplayTextAt** methods. This timer gives a “teletype” appearance when displaying the text.

InterCharacterWait is only used if the window is not in Marquee Mode — that is, **MarqueeType** must be **DISP_MT_NONE**.

When non-zero and the window is not in Marquee Mode, the window is in Teletype Mode: **DisplayText** and **DisplayTextAt** requests are enqueued and processed in the order they are received. The **InterCharacterWait** timer specifies the time to wait between outputting each character into the viewport. (Note that the system timer resolution may reduce the precision of the wait time.) If **CursorUpdate** is **TRUE**, **CursorRow** and **CursorColumn** are updated to their final values before **DisplayText** or **DisplayTextAt** returns, even though all of its data may not yet be displayed.

When the timer is zero and the window is not in Marquee Mode, Immediate Mode is in effect, so that characters are processed as quickly as possible. If some display requests are enqueued at the time that **InterCharacterWait** is set to zero, the requests are completed as quickly as possible.

If the capability **CapICharWait** is **FALSE**, then intercharacter wait is not supported, and the value of this property is not used.

This property is initialized to zero by the **Open** and **CreateWindow** methods, and is updated when **CurrentWindow** is set or **DestroyWindow** is called.

Return When this property is set, the following value is placed in the **ResultCode** property:

Value	Meaning
OPOS_SUCCESS	The property was set successfully.

See Also **DisplayText** Method

MarqueeFormat Property R/W

Syntax **LONG MarqueeFormat;**

Remarks Holds the marquee format for the current window.

Value	Meaning
DISP_MF_WALK	Begin the marquee by walking data from the opposite side. For example, if the marquee type is “left”, then the viewport is filled by bringing characters into the right side and scrolling them to the left.
DISP_MF_PLACE	Begin the marquee by placing data. For example, if the marquee type is “left”, then the viewport is filled by placing characters starting at the left side, and beginning scrolling only after the viewport is full.

The value of **MarqueeFormat** is initialized to DISP_MF_WALK by the **Open** and **CreateWindow** methods, and is updated when **CurrentWindow** is set or **DestroyWindow** is called.

MarqueeFormat is read when a transition is made to Marquee On Mode. It is not used when not in Marquee Mode.

When **MarqueeFormat** is DISP_MF_WALK, and a transition is made from Marquee Init Mode to Marquee On Mode, the following occurs:

1. Map the window to the viewport as follows:

<u>Marquee Type</u>	<u>Window</u>	=	<u>Viewport</u>
Left	First Column	=	Last Column
Up	First Row	=	Last Row
Right	Last Column	=	First Column
Down	Last Row	=	First Row

Fill the viewport with blanks. Continue to Step 2 without waiting.

2. Display the mapped portion of the window into the viewport, then wait **MarqueeUnitWait** milliseconds. Move the window mapping onto the viewport by one row or column in the marquee direction. Repeat until the viewport is full.
3. Refresh the viewport, then wait **MarqueeUnitWait** milliseconds. Move the window mapping by one row or column. Repeat until the the last row or column is scrolled into the viewport (in which case, omit the unit wait).
4. Wait **MarqueeRepeatWait** milliseconds. Then go to step back to Step 1.

When **MarqueeFormat** is DISP_MF_PLACE, and a transition is made from Marquee Init Mode to Marquee On Mode, the following occurs:

1. Map the window to the viewport as follows:

<u>Marquee Type</u>	<u>Window</u>	=	<u>Viewport</u>
Left	First Column	=	First Column
Up	First Row	=	First Row
Right	Last Column	=	Last Column
Down	Last Row	=	Last Row

Fill the viewport with blanks. Continue to Step 2 without waiting.

2. Display a row or column into viewport, then wait **MarqueeUnitWait** milliseconds. Repeat until the viewport is full.
3. Move the window mapping onto the viewport by one row or column in the marquee direction, and refresh the viewport, then wait **MarqueeUnitWait** milliseconds. Repeat until the the last row or column is scrolled into the viewport (in which case, omit the unit wait).
4. Wait **MarqueeRepeatWait** milliseconds. Then go to step back to Step 1.

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.
OPOS_E_ILLEGAL	An invalid property value was used, or attempted to change window 0.

See Also **MarqueeType** Property; **MarqueeUnitWait** Property; **MarqueeRepeatWait** Property

Example 1 Marquee Walk format.

- Assume a 2x20 display.
- A Visual Basic application has a line display object named LD.
- The application has performed:
 LD.CreateWindow(0, 3, 2, 3, 2, 5) ' 2x3 viewport of 2x5 window
 LD.DisplayText("0123456789", DISP_DT_NORMAL)

The window contains:

	0	1	2	3	4
0	0	1	2	3	4
1	5	6	7	8	9

and the display contains (assuming the other windows are all blank):

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0				0	1	2														
1				5	6	7														

If the application performs the sequence:

```
LD.MarqueeType = DISP_MT_INIT
LD.MarqueeFormat = DISP_MF_WALK
LD.DisplayTextAt(0, 4, "AB", DISP_DT_NORMAL)
```

the viewport is not changed (since we are in Marquee Init Mode), and the window becomes:

	0	1	2	3	4
0	0	1	2	3	A
1	B	6	7	8	9

If the application performs:

LD.MarqueeType = DISP_MT_LEFT

the window is not changed, and the viewport becomes:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0						0														
1						B														

After **MarqueeUnitWait** milliseconds, the viewport is changed to:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0					0	1														
1					B	6														

After **MarqueeUnitWait** milliseconds, the viewport is changed to:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0				0	1	2														
1				B	6	7														

After **MarqueeUnitWait** milliseconds, the viewport is changed to:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0				1	2	3														
1				6	7	8														

After **MarqueeUnitWait** milliseconds, the viewport is changed to:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0				2	3	A														
1				7	8	9														

The marquee has scrolled to the end of the window.

After **MarqueeRepeatWait** milliseconds, the marquee display restarts with the viewport changing to:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0						0														
1						B														

Example 2 Marquee Place format.

- Assume a 2x20 display.
- A Visual Basic application has a line display object named LD.
- The application has performed:
 - LD.CreateWindow(0, 3, 2, 3, 2, 5) ' 2x3 viewport of 2x5 window
 - LD.DisplayText("0123456789", DISP_DT_NORMAL)

The window contains:

	0	1	2	3	4
0	0	1	2	3	4
1	5	6	7	8	9

and display contains (assuming the other windows are all blank):

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0				0	1	2														
1				5	6	7														

If the application performs the sequence:

- LD.MarqueeType = DISP_MT_INIT
- LD.MarqueeFormat = DISP_MF_PLACE
- LD.DisplayTextAt(0, 4, "AB", DISP_DT_NORMAL)

the viewport is not changed (since we are in Marquee Init Mode), and the window becomes:

	0	1	2	3	4
0	0	1	2	3	A
1	B	6	7	8	9

If the application performs:

- LD.MarqueeType = DISP_MT_LEFT

the window is not changed, and the viewport becomes:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0				0																
1				B																

After **MarqueeUnitWait** milliseconds, the viewport is changed to:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0				0	1															
1				B	6															

After **MarqueeUnitWait** milliseconds, the viewport is changed to:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0				0	1	2														
1				B	6	7														

From this point to the end of the window, the marquee action is the same as with marquee walking...

After **MarqueeUnitWait** milliseconds, the viewport is changed to:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0				1	2	3														
1				6	7	8														

After **MarqueeUnitWait** milliseconds, the viewport is changed to:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0				2	3	A														
1				7	8	9														

The marquee has scrolled to the end of the window.

After **MarqueeRepeatWait** milliseconds, the marquee display restarts with the viewport changing to:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0				0																
1				B																

MarqueeRepeatWait Property R/W

Syntax **LONG MarqueeRepeatWait;**

Remarks Holds the wait time between scrolling the final character or row of the window into its viewport and restarting the marquee with the first or last character or row.

The wait time is the specified number of milliseconds. (Note that the timer resolution may reduce the precision of the wait time.)

This property is initialized to zero by the **Open** and **CreateWindow** methods, and is updated when **CurrentWindow** is set or **DestroyWindow** is called.

MarqueeRepeatWait is not used if not in Marquee Mode.

Return When this property is set, the following value is placed in the **ResultCode** property:

Value	Meaning
OPOS_SUCCESS	The property was set successfully.

See Also **MarqueeType** Property; **MarqueeFormat** Property; **MarqueeUnitWait** Property

MarqueeType Property R/W

Syntax **LONG MarqueeType;**

Remarks Holds the marquee type for the current window. When not DISP_MT_NONE, the window is in Marquee Mode.

Value	Meaning
DISP_MT_NONE	Marquees are disabled for this window.
DISP_MT_INIT	Marquee Init Mode. Changes to the window are not reflected in the viewport until MarqueeType is changed to another value.
DISP_MT_UP	Scroll the window up. Illegal unless Rows is greater than the <i>Height</i> parameter used for the window's CreateWindow call, and the capability CapVMarquee is TRUE.
DISP_MT_DOWN	Scroll the window down. Illegal unless Rows is greater than the <i>Height</i> parameter used for the window's CreateWindow call, and the capability CapVMarquee is TRUE.
DISP_MT_LEFT	Scroll the window left. Illegal unless Columns is greater than the <i>Width</i> parameter used for the window's CreateWindow call, and the capability CapHMarquee is TRUE.
DISP_MT_RIGHT	Scroll the window left. Illegal unless Columns is greater than the <i>Width</i> parameter used for the window's CreateWindow call, and the capability CapHMarquee is TRUE.

A marquee is typically initialized after entering Marquee Init Mode by setting **MarqueeType** to DISP_MT_INIT, then calling **ClearText** and **DisplayText(At)** methods. Then, when **MarqueeType** is changed to an “on” value, Marquee On Mode is entered, and the marquee begins to be displayed in the viewport beginning at the start of the window (or end if the type is right or down).

When the mode is changed from Marquee On Mode to off, the marquee stops in place. A subsequent transition from back to Marquee On Mode continues from the current position.

When the mode is changed from Marquee On Mode to Marquee Init Mode, the marquee stops. Changes may be made to the window, then the window may be returned to Marquee On Mode to restart the marquee with the new data.

MarqueeType is always DISP_MT_NONE for window 0 – the device window.

The value of **MarqueeType** is initialized to DISP_MT_NONE by the **Open** and **CreateWindow** methods, and is updated when **CurrentWindow** is set or **DestroyWindow** is called.

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.
OPOS_E_ILLEGAL	An invalid property value was used, or attempted to change window 0.

See Also **MarqueeFormat** Property; **MarqueeUnitWait** Property; **MarqueeRepeatWait** Property

MarqueeUnitWait Property R/W

Syntax **LONG MarqueeUnitWait;**

Remarks Holds the wait time between marquee scrolling of each column or row in the window.

The wait time is the specified number of milliseconds. (Note that the timer resolution may reduce the precision of the wait time.)

MarqueeUnitWait is not used if **MarqueeType** is DISP_MT_NONE.

This property is initialized to zero by the **Open** and **CreateWindow** methods, and is updated when **CurrentWindow** is set or **DestroyWindow** is called.

Return When this property is set, the following value is placed in the **ResultCode** property:

Value	Meaning
OPOS_SUCCESS	The property was set successfully.

See Also **MarqueeType** Property; **MarqueeFormat** Property; **MarqueeRepeatWait** Property

Rows Property

Syntax **LONG Rows;**

Remarks Holds the number of rows for this window.

For window 0, **Rows** is the same as **DeviceRows**.

For other windows, it may be less or greater than **DeviceRows**.

This property is initialized to **DeviceRows** by the **Open** method, and is updated when **CurrentWindow** is set or **CreateWindow** or **DestroyWindow** are called.

See Also **Columns** Property

Methods

ClearDescriptors Method

Syntax **LONG ClearDescriptors ();**

Remarks Turns off all descriptors.

This function is illegal if the capability **CapDescriptors** is FALSE.

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 does not support descriptors.
<i>Other Values</i>	See ResultCode .

See Also **SetDescriptor** Method; **DeviceDescriptors** Property

ClearText Method

Syntax **LONG ClearText ();**

Remarks Clears the current window to blanks, sets **CursorRow** and **CursorColumn** to zero, and resynchronizes the beginning of the window with the start of the viewport.

If in Immediate Mode or Teletype Mode, the viewport is also cleared immediately.

If in Marquee Init Mode, the viewport is not changed.

If in Marquee On Mode, **ClearText** is illegal.

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	In Marquee On Mode.
<i>Other Values</i>	See ResultCode .

See Also **DisplayText** Method

CreateWindow Method

Syntax **LONG CreateWindow (LONG ViewportRow, LONG ViewportColumn, LONG ViewportHeight, LONG ViewportWidth, LONG WindowHeight, LONG WindowWidth);**

Parameter	Description
<i>ViewportRow</i>	The viewport's start device row.
<i>ViewportColumn</i>	The viewport's start device column.
<i>ViewportHeight</i>	The number of device rows in the viewport.
<i>ViewportWidth</i>	The number of device columns in the viewport.
<i>WindowHeight</i>	The number of rows in the window.
<i>WindowWidth</i>	The number of columns in the window.

Remarks Creates a viewport over the portion of the display given by the first four parameters. The window size is given by the last two parameters. Valid window row values range from (0) to (*WindowHeight-1*) and column values range from (0) to (*WindowWidth-1*).

The window size must be at least as large as the viewport size.

The window size may be larger than the viewport size in one direction. Using the window marquee properties **MarqueeType**, **MarqueeFormat**, **MarqueeUnitWait**, and **MarqueeRepeatWait**, such a window may be continuously scrolled in a marquee fashion.

When successful, **CreateWindow** sets the **CurrentWindow** property to the window number assigned to this window. The following properties are maintained for each window, and are initialized as given:

Property	Value
Rows	Set to <i>WindowHeight</i> .
Columns	Set to <i>WindowWidth</i> .
CursorRow	Set to 0.
CursorColumn	Set to 0.
CursorUpdate	Set to TRUE.
MarqueeType	Set to DISP_MT_NONE.

	MarqueeFormat	Set to DISP_MF_WALK.
	MarqueeUnitWait	Set to 0.
	MarqueeRepeatWait	Set to 0.
	InterCharacterWait	Set to 0.
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	One or more parameters are out of their valid ranges, or all available windows are already in use.
	<i>Other Values</i>	See ResultCode .
See Also	DestroyWindow Method; CurrentWindow Property	

DestroyWindow Method

Syntax	LONG DestroyWindow ();	
Remarks	Destroys the current window. The characters displayed in its viewport are not changed. CurrentWindow is set to window 0. The device window and the associated window properties are updated.	
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 current window is 0. This window may not be destroyed.
	<i>Other Values</i>	See ResultCode .
See Also	CreateWindow Method; CurrentWindow Property	

DisplayText Method

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

Parameter	Description
<i>Data</i>	The string of characters to display.
<i>Attribute</i>	The display attribute for the text. Must be either DISP_DT_NORMAL or DISP_DT_BLINK.

Remarks The characters in *Data* are processed beginning at the location specified by **CursorRow** and **CursorColumn**, and continue in succeeding columns.

Character processing continues to the next row when the end of a window row is reached. If the end of the window is reached with additional characters to be processed, then the window is scrolled upward by one row and the bottom row is set to blanks. If **CursorUpdate** is TRUE, then **CursorRow** and **CursorColumn** are updated to point to the character following the last character of *Data*.

Note

Scrolling will not occur when the last character of *Data* is placed at the end of a row. In this case, when **CursorUpdate** is TRUE, then **CursorRow** is set to the row containing the last character, and **CursorColumn** is set to **Columns** (that is, to one more than the final character of the row).

This stipulation ensures that the display does not scroll when a character is written into its last position. Instead, the Control will wait until another character is written before scrolling the window.

The operation of **DisplayText** (and **DisplayTextAt**) varies for each mode:

- Immediate Mode (**MarqueeType** = DISP_MT_NONE and **InterCharacterWait** = 0): Updates the window and viewport immediately.
- Teletype Mode (**MarqueeType** = DISP_MT_NONE and **InterCharacterWait** not = 0): The *Data* is enqueued. Enqueued data requests are processed in order (typically by another thread within the Control), updating the window and viewport using a wait of **InterCharacterWait** milliseconds after each character is sent to the viewport.
- Marquee Init Mode (**MarqueeType** = DISP_MT_INIT): Updates the window, but doesn't change the viewport.
- Marquee On Mode (**MarqueeType** not = DISP_MT_INIT): Illegal.

If the capability **CapBlink** is DISP_CB_NOBLINK, then *Attribute* is ignored. If it is DISP_CB_BLINKALL, then the entire display will blink when one or more characters have been set to blink. If it is DISP_CB_BLINKEACH, then only those characters displayed with the blink attribute will blink.

Special character values within *Data* are:

Value	Meaning
New Line (13)	Change the next character's output position to the beginning of the current row.
Line Feed (10)	Change the next character's output position to the beginning of the next row. Scroll the window if the current row is the last row of the window.

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	<i>Attribute</i> is illegal, or the display is in Marquee On Mode.
<i>Other Values</i>	See ResultCode .

See Also **DisplayTextAt** Method; **ClearText** Method; **InterCharacterWait** Property

DisplayTextAt Method

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

Parameter	Description
<i>Row</i>	The start row for the text.
<i>Column</i>	The start column for the text.
<i>Data</i>	The string of characters to display.
<i>Attribute</i>	The display attribute for the text. Must be either DISP_DT_NORMAL or DISP_DT_BLINK.

Remarks The characters in *Data* are processed beginning at the window location specified by the *Row* and *Column* parameters, and continuing in succeeding columns.

This method has the same effect as setting the **CursorRow** to *Row*, setting **CursorColumn** to *Column*, and calling the **DisplayText** method.

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	<i>Row</i> or <i>Column</i> are out of range, <i>Attribute</i> is illegal, or in Marquee On Mode.
<i>Other Values</i>	See ResultCode .

See Also **DisplayText** Method; **ClearText** Method; **InterCharacterWait** Property

RefreshWindow Method

Syntax **LONG RefreshWindow (LONG Window);**

The *Window* parameter specifies which window must be refreshed.

Remarks Changes the current window to *Window*, then redisplay its viewport. Neither the mapping of the window to its viewport nor the window's cursor position is changed.

This function may be used to restore a window after another window has overwritten some of its viewport.

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	<i>Window</i> is larger than DeviceWindows or has not been created, or in Marquee On Mode.
<i>Other Values</i>	See ResultCode .

ScrollText Method

Syntax **LONG ScrollText (LONG *Direction*, LONG *Units*);**

The *Direction* parameter indicates the scrolling direction, which may be one of the following:

Value	Meaning
DISP_ST_UP	Scroll the window up.
DISP_ST_DOWN	Scroll the window down.
DISP_ST_LEFT	Scroll the window left.
DISP_ST_RIGHT	Scroll the window right.

The *Units* parameter indicates the number of columns or rows to scroll.

Remarks Scroll the current window.

ScrollText is only legal in Immediate Mode.

If the window size for the scroll direction matches its viewport size, then the window data is scrolled, the last *Units* rows or columns are set to spaces, and the viewport is updated.

If the window size for the scroll direction is larger than its viewport, then the window data is not changed. Instead, the mapping of the window into the viewport is moved in the specified direction. The window data is not altered, but the viewport is updated. If scrolling by *Units* would go beyond the beginning of the window data, then the window is scrolled so that the first viewport row or column contains the first window row or column. If scrolling by *Units* would go beyond the end of the window data, then the window is scrolled so that the last viewport row or column contains the last window row or column.

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	<i>Direction</i> is illegal, or in Teletype Mode or Marquee Mode.
<i>Other Values</i>	See ResultCode .

See Also **DisplayText** Method

- Example 1**
- Assume a 2x20 display.
 - A Visual Basic application has a line display object named LD.
 - The application has performed:
 - LD.CreateWindow(0, 3, 2, 4, 2, 4) ' 2x4 viewport of 2x4 window
 - LD.DisplayText("abcdABCD", DISP_DT_NORMAL)

The window contains:

	0	1	2	3
0	a	b	c	d
1	A	B	C	D

and the viewport on the display is:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0				a	b	c	d													
1				A	B	C	D													

If the method

LD.ScrollText (DISP_ST_LEFT, 2)

is called, the window data becomes:

	0	1	2	3
0	c	d		
1	C	D		

and the viewport becomes:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0				c	d															
1				C	D															

- Example 2**
- Assume a 2x20 display.
 - A Visual Basic application has a line display object named LD.
 - The application has performed:
 - LD.CreateWindow(0, 3, 2, 4, 2, 8) ' 2x4 viewport of 2x8 window
 - LD.DisplayText("abcdefghABCDEFGH", DISP_DT_NORMAL)

The window contains:

	0	1	2	3	4	5	6	7
0	a	b	c	d	e	f	g	h
1	A	B	C	D	E	F	G	H

and the viewport on the display is:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0				a	b	c	d													
1				A	B	C	D													

If the method

LD.ScrollText (DISP_ST_LEFT, 2)

is called, the window data is unchanged, and the viewport becomes:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0				c	d	e	f													
1				C	D	E	F													

If the method

LD.ScrollText (DISP_ST_UP, 1)

is called next, the window data becomes:

	0	1	2	3	4	5	6	7
0	A	B	C	D	E	F	G	H
1								

and the viewport becomes:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0				C	D	E	F													
1																				

SetDescriptor Method

Syntax **LONG SetDescriptor (LONG Descriptor, LONG Attribute);**

The *Descriptor* parameter indicates which descriptor to change. The value may range between zero and one less than **DeviceDescriptors**.

The *Attribute* parameter indicates the attribute for the descriptor. Values are:

Value	Meaning
DISP_SD_ON	Turns the descriptor on.
DISP_SD_BLINK	Sets the descriptor to blinking.
DISP_SD_OFF	Turns the descriptor off.

Remarks Sets the state of one of the descriptors, which are small indicators with a fixed label.

This function is illegal if the capability **CapDescriptors** is FALSE.

The device and its Service Object determine the mapping of *Descriptor* to its descriptors.

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 does not support descriptors, or one of the parameters contained an illegal value.
<i>Other Values</i>	See ResultCode .

See Also **ClearDescriptors** Method; **DeviceDescriptors** Property

CHAPTER 7

MICR - Magnetic Ink Character Recognition Reader

Summary

Properties

<i>Common</i>	<i>Type</i>	<i>Access</i>	<i>Initialized After</i>
CheckHealthText	String	R	Open
Claimed	Boolean	R	Open
DataEventEnabled	Boolean	R/W	Open
DeviceEnabled	Boolean	R/W	Open & Claim
FreezeEvents	Boolean	R/W	Open
OutputID	Long	R	<i>Not Supported</i>
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

<i>Specific</i>	<i>Type</i>	<i>Access</i>	<i>Initialized After</i>
CapValidationDevice	Boolean	R	Open
RawData	String	R	Open
AccountNumber	String	R	Open
Amount	String	R	Open
BankNumber	String	R	Open
EPC	String	R	Open
SerialNumber	String	R	Open
TransitNumber	String	R	Open
CheckType	Long	R	Open
CountryCode	Long	R	Open

Methods*Common***Open****Close****Claim****Release****CheckHealth****ClearInput****ClearOutput****DirectIO***Specific***BeginInsertion****EndInsertion****BeginRemoval****EndRemoval***May Use After*

--

Open

Open

Open & Claim

Open, Claim, & Enable

Open & Claim

Not Supported

Open

Open, Claim, & Enable

Open, Claim, & Enable

Open, Claim, & Enable

Open, Claim, & Enable

Events*Name***DataEvent****DirectIOEvent****ErrorEvent****OutputCompleteEvent****StatusUpdateEvent***May Occur After*

Open, Claim, & Enable

Open

Open, Claim, & Enable

*Not Supported**Not Supported*

General Information

The MICR Control's OLE programmatic ID is "OPOS.MICR".

Capabilities

The MICR Control has the following minimal set of capabilities:

- Reads magnetic ink characters from a check.
- Has programmatic control of check insertion, reading, and removal. For some MICR devices, this will require no processing in the Control since the device may automate many of these functions.
- Parses the MICR data into the output properties provided by this Control. This release of OPOS specifies parsing of fields specified in the ANSI MICR standard used in North America. For other countries, the application may need to parse the MICR data from the data in **RawData**.

The MICR may have the following additional capability:

- The MICR device may be physically attached to or incorporated into a check validation print device. If this is the case, once a check is inserted via MICR Control methods, the check can still be used by the Printer Control prior to check removal.

Some MICR devices support exception tables, which cause non-standard parsing of the serial number for specific check routing numbers. Exception tables are not directly supported by this OPOS release. However, a Service Object may choose to support them, and could assign registry entries under its device name key to define the exception entries. (See the appendix "OPOS Registry Usage", page 301.)

Model

The MICR Control follows the general **Input Model** (page 19). One point of difference is that the MICR Control requires the execution of methods to insert and remove the check for processing. Therefore, this Control requires more than simply setting the **DataEventEnabled** property to TRUE in order to receive data. The basic model is as follows:

- The MICR Control is opened, claimed, and enabled.
- When an application wishes to perform a MICR read, the application will call the **BeginInsertion** method, specifying a timeout value. This will result in the device being made ready to have a check inserted. The method will either return successfully if the check has been inserted before the timeout limit was expired, or a timeout status will be returned.

In the event of a timeout, the MICR device will remain in a state allowing a check to be inserted while the application provides any additional prompting required and then reissues the **BeginInsertion** method.
- Once a check is inserted, the method returns successfully and the application will call the **EndInsertion** method, which will result in the MICR device being taken out of check insertion mode and the check, if present, actually being read. When the application sets the **DataEventEnabled** property to TRUE, the data will be sent to the application via the **DataEvent**.
- Following the **DataEvent**, the application should query the **CapValidationDevice** property to determine if validation printing can be performed on the check prior to check removal. If this property is true, the application may call the Printer Control's **BeginInsertion** and **EndInsertion** methods. This will position the check for validation printing. The Printer Control's validation printing methods can then be used to perform validation printing. When validation printing is complete, the application should call the Printer Control's removal methods to remove the check.
- Once the check is no longer needed in the device, the application must call the **BeginRemoval** method, also specifying a timeout value. This method will either return successfully if the check has been removed, or timeout if the check has not been removed. If a timeout is returned, the application may perform any additional prompting prior to calling the method again. Once the check is removed, the application should call the **EndRemoval** method to take the MICR device out of removal mode.

Many models of MICR devices do not require any check handling processing from the application. Such devices may always be capable of receiving a check and require no commands to actually read and eject the check. For these types of MICR devices, the **BeginInsertion**, **EndInsertion**, **BeginRemoval** and **EndRemoval** methods simply return an OPOS_SUCCESS status, and the Control will buffer the data until the **DataEventEnabled** property is set to TRUE. However, applications should still use these methods to ensure application portability across different MICR devices.





Device Sharing

The MICR is an exclusive-use device, as follows:

- The application must claim the device before enabling it.
- The application must claim and enable the device before the device begins reading input, or before calling methods that manipulate the device.
- See the “Summary” table for precise usage prerequisites.

MICR Character Substitution

The E13B MICR format used by the ANSI MICR standard contains 15 possible characters. Ten of these are the numbers 0 through 9. A space character may also be returned. The other four characters are special to MICR data and are known as the *Transit*, *Amount*, *On-Us*, and *Dash* characters. These characters are used to mark the boundaries of certain special fields in MICR data. Since these four characters are not in the ASCII character set, the following lower-case characters will be used to represent them in properties and parameters to methods:

MICR Character	Name	Substitute Character
	Transit	t
	Amount	a
	On-Us	o
	Dash	-

Properties

AccountNumber Property

Syntax	BSTR AccountNumber;
Remarks	<p>A string containing the account number parsed from the most recently read MICR data.</p> <p>This account number will not include a check serial number if a check serial number is able to be separately parsed, even if the check serial number is embedded in the account number portion of the 'On Us' field.</p> <p>If the account number cannot be identified successfully, the string will be empty ("").</p> <p>Its value is set prior to a DataEvent being sent to the application.</p>
See Also	RawData Property; DataEvent

Amount Property

Syntax	BSTR Amount;
Remarks	<p>A string containing the amount field parsed from the most recently read MICR data.</p> <p>The amount field on a check consists of ten digits bordered by Amount symbols. All non space digits will be represented in the test string including leading 0's.</p> <p>If the amount is not present, the string will be empty ("").</p> <p>Its value is set prior to a DataEvent being sent to the application.</p>
See Also	RawData Property; DataEvent

BankNumber Property

Syntax **BSTR BankNumber;**

Remarks A string containing the bank number portion of the transit field parsed from the most recently read MICR data.

The bank number is contained in digits 5 through 8 of the transit field.

If the bank number or transit field is not present or successfully identified, the string will be empty (“”).

Its value is set prior to a **DataEvent** being sent to the application.

See Also **RawData** Property; **TransitNumber** Property; **DataEvent**

CapValidationDevice Property

Syntax **BOOL CapValidationDevice;**

Remarks Indicates if this device also performs validation printing via the POS Printer Control's slip station.

If its value is TRUE, a check does not have to be removed from the MICR device prior to performing validation printing. For devices that are both a MICR device as well as a POS Printer, the device will automatically position the check for validation printing after successfully performing a MICR read. Either the MICR Control's or the POS Printer Control's **BeginRemoval** and **EndRemoval** methods may be called to remove the check once processing is complete.

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

CheckType Property

Syntax LONG CheckType;

Remarks A number that represents the type of check parsed from the most recently read MICR data.

Values are:

Value	Meaning
MICR_CT_PERSONAL	The check is a personal check.
MICR_CT_BUSINESS	The check is a business or commercial check.
MICR_CT_UNKNOWN	Unknown type of check.

Its value is set prior to a **DataEvent** being sent to the application.

See Also RawData Property; DataEvent

CountryCode Property

Syntax LONG CountryCode;

Remarks A number that represents the country of origin of the check parsed from the most recently read MICR data.

Values are:

Value	Meaning
MICR_CC_USA	The check is from America.
MICR_CC_CANADA	The check is from Canada.
MICR_CC_MEXICO	The check is from Mexico.
MICR_CC_UNKNOWN	Check origination is unknown.

Its value is set prior to a **DataEvent** being sent to the application.

See Also RawData Property; DataEvent

EPC Property

- Syntax** **BSTR EPC;**
- Remarks** A string containing the Extended Processing Code (“EPC”) field parsed from the most recently read MICR data. The string will contain a single character 0 through 9 if the field is present. If not, the string will be empty (“”).
- Its value is set prior to a **DataEvent** being sent to the application.
- See Also** **RawData** Property; **DataEvent**

RawData Property

- Syntax** **BSTR RawData;**
- Remarks** A string containing the MICR data from the most recent MICR read.
- The string contains any of the 15 MICR characters with appropriate substitution to represent non-ASCII characters (see “MICR Character Substitution”, page 147). No parsing or special processing is done to the data returned in this string. A sample value may look like the following:
- “2t123456789t123 4 567890o 123 a0000001957a”
- Note that the property value will include spaces to represent spaces in the MICR data.
- Its value is set prior to a **DataEvent** being sent to the application.
- See Also** **AccountNumber** Property; **Amount** Property; **BankNumber** Property; **CheckType** Property; **CountryCode** Property; **EPC** Property; **SerialNumber** Property; **TransitNumber** Property; **DataEvent**

SerialNumber Property

Syntax BSTR SerialNumber;

Remarks A string containing the serial number of the check parsed from the most recently read MICR data.

If the serial number cannot be successfully parsed, the value of this property will be empty (“”).

Its value is set prior to a **DataEvent** being sent to the application.

See Also RawData Property; DataEvent

TransitNumber Property

Syntax BSTR TransitNumber;

Remarks A string containing the transit field of the check parsed from the most recently read MICR data.

The transmit number consists of all the characters read between the ‘Transit’ symbols on the check. It is a nine character string.

Its value is set prior to a **DataEvent** being sent to the application.

See Also RawData Property; DataEvent

Methods

BeginInsertion Method

Syntax **LONG BeginInsertion (LONG Timeout);**

The *Timeout* parameter gives the number of milliseconds before failing the method.

Remarks Called to initiate check insertion processing.

When called, the MICR is made ready to receive a check by opening the MICR's check handling "jaws" or activating a MICR's check insertion mode. This method is paired with the **EndInsertion** method for controlling check insertion. For MICR devices that do not require this sort of processing, these methods will always return OPOS_SUCCESS. However, the application should still use these methods to ensure application portability across different MICR devices.

If the MICR device cannot be placed into insertion mode, an error is returned to the application. Otherwise, the Control continues to monitor check insertion until either:

- The check is successfully inserted. In this case, the Control returns an OPOS_SUCCESS status.
- The check is not inserted before *Timeout* milliseconds have elapsed, or an error is reported by the MICR device. In this case, the Control either returns OPOS_E_TIMEOUT or another error. The MICR device remains in check insertion mode. This allows an application to perform some user interaction and reissue the **BeginInsertion** method without altering the MICR check handling mechanism.

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

Value	Meaning
OPOS_SUCCESS	The method was initiated successfully.
OPOS_E_BUSY	If the MICR is a combination device, the peer device may be busy.
OPOS_E_TIMEOUT	The specified time has elapsed without the check being properly inserted.
<i>Other Values</i>	See ResultCode .

See Also **EndInsertion** Method; **BeginRemoval** Method; **EndRemoval** Method

BeginRemoval Method

Syntax **LONG BeginRemoval (LONG Timeout);**

The *Timeout* property gives the number of milliseconds before failing the method.

Remarks Called to initiate check removal processing.

When called, the MICR is made ready to remove a check, by opening the MICR's check handling "jaws" or activating a MICR's check ejection mode. This method is paired with the **EndRemoval** method for controlling check removal. For MICR devices that do not require this sort of processing, these methods will always return OPOS_SUCCESS. However, the application should still use these methods to ensure application portability across different MICR devices.

If the MICR device cannot be placed into removal or ejection mode, an error is returned to the application. Otherwise, the Control continues to monitor check removal until either:

- The check is successfully removed. In this case, the Control returns an OPOS_SUCCESS status.
- The check is not removed before *Timeout* milliseconds have elapsed, or an error is reported by the MICR device. In this case, the Control either returns OPOS_E_TIMEOUT or another error. The MICR device remains in check removal mode. This allows an application to perform some user interaction and reissue the **BeginRemoval** method without altering the MICR check handling mechanism.

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

Value	Meaning
OPOS_SUCCESS	The method was initiated successfully.
OPOS_E_BUSY	If the MICR is a combination device, the peer device may be busy.
OPOS_E_TIMEOUT	The specified time has elapsed without the check being properly removed.
<i>Other Values</i>	See ResultCode .

See Also **BeginInsertion** Method; **EndInsertion** Method; **EndRemoval** Method

EndInsertion Method

Syntax **LONG EndInsertion ();**

Remarks Called to end check insertion processing.

When called, the MICR is taken out of check insertion mode. If a check is detected in the device, a successful status of OPOS_SUCCESS is returned to the application. If no check is present, an extended error status OPOS_EMICR_NOCHECK is returned. Upon completion of this method, the check will be read by the MICR device, and data will be available as soon as the **DataEventEnabled** property is set to TRUE. This allows an application to prompt the user prior to calling this method to ensure that the form is correctly positioned.

This method is paired with the **BeginInsertion** method for controlling check insertion. For MICR devices that do not require this sort of processing, these methods will always return OPOS_SUCCESS. However, the application should still use these methods to ensure application portability across different MICR devices.

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

Value	Meaning
OPOS_SUCCESS	The method was initiated successfully.
OPOS_E_ILLEGAL	The printer is not in check insertion mode.
OPOS_E_EXTENDED	ResultCodeExtended = OPOS_EMICR_NOCHECK: The device was taken out of insertion mode without a check being inserted.
<i>Other Values</i>	See ResultCode .

See Also **BeginInsertion** Method; **BeginRemoval** Method; **EndRemoval** Method

EndRemoval Method

Syntax **LONG EndRemoval ();**

Remarks Called to end check removal processing.

When called, the MICR is taken out of check removal or ejection mode. If no check is detected in the device, a successful status of OPOS_SUCCESS is returned to the application. If a check is present, an extended error status OPOS_EMICR_CHECK is returned.

This method is paired with the **BeginRemoval** method for controlling check removal. For MICR devices that do not require this sort of processing, these methods will always return OPOS_SUCCESS. However, the application should still use these methods to ensure application portability across different MICR devices.

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

Value	Meaning
OPOS_SUCCESS	The method was initiated successfully.
OPOS_E_ILLEGAL	The printer is not in check removal mode.
OPOS_E_EXTENDED	ResultCodeExtended = OPOS_EMICR_CHECK: The device was taken out of removal mode while a check is still present.
<i>Other Values</i>	See ResultCode .

See Also **BeginInsertion** Method; **EndInsertion** Method; **BeginRemoval** Method

Events

DataEvent Event

- Syntax** **VOID DataEvent (LONG Status);**
The *Status* parameter contains zero.
- Remarks** Fired when MICR data is read from a check.

Before firing this event, the MICR Control updates the **RawData** property and attempts to parse this data into the MICR data fields.
- See Also** **RawData** Property; **AccountNumber** Property; **Amount** Property; **BankNumber** Property; **CheckType** Property; **CountryCode** Property; **EPC** Property; **SerialNumber** Property; **TransitNumber** Property

ErrorEvent Event

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

Parameter	Description
<i>ResultCode</i>	Result code causing the error event. See ResultCode for values.
<i>ResultCodeExtended</i>	Extended result code causing the error event. See ResultCodeExtended for values.
<i>ErrorLocus</i>	Location of the error. See values below.
<i>pErrorResponse</i>	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.

The contents at the location pointed to by the *pErrorResponse* parameter is preset to a default value, based on the *ErrorLocus*. The application may change it to one of the following:

Value	Meaning
OPOS_ER_CLEAR	Clear the buffered input data. The error state is exited. Default when locus is OPOS_EL_INPUT.
OPOS_ER_CONTINUEINPUT	Use only when locus is OPOS_EL_INPUT_DATA. Acknowledges the error and directs the Control to continue processing. The Control remains in the error state and will fire additional DataEvents as directed by the DataEventEnabled property. When all input has been fired and the DataEventEnabled property is again set to TRUE, then another ErrorEvent is fired with locus

OPOS_EL_INPUT.

Default when locus is OPOS_EL_INPUT_DATA.

Remarks Fired when an error is detected while trying to read MICR data.

Input error events are not fired until the **DataEventEnabled** property is TRUE, so that proper application sequencing occurs.

See Also “Status, Result Code, and State Model”

CHAPTER 8

MSR - Magnetic Stripe Reader

Summary

Properties

<i>Common</i>	<i>Type</i>	<i>Access</i>	<i>Initialized After</i>
CheckHealthText	String	R	Open
Claimed	Boolean	R	Open
DataEventEnabled	Boolean	R/W	Open
DeviceEnabled	Boolean	R/W	Open & Claim
FreezeEvents	Boolean	R/W	Open
OutputID	Long	R	<i>Not Supported</i>
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

<i>Specific</i>	<i>Type</i>	<i>Access</i>	<i>Initialized After</i>
CapISO	Boolean	R	Open
CapJISOne	Boolean	R	Open
CapJISTwo	Boolean	R	Open
TracksToRead	Long	R/W	Open
DecodeData	Boolean	R/W	Open
ParseDecodeData	Boolean	R/W	Open
Track1Data	String	R	Open
Track2Data	String	R	Open
Track3Data	String	R	Open
AccountNumber	String	R	Open
ExpirationDate	String	R	Open
Title	String	R	Open
FirstName	String	R	Open
MiddleInitial	String	R	Open
Surname	String	R	Open
Suffix	String	R	Open
ServiceCode	String	R	Open
Track1DiscretionaryData	String	R	Open
Track2DiscretionaryData	String	R	Open

Methods

<i>Common</i>	<i>May Use After</i>
Open	--
Close	Open
Claim	Open
Release	Open & Claim
CheckHealth	Open, Claim, & Enable
ClearInput	Open, Claim, & Enable
ClearOutput	<i>Not Supported</i>
DirectIO	Open

Events

<i>Name</i>	<i>May Occur After</i>
DataEvent	Open, Claim, & Enable
DirectIOEvent	Open
ErrorEvent	Open, Claim, & Enable
OutputCompleteEvent	<i>Not Supported</i>
StatusUpdateEvent	<i>Not Supported</i>

General Information

The MSR Control's OLE programmatic ID is "OPOS.MSR".

Capabilities

The MSR Control has the following minimal set of capabilities:

- Reads encoded data from a magnetic stripe. Data is obtainable from any combination of tracks 1, 2, and 3.
- The alphanumeric data bytes may be decoded into their corresponding alphanumeric codes. Furthermore, this decoded alphanumeric data may be divided into specific fields accessed as device properties.

The MSR may have the following additional capability:

- Support for specific card types: ISO, JIS Type I, and/or JIS Type 2.

Model

Three writable properties control MSR data handling:

- The **TracksToRead** property controls which combination of the three tracks should be read. It is not an error to swipe a card containing less than this set of tracks. Rather, this property should be set to the set of tracks that the Application may need to process.
- The **DecodeData** property controls decoding of track data from raw format into displayable data.
- The **ParseDecodeData** property controls parsing of decoded data into fields, based on common MSR standards.

The MSR Control follows the general input model for event driven input:

- When input is received by the Control due to a card swipe, the Control generates a **DataEvent**, passing information about the swipe in the parameter *Data*. The actual card data is available by accessing the **Track1Data**, **Track2Data**, and **Track3Data** properties of the device. Based on application-settable properties, some data may be parsed into other properties.
- Just before firing the **DataEvent**, the Control disables further data events by setting the **DataEventEnabled** property to FALSE. This causes buffering of further input data at the Control while the application processes the current input and associated properties. When the application has finished the current input and is ready for more data, it reenables events by setting **DataEventEnabled** to TRUE.
- All buffered input may be deleted by calling the **ClearInput** method.

Device Sharing

The MSR is an exclusive-use device, as follows:

- The application must claim the device before enabling it.
- The application must claim and enable the device before the device begins reading input, or before calling methods that manipulate the device.
- See the “Summary” table for precise usage prerequisites.

Properties

AccountNumber Property

- Syntax** **BSTR AccountNumber;**
- Remarks** The account number obtained from the most recently swiped card.
- Set to the empty string if:
- The field was not included in the track data obtained, or,
 - The track data format was not one of those listed in the **ParseDecodeData** property section of this document, or,
 - **ParseDecodeData** is FALSE.

CapISO Property

- Syntax** **BOOL CapISO;**
- Remarks** If TRUE, the MSR device supports ISO cards; otherwise it is FALSE.
- This property is initialized by the **Open** method.

CapJISOne Property

- Syntax** **BOOL CapJISOne;**
- Remarks** If TRUE, the MSR device supports JIS Type-I cards; otherwise it is FALSE.
- JIS-I cards are a superset of ISO cards. Therefore, if **CapJISOne** is TRUE, then it is implied that **CapISO** is also TRUE.
- This property is initialized by the **Open** method.

CapJISTwo Property

Syntax **BOOL CapJISTwo;**

Remarks If TRUE, the MSR device supports JIS Type-II cards; otherwise it is FALSE.

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

DecodeData Property R/W

Syntax **BOOL DecodeData;**

Remarks If TRUE, each byte of track data contained within the **Track1Data**, **Track2Data**, and **Track3Data** properties is mapped from its original encoded bit sequence (as it exists on the magnetic card) to its corresponding decoded ASCII bit sequence. This conversion is mainly of relevance for data that is NOT of the 7-bit format, since 7-bit data needs no decoding to decipher its corresponding alphanumeric and/or Katakana characters.

The decoding that takes place is as follows for each card type, track, and track data format:

Card Type	Track	Data Format	Decoded Values
ISO	Track 1	6-Bit	0x20 thru 0x5F
	Track 2	4-Bit	0x30 thru 0x3F
	Track 3	4-Bit	0x30 thru 0x3F
JIS-I	Track 1	6-Bit	0x20 thru 0x5F
	Track 1	7-Bit	Data Unaltered
	Track 2	4-Bit	0x30 thru 0x3F
	Track 3	4-Bit	0x30 thru 0x3F
	Track 3	7-Bit	Data Unaltered
JIS-II	JIS Track on Front of Card	7-Bit	Data Unaltered

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

Setting this property to FALSE automatically sets the **ParseDecodeData** property to FALSE.

Return When this property is set, the following value is placed in the **ResultCode** property:

Value	Meaning
OPOS_SUCCESS	The property was set successfully.

See Also **ParseDecodeData** Property

ExpirationDate Property

Syntax **BSTR ExpirationDate;**

Remarks The expiration date obtained from the most recently swiped card.

Set to the empty string if:

- The field was not included in the track data obtained, or,
- The track data format was not one of those listed in the **ParseDecodeData** property section of this document, or,
- **ParseDecodeData** is FALSE.

FirstName Property

Syntax **BSTR FirstName;**

Remarks The first name obtained from the most recently swiped card.

Set to the empty string if:

- The field was not included in the track data obtained, or,
- The track data format was not one of those listed in the **ParseDecodeData** property section of this document, or,
- **ParseDecodeData** is FALSE.

MiddleInitial Property

Syntax **BSTR MiddleInitial;**

Remarks The middle initial obtained from the most recently swiped card.

Set to the empty string if:

- The field was not included in the track data obtained, or,
- The track data format was not one of those listed in the **ParseDecodeData** property section of this document, or,
- **ParseDecodeData** is FALSE.

ParseDecodedData Property R/W

Syntax **BOOL ParseDecodedData;**

Remarks If TRUE, the decoded data contained within the **Track1Data** and **Track2Data** properties is further separated into fields for access via various other properties. **Track3Data** is not parsed because its data content is of an open format defined by the card issuer. JIS-I Track 1 Format C and ISO Track 1 Format C data are not parsed for similar reasons.

The parsed data properties are the defined possible fields for cards with data consisting of the following formats:

- JIS-I / ISO Track 1 Format A
- JIS-I / ISO Track 1 Format B
- JIS-I / ISO Track 1 VISA Format (a de-facto standard)
- JIS-I / ISO Track 2 Format

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

Setting this property to TRUE automatically sets the **DecodeData** property to TRUE.

Return When this property is set, the following value is placed in the **ResultCode** property:

Value	Meaning
OPOS_SUCCESS	The property was set successfully.

See Also **DecodeData** Property; **Surname** Property; **Suffix** Property; **AccountNumber** Property; **FirstName** Property; **MiddleInitial** Property; **Title** Property; **ExpirationDate** Property; **ServiceCode** Property; **Track1DiscretionaryData** Property; **Track2DiscretionaryData** Property

ServiceCode Property

Syntax **BSTR ServiceCode;**

Remarks The service code obtained from the most recently swiped card.

Set to the empty string if:

- The field was not included in the track data obtained, or,
- The track data format was not one of those listed in the **ParseDecodeData** property section of this document, or,
- **ParseDecodeData** is FALSE.

Suffix Property

Syntax **BSTR Suffix;**

Remarks The suffix obtained from the most recently swiped card.

Set to the empty string if:

- The field was not included in the track data obtained, or,
- The track data format was not one of those listed in the **ParseDecodeData** property section of this document, or,
- **ParseDecodeData** is FALSE.

Surname Property

Syntax **BSTR Surname;**

Remarks The surname obtained from the most recently swiped card.

Set to the empty string if:

- The field was not included in the track data obtained, or,
- The track data format was not one of those listed in the **ParseDecodeData** property section of this document, or,
- **ParseDecodeData** is FALSE.

Title Property

Syntax **BSTR Title;**

Remarks The title obtained from the most recently swiped card.

Set to the empty string if:

- The field was not included in the track data obtained, or,
- The track data format was not one of those listed in the **ParseDecodeData** property section of this document, or,
- **ParseDecodeData** is FALSE.

Track1Data Property

Syntax **BSTR Track1Data;**

Remarks Contains either the track 1 data from the previous card swipe or an empty string.

The data returned by this property may also be parsed into other properties when the **ParseDecodeData** property is set.

An empty string indicates that the track was not accessible.

See Also **TracksToRead** Property

Track1DiscretionaryData Property

Syntax **BSTR Track1DiscretionaryData;**

Remarks The track 1 discretionary data obtained from the most recently swiped card.

Set to the empty string if:

- The field was not included in the track data obtained, or,
- The track data format was not one of those listed in the **ParseDecodeData** property section of this document, or,
- **ParseDecodeData** is FALSE.

The amount of data contained in this property varies widely depending upon the format of the track 1 data.

Track2Data Property

Syntax **BSTR Track2Data;**

Remarks Contains either the track 2 data from the previous card swipe or an empty string.

The data returned by this property may also be parsed into other properties when the **ParseDecodeData** property is set.

An empty string indicates that the track was not accessible.

See Also **TracksToRead** Property

Track2DiscretionaryData Property

- Syntax** **BSTR Track2DiscretionaryData;**
- Remarks** The track 2 discretionary data obtained from the most recently swiped card.
- Set to the empty string if:
- The field was not included in the track data obtained, or,
 - The track data format was not one of those listed in the **ParseDecodeData** property section of this document, or,
 - **ParseDecodeData** is FALSE.

Track3Data Property

- Syntax** **BSTR Track3Data;**
- Remarks** Contains either the track 3 data from the previous card swipe or an empty string.
- An empty string indicates that the track was not accessible.
- See Also** **TracksToRead** Property

TracksToRead Property R/W

Syntax **LONG TracksToRead;**

Remarks Indicates the track data that the application wishes to have placed into the **Track1Data**, **Track2Data**, and **Track3Data** properties following a card swipe.

Value	Meaning
MSR_TR_1	Obtain Track 1.
MSR_TR_2	Obtain Track 2.
MSR_TR_3	Obtain Track 3.
MSR_TR_1_2	Obtain Tracks 1 and 2.
MSR_TR_1_3	Obtain Tracks 1 and 3.
MSR_TR_2_3	Obtain Tracks 2 and 3.
MSR_TR_1_2_3	Obtain Tracks 1, 2, and 3.

Decreasing the required number of tracks may provide a greater swipe success rate and somewhat greater responsiveness by removing the processing for unaccessed data.

TracksToRead does not indicate a capability of the MSR hardware unit, but instead is an application configurable property representing which track(s) will have their data obtained, potentially decoded, and returned *if possible*. Cases such as an ISO type card being swiped through a JIS-II read head, cards simply not having data for particular tracks, and other factors may preclude desired data from being obtained.

This property is initialized to MSR_TR_1_2_3 by 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.
OPOS_E_ILLEGAL	An invalid track value was specified.

Events

DataEvent Event

Syntax VOID DataEvent (LONG Status);

The *Status* parameter is divided into four bytes with three of the bytes representing information about the three tracks, while the fourth byte is unused. The diagram below indicates how the parameter *Status* is divided:

High Word		Low Word	
High Byte	Low Byte	High Byte	Low Byte
Unused	Track 3	Track 2	Track 1

A value of zero (0) for a track byte means that no data was obtained from the swipe for that particular track. This might be due to the hardware device simply not having a read head for the track, or perhaps the application intentionally precluded incoming data from the track via the **TracksToRead** property.

A value greater than zero (> 0) indicates the length in bytes of the corresponding **Track x Data** property.

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

Before firing the event, the swiped data is placed into **Track1Data**, **Track2Data**, and **Track3Data**. If **DecodeData** is TRUE, then this data is decoded. If **ParseDecodedData** is TRUE, then the data is parsed into several additional properties.

ErrorEvent Event

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

Parameter	Description
<i>ResultCode</i>	Result code causing the error event. See values below.
<i>ResultCodeExtended</i>	Extended result code causing the error event. See values below.
<i>ErrorLocus</i>	Location of the error. See values below.
<i>pErrorResponse</i>	Pointer to the error event response. See values below.

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

Value	Meaning
OPOS_E_EXTENDED	ResultCodeExtended = OPOS_EMSR_START: Start sentinel error.
	ResultCodeExtended = OPOS_EMSR_END: End sentinel error.
	ResultCodeExtended = OPOS_EMSR_PARITY: Parity error.
	ResultCodeExtended = OPOS_EMSR_LRC: LRC error.
<i>Other Values</i>	See ResultCode .

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.

The contents at the location pointed to by the *pErrorResponse* parameter are preset to a default value, based on the *ErrorLocus*. The application may change it to one of the following:

Value	Meaning
OPOS_ER_CLEAR	Clear the buffered input data. The error state is exited. Default when locus is OPOS_EL_INPUT.
OPOS_ER_CONTINUEINPUT	Use only when locus is OPOS_EL_INPUT_DATA. Acknowledges the error and directs the Control to continue processing. The Control remains in the error state and will fire additional DataEvents as directed by the DataEventEnabled property. When all input has been fired and the DataEventEnabled property is again set to TRUE, then another ErrorEvent is fired with locus OPOS_EL_INPUT. Default when locus is OPOS_EL_INPUT_DATA.

- Remarks** Fired when an error is detected while trying to read MSR data.
- Input error events are not fired until the **DataEventEnabled** property is TRUE, so that proper application sequencing occurs.
- The particular track that caused the fault is not specified. However, this level of specificity is deemed irrelevant since this event will only be generated for a track or tracks that the application explicitly requested via **TracksToRead**.
- See Also** “Status, Result Code, and State Model”

CHAPTER 9

POS Keyboard

Summary

Properties

<i>Common</i>	<i>Type</i>	<i>Access</i>	<i>Initialized After</i>
CheckHealthText	String	R	Open
Claimed	Boolean	R	Open
DataEventEnabled	Boolean	R/W	Open
DeviceEnabled	Boolean	R/W	Open & Claim
FreezeEvents	Boolean	R/W	Open
OutputID	Long	R	<i>Not Supported</i>
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
<i>Specific</i>			
POSKeyData	String	R	Open

Methods

<i>Common</i>	<i>May Use After</i>
Open	--
Close	Open
Claim	Open
Release	Open & Claim
CheckHealth	Open, Claim, & Enable
ClearInput	Open, Claim, & Enable
ClearOutput	<i>Not Supported</i>
DirectIO	Open

Events

<i>Name</i>	<i>May Occur After</i>
DataEvent	Open, Claim, & Enable
DirectIOEvent	Open
ErrorEvent	Open, Claim, & Enable
OutputCompleteEvent	<i>Not Supported</i>
StatusUpdateEvent	<i>Not Supported</i>

General Information

The POS Keyboard is defined in OPOS Release 1.1.
The POS Keyboard Control's OLE programmatic ID is "OPOS.POSKeyboard".

Capabilities

The POS Keyboard Control has the following capability:

- Reads keys from a POS keyboard. A POS keyboard is a secondary key entry device, separate from the primary keyboard recognized and supported by the operating system.

Model

The POS Keyboard Control follows the general Input Model for event-driven input:

- When input is received by the Control, it generates a **DataEvent**.
- Just before firing this event, the Control disables further data events by setting the **DataEventEnabled** property to FALSE. This causes buffering of further input data at the Control while the application processes the current input and associated properties. When the application has finished the current input and is ready for more data, it reenables events by setting **DataEventEnabled** to TRUE.
- All input buffered at the Control may be deleted by calling the **ClearInput** method.

Device Sharing

The POS keyboard is an exclusive-use device, as follows:

- The application must claim the device before enabling it.
- The application must claim and enable the device before the device begins reading input.
- See the "Summary" table for precise usage prerequisites.

Properties

POSKeyData Property

Syntax **LONG POSKeyData;**

Remarks The data read from the POS keyboard.

The upper 16 bits (HIWORD) of **POSKeyData** contain the logical key code, and the lower 16 bits (LOWORD) contain the scan code. The logical key code and scan code may match a standard PC keyboard value, but this is not required. That is, a POS keyboard may return PC-style data or proprietary data, depending upon its implementation.

This property is set by the Control just before firing the **DataEvent**.

Events

DataEvent Event

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

The *Status* parameter contains zero.

Remarks Fired to present input data from the device to the application. The POS keyboard data is placed in the **POSKeyData** property before this event is fired.

ErrorEvent Event

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

Parameter	Description
<i>ResultCode</i>	Result code causing the error event. See ResultCode for values.
<i>ResultCodeExtended</i>	Extended result code causing the error event. See ResultCodeExtended for values.
<i>ErrorLocus</i>	Location of the error. See values below.
<i>pErrorResponse</i>	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.

The contents at the location pointed to by the *pErrorResponse* parameter is preset to a default value, based on the *ErrorLocus*. The application may change it to one of the following:

Value	Meaning
OPOS_ER_CLEAR	Clear the buffered input data. The error state is exited. Default when locus is OPOS_EL_INPUT.
OPOS_ER_CONTINUEINPUT	Use only when locus is OPOS_EL_INPUT_DATA. Acknowledges the error and directs the Control to continue processing. The Control remains in the error state and will fire additional DataEvents as directed by the DataEventEnabled property. When all input has been fired and the DataEventEnabled property is again set to TRUE, then another ErrorEvent is fired with locus OPOS_EL_INPUT. Default when locus is OPOS_EL_INPUT_DATA.

- Remarks** Fired when an error is detected while trying to read POS Keyboard data.
- Input error events are not fired until the **DataEventEnabled** property is TRUE, so that proper application sequencing occurs.
- See Also** “Status, Result Code, and State Model”

CHAPTER 10

POS Printer

Summary

Properties

<i>Common</i>		<i>Type</i>	<i>Access</i>	<i>Initialized After</i>
CheckHealthText		String	R	Open
Claimed		Boolean	R	Open
DataEventEnabled		Boolean	R/W	<i>Not Supported</i>
DeviceEnabled		Boolean	R/W	Open & Claim
FreezeEvents		Boolean	R/W	Open
OutputID		Long	R	Open
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

<i>Specific</i>		<i>Type</i>	<i>Access</i>	<i>Initialized After</i>
CapCharacterSet	1.1	Long	R	Open
CapConcurrentJrnRec	1.0	Boolean	R	Open
CapConcurrentJrnSlp	1.0	Boolean	R	Open
CapConcurrentRecSlp	1.0	Boolean	R	Open
CapCoverSensor	1.0	Boolean	R	Open
CapTransaction	1.1	Boolean	R	Open

<i>Specific (continued)</i>		<i>Type</i>	<i>Access</i>	<i>Initialized After</i>
CapJrnPresent	1.0	Boolean	R	Open
CapJrn2Color	1.0	Boolean	R	Open
CapJrnBold	1.0	Boolean	R	Open
CapJrnDhigh	1.0	Boolean	R	Open
CapJrnDwide	1.0	Boolean	R	Open
CapJrnDwideDhigh	1.0	Boolean	R	Open
CapJrnEmptySensor	1.0	Boolean	R	Open
CapJrnItalic	1.0	Boolean	R	Open
CapJrnNearEndSensor	1.0	Boolean	R	Open
CapJrnUnderline	1.0	Boolean	R	Open
CapRecPresent	1.0	Boolean	R	Open
CapRec2Color	1.0	Boolean	R	Open
CapRecBarCode	1.0	Boolean	R	Open
CapRecBitmap	1.0	Boolean	R	Open
CapRecBold	1.0	Boolean	R	Open
CapRecDhigh	1.0	Boolean	R	Open
CapRecDwide	1.0	Boolean	R	Open
CapRecDwideDhigh	1.0	Boolean	R	Open
CapRecEmptySensor	1.0	Boolean	R	Open
CapRecItalic	1.0	Boolean	R	Open
CapRecLeft90	1.0	Boolean	R	Open
CapRecNearEndSensor	1.0	Boolean	R	Open
CapRecPapercut	1.0	Boolean	R	Open
CapRecRight90	1.0	Boolean	R	Open
CapRecRotate180	1.0	Boolean	R	Open
CapRecStamp	1.0	Boolean	R	Open
CapRecUnderline	1.0	Boolean	R	Open

<i>Specific (continued)</i>		<i>Type</i>	<i>Access</i>	<i>Initialized After</i>
CapSlpPresent	1.0	Boolean	R	Open
CapSlpFullslip	1.0	Boolean	R	Open
CapSlp2Color	1.0	Boolean	R	Open
CapSlpBarCode	1.0	Boolean	R	Open
CapSlpBitmap	1.0	Boolean	R	Open
CapSlpBold	1.0	Boolean	R	Open
CapSlpDhigh	1.0	Boolean	R	Open
CapSlpDwide	1.0	Boolean	R	Open
CapSlpDwideDhigh	1.0	Boolean	R	Open
CapSlpEmptySensor	1.0	Boolean	R	Open
CapSlpItalic	1.0	Boolean	R	Open
CapSlpLeft90	1.0	Boolean	R	Open
CapSlpNearEndSensor	1.0	Boolean	R	Open
CapSlpRight90	1.0	Boolean	R	Open
CapSlpRotate180	1.0	Boolean	R	Open
CapSlpUnderline	1.0	Boolean	R	Open
AsyncMode	1.0	Boolean	R/W	Open
CharacterSet	1.0	Long	R/W	Open, Claim, & Enable
CharacterSetList	1.0	String	R	Open
CoverOpen	1.0	Boolean	R	Open, Claim, & Enable
ErrorLevel	1.1	Long	R	Open
ErrorStation	1.0	Long	R	Open
ErrorString	1.1	String	R	Open
FontTypefaceList	1.1	String	R	Open
FlagWhenIdle	1.0	Boolean	R/W	Open
MapMode	1.0	Long	R/W	Open
RotateSpecial	1.1	Long	R/W	Open

<i>Specific (continued)</i>		<i>Type</i>	<i>Access</i>	<i>Initialized After</i>
JrnLineChars	1.0	Long	R/W	Open, Claim, & Enable
JrnLineCharsList	1.0	String	R	Open
JrnLineHeight	1.0	Long	R/W	Open, Claim, & Enable
JrnLineSpacing	1.0	Long	R/W	Open, Claim, & Enable
JrnLineWidth	1.0	Long	R	Open, Claim, & Enable
JrnLetterQuality	1.0	Boolean	R/W	Open, Claim, & Enable
JrnEmpty	1.0	Boolean	R	Open, Claim, & Enable
JrnNearEnd	1.0	Boolean	R	Open, Claim, & Enable
RecLineChars	1.0	Long	R/W	Open, Claim, & Enable
RecLineCharsList	1.0	String	R	Open
RecLineHeight	1.0	Long	R/W	Open, Claim, & Enable
RecLineSpacing	1.0	Long	R/W	Open, Claim, & Enable
RecLineWidth	1.0	Long	R	Open, Claim, & Enable
RecLetterQuality	1.0	Boolean	R/W	Open, Claim, & Enable
RecEmpty	1.0	Boolean	R	Open, Claim, & Enable
RecNearEnd	1.0	Boolean	R	Open, Claim, & Enable
RecSidewaysMaxLines	1.0	Long	R	Open, Claim, & Enable
RecSidewaysMaxChars	1.0	Long	R	Open, Claim, & Enable
RecLinesToPaperCut	1.0	Long	R	Open, Claim, & Enable
RecBarCodeRotationList	1.1	String	R	Open
SlpLineChars	1.0	Long	R/W	Open, Claim, & Enable
SlpLineCharsList	1.0	String	R	Open
SlpLineHeight	1.0	Long	R/W	Open, Claim, & Enable
SlpLineSpacing	1.0	Long	R/W	Open, Claim, & Enable
SlpLineWidth	1.0	Long	R	Open, Claim, & Enable
SlpLetterQuality	1.0	Boolean	R/W	Open, Claim, & Enable
SlpEmpty	1.0	Boolean	R	Open, Claim, & Enable
SlpNearEnd	1.0	Boolean	R	Open, Claim, & Enable
SlpSidewaysMaxLines	1.0	Long	R	Open, Claim, & Enable
SlpSidewaysMaxChars	1.0	Long	R	Open, Claim, & Enable
SlpMaxLines	1.0	Long	R	Open, Claim, & Enable
SlpLinesNearEndToEnd	1.0	Long	R	Open, Claim, & Enable

SlpBarcodeRotationList 1.1 String R Open

Methods*Common*

		<i>May Use After</i>
Open	1.0	--
Close	1.0	Open
Claim	1.0	Open
Release	1.0	Open & Claim
CheckHealth	1.0	Open, Claim, & Enable
ClearInput	1.0	<i>Not Supported</i>
ClearOutput	1.0	Open, Claim, & Enable
DirectIO	1.0	Open

Specific

PrintNormal	1.0	Open, Claim, & Enable
PrintTwoNormal	1.0	Open, Claim, & Enable
PrintImmediate	1.0	Open, Claim, & Enable
BeginInsertion	1.0	Open, Claim, & Enable
EndInsertion	1.0	Open, Claim, & Enable
BeginRemoval	1.0	Open, Claim, & Enable
EndRemoval	1.0	Open, Claim, & Enable
CutPaper	1.0	Open, Claim, & Enable
RotatePrint	1.0	Open, Claim, & Enable
PrintBarCode	1.0	Open, Claim, & Enable
PrintBitmap	1.0	Open, Claim, & Enable
TransactionPrint	1.1	Open, Claim, & Enable
ValidateData	1.1	Open, Claim, & Enable
SetBitmap	1.0	Open, Claim, & Enable
SetLogo	1.0	Open, Claim, & Enable

Events*Name*

	<i>May Occur After</i>
DataEvent	<i>Not Supported</i>
DirectIOEvent	Open
ErrorEvent	Open, Claim, & Enable
OutputCompleteEvent	Open, Claim, & Enable
StatusUpdateEvent	Open, Claim, & Enable

General Information

The POS Printer Control's OLE programmatic ID is "OPOS.POSPrinter".

The printer OLE Control does not attempt to encapsulate the generic Windows graphics printer. Rather, for performance and ease of use considerations, the interfaces are defined to directly control a printer. Usually, an application will print one line to one station per method, for ease of use and accuracy in recovering from errors.

The printer model defines three stations with the following general uses:

- **Journal** Used for simple text to log transaction and activity information. Kept by the store for audit and other purposes.
- **Receipt** Used to print transaction information. Usually given to the customer. Also often used for store reports. Contains either a knife to cut the paper between transactions, or a tear bar to manually cut the paper.
- **Slip** Used to print information on a form. Usually given to the customer. Also used to print "validation" information on a form. The form type is typically a check or credit card slip.

Sometimes, limited forms-handling capability is integrated with the receipt or journal station to permit validation printing. Often this limits the number of print lines, due to the station's forms-handling throat depth. The Printer Control nevertheless addresses this printer functionality as a slip station.

Capabilities

The POS printer has the following capability:

- The default character set can print the ASCII characters 0x20 through 0x7F, which includes space, digits, uppercase, lowercase, and some special characters. (If the printer does not support all of these, then it should translate them to close approximations – such as lowercase to uppercase.)

The POS printer may have several additional capabilities. See the capabilities properties for specific information.

The following capabilities are not addressed in this version of the OPOS specification. A Service Object may choose to support them through the **DirectIO** mechanism.

- Downloadable character sets.
- Character substitution.
- General graphics printing, where each pixel of the printer line may be specified.

Model

The POS Printer follows the general output model, with some enhancements:

- The following methods are always performed synchronously: **BeginInsertion**, **EndInsertion**, **BeginRemoval**, **EndRemoval**, and **CheckHealth**. These methods will fail if asynchronous output is outstanding.
- The following method is also always performed synchronously: **PrintImmediate**. This method tries to print its data immediately (that is, as the very next printer operation). It may be called when asynchronous output is outstanding. **PrintImmediate** is primarily intended for use in exception conditions when asynchronous output is outstanding.
- The following methods are performed either synchronously or asynchronously, depending on the value of the **AsyncMode** property: **PrintNormal**, **PrintTwoNormal**, **CutPaper**, **RotatePrint**, **PrintBarCode**, and **PrintBitmap**. When **AsyncMode** is FALSE, then these methods print synchronously and return their completion status to the application.
- When **AsyncMode** is TRUE, then these methods operate as follows:
 - ◆ The Control buffers the request, sets the **OutputID** property to an identifier for this request, and returns as soon as possible. When the device completes the request successfully, then the Control fires an **OutputCompleteEvent**. A parameter of this event contains the **OutputID** of the completed request.

Asynchronous printer methods will not return an error status due to a printing problem, such as out of paper or printer fault. These errors will only be reported by an **ErrorEvent**. An error status is returned only if the printer is not claimed and enabled, a parameter is invalid, or the request cannot be enqueued. The first two error cases are due to an application error, while the last is a serious system resource exception.

- ◆ If an error occurs while performing an asynchronous request, an **ErrorEvent** is fired. The **ErrorStation** property is set to the station or stations that were printing when the error occurred. Beginning with Release 1.1, the **ErrorLevel** and **ErrorString** properties are also set. The event handler may call synchronous print methods (but not asynchronous methods), then can either retry the outstanding output or clear it.
 - ◆ OPOS guarantees that asynchronous output is performed on a first-in first-out basis.
 - ◆ All output buffered by OPOS may be deleted by calling the **ClearOutput** method. **OutputCompleteEvents** will not be fired for cleared output. This method also stops any output that may be in progress (when possible).
 - ◆ The property **FlagWhenIdle** may be set to cause the Control to fire a **StatusUpdateEvent** when all outstanding outputs have finished, whether successfully or because they were cleared.
- Beginning with Release 1.1, transaction mode printing is supported. A transaction is a sequence of print operations that are printed to a station as a unit. Print operations which may be included in a transaction are **PrintNormal**, **CutPaper**, **RotatePrint**, **PrintBarCode**, and **PrintBitmap**. During a transaction, the print operations are first validated. If valid, they are added to the transaction but not printed yet. Once the application has added as many operations as required, then the transaction print method is called. If the transaction is printed synchronously, then the returned status indicates either that the entire transaction printing successfully or that an error occurred during the print. If the transaction is printed asynchronously, then the asynchronous print rules listed above are followed. If an error occurs and the Error Event handler causes a retry, the entire transaction is retried.

The printer error reporting model is as follows:

- Printer out-of-paper and cover open conditions are reported by setting the **ResultCode** to **OPOS_E_EXTENDED** and then setting **ResultCodeExtended** to one of the following error conditions:
OPOS_EPTR_JRN_EMPTY,
OPOS_EPTR_REC_EMPTY,
OPOS_EPTR_SLP_EMPTY, or
OPOS_EPTR_COVER_OPEN.
- Other printer errors are reported by setting the **ResultCode** to **OPOS_E_FAILURE** or another standard error status. These failures are typically due to a printer fault or jam, or to a more serious error.

Device Sharing

The POS Printer is an exclusive-use device, as follows:

- The application must claim the device before enabling it.
- The application must claim and enable the device before accessing many printer-specific properties.
- The application must claim and enable the device before calling methods that manipulate the device.
- See the “Summary” table for precise usage prerequisites.

Data Characters and Escape Sequences

The default character set of all POS printers is assumed to support at least the ASCII characters 20-hex through 7F-hex, which include spaces, digits, uppercase, lowercase, and some special characters. If the printer does not support lowercase characters, then the Service Object may translate them to uppercase.

Every escape sequence begins with the escape character ESC, whose value is 27 decimal, followed by a vertical bar ('|'). This is followed by zero or more digits and/or lowercase alphabetic characters. The escape sequence is terminated by an uppercase alphabetic character.

To determine if an escape sequences or data can be performed on a printer station, the application can call the **ValidateData** method. (For some escape sequences, corresponding capability properties can also be used.)

The following escape sequences are recognized. If an escape sequence specifies an operation that is not supported by the printer station, then it is ignored.

One Shots Perform indicated action.

Name	Data	Remarks
Paper cut	ESC #P	Cuts receipt paper. The character '#' is replaced by an ASCII decimal string telling the percentage cut desired. If '#' is omitted, then a full cut is performed. For example: The C string "\x1B 75P" requests a 75% partial cut.
Feed and Paper cut	ESC #FP	Cuts receipt paper, after feeding the paper by the ReclinesToPaperCut lines. The character '#' is defined by the "Paper cut" escape sequence.
Feed, Paper cut, and Stamp	ESC #sP	Cuts and stamps receipt paper, after feeding the paper by the ReclinesToPaperCut lines. The character '#' is defined by the "Paper cut" escape sequence.
Fire stamp	ESC sL	Fires the stamp solenoid, which usually contains a graphical store emblem.
Print bitmap	ESC #B	Prints the pre-stored bitmap. The character '#' is replaced by the bitmap number.
Print top logo	ESC tL	Prints the pre-stored top logo.

Print bottom logo	ESC bL	Prints the pre-stored bottom logo.
Feed lines	ESC #lF	Feed the paper forward by lines. The character '#' is replaced by an ASCII decimal string telling the number of lines to be fed. If '#' is omitted, then one line is fed.
Feed units	ESC #uF	Feed the paper forward by mapping mode units. The character '#' is replaced by an ASCII decimal string telling the number of units to be fed. If '#' is omitted, then one unit is fed.
Feed reverse	ESC #rF	Feed the paper backward. The character '#' is replaced by an ASCII decimal string telling the number of lines to be fed. If '#' is omitted, then one line is fed.

Print Mode Characteristics that are remembered until explicitly changed.

Name	Data	Remarks
Font typeface selection	ESC #fT	Selects a new typeface for the following data. Values for the character '#' are: 0 = Default typeface. 1 = Select first typeface from the FontTypefaceList property. 2 = Select second typeface from the FontTypefaceList property. And so on.

Print Line Characteristics that are reset at the end of each print method or by a "Normal" sequence.

Name	Data	Remarks
Bold	ESC bC	Prints in bold or double-strike.
Underline	ESC #uC	Prints with underline. The character '#' is replaced by an ASCII decimal string telling the width of the underline in printer dot units. If '#' is omitted, then a printer-specific default width is used.
Italic	ESC iC	Prints in italics.

Alternate color (Red)	ESC rC	Prints in alternate color.
Reverse video	ESC rvC	Prints in a reverse video format.
Shading	ESC #sC	Prints in a shaded manner. The character '#' is replaced by an ASCII decimal string telling the percentage shading desired. If '#' is omitted, then a printer-specific default level of shading is used.
Single high & wide	ESC 1C	Prints normal size.
Double wide	ESC 2C	Prints double-wide characters.
Double high	ESC 3C	Prints double-high characters.
Double high & wide	ESC 4C	Prints double-high/double-wide characters.
Scale horizontally	ESC #hC	Prints with the width scaled '#' times the normal size, where '#' is replaced by an ASCII decimal string.
Scale vertically	ESC #vC	Prints with the height scaled '#' times the normal size, where '#' is replaced by an ASCII decimal string.
Center	ESC cA	Aligns following text in the center.
Right justify	ESC rA	Aligns following text at the right.
Normal	ESC N	Restores printer characteristics to normal condition.

Properties

AsyncMode Property R/W

Syntax **BOOL AsyncMode;**

Remarks If TRUE, then the print methods **PrintNormal**, **PrintTwoNormal**, **CutPaper**, **RotatePrint**, **PrintBarCode**, and **PrintBitmap** will be performed asynchronously. If FALSE, they will be printed synchronously.

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

Return When this property is set, the following value is placed in the **ResultCode** property:

Value	Meaning
OPOS_SUCCESS	The property was set successfully.

CapCharacterSet Property *Added in Release 1.1*

Syntax **LONG CapCharacterSet;**

Remarks Holds the default character set capability. It may be one of the following:

Value	Meaning
PTR_CCS_ALPHA	The default character set supports uppercase alphabetic plus numeric, space, minus, and period.
PTR_CCS_ASCII	The default character set supports all ASCII characters between 20-hex and 7F-hex.
PTR_CCS_KANA	The default character set supports partial code page 932, including ASCII characters 20-hex through 7F-hex and the Japanese Kana characters A1-hex through DF-hex, but excluding the Japanese Kanji characters.
PTR_CCS_KANJI	The default character set supports code page 932, including the Shift-JIS Kanji characters, Levels 1 and 2.

The default character set may contain a superset of these ranges. The initial **CharacterSet** property may be examined for additional information.

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

CapConcurrentJrnRec Property

Syntax **BOOL CapConcurrentJrnRec;**

Remarks If TRUE, then the Journal and Receipt stations can print at the same time. The **PrintTwoNormal** method may be used with the PTR_S_JOURNAL_RECEIPT station parameter.

If FALSE, the application should print to only one of the stations at a time, and minimize transitions between the stations. This will result in better performance, better reliability (such as less jams), or both.

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

CapConcurrentJrnSlp Property

- Syntax** **BOOL CapConcurrentJrnSlp;**
- Remarks** If TRUE, then the Journal and Slip stations can print at the same time. The **PrintTwoNormal** method may be used with the PTR_S_JOURNAL_SLIP station parameter.
- If FALSE, the application should insert form / print / remove form before printing to the journal station.
- This property is initialized by the **Open** method.

CapConcurrentRecSlp Property

- Syntax** **BOOL CapConcurrentRecSlp;**
- Remarks** If TRUE, then the Receipt and Slip stations can print at the same time. The **PrintTwoNormal** method may be used with the PTR_S_RECEIPT_SLIP station parameter.
- If FALSE, the application should insert form / print / remove form before printing to the receipt station.
- This property is initialized by the **Open** method.

CapCoverSensor Property

- Syntax** **BOOL CapCoverSensor;**
- Remarks** If TRUE, then the printer has a “cover open” sensor; otherwise it is FALSE.
- This property is initialized by the **Open** method.

CapJrn2Color Property

Syntax **BOOL CapJrn2Color;**

Remarks If TRUE, then the journal can print dark plus an alternate color;
otherwise it is FALSE.

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

CapJrnBold Property

Syntax **BOOL CapJrnBold;**

Remarks If TRUE, then the journal can print bold characters;
otherwise it is FALSE.

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

CapJrnDhigh Property

Syntax **BOOL CapJrnDhigh;**

Remarks If TRUE, then the journal can print double high characters;
otherwise it is FALSE.

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

CapJrnDwide Property

Syntax **BOOL CapJrnDwide;**

Remarks If TRUE, then the journal can print double wide characters;
otherwise it is FALSE.

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

CapJrnDwideDhigh Property

Syntax **BOOL CapJrnDwideDhigh;**

Remarks If TRUE, then the journal can print double high / double wide characters; otherwise it is FALSE.

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

CapJrnEmptySensor Property

Syntax **BOOL CapJrnEmptySensor;**

Remarks If TRUE, then the journal has an out-of-paper sensor; otherwise it is FALSE.

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

CapJrnItalic Property

Syntax **BOOL CapJrnItalic;**

Remarks If TRUE, then the journal can print italic characters; otherwise it is FALSE.

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

CapJrnNearEndSensor Property

Syntax **BOOL CapJrnNearEndSensor;**

Remarks If TRUE, then the journal has a low paper sensor; otherwise it is FALSE.

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

CapJrnPresent Property

Syntax **BOOL CapJrnPresent;**

Remarks If TRUE, then the journal print station is present;
 otherwise it is FALSE.

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

CapJrnUnderline Property

Syntax **BOOL CapJrnUnderline;**

Remarks If TRUE, then the journal can underline characters;
 otherwise it is FALSE.

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

CapRec2Color Property

Syntax **BOOL CapRec2Color;**

Remarks If TRUE, then the receipt can print dark plus an alternate color; otherwise it is FALSE.

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

CapRecBarCode Property

Syntax **BOOL CapRecBarCode;**

Remarks If TRUE, then the receipt has bar code printing capability; otherwise it is FALSE.

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

CapRecBitmap Property

Syntax **BOOL CapRecBitmap;**

Remarks If TRUE, then the receipt can print bitmaps; otherwise it is FALSE.

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

CapRecBold Property

Syntax **BOOL CapRecBold;**

Remarks If TRUE, then the receipt can print bold characters; otherwise it is FALSE.

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

CapRecDhigh Property

Syntax **BOOL CapRecDhigh;**

Remarks If TRUE, then the receipt can print double high characters;
otherwise it is FALSE.

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

CapRecDwide Property

Syntax **BOOL CapRecDwide;**

Remarks If TRUE, then the receipt can print double wide characters;
otherwise it is FALSE.

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

CapRecDwideDhigh Property

Syntax **BOOL CapRecDwideDhigh;**

Remarks If TRUE, then the receipt can print double high / double wide characters;
otherwise it is FALSE.

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

CapRecEmptySensor Property

Syntax **BOOL CapRecEmptySensor;**

Remarks If TRUE, then the receipt has an out-of-paper sensor;
otherwise it is FALSE.

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

CapRecItalic Property

- Syntax** **BOOL CapRecItalic;**
- Remarks** If TRUE, then the receipt can print italic characters;
 otherwise it is FALSE.
- This property is initialized by the **Open** method.

CapRecLeft90 Property

- Syntax** **BOOL CapRecLeft90;**
- Remarks** If TRUE, then the receipt can print in rotated 90 ° left mode;
 otherwise it is FALSE.
- This property is initialized by the **Open** method.

CapRecNearEndSensor Property

- Syntax** **BOOL CapRecNearEndSensor;**
- Remarks** If TRUE, then the receipt has a low paper sensor;
 otherwise it is FALSE.
- This property is initialized by the **Open** method.

CapRecPapercut Property

- Syntax** **BOOL CapRecPapercut;**
- Remarks** If TRUE, then the receipt can perform paper cuts;
 otherwise it is FALSE.
- This property is initialized by the **Open** method.

CapRecPresent Property

- Syntax** **BOOL CapRecPresent;**
- Remarks** If TRUE, then the receipt print station is present;
 otherwise it is FALSE.
- This property is initialized by the **Open** method.

CapRecRight90 Property

- Syntax** **BOOL CapRecRight90;**
- Remarks** If TRUE, then the receipt can print in a rotated 90° right mode;
 otherwise it is FALSE.
- This property is initialized by the **Open** method.

CapRecRotate180 Property

- Syntax** **BOOL CapRecRotate180;**
- Remarks** If TRUE, then the receipt can print in a rotated upside down mode;
 otherwise it is FALSE.
- This property is initialized by the **Open** method.

CapRecStamp Property

- Syntax** **BOOL CapRecStamp;**
- Remarks** If TRUE, then the receipt has a stamp capability;
 otherwise it is FALSE.
- This property is initialized by the **Open** method.

CapRecUnderline Property

- Syntax** **BOOL CapRecUnderline;**
- Remarks** If TRUE, then the receipt can underline characters;
 otherwise it is FALSE.
- This property is initialized by the **Open** method.

CapSlp2Color Property

- Syntax** **BOOL CapSlp2Color;**
- Remarks** If TRUE, then the slip can print dark plus an alternate color;
 otherwise it is FALSE.
- This property is initialized by the **Open** method.

CapSlpBarCode Property

- Syntax** **BOOL CapSlpBarCode;**
- Remarks** If TRUE, then the slip has bar code printing capability;
 otherwise it is FALSE.
- This property is initialized by the **Open** method.

CapSlpBitmap Property

- Syntax** **BOOL CapSlpBitmap;**
- Remarks** If TRUE, then the slip can print bitmaps;
 otherwise it is FALSE.
- This property is initialized by the **Open** method.

CapSlpBold Property

Syntax **BOOL CapSlpBold;**

Remarks If TRUE, then the slip can print bold characters;
otherwise it is FALSE.

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

CapSlpDhigh Property

Syntax **BOOL CapSlpDhigh;**

Remarks If TRUE, then the slip can print double high characters;
otherwise it is FALSE.

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

CapSlpDwide Property

Syntax **BOOL CapSlpDwide;**

Remarks If TRUE, then the slip can print double wide characters;
otherwise it is FALSE.

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

CapSlpDwideDhigh Property

Syntax **BOOL CapSlpDwideDhigh;**

Remarks If TRUE, then the slip can print double high / double wide characters;
otherwise it is FALSE.

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

CapSlpEmptySensor Property

Syntax **BOOL CapSlpEmptySensor;**

Remarks If TRUE, then the slip has a “slip in” sensor; otherwise it is FALSE.

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

CapSlpFullslip Property

Syntax **BOOL CapSlpFullslip;**

Remarks If TRUE, then the slip is a full slip station. It can print full-length forms..

If FALSE, then the slip is a “validation” type station. This usually limits the number of print lines, and disables access to the receipt and/or journal stations while the validation slip is being used.

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

CapSlpItalic Property

Syntax **BOOL CapSlpItalic;**

Remarks If TRUE, then the slip can print italic characters; otherwise it is FALSE.

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

CapSlpLeft90 Property

Syntax **BOOL CapSlpLeft90;**

Remarks If TRUE, then the slip can print in a rotated 90 ° left mode; otherwise it is FALSE.

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

CapSlpNearEndSensor Property

- Syntax** **BOOL CapSlpNearEndSensor;**
- Remarks** If TRUE, then the slip has a “slip near end” sensor;
otherwise it is FALSE.
- This property is initialized by the **Open** method.

CapSlpPresent Property

- Syntax** **BOOL CapSlpPresent;**
- Remarks** If TRUE, then the slip print station is present;
otherwise it is FALSE.
- This property is initialized by the **Open** method.

CapSlpRight90 Property

- Syntax** **BOOL CapSlpRight90;**
- Remarks** If TRUE, then the slip can print in a rotated 90 ° right mode;
otherwise it is FALSE.
- This property is initialized by the **Open** method.

CapSlpRotate180 Property

- Syntax** **BOOL CapSlpRotate180;**
- Remarks** If TRUE, then the slip can print in a rotated upside down mode;
otherwise it is FALSE.
- This property is initialized by the **Open** method.

CapSlpUnderline Property

Syntax **BOOL CapSlpUnderline;**

Remarks If TRUE, then the slip can underline characters;
 otherwise it is FALSE.

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

CapTransaction Property

Added in Release 1.1

Syntax **BOOL CapTransaction;**

Remarks If TRUE, then printer transactions are supported by each station;
 otherwise it is FALSE.

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

CharacterSet Property R/W

Syntax **LONG CharacterSet;**

Remarks The character set for printing characters.

This property is initialized when the device is first enabled following the **Open** method.

Values are:

Value	Meaning
Range 101 - 199	Device-specific character sets that do not match a code page or the ASCII or Windows ANSI character sets.
Range 400 - 990	Code page; matches one of the standard values.
PTR_CS_ASCII	The ASCII character set, supporting the ASCII characters between 0x20 and 0x7F. The value of this constant is 998.
PTR_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.

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.
OPOS_E_ILLEGAL	An invalid property value was used.
<i>Other Values</i>	See ResultCode .

See Also **CharacterSetList** Property

CharacterSetList Property

Syntax **BSTR CharacterSetList;**

Remarks 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 **CharacterSet** Property

CoverOpen Property

Syntax **BOOL CoverOpen;**

Remarks If TRUE, then the printer’s cover is open; otherwise it is FALSE.

If the **CapCoverSensor** property is FALSE, then the printer does not have a cover open sensor, and this property always returns FALSE.

This property is initialized and kept current while the device is enabled.

ErrorLevel Property *Added in Release 1.1*

Syntax **LONG ErrorLevel;**
Remarks The severity of the error condition.

Values are:

Value	Meaning
PTR_EL_NONE	No error condition is present.
PTR_EL_RECOVERABLE	A recoverable error has occurred. (Example: Out of paper.)
PTR_EL_FATAL	A non-recoverable error has occurred. (Example: Internal printer failure.)

This property is set by the Control just before firing an **ErrorEvent**. When the error is cleared, then the property is changed to PTR_EL_NONE.

ErrorStation Property

Syntax **LONG ErrorStation;**
Remarks Holds the station or stations that were printing when an error was detected.

This property will be set to one of the following values: PTR_S_JOURNAL, PTR_S_RECEIPT, PTR_S_SLIP, PTR_S_JOURNAL_RECEIPT, PTR_S_JOURNAL_SLIP, or PTR_S_RECEIPT_SLIP.

This property is set just before an **ErrorEvent** is fired.

ErrorString Property***Added in Release 1.1*****Syntax** **BSTR ErrorString;****Remarks** A vendor-supplied description of the current error.

This property is set by the Control just before firing an **ErrorEvent**. If no description is available, the property is set to an empty string. When the error is cleared, then the property is changed to an empty string.

FlagWhenIdle Property R/W**Syntax** **BOOL FlagWhenIdle;****Remarks** If TRUE, the Control will fire a **StatusUpdateEvent** if it is in the idle state. If FALSE, this event will not be fired.

FlagWhenIdle is automatically reset to FALSE when the status event is fired.

The main use of idle status event that is controlled by this property is to give the application control when all outstanding asynchronous outputs have been processed. The event will be fired if the outputs were completed successfully or if they were cleared by the **ClearOutput** method or by an **ErrorEvent** handler.

If the **State** is already set to OPOS_S_IDLE when the **FlagWhenIdle** property is set to TRUE, then a **StatusUpdateEvent** is fired immediately. The application can therefore depend upon the event, with no race condition between the starting of its last asynchronous output and the setting of this flag.

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

Return When this property is set, the following value is placed in the **ResultCode** property:

Value	Meaning
OPOS_SUCCESS	The property was set successfully.

FontTypefaceList Property *Added in Release 1.1*

Syntax **BSTR FontTypefaceList;**

Remarks A string that specifies the fonts and/or typefaces that are supported by the printer.

This property is initialized by the **Open** method. The string consists of font or typeface names separated by commas. The application selects a font or typeface for a printer station by using the font typeface selection escape sequence (ESC|#fT). The “#” character is replaced by the number of the font or typeface within the list: 1, 2, and so on.

In Japan, this property will frequently include the fonts “Mincho” and “Gothic”. Other fonts or typefaces may be commonly supported in other countries.

An empty string indicates that only the default typeface is supported.

See Also “Data Characters and Escape Sequences”

JrnEmpty Property

Syntax **BOOL JrnEmpty;**

Remarks If TRUE, the journal is out of paper.
If FALSE, journal paper is present.

If the capability **CapJrnEmptySensor** is FALSE, then the value of this property is always FALSE.

This property is initialized and kept current while the device is enabled.

See Also **JNearEnd** Property

JrnLetterQuality Property R/W

Syntax **BOOL JrnLetterQuality;**

Remarks If TRUE, prints in high quality mode.
 If FALSE, prints in high speed mode.

This property advises the Service Object that either high quality or high speed printing is desired. For example, printers with bi-directional print capability may be placed in unidirectional mode for high quality, so that column alignment is more precise.

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

Return When this property is set, the following value is placed in the **ResultCode** property:

Value	Meaning
OPOS_SUCCESS	The property was set successfully.

JrnLineChars Property R/W

Syntax **LONG JrnLineChars;**

Remarks The number of characters that may be printed on a journal line.

If changed to a line character width that can be supported, then the width is set to the specified value. If the exact width cannot be supported, then subsequent lines will be printed with a character size that most closely supports the specified characters per line.

If the character width cannot be supported, then an error is returned.

Setting **JrnLineChars** may also update **JrnLineWidth**, **JrnLineHeight**, and **JrnLineSpacing**, since the character pitch or font may be changed.

The value of **JrnLineChars** is initialized to the printer's default line character width 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.
OPOS_E_ILLEGAL	An invalid line character width was specified.

See Also **JrnLineCharsList** Property

JrnLineCharsList Property

Syntax **BSTR JrnLineCharsList;**

Remarks A string containing the line character widths supported by the journal station.

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 "32,36,40", then the station supports line widths of 32, 36, and 40 characters.

See Also **JrnLineChars** Property

JrnLineHeight Property R/W

Syntax **LONG JrnLineHeight;**

Remarks The journal print line height. Expressed in the unit of measure given by **MapMode**.

If changed to a height that can be supported with the current character width, then the line height is set to this value. If the exact height cannot be supported, then the height is set to the closest supported value.

When **JrnLineChars** is changed, **JrnLineHeight** is updated to the default line height for the selected width.

The value of **JrnLineHeight** is initialized to the printer's default line height when the device is first enabled following the **Open** method.

Return When this property is set, the following value is placed in the **ResultCode** property:

Value	Meaning
OPOS_SUCCESS	The property was set successfully.

JrnLineSpacing Property R/W

Syntax **LONG JrnLineSpacing;**

Remarks The space between journal print lines. Expressed in the unit of measure given by **MapMode**.

If changed to a spacing that can be supported by the printer, then the line spacing is set to this value. If the spacing cannot be supported, then the spacing is set to the closest supported value.

When **JrnLineChars** or **JrnLineHeight** is changed, **JrnLineSpacing** is updated to the default line spacing for the selected width or height.

The value of **JrnLineSpacing** is initialized to the printer's default line spacing when the device is first enabled following the **Open** method.

Return When this property is set, the following value is placed in the **ResultCode** property:

Value	Meaning
OPOS_SUCCESS	The property was set successfully.

JrnLineWidth Property

Syntax **LONG JrnLineWidth;**

Remarks The width of a line of **JrnLineChars** characters. Expressed in the unit of measure given by **MapMode**.

Setting **JrnLineChars** may also update **JrnLineWidth**.

The value of **JrnLineWidth** is initialized to the printer's default line width when the device is first enabled following the **Open** method.

JrnNearEnd Property

Syntax **BOOL JrnNearEnd;**

Remarks If TRUE, the journal paper is low.
 If FALSE, journal paper is not low.

If the capability **CapJrnNearEndSensor** is FALSE, then the value of this property is always FALSE.

This property is initialized and kept current while the device is enabled.

See Also **JrnEmpty** Property

MapMode Property R/W

Syntax LONG MapMode;

Remarks Contains the mapping mode of the printer. The mapping mode defines the unit of measure used for other properties, such as line heights and line spacings.

The following map modes are supported:

Value	Meaning
PTR_MM_DOTS	The printer's dot width. This width may be different for each printer station.
PTR_MM_TWIPS	1/1440 of an inch.
PTR_MM_ENGLISH	0.001 inch.
PTR_MM_METRIC	0.01 millimeter.

Setting **MapMode** may also change **JrnLineHeight**, **JrnLineSpacing**, **JrnLineWidth**, **RecLineHeight**, **RecLineSpacing**, **RecLineWidth**, **SlpLineHeight**, **SlpLineSpacing**, and **SlpLineWidth**.

The value of **MapMode** is initialized to PTR_MM_DOTS when the device is first enabled following the **Open** method.

Return When this property is set, the following value is placed in the **ResultCode** property:

Value	Meaning
OPOS_SUCCESS	The property was set successfully.
OPOS_E_ILLEGAL	An invalid mapping mode was specified.

RecBarCodeRotationList Property *Added in Release 1.1*

Syntax **BSTR RecBarCodeRotationList;**

Remarks A string that specifies the directions in which a receipt barcode may be rotated.

This property is initialized by the **Open** method. The string consists of rotation strings separated by commas. An empty string indicates that bar code printing is not supported. The legal rotation strings are:

Value	Meaning
0	Bar code may be printed in the normal orientation.
R90	Bar code may be rotated 90° to the right.
L90	Bar code may be rotated 90° to the left.
180	Bar code may be rotated 180° - upside down.

For example, if the string is “0,180”, then the printer can print normal bar codes and upside down bar codes.

See Also **RotateSpecial** Property; **PrintBarCode** Method

RecEmpty Property

Syntax **BOOL RecEmpty;**

Remarks If TRUE, the receipt is out of paper.
If FALSE, receipt paper is present.

If the capability **CapRecEmptySensor** is FALSE, then the value of this property is always FALSE.

This property is initialized and kept current while the device is enabled.

See Also **RecNearEnd** Property

RecLetterQuality Property R/W

Syntax **BOOL RecLetterQuality;**

Remarks If TRUE, prints in high quality mode.
 If FALSE, prints in high speed mode.

This property advises the Service Object that either high quality or high speed printing is desired.

For example:

- Printers with bi-directional print capability may be placed in unidirectional mode for high quality, so that column alignment is more precise.
- Bitmaps may be printed in a high-density graphics mode for high-quality, and in a low-density mode for high speed.

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

Return When this property is set, the following value is placed in the **ResultCode** property:

Value	Meaning
OPOS_SUCCESS	The property was set successfully.

RecLineChars Property R/W

Syntax **LONG RecLineChars;**

Remarks The number of characters that may be printed on a receipt line.

If changed to a line character width that can be supported, then the width is set to the specified value. If the exact width cannot be supported, then subsequent lines will be printed with a character size that most closely supports the specified characters per line.

If the character width cannot be supported, then an error is returned.

Setting **RecLineChars** may also update **RecLineWidth**, **RecLineHeight**, and **RecLineSpacing**, since the character pitch or font may be changed.

The value of **RecLineChars** is initialized to the printer's default line character width 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.
OPOS_E_ILLEGAL	An invalid line character width was specified.

See Also **RecLineCharsList** Property

RecLineCharsList Property

Syntax **BSTR RecLineCharsList;**

Remarks A string containing the line character widths supported by the receipt station.

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 "32,36,40", then the station supports line widths of 32, 36, and 40 characters.

See Also **RecLineChars** Property

RecLineHeight Property R/W

Syntax **LONG RecLineHeight;**

Remarks The receipt print line height. Expressed in the unit of measure given by **MapMode**.

If changed to a height that can be supported with the current character width, then the line height is set to this value. If the exact height cannot be supported, then the height is set to the closest supported value.

When **RecLineChars** is changed, **RecLineHeight** is updated to the default line height for the selected width.

The value of **RecLineHeight** is initialized to the printer's default line height when the device is first enabled following the **Open** method.

Return When this property is set, the following value is placed in the **ResultCode** property:

Value	Meaning
OPOS_SUCCESS	The property was set successfully.

See Also **RecLineChars** Property

RecLineSpacing Property R/W

Syntax	LONG RecLineSpacing;				
Remarks	<p>The space between receipt print lines. Expressed in the unit of measure given by MapMode.</p> <p>If changed to a spacing that can be supported by the printer, then the line spacing is set to this value. If the spacing cannot be supported, then the spacing is set to the closest supported value.</p> <p>When RecLineChars or RecLineHeight are changed, RecLineSpacing is updated to the default line spacing for the selected width or height.</p> <p>The value of RecLineSpacing is initialized to the printer's default line spacing when the device is first enabled following the Open method.</p>				
Return	When this property is set, the following value is placed in the ResultCode property:				
	<table> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>OPOS_SUCCESS</td> <td>The property was set successfully.</td> </tr> </tbody> </table>	Value	Meaning	OPOS_SUCCESS	The property was set successfully.
Value	Meaning				
OPOS_SUCCESS	The property was set successfully.				

RecLinesToPaperCut Property

Syntax	LONG RecLinesToPaperCut;
Remarks	<p>Holds the number of lines that must be advanced before the receipt paper is cut.</p> <p>If the capability CapRecPapercut is TRUE, then this is the line count before reaching the paper cut mechanism. Otherwise, this is the line count before the manual tear-off bar.</p> <p>Changing the properties RecLineChars, RecLineHeight, and RecLineSpacing may cause this property to change.</p> <p>This property is initialized when the device is first enabled following the Open method.</p>

RecLineWidth Property

Syntax **LONG RecLineChars;**

Remarks The width of a line of **RecLineChars** characters. Expressed in the unit of measure given by **MapMode**.

Setting **RecLineChars** may also update **RecLineWidth**.

The value of **RecLineWidth** is initialized to the printer's default line width when the device is first enabled following the **Open** method.

RecNearEnd Property

Syntax **BOOL RecNearEnd;**

Remarks If TRUE, the receipt paper is low.
If FALSE, receipt paper is not low.

If the capability **CapRecNearEndSensor** is FALSE, then the value of this property is always FALSE.

This property is initialized and kept current while the device is enabled.

See Also **RecEmpty** Property

RecSidewaysMaxChars Property

Syntax **LONG RecSidewaysMaxChars;**

Remarks Holds the maximum number of characters that may be printed on each line in sideways mode.

If the capabilities **CapRecLeft90** and **CapRecRight90** are both FALSE, then **RecSidewaysMaxChars** is zero.

Changing the properties **RecLineHeight**, **RecLineSpacing**, and **RecLineChars** may cause this property to change.

This property is initialized when the device is first enabled following the **Open** method.

See Also **RecSidewaysMaxLines** Property

RecSidewaysMaxLines Property

Syntax **LONG RecSidewaysMaxLines;**

Remarks Holds the maximum number of lines that may be printed in sideways mode.

If the capabilities **CapRecLeft90** and **CapRecRight90** are both FALSE, then **RecSidewaysMaxLines** is zero.

Changing the properties **RecLineHeight**, **RecLineSpacing**, and **RecLineChars** may cause this property to change.

This property is initialized when the device is first enabled following the **Open** method.

See Also **RecSidewaysMaxChars** Property

RotateSpecial Property R/W *Added in Release 1.1*

Syntax **LONG RotateSpecial;**

Remarks The rotation orientation for bar codes.

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

Values are:

Value	Meaning
PTR_RP_NORMAL	Print subsequent bar codes in normal orientation.
PTR_RP_RIGHT90	Rotate printing 90° to the right (clockwise).
PTR_RP_LEFT90	Rotate printing 90° to the left (counter-clockwise).
PTR_RP_ROTATE180	Rotate printing 180°, that is, print upside-down.

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.
OPOS_E_ILLEGAL	An invalid property value was used.

See Also **PrintBarcode** Method

SlpBarcodeRotationList Property *Added in Release 1.1*

Syntax **BSTR SlpBarcodeRotationList;**

Remarks A string that specifies the directions in which a slip barcode may be rotated.

This property is initialized by the **Open** method. The string consists of rotation strings separated by commas. An empty string indicates that bar code printing is not supported. The legal rotation strings are:

Value	Meaning
0	Bar code may be printed in the normal orientation.
R90	Bar code may be rotated 90° to the right.
L90	Bar code may be rotated 90° to the left.
180	Bar code may be rotated 180° - upside down.

For example, if the string is “0,180”, then the printer can print normal bar codes and upside down bar codes.

See Also **RotateSpecial** Property; **PrintBarcode** Method

SlpEmpty Property

Syntax **BOOL SlpEmpty;**

Remarks If TRUE, a slip form is not present.
 If FALSE, a slip form is present.

If the capability **CapSlpEmptySensor** is FALSE, then the value of this property is always FALSE.

This property is initialized and kept current while the device is enabled.

Note The “slip empty” sensor should be used primarily to determine whether a form has been inserted before printing, and can be monitored to determine whether a form is still in place. This sensor is usually placed one or more print lines above the slip print head.

However, the “slip near end” sensor (when present) should be used to determine when nearing the end of the slip. This sensor is usually placed one or more print lines below the slip print head.

See Also **SlpNearEnd** Property

SlpLetterQuality Property R/W

Syntax **BOOL SlpLetterQuality;**

Remarks If TRUE, prints in high quality mode.
If FALSE, prints in high speed mode.

This property advises the Service Object that either high quality or high speed printing is desired.

For example:

- Printers with bi-directional print capability may be placed in unidirectional mode for high quality, so that column alignment is more precise.
- Bitmaps may be printed in a high-density graphics mode for high-quality, and in a low-density mode for high speed.

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

Return When this property is set, the following value is placed in the **ResultCode** property:

Value	Meaning
OPOS_SUCCESS	The property was set successfully.

SlpLineChars Property R/W

Syntax **LONG SlpLineChars;**

Remarks The number of characters that may be printed on a slip line.

If changed to a line character width that can be supported, then the width is set to the specified value. If the exact width cannot be supported, then subsequent lines will be printed with a character size that most closely supports the specified characters per line.

If the character width cannot be supported, then an error is returned.

Setting **SlpLineChars** may also update **SlpLineWidth**, **SlpLineHeight**, and **SlpLineSpacing**, since the character pitch or font may be changed.

The value of **SlpLineChars** is initialized to the printer's default line character width 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.
OPOS_E_ILLEGAL	An invalid line character width was specified.

See Also **SlpLineCharsList** Property

SlpLineCharsList Property

Syntax **BSTR SlpLineCharsList;**

Remarks A string containing the line character widths supported by the slip station.

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 "32,36,40", then the station supports line widths of 32, 36, and 40 characters.

See Also **SlpLineChars** Property

SlpLineHeight Property R/W

Syntax **LONG SlpLineHeight;**

Remarks The slip print-line height. Expressed in the unit of measure given by **MapMode**.

If changed to a height that can be supported with the current character width, then the line height is set to this value. If the exact height cannot be supported, then the height is set to the closest supported value.

When **SlpLineChars** is changed, **SlpLineHeight** is updated to the default line height for the selected width.

The value of **SlpLineHeight** is initialized to the printer's default line height when the device is first enabled following the **Open** method.

Return When this property is set, the following value is placed in the **ResultCode** property:

Value	Meaning
OPOS_SUCCESS	The property was set successfully.

See Also **SlpLineChars** Property

SlpLinesNearEndToEnd Property

Syntax **LONG SlpLinesNearEndToEnd;**

Remarks Holds the number of lines that may be printed after the “slip near end” sensor is TRUE but before the printer reaches the end of the slip.

This property may be used to optimize the use of the slip, so that the maximum number of lines may be printed.

Changing the **SlpLineHeight**, **SlpLineSpacing**, or **SlpLineChars** properties may cause this property to change.

This property is initialized when the device is first enabled following the **Open** method.

See Also **SlpEmpty** Property; **SlpNearEnd** Property

SlpLineSpacing Property R/W

Syntax **LONG SlpLineSpacing;**

Remarks The space between slip print lines. Expressed in the unit of measure given by **MapMode**.

If changed to a spacing that can be supported by the printer, then the line spacing is set to this value. If the spacing cannot be supported, then the spacing is set to the closest supported value.

The value of **SlpLineSpacing** is initialized to the printer's default line spacing when the device is first enabled following the **Open** method. Also, when **SlpLineChars** or **SlpLineHeight** are changed, **SlpLineSpacing** is updated to the default line spacing for the selected width or height.

Return When this property is set, the following value is placed in the **ResultCode** property:

Value	Meaning
OPOS_SUCCESS	The property was set successfully.

SlpLineWidth Property

Syntax **LONG SlpLineWidth;**

Remarks The width of a line of **SlpLineChars** characters. Expressed in the unit of measure given by **MapMode**.

Setting **SlpLineChars** may also update **SlpLineWidth**.

The value of **SlpLineWidth** is initialized to the printer's default line width when the device is first enabled following the **Open** method.

SlpMaxLines Property

Syntax **LONG SlpMaxLines;**

Remarks Holds the maximum number of lines that can be printed on a form.

When the capability **CapSlpFullslip** is TRUE, then this value will be zero, indicating an unlimited maximum slip length.

When the capability is FALSE, then this value will be non-zero.

Changing the **SlpLineHeight**, **SlpLineSpacing**, or **SlpLineChars** properties may cause this property to change.

The value of **SlpMaxLines** is initialized when the device is first enabled following the **Open** method.

SlpNearEnd Property

Syntax **BOOL SlpNearEnd;**

Remarks If TRUE, the slip form is near its end.
If FALSE, the slip form is not near its end.

The “near end” sensor is also sometimes called the “trailing edge” sensor, referring to the bottom edge of the slip.

If the capability **CapSlpNearEndSensor** is FALSE, then the value of this property is always FALSE.

This property is initialized and kept current while the device is enabled.

Note The “slip empty” sensor should be used primarily to determine whether a form has been inserted before printing, and can be monitored to determine whether a form is still in place. This sensor is usually placed one or more print lines above the slip print head.

However, the “slip near end” sensor (when present) should be used to determine when nearing the end of the slip. This sensor is usually placed one or more print lines below the slip print head.

See Also **SlpEmpty** Property; **SlpLinesNearEndToEnd** Property

SlpSidewaysMaxChars Property

Syntax **LONG SlpSidewaysMaxChars;**

Remarks Holds the maximum number of characters that may be printed on each line in sideways mode.

If the capabilities **CapSlpLeft90** and **CapSlpRight90** are both FALSE, then **SlpSidewaysMaxChars** is zero.

Changing the properties **SlpLineHeight**, **SlpLineSpacing**, and **SlpLineChars** may cause this property to change.

This property is initialized when the device is first enabled following the **Open** method.

See Also **SlpSidewaysMaxLines** Property

SlpSidewaysMaxLines Property

Syntax **LONG SlpSidewaysMaxLines;**

Remarks Holds the maximum number of lines that may be printed in sideways mode.

If the capabilities **CapSlpLeft90** and **CapSlpRight90** are both FALSE, then **SlpSidewaysMaxLines** is zero.

Changing the properties **SlpLineHeight**, **SlpLineSpacing**, and **SlpLineChars** may cause this property to change.

This property is initialized when the device is first enabled following the **Open** method.

See Also **SlpSidewaysMaxChars** Property

Methods

BeginInsertion Method

Syntax **LONG BeginInsertion (LONG *Timeout*);**

The *Timeout* parameter gives the number of milliseconds before failing the method.

Remarks Called to initiate slip processing.

When called, the slip station is made ready to receive a form by opening the form's handling "jaws" or activating a form insertion mode. This method is paired with the **EndInsertion** method for controlling form insertion.

If the printer device cannot be placed into insertion mode, an error is returned to the application. Otherwise, the Control continues to monitor form insertion until either:

- The form is successfully inserted. In this case, the Control returns an OPOS_SUCCESS status.
- The form is not inserted before *Timeout* milliseconds have elapsed, or an error is reported by the printer device. In this case, the Control either returns OPOS_E_TIMEOUT or another error. The printer device remains in form insertion mode. This allows an application to perform some user interaction and reissue the **BeginInsertion** method without altering the form handling mechanism.

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

Value	Meaning
OPOS_SUCCESS	The method was initiated successfully.
OPOS_E_BUSY	Cannot perform while output is in progress.
OPOS_E_ILLEGAL	The slip station does not exist (see the CapSlpPresent property).
OPOS_E_TIMEOUT	The specified time has elapsed without the form being properly inserted.
<i>Other Values</i>	See ResultCode .

See Also **EndInsertion** Method; **BeginRemoval** Method; **EndRemoval** Method

BeginRemoval Method

Syntax **LONG BeginRemoval (LONG *Timeout*);**

The *Timeout* property gives the number of milliseconds before failing the method.

Remarks Called to initiate form removal processing.

When called, the printer is made ready to remove a form by opening the form handling “jaws” or activating a form ejection mode. This method is paired with the **EndRemoval** method for controlling form removal.

If the printer device cannot be placed into removal or ejection mode, an error is returned to the application. Otherwise, the Control continues to monitor form removal until either:

- The form is successfully removed. In this case, the Control returns an OPOS_SUCCESS status.
- The form is not removed before *Timeout* milliseconds have elapsed, or an error is reported by the printer device. In this case, the Control either returns OPOS_E_TIMEOUT or another error. The printer device remains in form removal mode. This allows an application to perform some user interaction and reissue the **BeginRemoval** method without altering the form handling mechanism.

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

Value	Meaning
OPOS_SUCCESS	The method was initiated successfully.
OPOS_E_BUSY	Cannot perform while output is in progress.
OPOS_E_ILLEGAL	The printer does not have a slip station (see the CapSlpPresent property).
OPOS_E_TIMEOUT	The specified time has elapsed without the form being properly removed.
<i>Other Values</i>	See ResultCode .

See Also **BeginInsertion** Method; **EndInsertion** Method; **EndRemoval** Method

EndInsertion Method

Syntax **LONG EndInsertion ();**

Remarks Called to end form insertion processing.

When called, the printer is taken out of form insertion mode. If the slip device has forms “jaws,” they are closed by this method. If a form is detected in the device, a successful status of OPOS_SUCCESS is returned to the application. If no form is present, an extended error status OPOS_EPTR_NOFORM is returned.

This method is paired with the **BeginInsertion** method for controlling form insertion. The application may choose to call this method immediately after a successful **BeginInsertion** if it wants to use the printer sensors to determine when a form is positioned within the slip printer. Alternatively, the application may prompt the user and wait for a key press before calling this method.

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

Value	Meaning
OPOS_SUCCESS	The method was initiated successfully.
OPOS_E_ILLEGAL	The printer is not in slip insertion mode.
OPOS_E_EXTENDED	<p>ResultCodeExtended = OPOS_EPTR_COVER_OPEN: The device was taken out of insertion mode while the printer cover was open.</p> <p>ResultCodeExtended = OPOS_EPTR_SLP_EMPTY: The device was taken out of insertion mode without a form being inserted.</p>
<i>Other Values</i>	See ResultCode .

See Also **BeginInsertion** Method; **BeginRemoval** Method; **EndRemoval** Method

EndRemoval Method

Syntax **LONG EndRemoval ();**

Remarks Called to end form removal processing.

When called, the printer is taken out of form removal or ejection mode. If no form is detected in the device, a successful status of OPOS_SUCCESS is returned to the application. If a form is present, an extended error status OPOS_EPTR_FORM is returned.

This method is paired with the **BeginRemoval** method for controlling form removal. The application may choose to call this method immediately after a successful **BeginRemoval** if it wants to use the printer sensors to determine when the form has been removed. Alternatively, the application may prompt the user and wait for a key press before calling this method.

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

Value	Meaning
OPOS_SUCCESS	The method was initiated successfully.
OPOS_E_ILLEGAL	The printer is not in slip removal mode.
OPOS_E_EXTENDED	ResultCodeExtended = OPOS_EPTR_SLP_FORM: The device was taken out of removal mode while a form was still present.
<i>Other Values</i>	See ResultCode .

See Also **BeginInsertion** Method; **EndInsertion** Method; **BeginRemoval** Method

CutPaper Method

Syntax **LONG CutPaper (LONG *Percentage*);**

The *Percentage* parameter indicates the percentage of paper to cut. The constant identifier PTR_CP_FULLCUT or the value 100 causes a full paper cut. Other values request a partial cut percentage.

Remarks Call to cut the receipt paper.

This method is performed synchronously if **AsyncMode** is FALSE, and asynchronously if **AsyncMode** is TRUE.

Many printers with paper cut capability can perform both full and partial cuts. Some offer gradations of partial cuts, such as a perforated cut and an almost-full cut. Although the exact type of cut will vary by printer capabilities, the following general guide may be used:

Value	Meaning
100	Full cut.
90	Leave only a small portion of paper for very easy final separation.
70	Perforate the paper for final separation that is somewhat more difficult and unlikely to occur by accidental handling.
50	Partial perforation of the paper.

The Service Object will select an appropriate type of cut based on the capabilities of its device and these general guidelines.

An escape sequence embedded in a **PrintNormal** or **PrintImmediate** method call may also be used to cause a paper cut.

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_BUSY	Cannot perform while output is in progress. (Can only be returned if AsyncMode is FALSE.)
OPOS_E_ILLEGAL	An invalid percentage was specified, the receipt station does not exist (see the CapRecPresent property), or the receipt printer does not have paper cutting ability (see the CapRecPapercut property).
OPOS_E_EXTENDED	<p>ResultCodeExtended = OPOS_EPTR_COVER_OPEN: The printer cover is open. (Can only be returned if AsyncMode is FALSE.)</p> <p>ResultCodeExtended = OPOS_EPTR_REC_EMPTY: The receipt station is out of paper. (Can only be returned if AsyncMode is FALSE.)</p>
<i>Other Values</i>	See ResultCode .

See Also “Data Characters and Escape Sequences”

PrintBarCode Method

Syntax **LONG PrintBarCode (LONG Station, BSTR Data, LONG Symbology, LONG Height, LONG Width, LONG Alignment, LONG TextPosition);**

Parameter	Description
<i>Station</i>	The printer station to be used. May be either PTR_S_RECEIPT or PTR_S_SLIP.
<i>Data</i>	Character string to be bar coded.
<i>Symbology</i>	Bar code symbol type to use. The Printer Control header file (OPOSPTR.H) defines several symbologies with constant names beginning with PTR_BCS. This parameter will be passed to the Service Object without validation, so that a Service Object may support additional symbologies than those specified in the header file.
<i>Height</i>	Bar code height. Expressed in the unit of measure given by MapMode .
<i>Width</i>	Bar code width. Expressed in the unit of measure given by MapMode .
<i>Alignment</i>	Placement of the bar code. See values below.
<i>TextPosition</i>	Placement of the readable character string.

The *Alignment* parameter values are:

Value	Meaning
PTR_BC_LEFT	Align with the left-most print column.
PTR_BC_CENTER	Align in the center of the station.
PTR_BC_RIGHT	Align with the right-most print column.
<i>Other Values</i>	Distance from the left-most print column to the start of the bar code. Expressed in the unit of measure given by MapMode .

The *TextPosition* parameter values are:

Value	Meaning
PTR_BC_TEXT_NONE	No text is printed. Only print the bar code.
PTR_BC_TEXT_ABOVE	Print the text above the bar code.
PTR_BC_TEXT_BELOW	Print the text below the bar code.

Remarks Call to print a bar code on the specified printer station.

This method is performed synchronously if **AsyncMode** is FALSE, and asynchronously if **AsyncMode** is TRUE.

If the property **RotateSpecial** indicates that the bar code is to be rotated, then perform the rotation. The *Height*, *Width*, and *TextPosition* parameters are applied to the bar code before the rotation. For example, if PTR_BC_TEXT_BELOW is specified and the bar code is rotated left, then the text will appear on the paper to the right of the bar code.

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	One of the following errors occurred: * <i>Station</i> does not exist * <i>Station</i> does not support bar code printing * <i>Height</i> or <i>Width</i> are zero or too big * <i>Alignment</i> is invalid or too big * <i>TextPosition</i> is invalid * The RotateSpecial rotation is not supported
OPOS_E_BUSY	Cannot perform while output is in progress. (Can only be returned if AsyncMode is FALSE.)

OPOS_E_EXTENDED ResultCodeExtended =
OPOS_EPTR_COVER_OPEN:
The printer cover is open.
(Can only be returned if **AsyncMode** is FALSE.)

ResultCodeExtended = OPOS_EPTR_REC_EMPTY:
The receipt station was specified but is out of paper.
(Can only be returned if **AsyncMode** is FALSE.)

ResultCodeExtended = OPOS_EPTR_SLP_EMPTY:
The slip station was specified, but a form is not inserted.
(Can only be returned if **AsyncMode** is FALSE.)

Other Values See **ResultCode**.

PrintBitmap Method

Syntax **LONG PrintBitmap (LONG Station, BSTR FileName, LONG Width, LONG Alignment);**

Parameter	Description
<i>Station</i>	The printer station to be used. May be either PTR_S_RECEIPT or PTR_S_SLIP.
<i>FileName</i>	Name of Windows bitmap file. The file must be in uncompressed format.
<i>Width</i>	Printed width of the bitmap to be performed. See values below.
<i>Alignment</i>	Placement of the bitmap. See values below.

The *Width* parameter values are:

Value	Meaning
PTR_BM_ASIS	Print the bitmap with one bitmap pixel per printer dot.
<i>Other Values</i>	Bitmap width expressed in the unit of measure given by MapMode .

The *Alignment* parameter values are:

Value	Meaning
PTR_BM_LEFT	Align with the left-most print column.
PTR_BM_CENTER	Align in the center of the station.
PTR_BM_RIGHT	Align with the right-most print column.
<i>Other Values</i>	Distance from the left-most print column to the start of the bitmap. Expressed in the unit of measure given by MapMode .

Remarks Call to print a bitmap on the specified printer station.

This method is performed synchronously if **AsyncMode** is FALSE, and asynchronously if **AsyncMode** is TRUE.

The *Width* parameter controls transformation of the bitmap. If *Width* is PTR_BM_ASIS, then no transformation is performed. The bitmap is printed with one bitmap pixel per printer dot. Advantages of this option are that it:

- Provides the highest performance bitmap printing.
- Works well for bitmaps tuned for a specific printer's aspect ratio between horizontal dots and vertical dots.

If *Width* is non-zero, then the bitmap will be transformed by stretching or compressing the bitmap such that its width is the specified width and the aspect ratio is unchanged. Advantages of this option are that it:

- Sizes a bitmap to fit a variety of printers.
- Maintains the bitmap's aspect ratio.

Disadvantages are:

- Lower performance than untransformed data.
- Some lines and images that are "smooth" in the original bitmap may show some "ratcheting."

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_BUSY	Cannot perform while output is in progress. (Can only be returned if AsyncMode is FALSE.)
OPOS_E_ILLEGAL	One of the following errors occurred: * <i>Station</i> does not exist * <i>Station</i> does not support bitmap printing * <i>Width</i> is too big * <i>Alignment</i> is invalid or too big
OPOE_E_NOEXIST	<i>FileName</i> was not found.
OPOS_E_EXTENDED	

ResultCodeExtended = OPOS_EPTR_TOOBIG:
The bitmap is either too wide to print without transformation, or it is too big to transform.

ResultCodeExtended =**OPOS_EPTR_COVER_OPEN:**

The printer cover is open.

(Can only be returned if **AsyncMode** is FALSE.)**ResultCodeExtended = OPOS_EPTR_BADFORMAT:**

The specified file is either not a bitmap file, or it is in an unsupported format.

ResultCodeExtended = OPOS_EPTR_REC_EMPTY:

The receipt station was specified but is out of paper.

(Can only be returned if **AsyncMode** is FALSE.)**ResultCodeExtended = OPOS_EPTR_SLP_EMPTY:**

The slip station was specified, but a form is not inserted.

(Can only be returned if **AsyncMode** is FALSE.)*Other Values*See **ResultCode**.

PrintImmediate Method

Syntax **LONG PrintImmediate (LONG Station, BSTR Data);**

Station The printer station to be used. May be PTR_S_JOURNAL, PTR_S_RECEIPT, or PTR_S_SLIP.

Data The characters to be printed. May consist of printable characters, escape sequences, carriage returns (13 decimal), and line feeds (10 decimal).

Remarks Call to print *Data* on the printer *Station* immediately.

This method tries to print its data immediately – that is, as the very next printer operation. It may be called when asynchronous output is outstanding.

PrintImmediate is primarily intended for use in exception conditions when asynchronous output is outstanding, such as within an error event handler.

Special character values within *Data* are:

Value	Meaning
Line Feed (10)	Print any data in the line buffer, and feed to the next print line. (A Carriage Return is not required in order to cause the line to be printed.)
Carriage Return (13)	<p>If a Carriage Return immediately precedes a Line Feed, or if the line buffer is empty, then it is ignored.</p> <p>Otherwise, the line buffer is printed and the printer does not feed to the next print line.</p> <p>On some printers, print without feed may be directly supported.</p> <p>On others, a print may always feed to the next line, in which case the Service Object will print the line buffer and perform a reverse line feed if supported.</p> <p>If the printer does not support either of these features, then Carriage Return acts like a Line Feed.</p> <p>The ValidateData method may be used to determine whether a Carriage Return without Line Feed is possible, and whether a reverse line feed is required to support it.</p>

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 specified <i>Station</i> does not exist. (See the CapJrnPresent , CapRecPresent , and CapSlpPresent properties.)
OPOS_E_EXTENDED	<p>ResultCodeExtended = OPOS_EPTR_COVER_OPEN: The printer cover is open.</p> <p>ResultCodeExtended = OPOS_EPTR_JRN_EMPTY: The journal station was specified but is out of paper.</p> <p>ResultCodeExtended = OPOS_EPTR_REC_EMPTY: The receipt station was specified but is out of paper.</p> <p>ResultCodeExtended = OPOS_EPTR_SLP_EMPTY: The slip station was specified, but a form is not inserted.</p> <p><i>Other Values</i> See ResultCode.</p>

See Also **PrintNormal** Method; **PrintTwoNormal** Method

PrintNormal Method

Syntax **LONG PrintNormal (LONG Station, BSTR Data);**

Station The printer station to be used. May be PTR_S_JOURNAL, PTR_S_RECEIPT, or PTR_S_SLIP.

Data The characters to be printed. May consist of printable characters, escape sequences, carriage returns (13 decimal), and line feeds (10 decimal).

Remarks Call to print *Data* on the printer *Station*.

This method is performed synchronously if **AsyncMode** is FALSE, and asynchronously if **AsyncMode** is TRUE.

Special character values within *Data* are:

Value	Meaning
Line Feed (10)	Print any data in the line buffer, and feed to the next print line. (A Carriage Return is not required in order to cause the line to be printed.)
Carriage Return (13)	<p>If a Carriage Return immediately precedes a Line Feed, or if the line buffer is empty, then it is ignored.</p> <p>Otherwise, the line buffer is printed and the printer does not feed to the next print line.</p> <p>On some printers, print without feed may be directly supported.</p> <p>On others, a print may always feed to the next line, in which case the Service Object will print the line buffer and perform a reverse line feed if supported.</p> <p>If the printer does not support either of these features, then Carriage Return acts like a Line Feed.</p> <p>The ValidateData method may be used to determine whether a Carriage Return without Line Feed is possible, and whether a reverse line feed is required to support it.</p>

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 specified <i>Station</i> does not exist. (See the CapJrnPresent , CapRecPresent , and CapSlpPresent properties.)
OPOS_E_BUSY	Cannot perform while output is in progress. (Can only be returned if AsyncMode is FALSE.)
OPOS_E_EXTENDED	<p>ResultCodeExtended = OPOS_EPTR_COVER_OPEN: The printer cover is open. (Can only be returned if AsyncMode is FALSE.)</p> <p>ResultCodeExtended = OPOS_EPTR_JRN_EMPTY: The journal station was specified but is out of paper. (Can only be returned if AsyncMode is FALSE.)</p> <p>ResultCodeExtended = OPOS_EPTR_REC_EMPTY: The receipt station was specified but is out of paper. (Can only be returned if AsyncMode is FALSE.)</p> <p>ResultCodeExtended = OPOS_EPTR_SLP_EMPTY: The slip station was specified, but a form is not inserted. (Can only be returned if AsyncMode is FALSE.)</p> <p><i>Other Values</i> See ResultCode.</p>

See Also **PrintImmediate** Method; **PrintTwoNormal** Method

PrintTwoNormal Method

Syntax **LONG PrintTwoNormal (LONG Stations, BSTR Data1, BSTR Data2);**

Parameter	Description
<i>Stations</i>	The printer stations to be used. May be PTR_S_JOURNAL_RECEIPT, PTR_S_JOURNAL_SLIP, or PTR_S_RECEIPT_SLIP.
<i>Data1</i>	The characters to be printed on the first station. May consist of printable characters and escape sequences. The characters must all fit on one printed line, so that the printer may attempt to print on both stations simultaneously.
<i>Data2</i>	The characters to be printed on the second station. (Restrictions are the same as <i>Data1</i> .) If this string is the empty string (“”), then print the same data as <i>Data1</i> . On some printers, using this format may give additional increased print performance.

Remarks Call to print two strings on two print stations simultaneously. When supported, this may give increased print performance.

This method is performed synchronously if **AsyncMode** is FALSE, and asynchronously if **AsyncMode** is TRUE.

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 specified <i>Stations</i> do not support concurrent printing. (See the CapConcurrentJrnRec , CapConcurrentJrnSlp , and CapConcurrentRecSlp properties.)
OPOS_E_BUSY	Cannot perform while output is in progress. (Can only be returned if AsyncMode is FALSE.)
OPOS_E_EXTENDED	ResultCodeExtended = OPOS_EPTR_COVER_OPEN: The printer cover is open. (Can only be returned if AsyncMode is FALSE.)

ResultCodeExtended = OPOS_EPTR_JRN_EMPTY:
The journal station was specified but is out of paper.
(Can only be returned if **AsyncMode** is FALSE.)

ResultCodeExtended = OPOS_EPTR_REC_EMPTY:
The receipt station was specified but is out of paper.
(Can only be returned if **AsyncMode** is FALSE.)

ResultCodeExtended = OPOS_EPTR_SLP_EMPTY:
The slip station was specified, but a form is not inserted.
(Can only be returned if **AsyncMode** is FALSE.)

Other Values

See **ResultCode**.

See Also **PrintNormal** Method

RotatePrint Method

Syntax **LONG RotatePrint (LONG Station, LONG Rotation);**

Parameter	Description
<i>Station</i>	The printer station to be used. May be PTR_S_RECEIPT or PTR_S_SLIP.
<i>Rotation</i>	Direction of rotation. See values below.
Value	Meaning
PTR_RP_RIGHT90	Rotate printing 90° to the right (clockwise).
PTR_RP_LEFT90	Rotate printing 90° to the left (counter-clockwise).
PTR_RP_ROTATE180	Rotate printing 180°, that is, print upside-down.
PTR_RP_NORMAL	End rotated printing.

Remarks Enters or exits rotated print mode.

This method is performed synchronously if **AsyncMode** is FALSE, and asynchronously if **AsyncMode** is TRUE.

If *Rotation* is PTR_RP_ROTATE180, then upside-down print mode is entered. Subsequent calls to **PrintNormal** or **PrintImmediate** will print the data upside-down until **RotatePrint** is called with the *Rotation* parameter set to PTR_RP_NORMAL.

Each print line is rotated by 180°. Lines are printed in the order that they are sent to the Control, with the start of each line justified at the right margin of the printer station. Only print methods **PrintNormal** and **PrintImmediate** may be used while in upside-down print mode.

If *Rotation* is PTR_RP_RIGHT90 or PTR_RP_LEFT90, then sideways print mode is entered. Subsequent calls to **PrintNormal** will buffer the print data (either at the printer or the Service Object, depending on the printer capabilities) until **RotatePrint** is called with the *Rotation* parameter set to PTR_RP_NORMAL. (In this case, **PrintNormal** only buffers the data – it does not initiate printing. Also, the value of the **AsyncMode** property does not affect its operation: No **OutputID** will be assigned to the request, nor will an **OutputCompleteEvent** be fired.) Each print line is rotated by 90°. If the lines are not all the same length, then they are justified at the start of each line. Only **PrintNormal** may be used while in sideways print mode.

If *Rotation* is PTR_RP_NORMAL, then rotated print mode is exited. If sideways-rotated print mode was in effect and some data was buffered by calls to the **PrintNormal** method, then the buffered data is printed. The entire rotated block of lines are treated as one message.

Changing the rotation mode may also change the station's line height, line spacing, line width, and other metrics.

Calling the **ClearOutput** method cancels rotated print mode. Any buffered sideways rotated print lines are also cleared.

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

Value	Meaning
OPOS_SUCCESS	The method was successful.
OPOS_E_ILLEGAL	The specified <i>Station</i> does not exist (see the CapJrnPresent , CapRecPresent , and CapSlpPresent properties), or the <i>Station</i> does not support the specified rotation (see the station's rotation capability properties).
OPOS_E_BUSY	Cannot perform while output is in progress. (Can only be returned if AsyncMode is FALSE.)
OPOS_E_EXTENDED	<p>ResultCodeExtended = OPOS_EPTR_COVER_OPEN: The printer cover is open. (Can only be returned if AsyncMode is FALSE.)</p> <p>ResultCodeExtended = OPOS_EPTR_REC_EMPTY: The receipt station was specified but is out of paper. (Can only be returned if AsyncMode is FALSE.)</p> <p>ResultCodeExtended = OPOS_EPTR_SLP_EMPTY: The slip station was specified, but a form is not inserted. (Can only be returned if AsyncMode is FALSE.)</p>

Other Values See **ResultCode**.

See Also "Data Characters and Escape Sequences"

SetBitmap Method

Syntax **LONG SetBitmap (LONG *BitmapNumber*, LONG *Station*, BSTR *FileName*, LONG *Width*, LONG *Alignment*);**

Parameter	Description
<i>BitmapNumber</i>	The number to be assigned to this bitmap. Two bitmaps, numbered 1 and 2, may be set.
<i>Station</i>	The printer station to be used. May be either PTR_S_RECEIPT or PTR_S_SLIP.
<i>FileName</i>	Name of Windows bitmap file. The file must be in uncompressed format. If set to an empty string (“”), then the bitmap is unset.
<i>Width</i>	Printed width of the bitmap to be performed. See PrintBitmap for values.
<i>Alignment</i>	Placement of the bitmap. See PrintBitmap for values.

Remarks Call to save information about a bitmap for later printing.

The bitmap may then be printed by calling the **PrintNormal** or **PrintImmediate** method with the print bitmap escape sequence in the print data. The print bitmap escape sequence will typically be included in a string for printing top and bottom transaction headers.

A Service Object may choose to cache the bitmap for later use to provide better performance. Regardless, the bitmap file and parameters are validated for correctness by this method.

The application must ensure that the printer station metrics, such as character width, line height, and line spacing are set for the *Station* before calling this method. The Service Object may perform transformations on the bitmap in preparation for later printing based upon the current values.

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	One of the following errors occurred: <ul style="list-style-type: none">* <i>BitmapNumber</i> is invalid* <i>Station</i> does not exist* <i>Station</i> does not support bitmap printing* <i>Width</i> is too big* <i>Alignment</i> is invalid or too big
OPOE_E_NOEXIST	<i>FileName</i> was not found.
OPOS_E_EXTENDED	ResultCodeExtended = OPOS_EPTR_TOOBIG: The bitmap is either too wide to print without transformation, or it is too big to transform. ResultCodeExtended = OPOS_EPTR_BADFORMAT: The specified file is either not a bitmap file, or it is in an unsupported format.
<i>Other Values</i>	See ResultCode .

See Also “Data Characters and Escape Sequences”; **PrintBitmap** Method

SetLogo Method

Syntax **LONG SetLogo (LONG *Location*, BSTR *Data*);**

Parameter	Description
<i>Location</i>	The logo to be set. May be PTR_L_TOP or PTR_L_BOTTOM.
<i>Data</i>	The characters that produce the logo. May consist of printable characters, escape sequences, carriage returns (13 decimal), and line feeds (10 decimal).

Remarks Call to save a data string as the top or bottom logo.

A logo may then be printed by calling the **PrintNormal**, **PrintTwoNormal**, or **PrintImmediate** method with the print top logo or print bottom logo escape sequence in the print data.

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	An invalid <i>Location</i> was specified.
<i>Other Values</i>	See ResultCode .

See Also “Data Characters and Escape Sequences”

TransactionPrint Method *Added in Release 1.1*

Syntax **LONG TransactionPrint (LONG Station, LONG Control);**

Parameter	Description
<i>Station</i>	The printer station to be used. May be PTR_S_JOURNAL, PTR_S_RECEIPT, or PTR_S_SLIP.
<i>Control</i>	Transaction control. See values below.

Value	Meaning
PTR_TP_TRANSACTION	Begin a transaction.
PTR_TP_NORMAL	End a transaction by printing the buffered data.

Remarks Enters or exits transaction mode.

If *Control* is PTR_TP_TRANSACTION, then transaction mode is entered. Subsequent calls to **PrintNormal**, **CutPaper**, **RotatePrint**, **PrintBarCode**, and **PrintBitmap** will buffer the print data (either at the printer or the Service Object, depending on the printer capabilities) until **TransactionPrint** is called with the *Control* parameter set to PTR_TP_NORMAL. (In this case, the print methods only validate the method parameters and buffer the data – they do not initiate printing. Also, the value of the **AsyncMode** property does not affect their operation: No **OutputID** will be assigned to the request, nor will an **OutputCompleteEvent** be fired.)

If *Control* is PTR_TP_NORMAL, then transaction mode is exited. If some data was buffered by calls to the methods **PrintNormal**, **CutPaper**, **RotatePrint**, **PrintBarCode**, and **PrintBitmap**, then the buffered data is printed. The entire transaction is treated as one message. This method is performed synchronously if **AsyncMode** is FALSE, and asynchronously if **AsyncMode** is TRUE.

Calling the **ClearOutput** method cancels transaction mode. Any buffered print lines are also cleared.

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

Value	Meaning
OPOS_SUCCESS	The method was successful.
OPOS_E_ILLEGAL	The specified <i>Station</i> does not exist (see the CapJrnPresent , CapRecPresent , and CapSlpPresent properties), or CapTransaction is FALSE.
OPOS_E_BUSY	Cannot perform while output is in progress. (Can only be returned if AsyncMode is FALSE.)
OPOS_E_EXTENDED	<p>ResultCodeExtended = OPOS_EPTR_COVER_OPEN: The printer cover is open. (Can only be returned if AsyncMode is FALSE.)</p> <p>ResultCodeExtended = OPOS_EPTR_JRN_EMPTY: The journal station was specified but is out of paper. (Can only be returned if AsyncMode is FALSE.)</p> <p>ResultCodeExtended = OPOS_EPTR_REC_EMPTY: The receipt station was specified but is out of paper. (Can only be returned if AsyncMode is FALSE.)</p> <p>ResultCodeExtended = OPOS_EPTR_SLP_EMPTY: The slip station was specified, but a form is not inserted. (Can only be returned if AsyncMode is FALSE.)</p>
<i>Other Values</i>	See ResultCode .

ValidateData Method *Added in Release 1.1*

Syntax **LONG ValidateData (LONG Station, BSTR Data);**

Parameter	Description
<i>Station</i>	The printer station to be used. May be PTR_S_JOURNAL, PTR_S_RECEIPT, or PTR_S_SLIP.
<i>Data</i>	The data to be validated. May include printable data and escape sequences.

Remarks Call to determine whether a data sequence, possibly including one or more escape sequences, is valid for the specified station, before calling the **PrintImmediate**, **PrintNormal**, or **PrintTwoNormal** methods.

This method does not cause any printing, but is used to determine the capabilities of the station.

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

Value	Meaning
OPOS_SUCCESS	The data is valid.
OPOS_E_ILLEGAL	Some of data is not precisely supported by the printer station, but the Control can select valid alternatives.
OPOS_E_FAILURE	Some of the data is not supported. No alternatives can be selected.

Cases which cause OPOS_E_ILLEGAL to be returned are:

Escape Sequence	Condition
Paper cut	The percentage '#' is not precisely supported: Control will select the closest supported value.
Feed and Paper cut	The percentage '#' is not precisely supported: Control will select the closest supported value.
Feed, Paper cut, and Stamp	The percentage '#' is not precisely supported: Control will select the closest supported value.

Feed units	The unit count '#' is not precisely supported: Control will select the closest supported value.
Feed reverse	The line count '#' is too large: Control will select the maximum supported value.
Shading	The percentage '#' is not precisely supported: Control will select the closest supported value.
Scale horizontally	The scaling factor '#' is not supported: Control will select the closest supported value.
Scale vertically	The scaling factor '#' is not supported: Control will select the closest supported value.

Data	Condition
<i>data1CRdata2LF</i>	(Where CR is a Carriage Return and LF is a Line Feed) In order to print data <i>data1</i> and remain on the same line, the Service Object will print with a line advance, then perform a reverse line feed. The data <i>data2</i> will then overprint <i>data1</i> .

Cases which will cause OPOS_E_FAILURE to be returned are:

Escape Sequence	Condition
(General)	The escape sequence format is not valid.
Paper cut	Not supported.
Feed and Paper cut	Not supported.
Feed, Paper cut, and Stamp	Not supported.
Fire stamp	Not supported.
Print bitmap	Bitmap printing is not supported, or the bitmap number '#' is out of range.
Feed reverse	Not supported.
Font typeface	The typeface '#' is not supported:

Bold	Not supported.
Underline	Not supported.
Italic	Not supported.
Alternate color	Not supported.
Reverse video	Not supported.
Shading	Not supported.
Single high & wide	Not supported.
Double wide	Not supported.
Double high	Not supported.
Double high & wide	Not supported.

Data	Condition
<i>data1</i> CR <i>data2</i> LF	(Where CR is a Carriage Return and LF is a Line Feed) Not able to print data and remain on the same line. The data <i>data1</i> will print on one line, and the data <i>data2</i> will print on the next line.

See Also “Data Characters and Escape Sequences”

Events

ErrorEvent Event

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

Parameter	Description
<i>ResultCode</i>	Result code causing the error event. See ResultCode for values.
<i>ResultCodeExtended</i>	Extended result code causing the error event. See values below.
<i>ErrorLocus</i>	Set to OPOS_EL_OUTPUT: Error occurred while processing asynchronous output.
<i>pErrorResponse</i>	Pointer to the error event response. See values below.

If *ResultCode* is OPOS_E_EXTENDED, then *ResultCodeExtended* is set to one of the following values:

Value	Meaning
OPOS_EPTR_COVER_OPEN	The printer cover is open.
OPOS_EPTR_JRN_EMPTY	The journal station is out of paper.
OPOS_EPTR_REC_EMPTY	The receipt station is out of paper.
OPOS_EPTR_SLP_EMPTY	A form is not inserted in the slip station.

The contents at the location pointed to by the *pErrorResponse* parameter are preset to the default value of OPOS_ER_RETRY. The application may set the value to one of the following:

Value	Meaning
OPOS_ER_RETRY	Retry the asynchronous output. The error state is exited.
OPOS_ER_CLEAR	Clear the asynchronous output. The error state is exited.

Remarks Fired when an error is detected and the Control's **State** transitions into the error state.

See Also “Status, Result Code, and State Model”

StatusUpdateEvent Event

Syntax `void StatusUpdateEvent (LONGData);`

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

Value	Meaning
PTR_SUE_COVER_OPEN	Printer cover is open.
PTR_SUE_COVER_OK	Printer cover is closed.
PTR_SUE_JRN_EMPTY	No journal paper.
PTR_SUE_JRN_NEAREMPTY	Journal paper is low.
PTR_SUE_JRN_PAPEROK	Journal paper is ready.
PTR_SUE_REC_EMPTY	No receipt paper.
PTR_SUE_REC_NEAREMPTY	Receipt paper is low.
PTR_SUE_REC_PAPEROK	Receipt paper is ready.
PTR_SUE_SLP_EMPTY	No slip form.
PTR_SUE_SLP_NEAREMPTY	Almost at the bottom of the slip form.
PTR_SUE_SLP_PAPEROK	Slip form is inserted.
PTR_SUE_IDLE	All asynchronous output has finished, either successfully or because output has been cleared. The printer State is now OPOS_S_IDLE. The FlagWhenIdle property must be TRUE for this event to be fired, and the Control automatically resets the property to FALSE just before firing the event.

Remarks Fired when a significant status event has occurred.

CHAPTER 11

Scale

Summary

Properties

<i>Common</i>	<i>Type</i>	<i>Access</i>	<i>Initialized After</i>
CheckHealthText	String	R	Open
Claimed	Boolean	R	Open
DataEventEnabled	Boolean	R/W	<i>Not Supported</i>
DeviceEnabled	Boolean	R/W	Open & Claim
FreezeEvents	Boolean	R/W	Open
OutputID	Long	R	<i>Not Supported</i>
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
<i>Specific</i>			
WeightUnits	Long	R	Open
MaximumWeight	Long	R	Open

Methods

Common

May Use After

Open

--

Close

Open

Claim

Open

Release

Open & Claim

CheckHealth

Open, Claim, & Enable

ClearInput

Not Supported

ClearOutput

Not Supported

DirectIO

Open

Specific

ReadWeight

Open, Claim, & Enable

Events

Name

May Occur After

DataEvent

Not Supported

DirectIOEvent

Open

ErrorEvent

Not Supported

OutputCompleteEvent

Not Supported

StatusUpdateEvent

Not Supported

General Information

The Scale Control's OLE programmatic ID is "OPOS.Scale".

Model

The Scale Control has the following functionality:

- Provides item weight to the application. The application may specify the measure of weight to be in grams, kilograms, ounces, or pounds.

The Scale Control returns weight from a synchronous method. No event-driven model is provided due to the typical usage of the device.

Device Sharing

The scale is an exclusive-use device, as follows:

- After opening the device, properties are readable.
- The application must claim the device before enabling it.
- The application must claim and enable the device before calling methods that manipulate the device.
- See the "Summary" table for precise usage prerequisites.

Properties

MaximumWeight Property

- Syntax** **LONG MaximumWeight;**
- Remarks** Holds the maximum weight measurement possible from the scale. The measurement unit is available via the **WeightUnit** property.
- MaxWeight** has an assumed decimal place located after the “thousands” digit position. For example, an actual value of 12345 represents 12.345, and an actual value of 5 represents 0.005.
- Changing the property **WeightUnit** will also cause this property to change.
- This property is initialized by the **Open** method.

WeightUnit Property

- Syntax** **LONG WeightUnit;**
- Remarks** Holds the unit of weight of scale data.

Valid units are:

Value	Meaning
SCAL_WU_GRAM	Unit is a gram.
SCAL_WU_KILOGRAM	Unit is a kilogram (= 1000 grams).
SCAL_WU_OUNCE	Unit is an ounce.
SCAL_WU_POUND	Unit is a pound (= 16 ounces).

This property is initialized to the scale’s weight unit by the **Open** method.

Methods

ReadWeight Method

Syntax **LONG ReadWeight (LONG* *pWeightData*, LONG *Timeout*);**

Parameter	Description
<i>pWeightData</i>	Points to the number where the weight is returned.
<i>Timeout</i>	The number of milliseconds to wait for a settled weight before failing the method.

Remarks Call to read a weight from the scale.

The weight returned at *pWeightData* has an assumed decimal place located after the “thousands” digit position. For example, an actual value of 12345 represents 12.345, and an actual value of 5 represents 0.005.

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

Value	Meaning
OPOS_SUCCESS	A valid weight was read and placed into the specified location.
OPOS_E_TIMEOUT	A stable non-zero weight was not available before <i>Timeout</i> milliseconds elapsed.
OPOS_E_EXTENDED	ResultCodeExtended = OPOS_ESCAL_OVERWEIGHT: The weight was over MaximumWeight .
<i>Other Values</i>	See ResultCode .

CHAPTER 12

Scanner (Bar Code Reader)

Summary

Properties

<i>Common</i>	<i>Type</i>	<i>Access</i>	<i>Initialized After</i>
CheckHealthText	String	R	Open
Claimed	Boolean	R	Open
DataEventEnabled	Boolean	R/W	Open
DeviceEnabled	Boolean	R/W	Open & Claim
FreezeEvents	Boolean	R/W	Open
OutputID	Long	R	<i>Not Supported</i>
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
<i>Specific</i>			
ScanData	String	R	Open

Methods

<i>Common</i>	<i>May Use After</i>
Open	--
Close	Open
Claim	Open
Release	Open & Claim
CheckHealth	Open, Claim, & Enable
ClearInput	Open, Claim, & Enable
ClearOutput	<i>Not Supported</i>
DirectIO	Open

Events

<i>Name</i>	<i>May Occur After</i>
DataEvent	Open, Claim, & Enable
DirectIOEvent	Open
ErrorEvent	Open, Claim, & Enable
OutputCompleteEvent	<i>Not Supported</i>
StatusUpdateEvent	<i>Not Supported</i>

General Information

The Scanner Control's OLE programmatic ID is "OPOS.Scanner".

Capabilities

The Scanner Control has the following capability:

- Reads encoded data from a label.

Model

The Scanner Control follows the general Input Model for event-driven input:

- When input is received by the Control, it generates a **DataEvent**.
- Just before firing this event, the Control disables further data events by setting the **DataEventEnabled** property to FALSE. This causes buffering of further input data at the Control while the application processes the current input and associated properties. When the application has finished the current input and is ready for more data, it reenables events by setting **DataEventEnabled** to TRUE.
- All input buffered at the Control may be deleted by calling the **ClearInput** method.

Device Sharing

The scanner is an exclusive-use device, as follows:

- The application must claim the device before enabling it.
- The application must claim and enable the device before the device begins reading input.
- See the "Summary" table for precise usage prerequisites.

Properties

ScanData Property

Syntax **BSTR ScanData;**

Remarks The data read from the scanner.

Scan data is, in general, in the format as delivered from the scanner. Message header and trailer information should be removed, however, since they do not contain useful information for an application and are likely to be scanner-specific.

Common header information is a prefix character (such as an STX character). Common trailer information is a terminator character (such as an ETX or CR character) and a block check character if one is generated by the scanner.

ScanData should include a symbology character if one is returned by the scanner (such as an 'A' for UPC-A). **ScanData** should also include check digits if they are present in the label and returned by the scanner. (Note that both symbology characters and check digits may or may not be present, depending upon the scanner configuration. The Scanner Control will return them if present, but will not generate or calculate them if they are absent.)

If a barcode is read that contains a supplemental code or multiple labels, the supplemental code or additional labels are delivered to the application as separate labels. This is necessary due to the current lack of standardization of these barcode types. One is not able to determine all variations based upon the individual barcode data. Therefore, the application will need to determine when a supplemental code or multiple label barcode has been read based upon the data returned.

This property is set by the Control just before firing the **DataEvent**.

Events

DataEvent Event

- Syntax** **void DataEvent (LONG Status);**
- The *Status* parameter contains zero.
- Remarks** Fired to present input data from the device to the application. The scanner data is placed in the **ScanData** property before this event is fired.

ErrorEvent Event

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

Parameter	Description
<i>ResultCode</i>	Result code causing the error event. See ResultCode for values.
<i>ResultCodeExtended</i>	Extended result code causing the error event. See ResultCodeExtended for values.
<i>ErrorLocus</i>	Location of the error. See values below.
<i>pErrorResponse</i>	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.

The contents at the location pointed to by the *pErrorResponse* parameter are preset to a default value, based on the *ErrorLocus*. The application may change the value to one of the following:

Value	Meaning
OPOS_ER_CLEAR	Clear the buffered input data. The error state is exited. Default when locus is OPOS_EL_INPUT.
OPOS_ER_CONTINUEINPUT	Use only when locus is OPOS_EL_INPUT_DATA. Acknowledges the error and directs the Control to continue processing. The Control remains in the error state, and will fire additional DataEvents as directed by the DataEventEnabled property. When all input has been fired and the DataEventEnabled property is again set to TRUE, then another ErrorEvent is fired with locus OPOS_EL_INPUT. Default when locus is OPOS_EL_INPUT_DATA.

Remarks Fired when an error is detected while trying to read scanner data.

Input error events are not fired until the **DataEventEnabled** property is TRUE, so that proper application sequencing occurs.

See Also “Status, Result Code, and State Model”

CHAPTER 13

Signature Capture

Summary

Properties

<i>Common</i>	<i>Type</i>	<i>Access</i>	<i>Initialized After</i>
CheckHealthText	String	R	Open
Claimed	Boolean	R	Open
DataEventEnabled	Boolean	R/W	Open
DeviceEnabled	Boolean	R/W	Open & Claim
FreezeEvents	Boolean	R/W	Open
OutputID	Long	R	<i>Not Supported</i>
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

<i>Specific</i>	<i>Type</i>	<i>Access</i>	<i>Initialized After</i>
CapDisplay	Boolean	R	Open
CapUserTerminated	Boolean	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

Methods

<i>Common</i>	<i>May Use After</i>
Open	--
Close	Open
Claim	Open
Release	Open & Claim
CheckHealth	Open, Claim, & Enable
ClearInput	Open, Claim, & Enable
ClearOutput	<i>Not Supported</i>
DirectIO	Open
 <i>Specific</i>	
BeginCapture	Open, Claim, & Enable
EndCapture	Open, Claim, & Enable

Events

<i>Name</i>	<i>May Occur After</i>
DataEvent	Open, Claim, & Enable
DirectIOEvent	Open
ErrorEvent	Open, Claim, & Enable
OutputCompleteEvent	<i>Not Supported</i>
StatusUpdateEvent	<i>Not Supported</i>

General Information

The Signature Capture Control's OLE programmatic ID is "OPOS.SigCap".

Capabilities

The Signature Capture Control has the following capability:

- Obtains a signature captured by a signature capture device. The captured signature data is in the form of lines consisting of a series of points. Each point lies within the coordinate system defined by the resolution of the device, where (0,0) is the upper-left point of the device, and (**MaximumX,MaximumY**) is the lower-right point.

The Signature Capture Control may have the following additional capabilities:

- Provides a way for the user to terminate signature capture – that is, to tell the device that she or he has completed the signature.
- Displays form/data on the signature capture device.

Model

The signature capture device usage model is:

- Open and claim the device.
- Enable the device and set **DataEventEnabled** to TRUE.
- Begin capturing a signature by calling the **BeginCapture** method. This method displays a form or data screen (if the device has a display) and enables the stylus.
- If the device provides a way for the user to terminate the signature, then when the user terminates, the Control fires a **DataEvent**. Otherwise, the application must call the **EndCapture** method to terminate the signature.
- Disable the device. If the device has a display, this also clears the display.

The Signature Capture Control follows the general Input Model for event-driven input:

- The capture of signature data begins when the **BeginCapture** method is called.
- When input is received by the Control, it generates a **DataEvent**.
- Just before firing this event, the Control disables further data events by setting the **DataEventEnabled** property to FALSE.
- If signature capture is terminated by calling **EndCapture**, then no **DataEvent** is fired.
- All input buffered at the Control may be deleted by calling the **ClearInput** method.

Device Sharing

The signature capture device is an exclusive-use device, as follows:

- The application must claim the device before enabling it.
- The application must claim and enable the device before calling methods that manipulate the device or before changing some writable properties.
- See the “Summary” table for precise usage prerequisites.

Properties

CapDisplay Property

- Syntax** **BOOL CapDisplay;**
- Remarks** Set to TRUE if the device is able to display a form or data entry screen; otherwise it is FALSE.
- This property is initialized by the **Open** method.

CapUserTerminated Property

- Syntax** **BOOL CapUserTerminated;**
- Remarks** Set to TRUE if the user is able to terminate signature capture by checking a completion box, pressing a completion button, or performing some other interaction with the device.
- Contains FALSE if the application must end signature capture by calling the **EndCapture** method.
- This property is initialized by the **Open** method.

DeviceEnabled Property R/W (Common)

- Syntax** **BOOL DeviceEnabled;**
- Remarks** Set to TRUE to enable the signature capture device.
- Set to FALSE to disable the device. If **CapDevice** is TRUE, then the display screen of the device is cleared.
- This property is initialized to FALSE by the **Open** method.

MaximumX Property

- Syntax** **LONG MaximumX;**
- Remarks** Contains the maximum horizontal coordinate of the signature capture device. It must be less than 65,536.
- This property is initialized by the **Open** method.

MaximumY Property

- Syntax** **LONG MaximumY;**
- Remarks** Contains the maximum vertical coordinate of the signature capture device. It must be less than 65,536.
- This property is initialized by the **Open** method.

RawData Property

- Syntax** **BSTR RawData;**
- Remarks** Contains the signature captured from the device in a device-specific format.
- 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 is set by the Control just before firing the **DataEvent** or by the **EndCapture** method.
- See Also** **TotalPoints** Property; **PointArray** Property

TotalPoints Property

Syntax **LONG TotalPoints;**

Remarks Contains the total number of signature points in the last captured signature.

This property is set by the Control just before firing the **DataEvent** or by the **EndCapture** method. It includes the line drawing terminators (see **PointArray**).

PointArray Property

Syntax **BSTR PointArray;**

Remarks Contains the signature captured from the device. It consists of an array of (x,y) 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).

Reconstruction of the signature using the points is accomplished by beginning with a line drawn from the first point in **PointArray** to the second point, then to the third, and so on.

Each line is terminated when the end of **PointArray** is reached or when a line terminator point with the values ($x=0xFFFF$, $y=0xFFFF$) is encountered. The next point is the first point for a new line. Almost all signatures are comprised of more than one line.

This property is set by the Control just before firing the **DataEvent** or by the **EndCapture** method.

See Also **RawData** Property

Methods

BeginCapture Method

Syntax **LONG BeginCapture (BSTR *FormName*);**

The *FormName* parameter contains the registry subkey name for obtaining form or data screen information for display on the device screen.

Remarks Call to start capturing a signature.

If **CapDisplay** is TRUE, then *FormName* is used to find information about the form or data screen to be displayed. The operating system registry key

```
\HKEY_LOCAL_MACHINE\SOFTWARE\OLEforRetail\ServiceOPOS\
SignatureCapture\DeviceName\FormName
```

is accessed to get this information. *DeviceName* is the Service Object's Device Name key.

The format and features of each signature capture device's form/data screen varies widely and is often built with proprietary tools. Therefore, this key's data and additional values and data under this key contain information that varies by Service Object. Typically, the registry key's data is set to a form/data screen file name, and extra registry values and data are set as needed to control its display. (See the appendix "OPOS Registry Usage", page 301.)

After displaying the form or data screen, when applicable, the signature capture stylus is enabled.

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

Value	Meaning
OPOS_SUCCESS	Signature capture successfully started.
OPOS_E_NOEXIST	<i>FormName</i> was not found.
<i>Other Values</i>	See ResultCode .

EndCapture Method

Syntax **LONG EndCapture ();**

Remarks Call to stop capturing a signature.

Terminates signature capture. If a signature was captured, then it is placed in the properties **TotalPoints**, **PointArray**, and **RawData**. If no signature was captured, then **TotalPoints** is set to zero, and **PointArray** and **RawData** are set to the empty string (“”).

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

Value	Meaning
OPOS_SUCCESS	Signature capture successfully stopped.
OPOS_E_ILLEGAL	Signature capture was not in progress.
<i>Other Values</i>	See ResultCode .

See Also **DataEvent**

Events

DataEvent Event

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

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

This event can only be fired if the user can terminate signature capture – that is, if **CapUserTerminated** is TRUE.

The *Status* parameter contains TRUE if the user has entered a signature before terminating capture. It contains FALSE if the user terminated capture with no signature.

Before firing the event, the properties **TotalPoints**, **PointArray**, and **RawData** are set to appropriate values.

See Also **EndCapture** Method

ErrorEvent Event

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

Parameter	Description
<i>ResultCode</i>	Result code causing the error event. See ResultCode for values.
<i>ResultCodeExtended</i>	Extended result code causing the error event. See ResultCodeExtended for values.
<i>ErrorLocus</i>	Location of the error. See values below.
<i>pErrorResponse</i>	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 .)

The contents at the location pointed to by the *pErrorResponse* parameter are preset to a default value, based on the *ErrorLocus*. The application may change the value to one of the following:

Value	Meaning
OPOS_ER_CLEAR	Clear the buffered input data. The error state is exited. Default when locus is OPOS_EL_INPUT.
OPOS_ER_CONTINUEINPUT	Use only when locus is OPOS_EL_INPUT_DATA. Acknowledges the error and directs the Control to continue processing. The Control remains in the error state, and will fire additional DataEvents as directed by the DataEventEnabled property. When all input has been fired and the DataEventEnabled property is again set to TRUE, then another ErrorException is fired with locus

OPOS_EL_INPUT.

Default when locus is OPOS_EL_INPUT_DATA.

Remarks Fired when an error is detected while trying to read signature capture data.

Input error events are not fired until the **DataEventEnabled** property is TRUE, so that proper application sequencing occurs.

With proper programming, an **ErrorEvent** with locus OPOS_EL_INPUT_DATA will not occur. This is because each signature requires an explicit **BeginCapture** method, which can generate at most one **DataEvent**. The application would need to defer the **DataEvent** by setting **DataEventEnabled** to FALSE and request another signature before an OPOS_EL_INPUT_DATA would be possible.

See Also “Status, Result Code, and State Model”

A P P E N D I X A

Change History

Release 1.01

Release 1.01 mostly adds clarifications and corrections, but the Line Display and Signature Capture chapters received substantive changes to correct deficiencies in their definition.

Release 1.01 replaces Release 1.0. The **ControlObjectVersion** for a compliant Control Object is 1000xxx, where xxx is a vendor-specific build number. The **ServiceObjectVersion** for a compliant Service Object is 1000xxx, where xxx is a vendor-specific build number.

Section	Change
Second Page	Add name of Microsoft Web site for OPOS information.
Introduction When ... Properties May Be Accessed	Update to say that capabilities are initialized at Open , others may not be initialized until DeviceEnabled = TRUE , and properties remain initialized until the Control is closed.
Introduction Device Sharing Model	If an exclusive device is Released , then re Claimed , settable device characteristics are restored to their state at Release .
Common Release method	If device is enabled, then disable before releasing.
Cash Drawer WaitForDrawerClose method	<i>BeepFrequency</i> is in hertz.

Hard Totals **General Information**

Recommend claiming necessary files before a **BeginTrans**, to ensure that **CommitTrans** does not fail.

Keylock **General Information**

Claim will return OPOS_E_ILLEGAL, not success.

Line Display **General Information**

Major clarification of line display usage modes; including intercharacter wait and marquees.

Line Display **MarqueeFormat** property

Add this property.

Line Display **MarqueeType** property

Add DISP_MT_INIT value.

Line Display **ClearText** and **RefreshWindow** methods

Clarify their functionality.

POS Printer **XxxLetterQuality** properties

Add initialization information.

POS Printer **XxxLineWidth** properties

Clarify these properties.

POS Printer **CapConcurrentXxxXxx** properties

Clarify that if a “concurrent” capability is false, then the application should print to only one of the stations at a time, and not alternate print lines between them.

POS Printer **CapXxxNearendSensor** properties

Rename to **CapXxxNearEndSensor** for consistency with **XxxNearEnd** properties.

POS Printer **CapXxxBarcode** properties

Rename to **CapXxxBarCode** for consistency with **PrintBarCode** method.

Scale **Summary**

Change **ClearInput** method to *Not Supported*. Scale input is not event-driven.

Scale **WeightUnit** property

Change to read-only property.

Signature Capture **MaximumX** and **MaximumY** properties

Clarify that maximum value is 65,535.

Signature Capture **TotalVectors** and **VectorArray** properties

Rename to **TotalPoints** and **PointArray**. Update the General Information and the property remarks sections for consistency.

Signature Capture **PointArray** property

Clarify that each point is represented by four characters: x (low 8 bits), x (high 8 bits), y (low 8 bits), y (high 8 bits).

Throughout

Update the property initialization details.

OPOSDISP.H header file

Add DISP_MT_INIT constant and **MarqueeFormat** constants.

Appendix C **Technical Details**

Add this appendix, with the sections:

- System strings and binary data.
- Event Handler Restrictions.

Release 1.1

Release 1.1 adds APIs based on requirements from OPOS-J, the Japanese OPOS consortium.

Release 1.1 is a superset of Release 1.01.

The **ControlObjectVersion** for a compliant Control Object is 1001xxx, where xxx is a vendor-specific build number. A Release 1.1 compliant Control Object will function correctly with Service Objects of either Release 1.01 or Release 1.1. If the application attempts to access a property or method that is new to Release 1.1 through a Release 1.01 Service Object, the Control Object will return a **ResultCode** value of OPOS_E_NOSERVICE.

The **ServiceObjectVersion** for a compliant Service Object is 1001xxx, where xxx is a vendor-specific build number. A Release 1.1 compliant Service Object will function correctly with Control Objects of either Release 1.01 or Release 1.1. Of course, the application will not be able to access access a property or method that is new to Release 1.1 through a Release 1.01 Control Object.

Section	Change
Second Page	Remove CompuServe reference.
POS Keyboard	New device: Add information in several locations, plus POS Keyboard chapter.
Line Display CapCharacterSet property	Add values for Kana and Kanji.
Line Display CharacterSet property	Add Windows code page information.

POS Printer Data Characters and Escape Sequences

Add new sequences for:

- Feed and Paper cut
- Feed, Paper cut, and Stamp
- Feed lines
- Feed units
- Feed reverse
- Font typeface selection
- Reverse video
- Shading
- Scale horizontally
- Scale vertically

Add width selection for underline sequence.

POS Printer: Add the following properties and methods:

- CapCharacterSet** property
- CapTransaction** property
- ErrorLevel** property
- ErrorString** property
- FontTypefaceList** property
- RecBarCodeRotationList** property
- RotateSpecial** property
- SlpBarCodeRotationList** property
- TransactionPrint** method
- ValidateData** method

POS Printer **CharacterSet** property

Add Windows code page information.

POS Printer **PrintBarCode** method

Add information on effects of the **RotateSpecial** property.

POS Printer **PrintImmediate** and **PrintNormal** methods

Clarify the effects of Carriage Return and Line Feed.

Scanner **ScanData** property

Clarify the data that is present in this property.

OPOSDISP.H header file

Add **CapCharacterSet** values for Kana and Kanji.

OPOSPTR.H header file

Add **CapCharacterSet** values.

Add **ErrorLevel** values.

Add **TransactionPrint** *Control* values.

A P P E N D I X B

OPOS Registry Usage

OPOS Controls require some data in the system registry in order for the Control Objects to locate the proper Service Object and initialize it for the device.

The registry is organized in a hierarchical structure, in which each level is named a “key.” Each key may contain:

- Additional keys (sometimes called “subkeys”).
- Zero or more named “values.” A value is assigned “data” of type string, binary, or double-word.
- One “default value” that may be assigned data of type string.

OPOS only defines string data.

Service Object Root Registry Key

All OPOS Service Object entries should be placed under the following main key:

HKEY_LOCAL_MACHINE\SOFTWARE\OLEforRetail\ServiceOPOS

The “HKEY_LOCAL_MACHINE\SOFTWARE” key is the recommended key for software configuration local to the PC. The “OLEforRetail” key will group all OLE for Retail related configuration information. The “ServiceOPOS” key maintains configuration information for OPOS Service Objects.

Device Class Keys

Each class has an identifying Device Class subkey under the main OPOS key. The following key names have been established:

CashDrawer
CoinDispenser
HardTotals
Keylock
LineDisplay
MICR

MSR
POSPrinter
Scale
Scanner
SignatureCapture

Device Name Keys and Values

Each device within a class is assigned a Device Name subkey under the class's key. This should be performed by a Service Object installation procedure. This Device Name key is passed to the Control Object's **Open** method by the application. The Device Name is not constrained, except that it must be unique among the names under the device class.

The default value of the Device Name key is the programmatic ID⁵ of the Service Object. This string is needed by the Control Object, so that the Service Object may be loaded and the OLE Automation interfaces established between the CO and the SO.

The device unit key's values and their data describe the characteristics of the actual device on the terminal or PC. The following values are strongly recommended for use by installation and support personnel:

Value	Data
Service	Filename of the Service Object.
Description	String describing the Service Object.
Version	String containing the Service Object version number. General format is: MajorVersion.MinorVersion.BuildVersion.

Other values may be defined as needed by the Service Object. Values might contain information such as:

Communications Port
Baud Rate
Serial Line Characteristics
Interrupt Request (IRQ) Values
Input/Output (I/O) Ports

⁵ A Programmatic ID, or "Prog ID", is the name of a key that must appear in the "HKEY_CLASSES_ROOT" section of the registry. This key must have a subkey named "CLSID", which is the Class ID associated with the Prog ID. The Class ID must be a key within the "HKEY_CLASSES_ROOT\CLSID" registry section. This key contains subkeys that specify the OLE Automation Server type and that instruct OLE how to start the Server.

Logical Device Name Values

An application may open a Control by passing the Device Name key to the **Open** method. In many cases, however, the application will want a level of isolation where the application specifies a “Logical Device Name” that is translated into a Device Name.

A Logical Device Name is added to the registry as a value contained in the Device Class key. The value name is set to the Logical Device Name, and its data must match a Device Name key contained in the same Device Class.

The application integrator is responsible for adding Logical Device Names to the registry. (They are not added by the Service Object install procedure.)

Service Provider Root Registry Key

The SO service providers may need to store some information in the registry that is common to some or all of its Service Objects. This data could include installation directories, installation date, and deinstall information. Service provider information should be placed under the following main key:

HKEY_LOCAL_MACHINE\SOFTWARE\OLEforRetail\ServiceInfo

The subkeys under this key should be the names of service provider companies. Subkeys and values within each service provider company subkey are provider-dependent.

Example

In this example, keys are listed in *italics*. Comments appear as **comment**.

Two device classes are given: POSPrinter and CashDrawer.

The POSPrinter class contains two Device Names. Also, two Logical Device Names are present, which point to the Device Names.

The CashDrawer class contains one Device Name and one Logical Device Name. The Service Object has a unique Prog ID but uses the same executable as one of the printers. This Service Object could use the example value “*Uses*” to point to some registry values of the printer device that can be used for the cash drawer parameters.

```

\HKEY_LOCAL_MACHINE
  1/2
  1/2® \SOFTWARE
    1/2 1/2
    1/2 1/2® \OLEforRetail
      -
      1/2 1/2
      1/2 1/2® \ServiceOPOS
        -
        1/2 1/2
        1/2 1/2® \POSPrinter Device Class Key
          1/2 1/2 1/2
          1/2 1/2 1/2® \NCR7156=NCR.Ptr7156.1 Device Name Key
            1/2 1/2 1/2 Service=C:\OPOS\NCR\PTR7156.DLL
            1/2 1/2 1/2 Description=NCR 7156 Serial Printer
            1/2 1/2 1/2 Version=1.0.12
            1/2 1/2 1/2 ...Service Object-specific values. Might include:
            1/2 1/2 1/2 Port=COM3
            1/2 1/2 1/2 BaudRate=9600
          1/2 1/2 1/2
          1/2 1/2 1/2® \Epson950=Epson.PtrTMU950.1 Device Name Key
            1/2 1/2 1/2 Service=TMU950.EXE
            1/2 1/2 1/2 Description=Epson TM-U950 Printer
            1/2 1/2 1/2 Version=1.0.7
            1/2 1/2 1/2 ...Service Object-specific values could go here.
          1/2 1/2 1/2
          1/2 1/2 1/2® PSI.Ptr.1=NCR7156 Logical Device Name
          1/2 1/2 1/2
          1/2 1/2 1/2® PSI.Ptr.2=Epson950 Logical Device Name
          1/2 1/2
          1/2 1/2® \CashDrawer Device Class Key
            1/2 1/2
            1/2 1/2® \EpsonCash=Epson.CD.1 Device Name Key
              1/2 1/2 Service=TMU950.EXE
              1/2 1/2 Description=Epson Cash Drawer Kickout on TM-U950
              1/2 1/2 Version=1.0.7
              1/2 1/2 ...Service Object-specific values. Might include:
              1/2 1/2 Uses=POSPrinter\Epson950
            1/2 1/2
            1/2 1/2® PSI.CD.1=EpsonCash Logical Device Name
          1/2
          1/2® \ServiceInfo
            1/2
            1/2® \EPSON
              1/2 InstallDir=C:\OPOS\EPSON
              1/2 InstallDate=1995/11/13
            -
  
```


APPENDIX C

OPOS Application Header Files

OPOS.H : Main OPOS Header File

```

/////////////////////////////////////////////////////////////////
//
// OPOS.H
//
// General header file for OPOS Applications.
//
// Modification history
// -----
// 95-12-08 OPOS Release 1.0 CRM
//
/////////////////////////////////////////////////////////////////

#ifndef OPOS_H
#define OPOS_H

/////////////////////////////////////////////////////////////////
// OPOS "State" Property Constants
/////////////////////////////////////////////////////////////////

const LONG OPOS_S_CLOSED          = 1;
const LONG OPOS_S_IDLE            = 2;
const LONG OPOS_S_BUSY            = 3;
const LONG OPOS_S_ERROR          = 4;

/////////////////////////////////////////////////////////////////
// OPOS "ResultCode" Property Constants
/////////////////////////////////////////////////////////////////

const LONG OPOSERR                = 100;
const LONG OPOSERREXT            = 200;

const LONG OPOS_SUCCESS           = 0;
const LONG OPOS_E_CLOSED          = 1 + OPOSERR;
const LONG OPOS_E_CLAIMED        = 2 + OPOSERR;
const LONG OPOS_E_NOTCLAIMED     = 3 + OPOSERR;
const LONG OPOS_E_NOSERVICE      = 4 + OPOSERR;
const LONG OPOS_E_DISABLED       = 5 + OPOSERR;
const LONG OPOS_E_ILLEGAL        = 6 + OPOSERR;
const LONG OPOS_E_NOHARDWARE     = 7 + OPOSERR;
const LONG OPOS_E_OFFLINE        = 8 + OPOSERR;
const LONG OPOS_E_NOEXIST        = 9 + OPOSERR;
const LONG OPOS_E_EXISTS         = 10 + OPOSERR;

```

```

const LONG OPOS_E_FAILURE      = 11 + OPOSERR;
const LONG OPOS_E_TIMEOUT     = 12 + OPOSERR;
const LONG OPOS_E_BUSY       = 13 + OPOSERR;
const LONG OPOS_E_EXTENDED    = 14 + OPOSERR;

/////////////////////////////////////////////////////////////////
// "CheckHealth" Method: "Level" Parameter Constants
/////////////////////////////////////////////////////////////////

const LONG OPOS_CH_INTERNAL    = 1;
const LONG OPOS_CH_EXTERNAL   = 2;
const LONG OPOS_CH_INTERACTIVE = 3;

/////////////////////////////////////////////////////////////////
// "ErrorEvent" Event: "ErrorLocus" Parameter Constants
/////////////////////////////////////////////////////////////////

const LONG OPOS_EL_OUTPUT     = 1;
const LONG OPOS_EL_INPUT     = 2;
const LONG OPOS_EL_INPUT_DATA = 3;

/////////////////////////////////////////////////////////////////
// "ErrorEvent" Event: "ErrorResponse" Constants
/////////////////////////////////////////////////////////////////

const LONG OPOS_ER_RETRY      = 11;
const LONG OPOS_ER_CLEAR     = 12;
const LONG OPOS_ER_CONTINUEINPUT= 13;

#endif // !defined(OPOS_H)

```

OPOSCASH.H : Cash Drawer Header File

```

/////////////////////////////////////////////////////////////////
//
// OPOSCASH.H
//
// Cash Drawer header file for OPOS Applications.
//
// Modification history
// -----
// 95-12-08 OPOS Release 1.0 CRM
//
/////////////////////////////////////////////////////////////////

#if !defined(OPOSCASH_H)
#define OPOSCASH_H

#include "opos.h"

// No definitions required for this version.

#endif // !defined(OPOSCASH_H)

```

OPOSCOIN.H : Coin Dispenser Header File

```

////////////////////////////////////
//
// OPOSCOIN.H
//
// Coin Dispenser header file for OPOS Applications.
//
// Modification history
// -----
// 95-12-08 OPOS Release 1.0 CRM
//
////////////////////////////////////

#if !defined(OPOSCOIN_H)
#define OPOSCOIN_H

#include "opos.h"

////////////////////////////////////
// "DispenserStatus" Property Constants
// "StatusUpdateEvent" Event: "Data" Parameter Constants
////////////////////////////////////

const LONG COIN_STATUS_OK = 1;
const LONG COIN_STATUS_EMPTY = 2;
const LONG COIN_STATUS_NEAREMPTY= 3;
const LONG COIN_STATUS_JAM = 4;

#endif // !defined(OPOSCOIN_H)

```

OPOSTOT.H : Hard Totals Header File

```
/////////////////////////////////////////////////////////////////
//
// OPOSTOT.H
//
//   Hard Totals header file for OPOS Applications.
//
// Modification history
// -----
// 95-12-08 OPOS Release 1.0                                CRM
//
/////////////////////////////////////////////////////////////////

#if !defined(OPOSTOT_H)
#define      OPOSTOT_H

#include "opos.h"

/////////////////////////////////////////////////////////////////
// "ResultCodeExtended" Property Constants for Hard Totals
/////////////////////////////////////////////////////////////////

const LONG OPOS_ETOT_NOROOM      = 1 + OPOSERREXT; // Create, Write
const LONG OPOS_ETOT_VALIDATION = 2 + OPOSERREXT; // Read, Write

#endif          // !defined(OPOSTOT_H)
```

OPOSDISP.H : Line Display Header File

```
/////////////////////////////////////////////////////////////////
//
// OPOSDISP.H
//
// Line Display header file for OPOS Applications.
//
// Modification history
// -----
// 95-12-08 OPOS Release 1.0 CRM
// 96-03-18 OPOS Release 1.01 CRM
// Add DISP_MT_INIT constant and MarqueeFormat constants.
// 96-04-22 OPOS Release 1.1 CRM
// Add CapCharacterSet values for Kana and Kanji.
//
/////////////////////////////////////////////////////////////////

#if !defined(OPOSDISP_H)
#define OPOSDISP_H

#include "opos.h"

/////////////////////////////////////////////////////////////////
// "CapBlink" Property Constants
/////////////////////////////////////////////////////////////////

const LONG DISP_CB_NOBLINK = 0;
const LONG DISP_CB_BLINKALL = 1;
const LONG DISP_CB_BLINKEACH = 2;

/////////////////////////////////////////////////////////////////
// "CapCharacterSet" Property Constants
/////////////////////////////////////////////////////////////////

const LONG DISP_CCS_NUMERIC = 0;
const LONG DISP_CCS_ALPHA = 1;
const LONG DISP_CCS_ASCII = 998;
const LONG DISP_CCS_KANA = 10;
const LONG DISP_CCS_KANJI = 11;

/////////////////////////////////////////////////////////////////
// "CharacterSet" Property Constants
/////////////////////////////////////////////////////////////////

const LONG DISP_CS_ASCII = 998;
const LONG DISP_CS_WINDOWS = 999;

/////////////////////////////////////////////////////////////////
// "MarqueeType" Property Constants
/////////////////////////////////////////////////////////////////

const LONG DISP_MT_NONE = 0;
const LONG DISP_MT_UP = 1;
const LONG DISP_MT_DOWN = 2;
```

```
const LONG DISP_MT_LEFT      = 3;
const LONG DISP_MT_RIGHT    = 4;
const LONG DISP_MT_INIT     = 5;

////////////////////////////////////
// "MarqueeFormat" Property Constants
////////////////////////////////////

const LONG DISP_MF_WALK      = 0;
const LONG DISP_MF_PLACE    = 1;

////////////////////////////////////
// "DisplayText" Method: "Attribute" Property Constants
// "DisplayTextAt" Method: "Attribute" Property Constants
////////////////////////////////////

const LONG DISP_DT_NORMAL   = 0;
const LONG DISP_DT_BLINK    = 1;

////////////////////////////////////
// "ScrollText" Method: "Direction" Parameter Constants
////////////////////////////////////

const LONG DISP_ST_UP       = 1;
const LONG DISP_ST_DOWN     = 2;
const LONG DISP_ST_LEFT     = 3;
const LONG DISP_ST_RIGHT    = 4;

////////////////////////////////////
// "SetDescriptor" Method: "Attribute" Parameter Constants
////////////////////////////////////

const LONG DISP_SD_OFF      = 0;
const LONG DISP_SD_ON       = 1;
const LONG DISP_SD_BLINK    = 2;

#endif // !defined(OPOSDISP_H)
```

OPOSLOCK.H : Keylock Header File

```

////////////////////////////////////
//
// OPOSLOCK.H
//
// Keylock header file for OPOS Applications.
//
// Modification history
// -----
// 95-12-08 OPOS Release 1.0 CRM
//
////////////////////////////////////

#if !defined(OPOSLOCK_H)
#define OPOSLOCK_H

#include "opos.h"

////////////////////////////////////
// "KeyPosition" Property Constants
// "WaitForKeylockChange" Method: "KeyPosition" Parameter
// "StatusUpdateEvent" Event: "Data" Parameter
////////////////////////////////////

const LONG LOCK_KP_ANY = 0; // WaitForKeylockChange Only
const LONG LOCK_KP_LOCK = 1;
const LONG LOCK_KP_NORM = 2;
const LONG LOCK_KP_SUPR = 3;

#endif // !defined(OPOSLOCK_H)

```

OPOSMICR.H : MICR Header File

```

/////////////////////////////////////////////////////////////////
//
// OPOSMICR.H
//
// MICR header file for OPOS Applications.
//
// Modification history
// -----
// 95-12-08 OPOS Release 1.0 CRM
//
/////////////////////////////////////////////////////////////////

#if !defined(OPOSMICR_H)
#define OPOSMICR_H

#include "opos.h"

/////////////////////////////////////////////////////////////////
// "CheckType" Property Constants
/////////////////////////////////////////////////////////////////

const LONG MICR_CT_PERSONAL = 1;
const LONG MICR_CT_BUSINESS = 2;
const LONG MICR_CT_UNKNOWN = 99;

/////////////////////////////////////////////////////////////////
// "CountryCode" Property Constants
/////////////////////////////////////////////////////////////////

const LONG MICR_CC_USA = 1;
const LONG MICR_CC_CANADA = 2;
const LONG MICR_CC_MEXICO = 3;
const LONG MICR_CC_UNKNOWN = 99;

/////////////////////////////////////////////////////////////////
// "ResultCodeExtended" Property Constants for MICR
/////////////////////////////////////////////////////////////////

const LONG OPOS_EMICR_NOCHECK = 1 + OPOSERREXT; // EndInsertion
const LONG OPOS_EMICR_CHECK = 2 + OPOSERREXT; // EndRemoval

#endif // !defined(OPOSMICR_H)

```


OPOSMSR.H : MSR Header File

```

////////////////////////////////////
//
// OPOSMSR.H
//
// Magnetic Stripe Reader header file for OPOS Applications.
//
// Modification history
// -----
// 95-12-08 OPOS Release 1.0 CRM
//
////////////////////////////////////

#if !defined(OPOSMSR_H)
#define OPOSMSR_H

#include "opos.h"

////////////////////////////////////
// "TracksToRead" Property Constants
////////////////////////////////////

const LONG MSR_TR_1 = 1;
const LONG MSR_TR_2 = 2;
const LONG MSR_TR_3 = 4;

const LONG MSR_TR_1_2 = MSR_TR_1 | MSR_TR_2;
const LONG MSR_TR_1_3 = MSR_TR_1 | MSR_TR_3;
const LONG MSR_TR_2_3 = MSR_TR_2 | MSR_TR_3;

const LONG MSR_TR_1_2_3 = MSR_TR_1 | MSR_TR_2 | MSR_TR_3;

////////////////////////////////////
// "ErrorEvent" Event: "ResultCodeExtended" Parameter Constants
////////////////////////////////////

const LONG OPOS_EMSR_START = 1 + OPOSERREXT;
const LONG OPOS_EMSR_END = 2 + OPOSERREXT;
const LONG OPOS_EMSR_PARITY = 3 + OPOSERREXT;
const LONG OPOS_EMSR_LRC = 4 + OPOSERREXT;

#endif // !defined(OPOSMSR_H)

```

OPOSKBD.H : POS Keyboard Header File

```
////////////////////////////////////  
//  
// OPOSKBD.H  
//  
// POS Keyboard header file for OPOS Applications.  
//  
// Modification history  
// -----  
// 96-04-22 OPOS Release 1.1 CRM  
//  
////////////////////////////////////  
  
#if !defined(OPOSKBD_H)  
#define OPOSKBD_H  
  
#include "opos.h"  
  
// No definitions required for this version.  
  
#endif // !defined(OPOSKBD_H)
```

OPOSPTR.H : POS Printer Header File

```

/////////////////////////////////////////////////////////////////
//
// OPOSPTR.H
//
// POS Printer header file for OPOS Applications.
//
// Modification history
// -----
// 95-12-08 OPOS Release 1.0                                CRM
// 96-04-22 OPOS Release 1.1                                CRM
// Add CapCharacterSet values.
// Add ErrorLevel values.
// Add TransactionPrint Control values.
/////////////////////////////////////////////////////////////////

#if !defined(OPOSPTR_H)
#define OPOSPTR_H

#include "opos.h"

/////////////////////////////////////////////////////////////////
// Printer Station Constants
/////////////////////////////////////////////////////////////////

const LONG PTR_S_JOURNAL          = 1;
const LONG PTR_S_RECEIPT         = 2;
const LONG PTR_S_SLIP            = 4;

const LONG PTR_S_JOURNAL_RECEIPT= PTR_S_JOURNAL | PTR_S_RECEIPT;
const LONG PTR_S_JOURNAL_SLIP   = PTR_S_JOURNAL | PTR_S_SLIP ;
const LONG PTR_S_RECEIPT_SLIP   = PTR_S_RECEIPT | PTR_S_SLIP ;

/////////////////////////////////////////////////////////////////
// "CapCharacterSet" Property Constants
/////////////////////////////////////////////////////////////////

const LONG PTR_CCS_ALPHA         = 1;
const LONG PTR_CCS_ASCII         = 998;
const LONG PTR_CCS_KANA          = 10;
const LONG PTR_CCS_KANJI         = 11;

/////////////////////////////////////////////////////////////////
// "CharacterSet" Property Constants
/////////////////////////////////////////////////////////////////

const LONG PTR_CS_ASCII          = 998;
const LONG PTR_CS_WINDOWS        = 999;

/////////////////////////////////////////////////////////////////
// "ErrorLevel" Property Constants
/////////////////////////////////////////////////////////////////

const LONG PTR_EL_NONE           = 1;

```

```

const LONG PTR_EL_RECOVERABLE = 2;
const LONG PTR_EL_FATAL      = 3;

/////////////////////////////////////////////////////////////////
// "MappingMode" Property Constants
/////////////////////////////////////////////////////////////////

const LONG PTR_MM_DOTS        = 1;
const LONG PTR_MM_TWIPS      = 2;
const LONG PTR_MM_ENGLISH    = 3;
const LONG PTR_MM_METRIC     = 4;

/////////////////////////////////////////////////////////////////
// "PaperCut" Method Constant
/////////////////////////////////////////////////////////////////

const LONG PTR_CP_FULLLCUT   = 100;

/////////////////////////////////////////////////////////////////
// "PrintBarCode" Method Constants:
/////////////////////////////////////////////////////////////////

// "Alignment" Parameter
//   Either the distance from the left-most print column to the start
//   of the bar code, or one of the following:

const LONG PTR_BC_LEFT      = -1;
const LONG PTR_BC_CENTER   = -2;
const LONG PTR_BC_RIGHT    = -3;

// "TextPosition" Parameter

const LONG PTR_BC_TEXT_NONE = -11;
const LONG PTR_BC_TEXT_ABOVE = -12;
const LONG PTR_BC_TEXT_BELOW = -13;

// "Symbology" Parameter:

//   One dimensional symbologies
const LONG PTR_BCS_UPCA     = 101; // Digits
const LONG PTR_BCS_UPCE     = 102; // Digits
const LONG PTR_BCS_JAN8     = 103; // (EAN)
const LONG PTR_BCS_JAN13    = 104; // (EAN)
const LONG PTR_BCS_TF       = 105; // (Discrete 2 of 5) Digits
const LONG PTR_BCS_ITF      = 106; // (Interleaved 2 of 5) Digits
const LONG PTR_BCS_Codabar  = 107; // Digits, -, $, :, /, ., +;
//   4 start/stop characters
//   (a, b, c, d)
const LONG PTR_BCS_Code39   = 108; // Alpha, Digits, Space, -, .,
//   $, /, +, %; start/stop (*)
//   Also has Full Ascii feature
const LONG PTR_BCS_Code93   = 109; // Same characters as Code 39
const LONG PTR_BCS_Code128  = 110; // 128 data characters

//   Two dimensional symbologies
const LONG PTR_BCS_PDF417   = 201;
const LONG PTR_BCS_MAXICODE = 202;

//   Start of Printer-Specific bar code symbologies

```

```
const LONG PTR_BCS_OTHER          = 501;

////////////////////////////////////////////////////////////////
// "PrintBitMap" Method Constants:
////////////////////////////////////////////////////////////////

// "Width" Parameter
//   Either bitmap width or:

const LONG PTR_BM_ASIS            = -11; // One pixel per printer dot

// "Alignment" Parameter
//   Either the distance from the left-most print column to the start
//   of the bitmap, or one of the following:

const LONG PTR_BM_LEFT            = -1;
const LONG PTR_BM_CENTER          = -2;
const LONG PTR_BM_RIGHT           = -3;

////////////////////////////////////////////////////////////////
// "RotatedPrint" Method: "Rotation" Parameter Constants
// "RotateSpecial" Property Constants (PTR_RP_NORMAL_ASYNC not legal)
////////////////////////////////////////////////////////////////

const LONG PTR_RP_NORMAL          = 0x0001;
const LONG PTR_RP_NORMAL_ASYNC    = 0x0002;

const LONG PTR_RP_RIGHT90         = 0x0101;
const LONG PTR_RP_LEFT90          = 0x0102;
const LONG PTR_RP_ROTATE180       = 0x0103;

////////////////////////////////////////////////////////////////
// "SetLogo" Method: "Location" Parameter Constants
////////////////////////////////////////////////////////////////

const LONG PTR_L_TOP              = 1;
const LONG PTR_L_BOTTOM           = 2;

////////////////////////////////////////////////////////////////
// "TransactionPrint" Method: "Control" Parameter Constants
////////////////////////////////////////////////////////////////

const LONG PTR_TP_TRANSACTION     = 11;
const LONG PTR_TP_NORMAL          = 12;

////////////////////////////////////////////////////////////////
// "StatusUpdateEvent" Event: "Data" Parameter Constants
////////////////////////////////////////////////////////////////

const LONG PTR_SUE_COVER_OPEN     = 11;
const LONG PTR_SUE_COVER_OK       = 12;

const LONG PTR_SUE_JRN_EMPTY      = 21;
const LONG PTR_SUE_JRN_NEAREMPTY  = 22;
const LONG PTR_SUE_JRN_PAPEROK    = 23;

const LONG PTR_SUE_REC_EMPTY      = 24;
```

```
const LONG PTR_SUE_REC_NEAREMPTY= 25;
const LONG PTR_SUE_REC_PAPEROK = 26;

const LONG PTR_SUE_SLP_EMPTY = 27;
const LONG PTR_SUE_SLP_NEAREMPTY= 28;
const LONG PTR_SUE_SLP_PAPEROK = 29;

const LONG PTR_SUE_IDLE = 1001;

////////////////////////////////////
// "ResultCodeExtended" Property Constants for Printer
////////////////////////////////////

const LONG OPOS_EPTR_COVER_OPEN = 1 + OPOSERREXT; // (Several)
const LONG OPOS_EPTR_JRN_EMPTY = 2 + OPOSERREXT; // (Several)
const LONG OPOS_EPTR_REC_EMPTY = 3 + OPOSERREXT; // (Several)
const LONG OPOS_EPTR_SLP_EMPTY = 4 + OPOSERREXT; // (Several)
const LONG OPOS_EPTR_SLP_FORM = 5 + OPOSERREXT; // EndRemoval
const LONG OPOS_EPTR_TOOBIG = 6 + OPOSERREXT; // PrintBitmap
const LONG OPOS_EPTR_BADFORMAT = 7 + OPOSERREXT; // PrintBitmap

#endif // !defined(OPOSPTR_H)
```

OPOSSCAL.H : Scale Header File

```

////////////////////////////////////
//
// OPOSSCAL.H
//
// Scale header file for OPOS Applications.
//
// Modification history
// -----
// 95-12-08 OPOS Release 1.0 CRM
//
////////////////////////////////////

#if !defined(OPOSSCAL_H)
#define OPOSSCAL_H

#include "opos.h"

////////////////////////////////////
// "WeightUnit" Property Constants
////////////////////////////////////

const LONG SCAL_WU_GRAM = 1;
const LONG SCAL_WU_KILOGRAM = 2;
const LONG SCAL_WU_OUNCE = 3;
const LONG SCAL_WU_POUND = 4;

////////////////////////////////////
// "ResultCodeExtended" Property Constants for Scale
////////////////////////////////////

const LONG OPOS_ESCAL_OVERWEIGHT= 1 + OPOSERREXT; // ReadWeight

#endif // !defined(OPOSSCAL_H)

```

OPOSSCAN.H : Bar Code Scanner Header File

```

////////////////////////////////////
//
// OPOSSCAN.H
//
// Scanner header file for OPOS Applications.
//
// Modification history
// -----
// 95-12-08 OPOS Release 1.0 CRM
//
////////////////////////////////////

#if !defined(OPOSSCAN_H)
#define OPOSSCAN_H

#include "opos.h"

// No definitions required for this version.

#endif // !defined(OPOSSCAN_H)

```

OPOSSIG.H : Signature Capture Header File

```

////////////////////////////////////
//
// OPOSSIG.H
//
// Signature Capture header file for OPOS Applications.
//
// Modification history
// -----
// 95-12-08 OPOS Release 1.0 CRM
//
////////////////////////////////////

#if !defined(OPOSSIG_H)
#define OPOSSIG_H

#include "opos.h"

// No definitions required for this version.

#endif // !defined(OPOSSIG_H)

```


A P P E N D I X D

Technical Details

System Strings (BSTR)

System String Characteristics

OPOS uses OLE system strings to pass and return data of variable length. System strings are often referred to as BStrings, and are assigned the type BSTR by Microsoft Visual C++.

A system string consists of a sequence of Unicode characters, which are each 16-bits wide. Thus, they are also referred to as “wide” characters. The string is followed by a NUL, or zero, character. The string is preceded by an unsigned long count of the bytes in the string, not including the NUL. Divide this count by two to obtain the number of characters in the string.

Most of the time, OPOS uses system strings to pass character data back and forth among the Application, Control Object, and System Object. A system string (BSTR) is used to pass string parameters by methods and to return string properties. A pointer to a system string (BSTR*) is used as a method parameter when the method must return string data.

System String Usage

Visual Basic 4.0 both receives and sends system strings without any complications. The internal representation of VB strings is as wide characters with a length component. A BSTR may be passed using a variable, a string expression, or a literal. A BSTR* requires use of a variable, so that the data may be modified by the method.

Visual C++ 4.0, however, requires more consideration.

BSTR is usually quite straightforward to use:

- BSTR Method Parameters
 - ◆ **Calling Function** Calling an OLE automation method with a BSTR parameter is treated by VC++ as a pointer to a character string, LPCTSTR. If the VC++ ANSI option is used, MFC takes care of conversion from ANSI to Unicode.
 - ◆ **Called Function** The function implementing an OLE automation method receives a BSTR parameter as a pointer to a character string, LPCTSTR. If the VC++ ANSI option is used, then MFC performs an automatic conversion from Unicode into ANSI before passing control to the function. The string length immediately precedes the string pointer.
- BSTR Return Type (used for getting properties)
 - ◆ **Calling Function** An OLE automation method returning a BSTR result is automatically converted by MFC into a CString.
 - ◆ **Called Function** An automation method returns a BSTR result by placing the data into an MFC CString object, and returning the result of the CString's "AllocSysString" member function. If the VC++ ANSI option is used, then this function automatically converts the string from ANSI into Unicode.

BSTR* can be a little more difficult to use in ANSI mode, since the string remains in Unicode format.

- To get the string, it must be converted from Unicode to MBCS. Some macros are available that make this conversion easier, such as T2OLE and OLE2T. (These do not handle NUL characters embedded in the string, however.)
- To set the string, place the data into an MFC CString object, and use CString's "SetSysString" member function.

System Strings and Binary Data

Sometimes OPOS uses BSTR and BSTR* to pass binary data. The current cases are:

- Hard Totals: **Write** (BSTR) and **Read** (BSTR*).
- MSR (when **DecodeData** is FALSE): **Track1Data** (BSTR), **Track1DiscretionaryData** (BSTR), **Track2Data** (BSTR), **Track2DiscretionaryData** (BSTR), **Track3Data** (BSTR).
- Signature Capture: **RawData** (BSTR) and **PointArray** (BSTR).
- All: **DirectIO** (BSTR*) and **DirectIOEvent** (BSTR*).

These cases may return byte data in the range 00-hex to FF-hex. Each 16-bit character of the system string contains one byte of binary data in the lower 8 bits. The upper 8 bits are zero. This ensures that translations between ANSI and Unicode formats maintain one byte per string character.

The troublesome character within binary data is the NUL character, or zero. This is because although system strings have a length component, some software still relies upon the NUL character to determine the end of the string.

System String Usage with Binary Data

Visual Basic 4.0 can build binary string data by using the **Chr**(*number*) function to create each character, where *number* ranges from 0 to 255. Each byte of binary data may be extracted by using **AscB**(**Mid**(*string*, *charindex*, 1)).

Visual C++ 4.0, again, requires more consideration.

Looking at the cases as with non-binary data, BSTR handling is as follows:

- BSTR Method Parameters
 - ◆ **Calling Function** This is the most difficult case. The automatic conversion from a LPCTSTR to a system string cannot be used if the data may contain NULs, since it terminates upon finding a NUL. See “Calling Methods with Binary BSTR Data” below for steps to handle this case.
 - ◆ **Called Function** The function receives a pointer to a character string, LPCTSTR. It must use the string length immediately preceding the string pointer.

- BSTR Return Type (used for getting properties)
 - ◆ **Calling Function** The automatic conversion by MFC into a CString properly handles binary data.
 - ◆ **Called Function** The CString “AllocSysString” member function properly handles binary data.

BSTR* handling for ANSI is as follows:

- To get the string, it must be converted from Unicode to MBCS. The conversion macros, such as T2OLE and OLE2T, stop on the first NUL character. Therefore, the function “WideCharToMultiByte” must be used.
- To set the string, place the data into an MFC CString object, and use CString's “SetSysString” member function.

Calling Methods with Binary BSTR Data

When a VC++ project inserts an OLE Control, VC++ generates a wrapper class for the control, so that the methods and properties may be accessed. Member functions of this class handle placing parameters into the format required to call across the OLE IDispatch interface into the control.

The generated member functions for calling a method with a BSTR parameter or for setting a BSTR property use LPCTSTR as the input parameter, and convert this NUL-terminated string into a system string. Thus, this member function may not be used for passing binary data with NULs.

The solution involves manually overloading the generated method to accept a “const CString&”. Then, the application may set a CString to the binary data and call the new function.

For example, if the control has a method “long SendBstring(BSTR String)”, the generated wrapper class will have a function similar to the following:

```
long xxx::SendBstring(LPCTSTR String)
{
    long result;
    static BYTE parms[] = VTS_BSTR;
    InvokeHelper(???, // ??? is the dispatch ID for the method.
                DISPATCH_METHOD,
                VT_I4, (void*)&result, // Returns a 4-byte integer.
                parms, String);        // Sends one BSTR parameter.
    return result;
}
```

Add the following overloaded function to the class declaration header file:

```
long SendBstring(const CString& String);
```

and add the following definition to the class definition source file:

```
long xxx::SendBstring(const CString& String);
{
    long result;
    static BYTE parms[] = VTS_VARIANT;
    VARIANT VarString;
    VariantInit(&VarString);
    VarString.vt = VT_BSTR;
    VarString.bstrVal = String.AllocSysString();
    InvokeHelper(???, // ??? is the dispatch ID for the method.
        DISPATCH_METHOD,
        VT_I4, (void*)&result, // Returns a 4-byte integer.
        parms, &VarString); // Sends one VARIANT parameter.
    VariantClear(&VarString);
    return result;
}
```

To call the method with binary data, use a sequence such as:

```
CString s;
.... Put string (which may contain NULs) into "s" ....
.... Then, assuming that bs is an instance of the class "xxx":
long r = bs.SendBstring(s);
```

Event Handlers

Event Characteristics

Events require some special consideration, due to the environment in which they are fired into the application: OPOS events are fired by a thread created by the Service Object, and not by one of the application's threads. (The Service Object ensures that it calls at most one event handler at a time.)

A separate event thread is used for two reasons:

- Events should be delivered to the application as soon as possible. Waiting for an application thread to become idle so that it can fire events might delay the event for an indefinite period of time. Timeliness is especially important for the **ErrorEvent**.
- Events may be delayed by the **DataEventEnabled** property being FALSE (for **DataEvent** and input **ErrorEvent**), by the container freezing events, or by the application setting **FreezeEvents** to TRUE. A separate event firing thread can enqueue events and fire them as soon as these conditions are cleared.

On the other hand, a separate thread introduces some complexities due to the asynchronous nature of fired events: They may compete with application threads for access to the Control, user interface objects, and so on.

In most cases, problems may be avoided by restricting event handlers both in Control access and user interface. Event handlers usually may be limited to manipulating application flags and variables and returning from the event.

Occasionally, the event handling may need to be more robust. Most notably, the POS printer **ErrorEvent** handler may need to prompt for reinsertion of slips, replacement of paper, and so on.

Restrictions and Cautions

The application should refrain from certain actions on a Control from within an event handler. The following table lists the methods which may be called and the properties which may be written within an event handler. Properties may be read within a handler.

Event Type	Callable Methods	Writable Properties
ErrorEvent	<i>Class-specific restrictions</i>	DataEventEnable All class-specific Properties
DataEvent	None	DataEventEnable All class-specific Properties
OutputCompleteEvent	None	None
StatusUpdateEvent	None	None

The *Class-specific restrictions* on **ErrorEvent** callable methods are:

- Printer: May call all class-specific methods plus **ClearOutput**.
- MICR: May call all class-specific methods plus **ClearInput**.
- MSR, Scanner, and Signature Capture: May call **ClearInput**.

An event handler must use extreme care if it needs to manipulate or access windows created by an application thread. Win32 requires that the thread which created the window is the only thread that manipulates it. If the thread of the event handler accesses a window created by the application, then interthread messages are sent to the window and will be processed when that window's thread accesses its message queue. Since the application thread continues running while the event handler runs, some synchronization may be needed. Notice that if the application thread is waiting in its message loop, there is no issue. However, if the thread is actively updating the display, then contention occurs.

A better and safer alternative for getting user input may be for the event handler to create a user-interface thread which creates and processes a dialog window for user input, then destroys the window and thread before returning from the handler. (This applies to Visual C++ applications rather than Visual Basic.) Also, the event handler may be able to use a message box with Retry and Cancel options.

In general, the application developer must carefully design the interaction between the main application thread(s) and event handlers, so that conflicts between them may be avoided.

End of Application Programmer's Guide