OLE for Retail POS

Application Programmer's Guide

Release 1.4 **September 23, 1998**

International Standard

Windows 95/98, Windows NT, or other OLE/ActiveX compliant 32-bit operating system

OLE for Retail POS Committee

Core Companies
Epson
Fujitsu/ICL
Microsoft
NCR

plus
OPOS-Japan
OPOS-Europe

OLE for Retail POS

Application Programmer’ s Guide

Information in this document is subject to change without notice.

Also see the following Web sites for OPOS information:

“OPOS Home Page” – Primary repository of OPOS documentation:
    http://www.ncr.com/product/retail/products/software/OposHome.html
Microsoft Retail Industry Page:
    http://www.microsoft.com/industry/retail_dist/

Table of Contents

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 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.

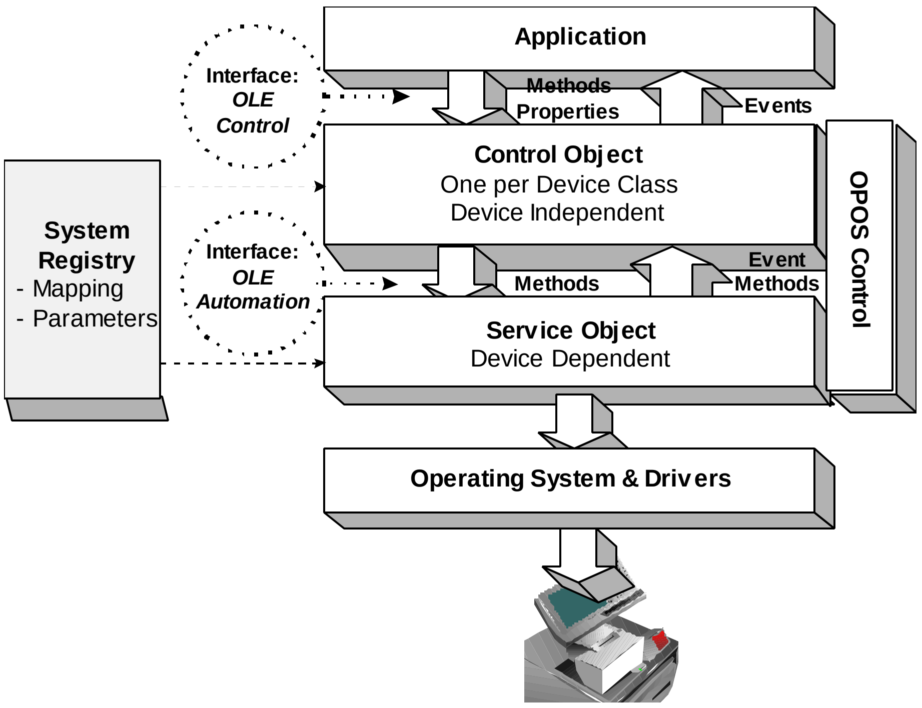
This guide assumes that the reader is familiar with the following:

- 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 or Microsoft Visual C++, 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 the layers 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 delivered 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). In theory, it may also be implemented as a local out-proc server (in a separate executable process). However, we have found that, in practice, out-proc servers do not work well for OPOS Service Objects, and do not recommend their use.

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**, **AutoDisable, DataCount,** and **BinaryConversion,** as well as descriptions and version numbers of the OPOS Control layers. Additional class-specific properties may also be initialized.

Several applications may have an OPOS Control open at the same time. Therefore, after the device is opened, the application will often need to call the **Claim** method to gain exclusive access to the device. Many devices must be **Claim**ed 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 20 for more information).
- **DeviceEnabled** property: Set to TRUE to make the device operational. (For sharable devices, the device may be enabled without first **Claim**ing 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**
Boolean	FALSE
Long	0
String	"[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. ***Release 1.3 and later:*** All devices may be able to report device power state. See "Device Power Reporting Model" on page 28.
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.
- The Control must select one of the result codes listed on page 51. If the Control sets **ResultCode** to OPOS_E_EXTENDED, then it must set **ResultCodeExtended** to one of the values specified in the device class documentation. (That is, when this **ResultCode** value is selected, then **ResultCodeExtended** may only contain one of the values listed in this document for the device class, in the appropriate method or property section.)

  If the Control sets **ResultCode** to a value other than OPOS_E_EXTENDED, then the Service Object may set the **ResultCodeExtended** property to any 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 – 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.
    If the locus is Input: Some devices support retrying the input, if retry can be controlled by the Service Object.
    "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 deliver additional data events as directed by the **DataEventEnabled** property. When all input has been delivered and the **DataEventEnabled** property is again set to TRUE, then another **ErrorEvent** is delivered 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 deliver any other **ErrorEvent**s.

Device Sharing Model

The OLE for Retail POS device sharing model supports devices that are to be used exclusively by one application[3] 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 14, for other details.) All OPOS Controls may be opened by more than one application at a given time. Some or many of the activities that an application can perform with the Control, however, may be restricted to an application that claims access to the device.

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 **Claim**ed it, or may fire some events only to this application.

[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.

Note

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

Events

OLE for Retail POS uses events to inform an application of various activities or changes with the OPOS Control. The five event types follow. Subsequent sections will clarify their definitions.

- **DataEvent**: Input data has been placed into device class-specific properties.
- **ErrorEvent**: An error has occurred during event-driven input or asynchronous output.
- **StatusUpdateEvent**: Reports a change in the device’ s status.
- **OutputCompleteEvent**: An asynchronous output has successfully completed.
- **DirectIOEvent**: This event may be defined by a Service Object provider for purposes not covered by the specification.

The Service Object enqueues events as they occur. Often these events will be enqueued by worker threads, rather than the application’ s thread. Enqueued events are delivered to the application when conditions are correct. Conditions which delay the delivery of events include:

- The application thread is busy processing other messages.
  OPOS Controls are to follow the OLE Apartment Threading model. According to OLE Apartment Threading rules, events are to be delivered on the thread that created the COM object, which will usually be the application’ s main thread. If the application is processing another message, then event delivery must wait until this processing has finished.
- The application has set the property **FreezeEvents** to TRUE. (See page 47.)
- The event type is **DataEvent** or **ErrorEvent** but the property **DataEventEnabled** is FALSE. (See “ Input Model” on page 24.)

If the oldest enqueued event is blocked for one of these reasons, then all newer events may also be blocked. That is, the delivery of enqueued events is typically in a strict first in, first out order. Priority is not given to any event types on the queue.

Note – Terminology

The following event terminology is used rather consistently in this document. Some implementations may vary from the model described here, but the net effect is similar:

- **Enqueue**: When the Service Object determines that an event needs to be fired to the Application, it enqueues the event on an internal event queue. Event queuing typically occurs from one or more internal Service Object worker threads.
- **Deliver**: When the event queue is non-empty and all conditions are met for the top event on the queue, this event is removed from the queue and delivered to the Application. Event delivery is typically managed by a dedicated internal Service Object worker thread. This thread ensures that events are delivered in the context of the thread that created the Control, in order to adhere to the Apartment Threading model.
- **Fire**: The combination of enqueuing and delivering an event.
  Sometimes, the term is used more loosely and may only refer to one of these steps. The reader should differentiate these cases by context.

Rules on the management of the queue of events are:

- The Control may only enqueue new events while the device is enabled.
- The Control may deliver enqueued events until the application calls the **Release** method (for exclusive-use devices) or the **Close** method (for any device), at which time any remaining events are deleted.
- For input devices, the **ClearInput** method clears data and error events.

While within an event handler, the application may access properties and call methods. However, the application must not call the **Release** or **Close** methods from an event handler, since **Release** may shut down event handling (possibly including a thread that caused the event to be delivered) and **Close** must shut down event handling before returning.

Input Model

The OLE for Retail POS input model supports event-driven input. Event-driven input allows input data to be received after **DeviceEnabled** is set to TRUE. Received data is enqueued as a **DataEvent**, which is delivered to the application when preconditions are correct. If the **AutoDisable** property is TRUE when data is received, then the control will automatically disable itself, setting **DeviceEnabled** to FALSE. This will inhibit the Control from enqueuing further input and, when possible, physically disable the device.

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 enqueues and delivers a **DataEvent**. (If input has already been enqueued, the **DataEvent** will be delivered.) 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 delivering this event, the Control disables further data events by setting the **DataEventEnabled** property to FALSE. This causes subsequent input data to be enqueued by 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.

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 the Control 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 **Claim**s 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 the Control encounters an error while gathering or processing event-driven input, then the Control changes its state to Error, and enqueues one or two **ErrorEvents** to alert the application of the error condition. This event (or events) is not delivered until the **DataEventEnabled** property is TRUE, so that orderly application sequencing occurs. Error events are delivered with the following loci:

- InputWithData (OPOS_EL_INPUT_DATA) – Only enqueued if the error occurred while one or more **DataEvent**s are enqueued. It is enqueued ahead of all other **DataEvent**s. (A typical implementation would place it at the head of the event queue.) 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_EL_INPUT) – Delivered when an error has occurred and there is no data available. (A typical implementation would place it at the tail of the event queue.) If some input data was already enqueued when the error occurred, then an **ErrorEvent** with the locus "InputWithData" was enqueued and delivered first, and then this error event is delivered after all **DataEvent**s have been fired. (If an "InputWithData" event was delivered and the application event handler responded with a "Clear", then this "InputNoData" event is not delivered.)

The Control exits the Error state when one of the following occurs:

- The application returns from the InputNoData **ErrorEvent**.
- The application calls the **ClearInput** method.

For some Controls, the Application must call a method to begin event driven input. After the input is received by the Control, then typically no additional input will be received until the method is called again to reinitiate input. Examples are the MICR and Signature Capture devices. This variation of event driven input is sometimes called "asynchronous input."

The **DataCount** property may be read to obtain the number of **DataEvent**s enqueued by the Control.

All input enqueued by a Control may be deleted by calling the **ClearInput** method. **ClearInput** may be called after **Open** for sharable devices and after **Claim** for exclusive-use devices.

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. An example is the Keylock device. This type of input is sometimes called "synchronous input."

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. The Control is in the Error state while the **ErrorEvent** is in progress. (Note that if the condition causing the error was not corrected, then the Control may immediately reenter the Error state and fire another **ErrorEvent**.)

Asynchronous output is performed on a first-in first-out basis.

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

Device Power Reporting Model

Added in OPOS Release 1.3.

Applications frequently need to know the power state of the devices they use. Earlier versions of OPOS had no consistent method for reporting this information. **Note**: This model is not intended to report PC or POS Terminal power conditions (such as "on battery" and "battery low"). Reporting of these conditions is left to PC power management standards and APIs.

Model

OPOS segments device power into three states:

- ONLINE: The device is powered on and ready for use. This is the "operational" state.
- OFF: The device is powered off or detached from the terminal. This is a "non-operational" state.
- OFFLINE: The device is powered on but is either not ready or not able to respond to requests. It may need to be placed online by pressing a button, or it may not be responding to terminal requests. This is a "non-operational" state.

In addition, one combination state is defined:

- OFF_OFFLINE: The device is either off or offline, and the Service Object cannot distinguish these states.

Power reporting only occurs while the device is Open, Claimed (if the device is exclusive-use), and Enabled.

Note – Enabled/Disabled vs. Power States

These states are different and usually independent. OPOS defines "disabled" / "enabled" as a logical state, whereas the power state is a physical state. A device may be logically "enabled" but physically "offline". It may also be logically "disabled" but physically "online". Regardless of the physical power state, OPOS only reports the state while the device is enabled. (This restriction is necessary because a Service Object typically can only communicate with the device while enabled.)

If a device is "offline", then a Service Object may choose to fail an attempt to "enable" the device. However, once enabled, the Service Object may not disable a device based on its power state.

Properties

The OPOS device power reporting model adds the following common elements across all device classes:

- **CapPowerReporting** property: Identifies the reporting capabilities of the device. This property may be one of:
  - ♦ OPOS_PR_NONE: The Service Object cannot determine the state of the device. Therefore, no power reporting is possible.
  - ♦ OPOS_PR_STANDARD: The Service Object can determine and report two of the power states – OFF_OFFLINE (that is, off or offline) and ONLINE.
  - ♦ OPOS_PR_ADVANCED: The Service Object can determine and report all three power states – ONLINE, OFFLINE, and OFF.
- **PowerState** property: Maintained by the Service Object at the current power condition, if it can be determined. This property may be one of:
  - ♦ OPOS_PS_UNKNOWN
  - ♦ OPOS_PS_ONLINE
  - ♦ OPOS_PS_OFF
  - ♦ OPOS_PS_OFFLINE
  - ♦ OPOS_PS_OFF_OFFLINE
- **PowerNotify** property: The Application may set this property to enable power reporting via **StatusUpdateEvents** and the **PowerState** property. This property may only be set before the device is enabled (that is, before **DeviceEnabled** is set to TRUE). This restriction allows simpler implementation of power notification with no adverse effects on the application. The application is either prepared to receive notifications or doesn’ t want them, and has no need to switch between these cases. This property may be one of:
  - ♦ OPOS_PN_DISABLED
  - ♦ OPOS_PN_ENABLED

Power Reporting Requirements for DeviceEnabled

The following semantics are added to **DeviceEnabled** when
CapPowerReporting is not OPOS_PR_NONE, and
PowerNotify is OPOS_PN_ENABLED:

- When the Control changes from **DeviceEnabled** FALSE to TRUE, then begin monitoring the power state:
    - ♦ If the device is ONLINE, then:
        - ▪ **PowerState** is set to OPOS_PS_ONLINE.
        - ▪ A **StatusUpdateEvent** is fired with *Status* parameter set to OPOS_SUE_POWER_ONLINE.
    - ♦ If the device power state is OFF, OFFLINE, or OFF_OFFLINE, then the Control may choose to fail the enable, setting **ResultCode** to OPOS_E_NOHARDWARE or OPOS_E_OFFLINE.

      However, if there are no other conditions that cause the enable to fail, and the Control chooses to return success for the enable, then:
        - ▪ **PowerState** is set to OPOS_PS_OFF, OPOS_PS_OFFLINE, or OPOS_PS_OFF_OFFLINE.
        - ▪ A **StatusUpdateEvent** is fired with *Status* parameter set to OPOS_SUE_POWER_OFF, OPOS_SUE_POWER_OFFLINE, or OPOS_SUE_POWER_OFF_OFFLINE.
- When the Control changes from **DeviceEnabled** TRUE to FALSE, then OPOS assumes that the Control is no longer monitoring the power state. Therefore:

  PowerState is set to OPOS_PS_UNKNOWN.

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:

- Bump Bar ***Added in Release 1.3***
- Cash Changer ***Added in Release 1.2***
- Cash Drawer
- Credit Authorization Terminal (CAT) ***Added in Release 1.4***
- Coin Dispenser (Largely superseded by the Cash Changer in Release 1.2)
- Fiscal Printer ***Added in Release 1.3***
- Line Display
- Hard Totals
- Keylock
- Magnetic Ink Character Recognition (MICR) Reader
- Magnetic Stripe Reader (MSR)
- PIN Pad ***Added in Release 1.3***
- POS Keyboard ***Added in Release 1.1***
- POS Printer
- Remote Order Display ***Added in Release 1.3***
- Scale
- Scanner – Bar Code Reader
- Signature Capture
- Tone Indicator ***Added in Release 1.2***

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 723) for more information.
BSTR*	A pointer to a character string.
LONG	An integer with a size of 32 bits.
LONG*	A pointer to a 32-bit integer.
CURRENCY	***Release 1.3 and later*** A monetary value. An integer with a size of 64 bits. The value assumes four decimal places. For example, if the integer is "1234567", then the value is "123.4567".
CURRENCY*	***Release 1.3 and later*** A pointer to a CURRENCY value.

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

Name		*Type*	*Access*
AutoDisable	1.2	Boolean	R/W
BinaryConversion	1.2	Long	R/W
CapPowerReporting	1.3	Long	R
CheckHealthText	1.0	String	R
Claimed	1.0	Boolean	R
DataCount	1.2	Long	R
DataEventEnabled	1.0	Boolean	R/W
DeviceEnabled	1.0	Boolean	R/W
FreezeEvents	1.0	Boolean	R/W
OutputID	1.0	Long	R
PowerNotify	1.3	Long	R/W
PowerState	1.3	Long	R
ResultCode	1.0	Long	R
ResultCodeExtended	1.0	Long	R
State	1.0	Long	R
ControlObjectDescription	1.0	String	R
ControlObjectVersion	1.0	Long	R
ServiceObjectDescription	1.0	String	R
ServiceObjectVersion	1.0	Long	R
DeviceDescription	1.0	String	R
DeviceName	1.0	String	R

Methods

Name	
Open	1.0
Close	1.0
Claim	1.0
Release	1.0
CheckHealth	1.0
ClearInput	1.0
ClearOutput	1.0
DirectIO	1.0

Events

Name	
DataEvent	1.0
DirectIOEvent	1.0
ErrorEvent	1.0
OutputCompleteEvent	1.0
StatusUpdateEvent	1.0

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 that do not apply to that class as “Not Supported.” These are not present in the class’ controls.

Properties

AutoDisable Property R/W *Added in Release 1.2*

Syntax **BOOL AutoDisable;**

Remarks This property applies to event-driven input devices. It provides the application with an additional option for controlling the receipt of input data. If an application wants to receive and process only one input, or only one input at a time, then this property may be set to TRUE.

When TRUE, then as soon as the Service Object receives and enqueues data to be fired as a **DataEvent**, then it sets **DeviceEnabled** = FALSE. Before any additional input can be received, the application must set **DeviceEnabled** = TRUE.

When FALSE, the Service Object does not automatically disable the device when data is received. This is the behavior of OPOS controls prior to Release 1.2.

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.

See Also "Input Model"

BinaryConversion Property R/W *Added in Release 1.2*

Syntax **LONG BinaryConversion;**

Remarks OPOS passes multicharacter input and output using BStrings. BStrings may be safely used for text data. As the BStrings are passed between the application and the OPOS Control, OLE may perform language-specific translations to or from Unicode.

When BStrings are used to pass binary data, then these translations may alter the data such that the data byte in a BString character at the application does not match the corresponding byte at the Control. This mismatch is more likely when BString pointers are used, since the Unicode characters are presented to the application and/or Control, and a language difference between them may cause misinterpretation. (This was first reported with Japanese, which uses the MBCS Code Page 932, but can occur with other languages, also.)

Characters between 0x00 and 0x7F may be sent without fear of language-specific translation. Only characters between 0x80 and 0xFF sometimes cause incorrect translations.

This document specifies those properties and method parameters that are affected by **BinaryConversion** in the individual property and method descriptions. The following line is added to their description:

> The format of this data depends upon the value of the **BinaryConversion** property. See page 37.

The binary conversion values are:

Value	**Meaning**
OPOS_BC_NONE	Data is placed one byte per BString character, with no conversion. (This is the default, and is the behavior of OPOS Service Objects prior to 1.2.)
OPOS_BC_NIBBLE	Each byte is converted into two characters. (This option provides for the fastest conversion between binary and ASCII characters.) Each data byte is converted as follows: First character = 0x30 + bits 7-4 of the data byte. Second character = 0x30 + bits 3-0 of the data byte.

	Example: Byte value 154 = 0x9A is converted into the characters 0x39 0x3A (= the string "9:"). Note that this conversion is not the more common hexadecimal ASCII, which would have converted 154 to 0x39 0x41 (= the string "9A").
OPOS_BC_DECIMAL	Each byte is converted into three characters. (This option provides for the easiest conversion between binary and ASCII characters for Visual Basic and similar languages.) VAL(*string)* may be used on each 3 characters to convert from ASCII to binary. RIGHT("^^"+STR(*byte*), 3) may be used to produce 3 ASCII characters from each byte, where '^' represents the space character. Example 1: Byte value 154 = 0x9A becomes the characters 0x31 0x35 0x34 (= the string "154"). Example 2: Byte value 8 becomes the characters 0x30 0x30 0x38 (= the string "008"). Requirements for a Service Object are: (1) When the Service Object converts from ASCII to binary, it must allow either leading spaces or ASCII zeroes, since STR(*byte*) produces a leading space. (For example, the application may pass "^^8^27", where '^' represents the space character, which will be interpreted as the two bytes 8 (0x08) and 27 (0x1B).) (2) When the Service Object converts from binary to ASCII, is must always convert each byte into exactly three ASCII decimal characters (range 0x30 to 0x39).

When **BinaryConversion** is on (that is, not OPOS_BC_NONE) and the property or method parameter description specifies that **BinaryConversion** applies, then the application has the following responsibilities:

- Before setting the property or passing the method parameter, convert the string data into the format specified by the **BinaryConversion** value.
- After getting the property or receiving the method parameter, convert the string data from the format specified by the **BinaryConversion** value.

This property is initialized to OPOS_BC_NONE 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 illegal value was specified.

CapPowerReporting Property *Added in Release 1.3*

Syntax **LONG CapPowerReporting;**

Remarks Identifies the reporting capabilities of the device.

The power reporting values are:

Value	Meaning
OPOS_PR_NONE	The Service Object cannot determine the state of the device. Therefore, no power reporting is possible.
OPOS_PR_STANDARD	The Service Object can determine and report two of the power states – OFF_OFFLINE (that is, off or offline) and ONLINE.
OPOS_PR_ADVANCED	The Service Object can determine and report all three power states – OFF, OFFLINE, and ONLINE.

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

See Also “Device Power Reporting Model”; **PowerState** Property, **PowerNotify** Property

CheckHealthText Property

Syntax **BSTR CheckHealthText;**

Remarks Holds the results of the most recent call to the **CheckHealth** method. The following examples illustrate some possible diagnoses:

- "Internal HCheck: Successful"
- "External HCheck: Not Responding"
- "Interactive HCheck: Complete"

Before the first **CheckHealth** method call, its value is uninitialized.

See Also **CheckHealth** Method

Claimed Property

Syntax **BOOL Claimed;**

Remarks If TRUE, the device is claimed for exclusive access.
If FALSE, the device is released for sharing with other applications.

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.

The value of **Claimed** is initialized to FALSE by the **Open** method.

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

Note

A Control Object for a device class will operate with any Service Object for that class, as long as its major version number matches the Service Object’ s major version number. If they match, but the Control Object’ s minor version number is greater than the Service Object’ s minor version number, then the Control Object may support some new methods or properties that are not supported by the Service Object’ s release.

The following rules apply to APIs supported by the Control Object’ s release but not supported by the Service Object’ s older release:

- Reading an unsupported property: The Control Object returns the property’ s uninitialized value. (See page 14 for uninitialized property default values.)
- Writing an unsupported property: The Control Object returns, but must remember that an unsupported property write or method call occurred. Then, if the application reads the **ResultCode** property, the Control Object must return a value of OPOS_E_NOSERVICE (rather than reading the current **ResultCode** from the Service Object). It must do this until the next property write or method call, at which time **ResultCode** is set by that API.
- Calling an unsupported method: The Control Object returns a value of OPOS_E_NOSERVICE , and must remember that an unsupported property write or method call occurred. Then, if the application reads the **ResultCode** property, the Control Object must return a value of OPOS_E_NOSERVICE (rather than reading the current **ResultCode** from the Service Object). It must do this until the next property write or method call, at which time **ResultCode** is set by that API.

DataCount Property *Added in Release 1.2*

Syntax **LONG DataCount;**

Remarks Holds the number of enqueued **DataEvent**s at the control.

The application may interrogate **DataCount** to determine whether additional input is enqueued from a device, but has not yet been delivered because of other application processing, freezing of events, or other causes.

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

See Also “Input Model”; **DataEvent**

DataEventEnabled Property R/W

Syntax **BOOL DataEventEnabled;**

Remarks When TRUE, a **DataEvent** will be delivered as soon as input data is enqueued. If changed to TRUE and some input data is already queued, then a **DataEvent** is delivered immediately. (Note that other, less likely, conditions may delay "immediate" delivery: If **FreezeEvents** is TRUE or another event is already being processed at the application, the **DataEvent** will remain enqueued at the Service Object until the condition is corrected.)

When FALSE, input data is queued for later delivery to the application. Also, if an input error occurs, the **ErrorEvent** is not delivered while **DataEventEnabled** is FALSE.

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.

See Also "Input Model"; **DataEvent**

DeviceDescription Property

Syntax **BSTR DeviceDescription;**

Remarks String identifying the device.

The property identifies the device and any pertinent information about it. A sample returned string is:

```
“NCR 7192-0184 Printer, Japanese Version”
```

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

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.

Release 1.3 and later: The device' s power state may be reported while **DeviceEnabled** is TRUE. See "Device Power Reporting Model" for details.

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.
Other Values	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 deliver events. Events will be held by the Control until events are unfrozen.

When FALSE, the application allows events to be delivered. If some events have been held while events were frozen and all other conditions are correct for delivering the events, then changing **FreezeEvents** to FALSE will cause these events to be delivered.[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] 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.

Other conditions are described in the section "Events" on page 22.

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**

PowerNotify Property R/W *Added in Release 1.3*

Syntax **LONG PowerNotify;**

Remarks Contains the type power notification selection made by the Application.

The power notification values are:

Value	Meaning
OPOS_PN_DISABLED	The Control will not provide any power notifications to the application. No power notification **StatusUpdateEvent**s will be fired, and **PowerState** may not be set.
OPOS_PN_ENABLED	The Control will fire power notification **StatusUpdateEvent**s and update **PowerState**, beginning when **DeviceEnabled** is set to TRUE. The level of functionality depends upon **CapPowerReporting**.

PowerNotify may only be set while the device is disabled, that is, while **DeviceEnabled** is FALSE.

This property is initialized to OPOS_PN_DISABLED by the **Open** method. This value provides compatibility with earlier releases.

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	One of the following occurred: • The device is already enabled. • **PowerNotify** = OPOS_PN_ENABLED but **CapPowerReporting =** OPOS_PR_NONE.
Other Values	See **ResultCode**.

See Also "Device Power Reporting Model"; **CapPowerReporting** Property, **PowerState** Property

PowerState Property *Added in Release 1.3*

Syntax **LONG PowerState;**

Remarks Contains the current power condition, if it can be determined.

The power reporting values are:

Value	Meaning
OPOS_PS_UNKNOWN	Cannot determine the device's power state, for one of the following reasons: • **CapPowerReporting** = OPOS_PR_NONE. Device does not support power reporting. • **PowerNotify** = OPOS_PN_DISABLED. Power notifications are disabled. • **DeviceEnabled** = FALSE. Power state monitoring does not occur until the device is enabled.
OPOS_PS_ONLINE	The device is powered on and ready for use. Can be returned if **CapPowerReporting** = OPOS_PR_STANDARD or OPOS_PR_ADVANCED.
OPOS_PS_OFF	The device is off or detached from the terminal. Can only be returned if **CapPowerReporting** = OPOS_PR_ADVANCED.
OPOS_PS_OFFLINE	The device is powered on but is either not ready or not able to respond to requests. Can only be returned if **CapPowerReporting** = OPOS_PR_ADVANCED.
OPOS_PS_OFF_OFFLINE	The device is either off or offline. Can only be returned if **CapPowerReporting** = OPOS_PR_STANDARD.

This property is initialized to OPOS_PS_UNKNOWN by the **Open** method. When **PowerNotify** is set to enabled and **DeviceEnabled** is TRUE, then this property is updated as the Service Object detects power condition changes.

See Also “Device Power Reporting Model”; **CapPowerReporting** Property, **PowerNotify** Property

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	The current Service Object state does not allow this request. For example, if asynchronous output is in progress, certain methods may not be allowed.
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, and must match one of the values given in this document under the appropriate device class section.

When the **ResultCode** is set to any other value, this property may be set by the Service Object to any 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.

Note

A Service Object for a device class will operate with any Control Object for that class, as long as its major version number matches the Control Object’ s major version number. If they match, but the Service Object’ s minor version number is greater than the Control Object’ s minor version number, then the Service Object may support some methods or properties that cannot be accessed from the Control Object’ s release.

If the application requires such features, then it will need to be updated to use a later version of the Control Object.

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.
Other Values	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, then returns the appropriate status immediately.
If OPOS_FOREVER (-1), the method waits as long as needed until exclusive access is satisfied.

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, or an invalid *Timeout* parameter was specified.
OPOS_E_TIMEOUT	Another application has exclusive access to the device, and did not relinquish control before *Timeout* 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 enqueued – 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.
OPOS_E_CLAIMED	The device is claimed by another process.
OPOS_E_NOTCLAIMED	The device must be claimed before this method can be used.

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 enqueued – 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.
OPOS_E_CLAIMED	The device is claimed by another process.
OPOS_E_NOTCLAIMED	The device must be claimed before this method can be used.

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.
Other Values	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
Command	Command number. Specific values assigned by the Service Object.
pData	Pointer to additional numeric data. Specific values vary by *Command* and Service Object.
pString	Pointer to additional string data. Specific values vary by *Command* and Service Object. The format of this data depends upon the value of the **BinaryConversion** property. See page 37.

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.
Other Values	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 "APPENDIX B OPOS Registry Usage", page 683.)

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:

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

Note

The value of the **ResultCode** property after calling the **Open** method may not be the same as the **Open** method return value for the following two cases:

1. The Control was closed and the **Open** method failed: The **ResultCode** property will continue to return OPOS_E_CLOSED.
2. The Control was already opened: The **Open** method will return OPOS_E_ILLEGAL, but the **ResultCode** property may continue to return the value it held before the **Open** method.

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, and the device is an exclusive-use device, then the device is first disabled. (**Release** does not change the device enabled state of sharable devices.)

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 delivered. (When this property is subsequently changed back to TRUE, the event will be delivered 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
EventNumber	Event number. Specific values are assigned by the Service Object.
pData	Pointer to additional numeric data. Specific values vary by *EventNumber* and the Service Object.
pString	Pointer to additional string data. Specific values vary by *EventNumber* and the Service Object. The format of this data depends upon the value of the **BinaryConversion** property. See page 37.

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
ResultCode	Result code causing the error event. See **ResultCode** for values.
ResultCodeExtended	Extended result code causing the error event. See **ResultCodeExtended** for values.
ErrorLocus	Location of the error. See values below.
pErrorResponse	Pointer to the error event response. See values below.

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

Value	Meaning
OPOS_EL_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	Typically valid only when locus is OPOS_EL_OUTPUT. Retry the asynchronous output. The error state is exited. May be valid when locus is OPOS_EL_INPUT. 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 deliver additional **DataEvent**s as directed by the **DataEventEnabled** property. When all input has been delivered and the **DataEventEnabled** property is again set to TRUE, then another **ErrorEvent** is delivered 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 delivered 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** *Status***);**

The *Status* 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).

When a device is enabled, then the Control may fire initial **StatusUpdateEvent**s to inform the application of the device state. This behavior, however, is not required.

Release 1.3 and later – Power State Reporting

All device classes may fire **StatusUpdateEvent**s with at least the following *Status* parameter values, if **PowerNotify** = OPOS_PN_ENABLED:

Value	Meaning
OPOS_SUE_POWER_ONLINE	The device is powered on and ready for use. Can be returned if **CapPowerReporting** = OPOS_PR_STANDARD or OPOS_PR_ADVANCED.
OPOS_SUE_POWER_OFF	The device is off or detached from the terminal. Can only be returned if **CapPowerReporting** = OPOS_PR_ADVANCED.
OPOS_SUE_POWER_OFFLINE	The device is powered on but is either not ready or not able to respond to requests. Can only be returned if **CapPowerReporting** = OPOS_PR_ADVANCED.
OPOS_SUE_POWER_OFF_OFFLINE	The device is either off or offline. Can only be returned if **CapPowerReporting** = OPOS_PR_STANDARD.

The common property **PowerState** is also maintained at the current power state of the device.

See Also “Status, Result Code, and State Model”; “Device Power Reporting Model”; **CapPowerReporting** Property, **PowerNotify** Property

C H A P T E R   2

Bump Bar

Summary

Properties

Common		*Type*	*Access*	*Initialized After*
AutoDisable	1.3	Boolean	R/W	*Not Supported*
BinaryConversion	1.3	Long	R/W	Open
CapPowerReporting	1.3	Long	R	Open
CheckHealthText	1.3	String	R	Open
Claimed	1.3	Boolean	R	Open
DataCount	1.3	Long	R	Open
DataEventEnabled	1.3	Boolean	R/W	Open
DeviceEnabled	1.3	Boolean	R/W	Open; Claim
FreezeEvents	1.3	Boolean	R/W	Open
OutputID	1.3	Long	R	Open
PowerNotify	1.3	Long	R/W	Open
PowerState	1.3	Long	R	Open
ResultCode	1.3	Long	R	--
ResultCodeExtended	1.3	Long	R	Open
State	1.3	Long	R	--
ControlObjectDescription	1.3	String	R	--
ControlObjectVersion	1.3	Long	R	--
ServiceObjectDescription	1.3	String	R	Open
ServiceObjectVersion	1.3	Long	R	Open
DeviceDescription	1.3	String	R	Open
DeviceName	1.3	String	R	Open

Properties (continued)

Specific		*Type*	*Access*	*Initialized After*
AsyncMode	1.3	Boolean	R/W	Open, Claim, & Enable
Timeout	1.3	Long	R/W	Open
UnitsOnline	1.3	Long	R	Open, Claim, & Enable
CurrentUnitID	1.3	Long	R/W	Open, Claim, & Enable
CapTone	1.3	Boolean	R	Open, Claim, & Enable
AutoToneDuration	1.3	Long	R/W	Open, Claim, & Enable
AutoToneFrequency	1.3	Long	R/W	Open, Claim, & Enable
BumpBarDataCount	1.3	Long	R	Open, Claim, & Enable
Keys	1.3	Long	R	Open, Claim, & Enable
ErrorUnits	1.3	Long	R	Open
ErrorString	1.3	String	R	Open
EventUnitID	1.3	Long	R	Open, Claim
EventUnits	1.3	Long	R	Open, Claim
EventString	1.3	String	R	Open, Claim

Methods

Common		*Prerequisites*
Open	1.3	None
Close	1.3	Open
Claim	1.3	Open
Release	1.3	Open, Claim
CheckHealth	1.3	Open, Claim, & Enable
ClearInput	1.3	Open, Claim
ClearOutput	1.3	Open, Claim
DirectIO	1.3	Open
Specific		
BumpBarSound	1.3	Open, Claim, & Enable
SetKeyTranslation	1.3	Open, Claim, & Enable

Events

Name		*May Occur After*
DataEvent	1.3	Open, Claim, & Enable
DirectIOEvent	1.3	Open, Claim
ErrorEvent	1.3	Open, Claim, & Enable
OutputCompleteEvent	1.3	Open, Claim, & Enable
StatusUpdateEvent	1.3	Open, Claim, & Enable

General Information

The Bump Bar Control’ s OLE Programmatic ID is “OPOS.BumpBar”.

This device was added in OPOS Release 1.3.

Capabilities

The Bump Bar Control has the following minimal set of capabilities:

- Broadcast methods that can communicate with one, a range, or all bump bar units online.
- Supports bump bar input (keys 0-255).

The Bump Bar Control may also have the following additional capabilities:

- Supports bump bar enunciator output with frequency and duration.
- Supports tactile feedback via an automatic tone when a bump bar key is pressed.

Model

The general model of a bump bar is:

- The bump bar device class is a subsystem of bump bar units. The initial targeted environment is food service, to control the display of order preparation and fulfillment information. Bump bars typically are used in conjunction with remote order displays.

  The subsystem can support up to 32 bump bar units.

  One Application on one PC or POS Terminal will typically manage and control the entire subsystem of bump bars. If Applications on the same or other PCs and POS Terminals will need to access the subsystem, then this Application must act as a subsystem server and expose interfaces to other Applications.
- All specific methods are broadcast methods. This means that the method can apply to one unit, a selection of units or all online units. The *Units* parameter is a **LONG**, with each bit identifying an individual bump bar unit. (One or more of the constants BB_UID_1 through BB_UID_32 are bitwise ORed to form the bitmask.) The service object will attempt to satisfy the method for all unit(s) indicated in the *Units* parameter. If an error is received from one or more units, the **ErrorUnits** property is updated with the appropriate units in error. The **ErrorString** property is updated with a description of the error or errors received. The method will then return with the corresponding OPOS error. In the case where two or more units encounter different errors, the service object should determine the most severe OPOS error to return.
- The common methods **CheckHealth, ClearInput,** and **ClearOutput** are not broadcast methods and use the unit ID indicated in the **CurrentUnitID** property. (One of the constants BB_UID_1 through BB_UID_32 are selected.) See the description of these common methods to understand how the current unit ID property is used.
- When the current unit ID property is set by the application, all the corresponding properties are updated to reflect the settings for that unit.

  If the **CurrentUnitID** property is set to a unit ID that is not online, the dependent properties will contain non-initialized values.

  The **CurrentUnitID** uniquely represents a single bump bar unit. The definitions range from BB_UID_1 to BB_UID_32. These definitions are also used to create the bitwise parameter, *Units,* used in the broadcast methods. See the Examples section below for usage.

Input – Bump Bar

The Bump Bar Control follows the general "Input Model" for event-driven input with some differences:

- When input is received by the Control, it enqueues a **DataEvent**.
- This device does not support the **AutoDisable** property, so the control will not automatically disable itself when a **DataEvent** is enqueued.
- An enqueued **DataEvent** can be delivered to the application when the **DataEventEnabled** property is TRUE and other event delivery requirements are met. Just before delivering this event, the Control copies the data into properties, and disables further data events by setting the **DataEventEnabled** property to FALSE. This causes subsequent input data to be enqueued by 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.
- An **ErrorEvent** or events are enqueued if the Control encounters an error while gathering or processing input, and are delivered to the application when the **DataEventEnabled** property is TRUE and other event delivery requirements are met.
- The **BumpBarDataCount** property may be read to obtain the number of bump bar **DataEvent**s for a specific unit ID enqueued by the Control. The **DataCount** property can be read to obtain the total number of data events enqueued by the Control.
- Input enqueued by the Control may be deleted by calling the **ClearInput** method. See **ClearInput** method description for more details.

The Bump Bar Control must supply a method for translating its internal key scan codes into user-defined codes which are returned by the data event. Note that this translation *must* be end-user configurable. The default translated key value is the scan code value.

Output – Tone

The bump bar follows the general "Output Model", with some enhancements:

- The **BumpBarSound** method is performed either synchronously or asynchronously, depending on the value of the **AsyncMode** property. When **AsyncMode** is FALSE, then this method operates synchronously and returns its completion status to the application.
- When **AsyncMode** is TRUE, then this method operates 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 updates the **EventUnits** property and fires an **OutputCompleteEvent**. A parameter of this event contains the output ID of the completed request.

      Asynchronous methods will not return an error status due to a bump bar problem, such as communications failure. These errors will only be reported by an **ErrorEvent**. An error status is returned only if the bump bar 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 **EventUnits** property is set to the unit or units in error. The **EventString** property is also set.
      *Note: **ErrorEvent** updates **EventUnits** and **EventString**. If an error is reported by a broadcast method, then **ErrorUnits** and **ErrorString** are set instead.*

      The event handler may call synchronous bump bar methods (but not asynchronous methods), then can either retry the outstanding output or clear it.
    - ♦ The Control guarantees that asynchronous output is performed on a first-in first-out basis.
    - ♦ All output buffered by the Control may be deleted by setting the **CurrentUnitID** property and calling the **ClearOutput** method. **OutputCompleteEvent**s will not be fired for cleared output. This method also stops any output that may be in progress (when possible).

Example

Sounds one tone on unit ID 1 and unit ID 4. The frequency is set to 64 Hertz and will sound for 100 milliseconds.

```
BB.BumpBarSound( BB_UID_1 | BB_UID_4, 64, 100, 1, 0 )
```

Device Sharing

The bump bar 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 many bump bar specific properties.
- The application must claim and enable the device before calling methods that manipulate the device.
- When a **Claim** method is called again, settable device characteristics are restored to their condition at **Release**.
- See the "Summary" table for precise usage prerequisites.

Properties

AsyncMode Property R/W

Syntax **BOOL AsyncMode;**

Remarks If TRUE, then the **BumpBarSound** method will be performed asynchronously.
If FALSE, tones are generated 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.

See Also **BumpBarSound** Method; "Output Model"

AutoToneDuration Property R/W

Syntax **LONG AutoToneDuration;**

Remarks Sets the duration (in milliseconds) of the automatic tone for the bump bar unit specified by the **CurrentUnitID** property.

This property is initialized to the default value for each online bump bar unit 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 illegal value was specified. The **ErrorString** property is updated before return.

See Also **CurrentUnitID** Property

AutoToneFrequency Property R/W

Syntax **LONG AutoToneFrequency;**

Remarks Sets the frequency (in Hertz) of the automatic tone for the bump bar unit specified by the **CurrentUnitID** property.

This property is initialized to the default value for each online bump bar unit 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 illegal value was specified. The **ErrorString** property is updated before return.

See Also **CurrentUnitID** Property

BumpBarDataCount Property

Syntax **LONG BumpBarDataCount;**

Remarks Indicates the number of **DataEvent**s enqueued for the bump bar unit specified by the **CurrentUnitID** property.

The application may interrogate **BumpBarDataCount** to determine whether additional input is enqueued from a bump bar unit, but has not yet been delivered because of other application processing, freezing of events, or other causes.

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

See Also **CurrentUnitID** Property; **DataEvent** Event

CapTone Property

Syntax **BOOL CapTone;**

Remarks If TRUE, the bump bar unit specified by the **CurrentUnitID** property supports an enunciator; otherwise it is FALSE

This property is initialized when the device is first enabled following the **Open** method.

See Also **CurrentUnitID** Property

CurrentUnitID Property R/W

Syntax **LONG CurrentUnitID;**

Remarks Selects the current bump bar unit ID. Some properties and methods apply only to the selected bump bar unit ID as noted. Up to 32 units are allowed for one bump bar device. The unit ID definitions range from BB_UID_1 to BB_UID_32.

The following properties and methods apply only to the selected bump bar ID:

- Properties: **AutoToneDuration, AutoToneFrequency, BumpBarDataCount, CapTone, Keys.**

  Setting **CurrentUnitID** will update these properties to the current values for the specified unit.
- Methods: **CheckHealth, ClearInput, ClearOutput.**

This property is initialized to BB_UID_1 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 illegal unit ID was specified. The **ErrorString** property is updated before return.

DataCount Property (Common)

Syntax **LONG DataCount;**

Remarks Indicates the total number of **DataEvent**s enqueued at the control. All units online are included in this value. The number of enqueued events for a specific unit ID is stored in the **BumpBarDataCount** property.

The application may interrogate **DataCount** to determine whether additional input is enqueued from a device, but has not yet been delivered because of other application processing, freezing of events, or other causes.

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

See Also **BumpBarDataCount** Property; **DataEvent** Event; "Input Model"

ErrorString Property

Syntax **BSTR ErrorString;**

Remarks When an error occurs for any method that acts on a bitwise set of bump bar units, the **ErrorString** will contain a description of the error which occurred to the unit(s) specified by the **ErrorUnits** property.

If an error occurs during processing of an asynchronous request, the **ErrorEvent** updates the property **EventString** instead.

This property is initialized to an empty string by the **Open** method.

See Also **ErrorUnits** Property

ErrorUnits Property

Syntax **LONG ErrorUnits;**

Remarks When an error occurs for any method that acts on a bitwise set of bump bar units, the **ErrorUnits** will contain a bitwise mask of the unit(s) that encountered an error.

If an error occurs during processing of an asynchronous request, the **ErrorEvent** updates the property **EventUnits** instead.

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

See Also **ErrorString** Property

EventString Property

Syntax **BSTR EventString;**

Remarks When an **ErrorEvent** is delivered, this property is set to a description of the error which occurred to the unit(s) specified by the **EventUnits** property.

This property is initialized to an empty string by the **Open** method.

See Also **EventUnits** Property; **ErrorEvent**

EventUnitID Property

Syntax **LONG EventUnitID;**

Remarks Just before the Control delivers a **DataEvent** to the Application, it sets this property to the bump bar unit ID causing the event. The unit ID definitions range from BB_UID_1 to BB_UID_32.

See Also **DataEvent**

EventUnits Property

Syntax **LONG EventUnits;**

Remarks When an **OutputCompleteEvent,** output **ErrorEvent**, or **StatusUpdateEvent** is delivered, the **EventUnits** property will contain a bitwise mask of the unit(s).

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

See Also **OutputCompleteEvent**, **ErrorEvent, StatusUpdateEvent**

Keys Property

Syntax **LONG Keys;**

Remarks Indicates the number of keys on the bump bar unit specified by the **CurrentUnitID** property.

This property is initialized when the device is first enabled following the **Open** method.

See Also **CurrentUnitID** Property

Timeout Property R/W

Syntax **LONG Timeout;**

Remarks Timeout value in milliseconds used by the bump bar device to complete all output methods supported. If the device cannot successfully complete an output method within the timeout value, then the method returns a failure status if **AsyncMode** is FALSE, or enqueues an **ErrorEvent** if **AsyncMode** is TRUE.

This property is initialized to a Service Object dependent timeout 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 negative timeout value was specified. The **ErrorString** property is updated before return.

See Also **AsyncMode** Property; **ErrorString** Property ; **BumpBarSound** Method

UnitsOnline Property

Syntax **LONG UnitsOnline;**

Remarks Bitwise mask indicating the bump bar units online, where zero or more of the unit constants BB_UID_1 (bit 0 on) through BB_UID_32 (bit 31 on) are bitwise ORed. 32 units are supported.

This property is initialized when the device is first enabled following the **Open** method. This property is updated as changes are detected, such as before a **StatusUpdateEvent** is fired and during the **CheckHealth** method.

See Also **CheckHealth** Method; **StatusUpdateEvent** Event; "Model" Discussion Section

Methods

BumpBarSound Method

Syntax **LONG BumpBarSound (LONG** *Units***, LONG** *Frequency,* **LONG** *Duration,* **LONG** *NumberOfCycles,* **LONG** *InterSoundWait***);**

Parameter	**Description**
Units	Bitwise mask indicating which bump bar unit(s) to operate on.
Frequency	Tone frequency in Hertz.
Duration	Tone duration in milliseconds.
NumberOfCycles	If OPOS_FOREVER, then start bump bar sounding and, repeat continuously. Else perform the specified number of cycles.
InterSoundWait	When *NumberOfCycles* is not one, then pause for *InterSoundWait* milliseconds before repeating the tone cycle (before playing the tone again)

Remarks Sound the bump bar enunciator for the bump bar(s) specified by the *Units* parameter.

This method is performed synchronously if **AsyncMode** is FALSE, and asynchronously if **AsyncMode** is TRUE.

The duration of a tone cycle is:

Duration parameter +
InterSoundWait parameter (except on the last tone cycle)

After the bump bar has started an asynchronous sound, then the sound may be stopped by using the **ClearOutput** method. (When an *InterSoundWait* value of OPOS_FOREVER was used to start the sound, then the application must use **ClearOutput** to stop the continuous sounding of tones.)

If the **CapTone** property is FALSE for the selected unit(s), an OPOS_E_ILLEGAL is returned.

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: • *NumberOfCycles* is neither a positive, non-zero value nor OPOS_FOREVER. • *NumberOfCycles* is OPOS_FOREVER when **AsyncMode** is FALSE. • A negative *InterSoundWait* was specified. • *Units* is zero or a non-existent unit was specified. • A unit in *Units* does not support the **CapTone** capability. The **ErrorUnits** and **ErrorString** properties may be updated before return.
OPOS_E_FAILURE	An error occurred while communicating with one of the bump bar units specified by the *Units* parameter. The **ErrorUnits** and **ErrorString** properties are updated before return. (Can only be returned if **AsyncMode** is FALSE.)
Other Values	See **ResultCode**.

See Also **AsyncMode** Property; **ErrorString** Property; **ErrorString** Property ; **CapTone** Property; **ClearOutput** Method

CheckHealth Method (Common)

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.
OPOS_CH_INTERACTIVE	Perform an interactive test of the device. The Service Object will typically display a modal dialog box to present test options and results.

Remarks When OPOS_CH_INTERNAL or OPOS_CH_EXTERNAL level is requested, the method will check the health of the bump bar unit specified by the **CurrentUnitID** property. When the current unit ID property is set to a unit that is not currently online, the device will attempt to check the health of the bump bar unit and report a communication error if necessary. The OPOS_CH_INTERACTIVE health check operation is up to the service object designer.

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

The **UnitsOnline** property will be updated with any changes before returning to the application.

The **CheckHealth** method is always synchronous.

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

Value	Meaning
OPOS_SUCCESS	Indicates that the health check 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_FAILURE	An error occurred while communicating with the bump bar unit specified by the **CurrentUnitID** property.
Other Values	See **ResultCode**.

See Also **CurrentUnitID** Property; **UnitsOnline** Property

ClearInput Method (Common)

Syntax **LONG ClearInput ();**

Remarks Called to clear the device input that has been buffered for the unit specified by the **CurrentUnitID** property.

Any data events that are enqueued – usually waiting for **DataEventEnabled** to be set to TRUE and **FreezeEvents** to be set to FALSE – are also cleared.

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	The device is claimed by another process.
OPOS_E_NOTCLAIMED	The device must be claimed before this method can be used.

See Also **CurrentUnitID** Property; "Input Model"

ClearOutput Method (Common)

Syntax **LONG ClearOutput ();**

Remarks Called to clear the tone outputs that have been buffered for the unit specified by the **CurrentUnitID** property.

Any output complete and output error events that are enqueued – usually waiting for **DataEventEnabled** to be set to TRUE and **FreezeEvents** to be set to FALSE – are also cleared.

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	The device is claimed by another process.
OPOS_E_NOTCLAIMED	The device must be claimed before this method can be used.

See Also **CurrentUnitID** Property; "Output Model"

SetKeyTranslation Method

Syntax **LONG SetKeyTranslation (LONG** *Units*, **LONG** *ScanCode*, **LONG** *LogicalKey***);**

Parameter	Description
Units	Bitwise mask indicating which bump bar unit(s) to set key translation for.
ScanCode	The bump bar generated key scan code. Valid values 0-255.
LogicalKey	The translated logical key value. Valid values 0-255.

Remarks This method will assign a logical key value to a device-specific key scan code for the bump bar unit(s) specified by the *Units* parameter. The logical key value is used during translation during the **DataEvent**.

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: • *ScanCode* or *LogicalKey* are out of range. • *Units* is zero or a non-existent unit was specified. The **ErrorUnits** and **ErrorString** properties are updated before return.
Other Values	See **ResultCode**.

See Also **ErrorUnits** Properties; **ErrorString** Properties; **DataEvent** Event

Events

DataEvent Event

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

The *Status* parameter is divided into four bytes. Depending on the Event Type, located in the low word, the remaining 2 bytes will contain additional data. The diagram below indicates how the parameter *Status* is divided:

High Word		Low Word (Event Type)
High Byte	Low Byte	
Unused. Always zero.	LogicalKeyCode	BB_DE_KEY

Remarks Fired to present input data from a bump bar unit to the Application. The low word contains the Event Type. The high word contains additional data depending on the Event Type. When the Event Type is BB_DE_KEY, the low byte of the high word contains the LogicalKeyCode for the key pressed on the bump bar unit. The LogicalKeyCode value is device independent; it has been translated by the Service Object from its original hardware specific value. Valid ranges are 0-255.

The **EventUnitID** property is updated before delivering the event.

See Also "Input Model"; **EventUnitID** Property; **DataEventEnabled** Property; **FreezeEvents** Property

OutputCompleteEvent Event

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

The *OutputID* parameter indicates the ID number of the asynchronous output request that is complete. The **EventUnits** property is updated before delivering.

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

See Also **EventUnits** Property; "Output Model"

StatusUpdateEvent Event

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

The *Status* parameter reports a change in the power state of a bump bar unit.

Remarks Fired when the bump bar device detects a power state change.

Deviation from the standard **StatusUpdateEvent** (see page 68):

- Before delivering the event, the **EventUnits** property is set to the units for which the new power state applies.
- When the bump bar device is enabled, then the Control will fire a **StatusUpdateEvent** to specify the bitmask of online units.
- While the bump bar device is enabled, a **StatusUpdateEvent** is fired when the power state of one or more units change. If more than one unit changes state at the same time, the Service Object may choose to either fire multiple events or to coalesce the information into a minimal number of events applying to **EventUnits**.

See Also **EventUnits** Property

ErrorEvent Event

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

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

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

Value	Meaning
OPOS_EL_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 the value 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 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 deliver additional **DataEvent**s as directed by the **DataEventEnabled** property. When all input has been delivered and the **DataEventEnabled** property is again set to TRUE, then another **ErrorEvent** is delivered with locus OPOS_EL_INPUT. Default when locus is OPOS_EL_INPUT_DATA.

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

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

The **EventUnits** and **EventString** properties are updated before return.

See Also "Status, Result Code, and State Model"; **DataEventEnabled** Property; **EventUnits** Property; **EventString** Property

C H A P T E R 3

Cash Changer

Summary

Properties

Common		*Type*	*Access*	*Initialized After*
AutoDisable	1.2	Boolean	R/W	*Not Supported*
BinaryConversion	1.2	Long	R/W	Open
CapPowerReporting	1.3	Long	R	Open
CheckHealthText	1.2	String	R	Open
Claimed	1.2	Boolean	R	Open
DataCount	1.2	Long	R	*Not Supported*
DataEventEnabled	1.2	Boolean	R/W	*Not Supported*
DeviceEnabled	1.2	Boolean	R/W	Open & Claim
FreezeEvents	1.2	Boolean	R/W	Open
OutputID	1.2	Long	R	*Not Supported*
PowerNotify	1.3	Long	R/W	Open
PowerState	1.3	Long	R	Open
ResultCode	1.2	Long	R	--
ResultCodeExtended	1.2	Long	R	Open
State	1.2	Long	R	--
ControlObjectDescription	1.2	String	R	--
ControlObjectVersion	1.2	Long	R	--
ServiceObjectDescription	1.2	String	R	Open
ServiceObjectVersion	1.2	Long	R	Open
DeviceDescription	1.2	String	R	Open
DeviceName	1.2	String	R	Open

Specific		*Type*	*Access*	*Initialized After*
CapDiscrepancy	1.2	Boolean	R	Open
CapEmptySensor	1.2	Boolean	R	Open
CapFullSensor	1.2	Boolean	R	Open
CapNearEmptySensor	1.2	Boolean	R	Open
CapNearFullSensor	1.2	Boolean	R	Open
AsyncMode	1.2	Boolean	R/W	Open
AsyncResultCode	1.2	Long	R	Open, Claim, & Enable
AsyncResultCodeExtended	1.2	Long	R	Open, Claim, & Enable
CurrencyCashList	1.2	String	R	Open
CurrencyCode	1.2	String	R/W	Open
CurrencyCodeList	1.2	String	R	Open
CurrentExit	1.2	Long	R/W	Open
DeviceExits	1.2	Long	R	Open
ExitCashList	1.2	String	R	Open
DeviceStatus	1.2	Long	R	Open, Claim, & Enable
FullStatus	1.2	Long	R	Open, Claim, & Enable

Methods

Common		*May Use After*
Open	1.2	--
Close	1.2	Open
Claim	1.2	Open
Release	1.2	Open & Claim
CheckHealth	1.2	Open, Claim, & Enable
ClearInput	1.2	*Not Supported*
ClearOutput	1.2	*Not Supported*
DirectIO	1.2	Open
Specific		
DispenseCash	1.2	Open, Claim, & Enable
DispenseChange	1.2	Open, Claim, & Enable
ReadCashCounts	1.2	Open, Claim, & Enable

Events

Name		*May Occur After*
DataEvent	1.2	*Not Supported*
DirectIOEvent	1.2	Open , Claim
ErrorEvent	1.2	*Not Supported*
OutputCompleteEvent	1.2	*Not Supported*
StatusUpdateEvent	1.2	Open, Claim, & Enable

General Information

The Cash Changer Control’ s OLE programmatic ID is “OPOS.CashChanger”.

This device was added in OPOS Release 1.2.

Capabilities

The Cash Changer has the following capabilities:

- Supports reporting the cash units and corresponding unit counts available in the Cash Changer.
- Supports dispensing of a specified amount of cash from the device in either bills, coins, or both into a user-specified exit.
- Supports dispensing of a specified number of cash units from the device in either bills, coins, or both into a user-specified exit.
- Supports reporting of jam conditions within the device.
- Support for more than one currency.

The Cash Changer may also have the following additional capabilities:

- Reporting the fullness levels of the Cash Changer’ s cash units. Conditions which may be indicated include empty, near empty, full, and near full states.
- Reporting of a possible (or probable) cash count discrepancy in the data reported by the **ReadCashCounts** method.

Model

The general model of a Cash Changer is:

- The Cash Changer may support several cash types such as coins, bills, and combinations of coins and bills. The supported cash type for a particular currency is noted by the list of cash units in the **CurrencyCashList** property.
- A Cash Changer device may consist of any combination of features to aid in the cash processing functions such as a cash entry holding bin, a number of slots or bins which can hold the cash, and cash exits.
- The current model of the Cash Changer device class provides programmatic control *only for the dispensing of cash*. The accepting of cash by the device (for example, to replenish cash) cannot be controlled by the APIs provided in this model. The application can call the **ReadCashCounts** method to retrieve the current unit count for each cash unit, but cannot control when or how cash is added to the device.
- A Cash Changer device may have multiple exits. The number of exits is specified in the **DeviceExits** property. The application chooses a dispensing exit by setting the **CurrentExit** property. The cash units which may be dispensed to the current exit are indicated by the **ExitCashList** property. When the **CurrentExit** value is 1, the exit is considered the "primary exit" which is typically used during normal processing for dispensing cash to a customer following a retail transaction. When **CurrentExit** is a value greater than 1, the exit is considered an "auxiliary exit". An "auxiliary exit" typically is used for special purposes such as dispensing quantities or types of cash not targeted for the "primary exit".
- Dispensing of funds into the exit specified by the **CurrentExit** property is performed by calling either the **DispenseChange** or **DispenseCash** method. With the **DispenseChange** method, the application specifies a total amount to be dispensed, and it is the responsibility of the Cash Changer device or the Control to dispense the proper amount of cash from the various slots or bins. With the **DispenseCash** method, the application specifies a count of each cash unit to be dispensed.
- Cash dispensing can be performed either synchronously or asynchronously, depending on the value of the **AsyncMode** property.

  When **AsyncMode** is FALSE, then the cash dispensing methods are performed synchronously and the dispense method returns the completion status to the application.

When **AsyncMode** is TRUE and OPOS_SUCCESS is returned by either **DispenseChange** or **DispenseCash**, then the method is performed asynchronously and its completion is indicated by a **StatusUpdateEvent** event containing CHAN_STATUS_ASYNC as its *Status* value. The method's completion status is set in the **AsyncResultCode** and **AsyncResultCodeExtended** properties.

The values of the **AsyncResultCode** and **AsyncResultCodeExtended** properties are same as those returned in the **ResultCode** and **ResultCodeExtended** properties when synchronous dispensing is chosen.

Nesting of asynchronous Cash Changer operations is illegal; only one asynchronous method can be processed at a time.

ReadCashCounts may not be performed while an asynchronous method is being performed since doing so could likely report incorrect cash counts.

- The Cash Changer may support more than one currency. The **CurrencyCode** property may be set to the currency, selecting from a currency in the list **CurrencyCodeList**. The properties and methods **CurrencyCashList**, **ExitCashList, DispenseCash**, **DispenseChange**, and **ReadCashCounts** all act upon the current currency only.
- The cash slot (or cash bin) conditions are set in the **DeviceStatus** property to show empty and near empty status, and in the **FullStatus** property to show full and near full status. If there are one or more empty cash slots, then **DeviceStatus** property is CHAN_STATUS_EMPTY, and if there are one or more full cash slots, then **FullStatus** property is CHAN_STATUS_FULL.

Device Sharing

The Cash Changer 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 or collecting, or receiving status update events.
- See the "Summary" table for precise usage prerequisites.

Properties

AsyncMode Property R/W

Syntax **BOOL AsyncMode;**

Remarks If TRUE, then the **DispenseCash** and **DispenseChange** methods will be performed asynchronously.
If FALSE, these methods will be performed 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.

See Also **DispenseCash** Method; **DispenseChange** Method; **AsyncResultCode** Property; **AsyncResultCodeExtended** Property

AsyncResultCode Property

Syntax **LONG AsyncResultCode;**

Remarks When methods are asynchronously performed, they return their completion status to the application in this property. This property is set by the control before a **StatusUpdateEvent** event is delivered with a *Status* value of CHAN_STATUS_ASYNC.

The value of this property is same as the value that would have been in the **ResultCode** property had the method been performed synchronously.

See Also **DispenseCash** Method; **DispenseChange** Method; **AsyncMode** Property

AsyncResultCodeExtended Property

Syntax **LONG AsyncResultCodeExtended;**

Remarks When methods are asynchronously performed, they return their extended completion status to the application in this property. This property is set by the control before a **StatusUpdateEvent** event is delivered with a *Status* value of CHAN_STATUS_ASYNC. The value of this property is same as the value that would have been in the **ResultCodeExtended** property had the method been performed synchronously.

See Also **DispenseCash** Method; **DispenseChange** Method; **AsyncMode** Property

CapDiscrepancy Property

Syntax **BOOL CapDiscrepancy;**

Remarks If TRUE, the **ReadCashCounts** method can report effective *pDiscrepancy* value; otherwise it is FALSE.

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

See Also **ReadCashCounts** method

CapEmptySensor Property

Syntax **BOOL CapEmptySensor;**

Remarks If TRUE, the Cash Changer can report the condition that some cash slots are empty; otherwise it is FALSE.

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

See Also **DeviceStatus** Property; **StatusUpdateEvent**

CapFullSensor Property

Syntax **BOOL CapFullSensor;**

Remarks If TRUE, the Cash Changer can report the condition that some cash slots are full; otherwise it is FALSE.

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

See Also **FullStatus** Property; **StatusUpdateEvent**

CapNearEmptySensor Property

Syntax **BOOL CapNearEmptySensor;**

Remarks If TRUE, the Cash Changer can report the condition that some cash slots are nearly empty;
otherwise it is FALSE.

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

See Also **DeviceStatus** Property; **StatusUpdateEvent**

CapNearFullSensor Property

Syntax **BOOL CapNearFullSensor;**

Remarks If TRUE, the Cash Changer can report the condition that some cash slots are nearly full;
otherwise it is FALSE.

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

See Also **FullStatus** Property; **StatusUpdateEvent**

CurrencyCashList Property

Syntax **BSTR CurrencyCashList;**

Remarks A string value denoting the cash units supported in the Cash Changer for the currency represented by the **CurrencyCode** property.

The string consists of an ASCII numeric comma delimited values which denote the units of coins, then the ASCII semicolon character (“;”) followed by ASCII numeric comma delimited values for the bills that can be used with the Cash Changer. If a semicolon (“;”) is absent, then all units represent coins.

Below are sample **CurrencyCashList** values in Japan.

- “1,5,10,50,100,500” —
  1, 5, 10, 50, 100, 500 yen coin.
- “1,5,10,50,100,500;1000,5000,10000” —
  1, 5, 10, 50, 100, 500 yen coin and 1000, 5000, 10000 yen bill.
- “;1000,5000,10000” —
  1000, 5000, 10000 yen bill.

This property is initialized by the **Open** method, and is updated when **CurrencyCode** is set.

See Also **CurrencyCode** Property

CurrencyCode Property R/W

Syntax **BSTR CurrencyCode;**

Remarks Contains the active currency code to be used by Cash Changer operations.

This property is initialized to an appropriate value by the **Open** method. This value is guaranteed to be one of the set of currencies specified by the **CurrencyCodeList** property.

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	A value was specified that is not within **CurrencyCodeList**.

See Also **CurrencyCodeList** Property

CurrencyCodeList Property

Syntax **BSTR CurrencyCodeList;**

Remarks A string of currency code indicators.

This property is initialized by the **Open** method. The string consists of a list of ASCII three-character ISO 4217 currency codes separated by commas.

For example, if the string is “JPY,USD”, then the Cash Changer supports both Japanese and U.S. monetary units.

See Also **CurrencyCode** Property

CurrentExit Property R/W

Syntax **LONG CurrentExit;**

Remarks The current cash dispensing exit. The value 1 represents the primary exit (or *normal* exit), while values greater then 1 are considered auxiliary exits. Legal values range from 1 to **DeviceExits**.

This property is initialized to 1 by the **Open** method.

Examples below are samples of typical property value sets in Japan.
CurrencyCode is "JPY" and **CurrencyCodeList** is "JPY".

- Cash Changer supports coins; only one exit supported :
  CurrencyCashList = "1,5,10,50,100,500"
  DeviceExits = 1
  CurrentExit = 1 : **ExitCashList** = "1,5,10,50,100,500"
- Cash Changer supports both coins and bills; an auxiliary exit is used for larger quantities of bills :
  CurrencyCashList = "1,5,10,50,100,500;1000,5000,10000"
  DeviceExits = 2
  When **CurrentExit** = 1 : **ExitCashList** = "1,5,10,50,100,500;1000,5000"
  When **CurrentExit** = 2 : **ExitCashList** = ";1000,5000,10000"
- Cash Changer supports bills; an auxiliary exit is used for larger quantities of bills :
  CurrencyCashList = ";1000,5000,10000"
  DeviceExits = 2
  When **CurrentExit** = 1 : **ExitCashList** = ";1000,5000"
  When **CurrentExit** = 2 : **ExitCashList** = ";1000,5000,10000"

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 **CurrentExit** value was specified.

See Also **CurrencyCashList** Property; **DeviceExits** Property; **ExitCashList** Property

DeviceExits Property

Syntax **LONG DeviceExits;**

Remarks The number of exits for dispensing cash.

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

See Also **CurrentExit** Property

DeviceStatus Property

Syntax **LONG DeviceStatus;**

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

Value	**Meaning**
CHAN_STATUS_OK	The current condition of the Cash Changer is satisfactory.
CHAN_STATUS_EMPTY	Some cash slots are empty.
CHAN_STATUS_NEAREMPTY	Some cash slots are nearly empty.
CHAN_STATUS_JAM	A mechanical fault has occurred.

This property is initialized and kept current while the device is enabled. If more than one condition is present, then the order of precedence starting at the highest is fault, empty, and near empty.

ExitCashList Property

Syntax **BSTR ExitCashList;**

Remarks A string value denoting the cash units which may be dispensed to the exit which is denoted by **CurrentExit** property. The supported cash units are either the same as **CurrencyCashList**, or a subset of it. The string format is identical to that of **CurrencyCashList**.

This property is initialized by the **Open** method, and is updated when **CurrencyCode** or **CurrentExit** is set.

See Also **CurrencyCode** Property; **CurrencyCashList** Property; **CurrentExit** Property

FullStatus Property

Syntax **LONG FullStatus;**

Remarks Holds the current full status of the cash slots. It may be one of the following:

Value	**Meaning**
CHAN_STATUS_OK	All cash slots are neither nearly full nor full.
CHAN_STATUS_FULL	Some cash slots are full.
CHAN_STATUS_NEARFULL	Some cash slots are nearly full.

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

Methods

DispenseCash Method

Syntax **LONG DispenseCash (BSTR** *CashCounts***);**

The *CashCounts* parameter contains the dispensing cash units and counts, represented by the format of "cash unit:cash counts, ..;.., cash unit:cash counts". Units before ";" represent coins, and units after ";" represent bills. If ";" is absent, then all units represent coins.

Remarks Dispenses the cash from the Cash Changer into the exit specified by **CurrentExit**. The cash dispensed is specified by pairs of cash units and counts.

This Method is performed synchronously if **AsyncMode** is FALSE, and asynchronously if **AsyncMode** is TRUE.

CashCounts examples, using Japanese Yen as the currency, are below.

- "10:5,50:1,100:3,500:1"
  Dispense 5 ten yen coins, 1 fifty yen coins, 3 one hundred yen coins, 1 five hundred yen coins.
- "10:5,100:3;1000:10"
  Dispense 5 ten yen coins, 3 one hundred yen coins, and 10 one thousand yen bills.
- ";1000:10,10000:5"
  Dispense 10 one thousand yen bills and 5 ten thousand yen bills.

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

Value	**Meaning**
OPOS_SUCCESS	The specified cash was dispensed successfully, or **DispenseCash** method was performed asynchronously.
OPOS_E_BUSY	Cash cannot be dispensed because an asynchronous method is outstanding.
OPOS_E_ILLEGAL	A *CashCounts* parameter value was illegal for the current exit.

OPOS_E_EXTENDED	**ResultCodeExtended** = OPOS_ECHAN_OVERDISPENSE : The specified cash cannot be dispensed because of a cash shortage.
Other Values	See **ResultCode**.

See Also **AsyncMode** Property; **CurrentExit** Property

DispenseChange Method

Syntax **LONG DispenseChange (LONG** *Amount***);**

The *Amount* parameter contains the amount of change to be dispensed. It is up to the Cash Changer to determine what combination of bills and coins will satisfy the tender requirements from its available supply of cash.

Remarks Dispenses the specified amount of cash from the Cash Changer into the exit represented by **CurrentExit**.

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 specified change was dispensed successfully, or **DispenseChange** method was performed asynchronously.
OPOS_E_BUSY	The specified change cannot be dispensed because an asynchronous method is outstanding.
OPOS_E_ILLEGAL	A negative or zero *Amount* was specified, or It is impossible to dispense the *Amount* based on the values specified in **ExitCashList** for the current exit.
OPOS_E_EXTENDED	**ResultCodeExtended** = OPOS_ECHAN_OVERDISPENSE : The specified change cannot be dispensed because of a cash shortage.

Other Values See **ResultCode**.

See Also **AsyncMode** Property; **CurrentExit** Property

ReadCashCounts Method

Syntax **LONG ReadCashCounts (BSTR*** *pCashCounts*, **BOOL*** *pDiscrepancy*)**;**

Parameter	Description
pCashCounts	The cash count data is placed into the string pointed to by *pCashCounts*.
pDiscrepancy	If the integer pointed to by *pDiscrepancy* is set to TRUE by this method, then there is some cash which was not able to be included in the counts reported in *pCashCounts*; otherwise it is FALSE.

Remarks The format of the string pointed to by *pCashCounts* is the same as *CashCounts* in the **DispenseCash** method. Each unit in *pCashCounts* matches a unit in the **CurrencyCashList** property, and is in the same order.

For example if the currency is Japanese yen and string returned at the *pCashCounts* parameter is set to
1:80,5:77,10:0,50:54,100:0,500:87
as a result of calling the **ReadCashCounts** method, then there would be 80 one yen coins, 77 five yen coins, 54 fifty yen coins, and 87 five hundred yen coins in the Cash Changer.

If **CapDiscrepancy** property is FALSE, then *pDiscrepancy* is always FALSE.

Usually, the cash total calculated by *pCashCounts* parameter is equal to the cash total in a Cash Changer. But, there are some cases where a discrepancy may occur because of existing uncountable cash in a Cash Changer. An example would be when a cash slot is "overflowing" such that the device has lost its ability to accurately detect and monitor the cash.

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

Value	Meaning
OPOS_SUCCESS	This method was successful.
OPOS_E_BUSY	Cash units and counts cannot be read because an asynchronous method is outstanding.
Other Values	See **ResultCode**.

See Also **DispenseCash** Method; **CapDiscrepancy** Property; **CurrencyCashList** Property

Events

StatusUpdateEvent

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

The *Status* parameter contains the Cash Changer status condition:

Value	**Meaning**
CHAN_STATUS_EMPTY	Some cash slots are empty.
CHAN_STATUS_NEAREMPTY	Some cash slots are nearly empty.
CHAN_STATUS_EMPTYOK	No cash slots are either empty or nearly empty.
CHAN_STATUS_FULL	Some cash slots are full.
CHAN_STATUS_NEARFULL	Some cash slots are nearly full.
CHAN_STATUS_FULLOK	No cash slots are either full or nearly full.
CHAN_STATUS_JAM	A mechanical fault has occurred.
CHAN_STATUS_JAMOK	A mechanical fault has recovered.
CHAN_STATUS_ASYNC	Asynchronously performed method has completed.

Remarks Fired when the Cash Changer detects a status change.

For changes in the fullness levels, the Cash Changer is only able to fire **StatusUpdateEvent**s when the device has a sensor capable of detecting the full, near full, empty, and/or near empty states and the corresponding capability properties for these states are set.

Jam conditions may be reported whenever this condition occurs; likewise for asynchronous method completion.

The completion statuses of asynchronously performed methods are placed in the **AsyncResultCode** and **AsyncResultCodeExtended** properties.

C H A P T E R   4

Cash Drawer

Summary

Properties

Common		*Type*	*Access*	*Initialized After*
AutoDisable	1.2	Boolean	R/W	*Not Supported*
BinaryConversion	1.2	Long	R/W	Open
CapPowerReporting	1.3	Long	R	Open
CheckHealthText	1.0	String	R	Open
Claimed	1.0	Boolean	R	Open
DataCount	1.2	Long	R	*Not Supported*
DataEventEnabled	1.0	Boolean	R/W	*Not Supported*
DeviceEnabled	1.0	Boolean	R/W	Open
FreezeEvents	1.0	Boolean	R/W	Open
OutputID	1.0	Long	R	*Not Supported*
PowerNotify	1.3	Long	R/W	Open
PowerState	1.3	Long	R	Open
ResultCode	1.0	Long	R	--
ResultCodeExtended	1.0	Long	R	Open
State	1.0	Long	R	--
ControlObjectDescription	1.0	String	R	--
ControlObjectVersion	1.0	Long	R	--
ServiceObjectDescription	1.0	String	R	Open
ServiceObjectVersion	1.0	Long	R	Open
DeviceDescription	1.0	String	R	Open
DeviceName	1.0	String	R	Open

Specific		*Type*	*Access*	*Initialized After*
CapStatus	1.0	Boolean	R	Open
DrawerOpened	1.0	Boolean	R	Open & Enable

Methods

Common		*May Use After*
Open	1.0	--
Close	1.0	Open
Claim	1.0	Open
Release	1.0	Open & Claim
CheckHealth	1.0	Open & Enable; *Note*
ClearInput	1.0	*Not Supported*
ClearOutput	1.0	*Not Supported*
DirectIO	1.0	Open
Specific		
OpenDrawer	1.0	Open & Enable; *Note*
WaitForDrawerClose	1.0	Open & Enable; *Note*

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

Events

Name		*May Occur After*
DataEvent	1.0	*Not Supported*
DirectIOEvent	1.0	Open
ErrorEvent	1.0	*Not Supported*
OutputCompleteEvent	1.0	*Not Supported*
StatusUpdateEvent	1.0	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 **WaitForDrawerClose** 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.
Other Values	See **ResultCode**.

WaitForDrawerClose Method

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

Parameter	**Description**
BeepTimeout	Number of milliseconds to wait before starting an alert beeper.
BeepFrequency	Audio frequency of the alert beeper in hertz.
BeepDuration	Number of milliseconds that the beep tone will be sounded.
BeepDelay	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.
Other Values	See **ResultCode**.

Events

StatusUpdateEvent Event

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

The *Status* parameter contains the updated drawer status.

Release 1.0 - 1.2

If *Status* contains a non-zero value, then the drawer is open.
If *Status* contains a zero value, then the drawer is closed.

Release 1.3 and later

One of the following values may be returned:

Value	Meaning
CASH_SUE_DRAWERCLOSED (= 0)	The drawer is closed.
CASH_SUE_DRAWEROPEN (= 1)	The drawer is open.
*Power reporting **StatusUpdateEvent** values*	See **StatusUpdateEvent** description on page 68. (Can only be returned if the application sets **PowerNotify** to OPOS_PN_ENABLED.)

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.

C H A P T E R   5

CAT - Credit Authorization Terminal

Summary

Properties

Common		*Type*	*Access*	*Initialized After*
AutoDisable	1.4	Boolean	R/W	*Not Supported*
BinaryConversion	1.4	Long	R/W	Open
CapPowerReporting	1.4	Long	R	Open
CheckHealthText	1.4	String	R	Open
Claimed	1.4	Boolean	R	Open
DataCount	1.4	Long	R	*Not Supported*
DataEventEnabled	1.4	Boolean	R/W	*Not Supported*
DeviceEnabled	1.4	Boolean	R/W	Open & Claim
FreezeEvents	1.4	Boolean	R/W	Open
OutputID	1.4	Long	R	Open
PowerNotify	1.4	Long	R/W	Open
PowerState	1.4	Long	R	Open
ResultCode	1.4	Long	R	--
ResultCodeExtended	1.4	Long	R	Open
State	1.4	Long	R	--
ControlObjectDescription	1.4	String	R	--
ControlObjectVersion	1.4	Long	R	--
ServiceObjectDescription	1.4	String	R	Open
ServiceObjectVersion	1.4	Long	R	Open
DeviceDescription	1.4	String	R	Open
DeviceName	1.4	String	R	Open

Specific		*Type*	*Access*	*Initialized After*
AccountNumber	1.4	String	R	Open
AdditionalSecurityInformation	1.4	String	R/W	Open
ApprovalCode	1.4	String	R	Open
AsyncMode	1.4	Boolean	R/W	Open
CapAdditionalSecurityInformation	1.4	Boolean	R	Open
CapAuthorizeCompletion	1.4	Boolean	R	Open
CapAuthorizePreSales	1.4	Boolean	R	Open
CapAuthorizeRefund	1.4	Boolean	R	Open
CapAuthorizeVoid	1.4	Boolean	R	Open
CapAuthorizeVoidPreSales	1.4	Boolean	R	Open
CapCenterResultCode	1.4	Boolean	R	Open
CapCheckCard	1.4	Boolean	R	Open
CapDailyLog	1.4	Long	R	Open
CapInstallments	1.4	Boolean	R	Open
CapPaymentDetail	1.4	Boolean	R	Open
CapTaxOthers	1.4	Boolean	R	Open
CapTransactionNumber	1.4	Boolean	R	Open
CapTrainingMode	1.4	Boolean	R	Open
CardCompanyID	1.4	String	R	Open
CenterResultCode	1.4	String	R	Open
DailyLog	1.4	String	R	Open
PaymentCondition	1.4	Long	R	Open
PaymentDetail	1.4	String	R	Open
SequenceNumber	1.4	Long	R	Open
SlipNumber	1.4	String	R	Open
TrainingMode	1.4	Boolean	R/W	Open
TransactionNumber	1.4	Long	R	Open
TransactionType	1.4	Long	R	Open

Methods

Common		*May Use After*
Open	1.4	--
Close	1.4	Open
Claim	1.4	Open & Claim
Release	1.4	Open & Claim
CheckHealth	1.4	Open, Claim, & Enable
ClearInput	1.4	*Not Supported*
ClearOutput	1.4	Open & Claim
DirectIO	1.4	Open & Claim

Specific		
AccessDailyLog	1.4	Open, Claim, & Enable
AuthorizeCompletion	1.4	Open, Claim, & Enable
AuthorizePreSales	1.4	Open, Claim, & Enable
AuthorizeRefund	1.4	Open, Claim, & Enable
AuthorizeSales	1.4	Open, Claim, & Enable
AuthorizeVoid	1.4	Open, Claim, & Enable
AuthorizeVoidPreSales	1.4	Open, Claim, & Enable
CheckCard	1.4	Open, Claim, & Enable

Events

Name		*May Use After*
DataEvent	1.4	*Not Supported*
DirectIOEvent	1.4	Open & Claim
ErrorEvent	1.4	Open, Claim, & Enable
OutputCompleteEvent	1.4	Open, Claim, & Enable
StatusUpdateEvent	1.4	Open, Claim, & Enable

General Information

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

This device was added in OPOS Release 1.4.

Description of terms

- Authorization method

  Methods defined by this device class that have the *Authorize* prefix in their name. These methods require communication with an approval agency.

- Authorization operation

  The period from the invocation of an authorization method until the authorization is completed. This period differs depending upon whether operating in synchronous or asynchronous mode.

- Credit Authorization Terminal (CAT) Device

  A CAT device typically consists of a display, keyboard, magnetic stripe card reader, receipt printing device, and a communications device. CAT devices are predominantly used in Japan where they are required by law. Essentially a CAT device can be considered a device that shields the encryption, message formatting, and communication functions of an electronic funds transfer (EFT) operation from an application.

- Purchase

  The transaction that allows credit card payment at the POS. It is independent of payment methods (for example, lump-sum payment, payment in installments, revolving payment, etc.).

- Cancel Purchase

  The transaction to request voiding a purchase *on the date of purchase.*

- Refund Purchase

  The transaction to request voiding a purchase *after the date of purchase*. This differs from cancel purchase in that a cancel purchase operation can often be handled by updating the daily log at the CAT device, while the refund purchase operation typically requires interaction with the approval agency.

- Authorization Completion

  The state of a purchase when the response from the approval agency is "suspended". The purchase is later completed after a voice approval is received from the card company.

- Pre-Authorization

  The transaction to reserve an estimated amount in advance of the actual purchase with customer's credit card presentation and card entry at CAT.

- Cancel Pre-Authorization

  The transaction to request canceling pre-authorization.

- Card Check

  The transaction to perform a negative card file validation of the card presented by the customer. Typically negative card files contain card numbers that are known to fail approval. Therefore the Card Check operation removes then need for communication to the approval agency in some instances.

- Daily log

  The daily log of card transactions that have been approved by the card companies.

- Payment condition

  Condition of payment such as lump-sum payment, payment by bonus, payment in installments, revolving payment, and the combination of those payments. See the **PaymentCondition** and **PaymentDetail** properties for details.

- Approval agency

  The agency to decide whether or not to approve the purchase based on the card information, the amount of purchase, and payment type. The approval agency is generally the card company.

Capabilities

The CAT control is capable of the following general mode of operation:

- This standard defines the application interface with the CAT control and does not depend on the CAT device hardware implementation. Therefore, the hardware implementation of a CAT device may be as follows:
  - ♦ Separate type (POS interlock)

    The dedicated CAT device is externally connected to the POS (for instance, via an RS-232 connection).
  - ♦ Built-in type

    The hardware structure is the same as the separate type but is installed within the POS housing.
- The CAT device receives each authorization request containing a purchase amount and tax from CAT control.
- The CAT device generally requests the user to swipe a magnetic card when it receives an authorization request from CAT control.
- Once a magnetic card is swiped at the CAT device, the device sends the purchase amount and tax to the approval agency using the communications device.
- The CAT device returns the result from the approval agency to the CAT control. The returned data will be stored in the authorization properties by the CAT control for access by applications.

Model

The general models for the CAT control are shown below:

- The CAT control basically follows the output device model. However, multiple methods cannot be issued for asynchronous output; only 1 outstanding asynchronous request is allowed.
- The CAT control issues requests to the CAT device for different types of authorization by invoking the following methods.

Function	Method name	Corresponding **Cap** property
Purchase	**AuthorizeSales**	None
Cancel Purchase	**AuthorizeVoid**	**CapAuthorizeVoid**
Refund Purchase	**AuthorizeRefund**	**CapAuthorizeRefund**
Authorization Completion	**AuthorizeCompletion**	**CapAuthorizeCompletion**
Pre-Authorization	**AuthorizePreSales**	**CapAuthorizePreSales**
Cancel Pre-Authorization	**AuthorizeVoidPreSales**	**CapAuthorizeVoidPreSales**

- The CAT control issues requests to the CAT device for special processing local to the CAT device by invoking the following methods.

Function	Method name	Corresponding **Cap** property
Card Check	**CheckCard**	**CapCheckCard**
Daily log	**AccessDailyLog**	**CapDailyLog**

- The CAT control stores the authorization results in the following properties when an authorization operation sucessfully completes:

Description	Property Name	Corresponding **Cap** Property
Account number	**AccountNumber**	None
Additional information	**AdditionalSecurityInformation**	**CapAdditionalSecurityInformation**
Approval code	**ApprovalCode**	None
Card company ID	**CardCompanyID**	None
Code from the approval agency	**CenterResultCode**	**CapCenterResultCode**
Payment condition	**PaymentCondition**	None
Payment detail	**PaymentDetail**	**CapPaymentDetail**
Sequence number	**SequenceNumber**	None
Slip number	**SlipNumber**	None
Center transaction number	**TransactionNumber**	**CapTransactionNumber**
Transaction type	**TransactionType**	None

- The **AccessDailyLog** method sets the following property:

Description	Property Name	Corresponding **Cap** Property
Daily log	**DailyLog**	**CapDailyLog**

- Sequence numbers are used to validate that the properties set at completion of a method are indeed associated with the completed method. An incoming *SequenceNumber* argument for each method is compared with the resulting **SequenceNumber** property after the operation associated with the method has completed. If the numbers do not match, or if an application fails to identify the number, there is no guarantee that the values of the properties listed in the two tables correspond to the completed method.

- The **AsyncMode** property determines if methods are run synchronously or asynchronously.
  - ✓ When **AsyncMode** is FALSE, methods will be executed synchronously and their corresponding properties will contain data when the method returns.
  - ✓ When **AsyncMode** is TRUE, methods will return immediately to the application. When the operation associated with the method completes, each corresponding property will be updated by the CAT control prior to an **OutputCompleteEvent**. When **AsyncMode** is TRUE, methods cannot be issued immediately after issuing a prior method; only one outstanding asynchronous method is allowed at a time. However, **ClearOutput** is an exception because its purpose is to cancel an outstanding asynchronous method.
- The methods supported and their corresponding properties vary depending on the CAT control implementation. Applications should verify that particular **Cap** properties are supported before utilizing the capability dependent methods and properties.
- Results of synchronous calls to methods and writable properties will be stored in **ResultCode**. Results of asynchronous processing will be indicated by an **OutputCompleteEvent** or returned in the *Resultcode* argument of an **ErrorEvent**. If **ResultCode** or the *ResultCode* argument is OPOS_E_EXTENDED, detailed device specific information may be stored to **ResultCodeExtended** in synchronous mode and stored to **ErrorEvent** argument *ResultCodeExtended* in asynchronous mode. The result code from the approval agency will be stored in **CenterResultCode** in either mode.
- Training mode occurs continually when **TrainingMode** is TRUE. To discontinue training mode, set **TrainingMode** to FALSE.
- An outstanding asynchronous method can be canceled via the **ClearOutput** method.
- The Daily log can be collected by the **AccessDailyLog** method. Collection will be run either synchronously or asynchronously according to the value of **AsyncMode**.

- Following is the general usage sequence of the CAT control.

  Synchronous Mode:

  - **Open**
  - **Claim**
  - **DeviceEnabled**=TRUE
  - Definition of the argument *SequenceNumber*
  - **AuthorizeSales**()
  - Check **ResultCode**
  - Verify that the **SequenceNumber** property matches the value of the **AuthorizeSales()** *SequenceNumber* argument
  - Access the properties set by **AuthorizeSales()**
  - **DeviceEnabled**=FALSE
  - **Release**
  - **Close**

  Asynchronous Mode:

  - **Open**
  - **Claim**
  - **DeviceEnabled**=TRUE
  - **AsyncMode**=TRUE
  - Definition of the argument *SequenceNumber*
  - **AuthorizeSales**()
  - Check **ResultCode**
  – Wait for **OutputCompleteEvent**
  - Check the argument *ResultCode*
  - Verify that the **SequenceNumber** property matches the value of the **AuthorizeSales()** *SequenceNumber* argument
  - Access the properties set by **AuthorizeSales()**
  - **DeviceEnabled**=FALSE
  - **Release**
  - **Close**

Device sharing

The CAT 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

AccountNumber Property R

Syntax **BSTR AccountNumber;**

Remarks This property is initialized to NULL by the **Open** method and is updated when an authorization operation successfully completes.

AdditionalSecurityInformation Property R/W

Syntax **BSTR AdditionalSecurityInformation;**

Remarks An application can send data to the CAT device by setting this property before issuing an authorization method. Also, data obtained from the CAT device and not stored in any other property as the result of an authorization operation (for example, the account code for a loyalty program) can be provided to an application by storing it in this property. Since the data stored here is device specific, this should not be used for any development that requires portability. The format of this data depends on **BinaryConversion** property. See **BinaryConversion** property for more details.

See Also CapAdditionalSecurityInformation Property; BinaryConversion Property

ApprovalCode Property R

Syntax **BSTR ApprovalCode;**

Remarks This property is initialized to NULL by the **Open** method and is updated when an authorization operation successfully completes.

AsyncMode Property R/W

Syntax **BOOL AsyncMode;**

Remarks If TRUE, the authorization methods will run asynchronously.

If FALSE, the authorization methods will run 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	Property has been properly set up.

See Also Authorization Methods

CapAdditionalSecurityInformation Property

Syntax **BOOL CapAdditionalSecurityInformation;**

Remarks If TRUE, the **AdditionalSecurityInformation** property may be utilized; otherwise it is FALSE.

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

See Also **AdditionalSecurityInformation** Property

CapAuthorizeCompletion Property

Syntax **BOOL CapAuthorizeCompletion;**

Remarks If TRUE, the **AuthorizeCompletion** method has been implemented; otherwise it is FALSE.

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

See Also **AuthorizeCompletion** Method

CapAuthorizePreSales Property

Syntax **BOOL CapAuthorizePreSales;**

Remarks If TRUE, the **AuthorizePreSales** method has been implemented; otherwise it is FALSE.

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

See Also **AuthorizePreSales** Method

CapAuthorizeRefund Property

Syntax **BOOL CapAuthorizeRefund;**

Remarks If TRUE, the **AuthorizeRefund** method has been implemented; otherwise it is FALSE.

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

See Also **AuthorizeRefund** Method

CapAuthorizeVoid Property

Syntax **BOOL CapAuthorizeVoid;**

Remarks If TRUE, the **AuthorizeVoid** method has been implemented; otherwise it is FALSE.

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

See Also **AuthorizeVoid** Method

CapAuthorizeVoidPreSales Property

Syntax **BOOL CapAuthorizeVoidPreSales;**

Remarks If TRUE, the **AuthorizeVoidPreSales** method has been implemented; otherwise it is FALSE.

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

See Also **AuthorizeVoidPreSales** Method

CapCenterResultCode property

Syntax **BOOL CapCenterResultCode;**

Remarks If TRUE, the **CenterResultCode** property has been implemented; otherwise it is FALSE.

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

See Also **CenterResultCode** Property

CapCheckCard Property

Syntax **BOOL CapCheckCard;**

Remarks If TRUE, the **CheckCard** method has been implemented; otherwise it is FALSE.

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

See Also **CheckCard** Method

CapDailyLog Property

Syntax **LONG CapDailyLog;**

Remarks Shows the daily log ability of the device.

Value	Meaning
CAT_DL_NONE	The CAT device does not have the daily log functions.
CAT_DL_REPORTING	The CAT device only has an intermediate total function which reads the daily log but does not erase the log.
CAT_DL_SETTLEMENT	The CAT device only has the "final total" and "erase daily log" functions.
CAT_DL_REPORTING_SETTLEMENT	The CAT device has both the intermediate total function and the final total and erase daily log function.

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

See Also **DailyLog** Property; **AccessDailyLog** Method

CapInstallments Property

Syntax **BOOL CapInstallments;**

Remarks If TRUE, the item "Installments" which is stored in the **DailyLog** property as the result of **AccessDailyLog** will be provided; otherwise it is FALSE.

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

See Also **DailyLog** Property

CapPaymentDetail Property

Syntax **BOOL CapPaymentDetail;**

Remarks If TRUE, the **PaymentDetail** property has been implemented; otherwise it is FALSE.

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

See Also **PaymentDetail** Property

CapTaxOthers Property

Syntax **BOOL CapTaxOthers;**

Remarks If TRUE, the item “TaxOthers” which is stored in the **DailyLog** property as the result of **AccessDailyLog** will be provided; otherwise it is FALSE.

Note that this property is not related to the “TaxOthers” argument used with the authorization methods.

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

See Also **DailyLog** Property

CapTransactionNumber Property

Syntax **BOOL CapTransactionNumber;**

Remarks If TRUE, the **TransactionNumber** property has been implemented; otherwise it is FALSE.

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

See Also **TransactionNumber** Property

CapTrainingMode Property

Syntax **BOOL CapTrainingMode;**

Remarks If TRUE, the **TrainingMode** property has been implemented; otherwise it is FALSE.

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

See Also **TrainingMode** Property

CardCompanyID Property R

Syntax **BSTR CardCompanyID;**

Remarks This property is initialized to NULL by the **Open** method and is updated when an authorization operation successfully completes.

The length of the ID string varies depending upon the CAT device.

CenterResultCode Property R

Syntax **BSTR CenterResultCode;**

Remarks Contains the code from the approval agency. Check the approval agency for the actual codes to be stored.

This property is initialized to NULL by the **Open** method and is updated when an authorization operation successfully completes

DailyLog Property R

Syntax **BSTR DailyLog;**

Remarks Stores the result of the **AccessDailyLog** method. The data is delimited by CR(13)+LF(10) for each transaction and is stored in ASCII code. The detailed data of each transaction is comma separated [i.e. delimited by "," (44)].

The details of one transaction are shown as follows:

No.	Item	Property	Corresponding **Cap** Property
1	Card company ID	**CardCompanyID**	None
2	Transaction type	**TransactionType**	None
3	Transaction date Note 1)	None	None
4	Transaction number Note 3)	**TransactionNumber**	**CapTransactionNumber**
5	Payment condition	**PaymentCondition**	None
6	Slip number	**SlipNumber**	None
7	Approval code	**ApprovalCode**	None
8	Purchase date Note 5)	None	None
9	Account number	**AccountNumber**	None
10	Amount Note 4)	The argument *Amount* of the authorization method or the amount actually approved.	None
11	Tax/others Note 3)	The argument *TaxOthers* of the authorization method.	**CapTaxOthers**
12	Installments Note 3)	None	**CapInstallments**
13	Additional data Note 2)	**AdditionalSecurityInformation**	**CapAdditionalSecurityInformation**

Notes from the previous table:

1) Format

Item	Format
Transaction date	YYYYMMDDHHMMSS
Purchase date	MMDD

Some CAT devices may not support seconds by the internal clock. In that case, the seconds field of the transaction date is filled with “00”

2) Additional data

The area where the CAT device stores the vendor specific data. This enables an application to receive data other than that defined in this specification. The data stored here is vendor specific and should not be used for development which places an importance on portability.

3) If the corresponding **Cap** property is FALSE

Cap property is set to FALSE if the CAT device provides no corresponding data. In such instances, the item can’ t be displayed so the next comma delimiter immediately follows. For example, if “Amount” is 1234 yen and “Tax/others” is missing and “Installments” is 2, the description will be “1234,,2”. This makes the description independent of **Cap** property and makes the position of each data item consistent.

4) Amount

Amount always includes “Tax/others” even if item 11 is present.

5) Purchase date

The date manually entered for the purchase transaction after approval.

Example An example of daily log content is shown below.

Item	Description	Meaning
Card company ID	102	JCB
Transaction type	**CAT_TRANSACTION_SALES**	Purchase
Transaction date	19980116134530	1/16/1998 13:45:30
Transaction number	123456	123456
Payment condition	**CAT_PAYMENT_INSTALLMENT_1**	Installment 1
Slip number	12345	12345
Approval code	0123456	0123456
Purchase date	None	None
Account number	1234123412341234	1234-1234-1234-1234
Amount	12345	12345JPY
Tax/others	None	None
Number of payments	2	2
Additional data	12345678	Specific information

The actual data stored in **DailyLog** will be as follows.

```
102,10,19980116134530,123456,61,12345,0123456,,12341234123412
34,12345,,2,12345678[CR][LF]
```

See Also **CapDailyLog** Property; **AccessDailyLog** Method

PaymentCondition Property R

Syntax **LONG PaymentCondition;**

Remarks Holds the payment condition of the most recent successful authorization operation.

This property will be set to one of the following values. See **PaymentDetail** for the detailed payment string that correlates to the following **PaymentCondition** values.

Value	Meaning
CAT_PAYMENT_LUMP	Lump-sum
CAT_PAYMENT_BONUS_1	Bonus 1
CAT_PAYMENT_BONUS_2	Bonus 2
CAT_PAYMENT_BONUS_3	Bonus 3
CAT_PAYMENT_BONUS_4	Bonus 4
CAT_PAYMENT_BONUS_5	Bonus 5
CAT_PAYMENT_INSTALLMENT_1	Installment 1
CAT_PAYMENT_INSTALLMENT_2	Installment 2
CAT_PAYMENT_INSTALLMENT_3	Installment 3
CAT_PAYMENT_BONUS_COMBINATION_1	Bonus combination payments 1
CAT_PAYMENT_BONUS_COMBINATION_2	Bonus combination payments 2
CAT_PAYMENT_BONUS_COMBINATION_3	Bonus combination payments 3
CAT_PAYMENT_BONUS_COMBINATION_4	Bonus combination payments 4
CAT_PAYMENT_ REVOLVING	Revolving

See Also **PaymentDetail** Property

PaymentDetail Property R

Syntax **BSTR PaymentDetail;**

Remarks Contains payment condition details as the result of an authorization operation. Payment details vary depending on the value of **PaymentCondition**. The data will be stored as comma separated ASCII code. NULL means that no data is stored and represents a **BSTR** with zero length data.

PaymentCondition	**PaymentDetail**
CAT_PAYMENT_LUMP	NULL
CAT_PAYMENT_BONUS_1	NULL
CAT_PAYMENT_BONUS_2	Number of bonus payments
CAT_PAYMENT_BONUS_3	1st bonus month
CAT_PAYMENT_BONUS_4*	Number of bonus payments, 1st bonus month, 2nd bonus month, 3rd bonus month, 4th bonus month, 5th bonus month, 6th bonus month
CAT_PAYMENT_BONUS_5*	Number of bonus payments, 1st bonus month, 1st bonus amount, 2nd bonus month, 2nd bonus amount, 3rd bonus month, 3rd bonus amount, 4th bonus month, 4th bonus amount, 5th bonus month, 5th bonus amount, 6th bonus month, 6th bonus amount
CAT_PAYMENT_INSTALLMENT_1	1st billing month, Number of payments
CAT_PAYMENT_INSTALLMENT_2*	1st billing month, Number of payments, 1st amount, 2nd amount, 3rd amount, 4th amount, 5th amount, 6th amount
CAT_PAYMENT_INSTALLMENT_3	1st billing month, Number of payments, 1st amount
CAT_PAYMENT_BONUS_COMBINATION_1	1st billing month, Number of payments
CAT_PAYMENT_BONUS_COMBINATION_2	1st billing month, Number of payments, bonus amount
CAT_PAYMENT_BONUS_COMBINATION_3*	1st billing month, Number of payments, number of bonus payments, 1st bonus month, 2nd bonus month, 3rd bonus month, 4th bonus month, 5th bonus month, 6th bonus month
CAT_PAYMENT_BONUS_COMBINATION_4*	1st billing month, Number of payments, number of bonus payments, 1st bonus month, 1st bonus amount, 2nd bonus month, 2nd bonus amount, 3rd bonus month, 3rd bonus amount, 4th bonus month, 4th bonus amount, 5th bonus month, 5th bonus amount, 6th bonus month, 6th bonus amount

CAT_PAYMENT_REVOLVING	NULL

*Maximum 6 installments

The payment types and names vary depending on the CAT device. The following are the payment types and terms available for CAT devices. Note that there are some differences between OPOS terms and those used by the CAT devices. The goal of this table is to synchronize these terms.

General Payment Category	Entry item	**Payment Condition** Value	CAT Name	CAT (Old CAT)	G-CAT	JET-S	SG-CAT	Master-T
			Credit Card	Not specified	Not specified	JCB	VISA	MASTER
			OPOS Term	Card Company Terms				
Lump-sum	(None)	10	Lump-sum	Lump-sum	Lump-sum	Lump-sum	Lump-sum	Lump-sum
Bonus	(None)	21	Bonus 1	Bonus 1	Bonus 1	Bonus 1	Bonus 1	Bonus 1
	Number of bonus payments	22	Bonus 2	Bonus 2	Bonus 2	Bonus 2	Bonus 2	Bonus 2
	Bonus month(s)	23	Bonus 3	Bonus 3	Does not exist.	Does not exist.	Bonus 3	Bonus 3

Number of bonus payments Bonus month (1) Bonus month (2) Bonus month (3) Bonus month (4) Bonus month (5) Bonus month (6)	24	Bonus 4	Bonus 4	Bonus 3	Bonus 3	Bonus 4 (Up to two entries for bonus month)	Bonus 4

	Number of bonus payments Bonus month (1) Bonus amount (1) Bonus month (2) Bonus amount (2) Bonus month (3) Bonus amount (3) Bonus month (4) Bonus amount (4) Bonus month (5) Bonus amount (5) Bonus month (6) Bonus amount (6)	25	Bonus 5	Bonus 5	Does not exist.	Does not exist.	Does not exist.	Bonus 5
Installment	Payment start month Number of payments	61	Installment 1	Installment 1	Installm ent 1	Installment 1	Installment 1	Installment 1

	Payment start month Number of payments Installment amount (1) Installment amount (2) Installment amount (3) Installment amount (4) Installment amount (5) Installment amount (6)	62	Installment 2	Installment 2	Does not exist.	Does not exist.	Does not exist.	Does not exist.
	Payment start month Number of payments Initial amount	63	Installment 3	Installment 3	Installm ent 2	Installment 2	Does not exist.	Installment 2
Combination	Payment start month Number of payments	31	Bonus Combination 1	Bonus Combination 1	Bonus Combina tion 1	Bonus Combinati on 1	Bonus Combinati on 1	Bonus Combinati on 1
	Payment start month Number of payments Bonus amount	32	Bonus Combination 2	Bonus Combination 2	Does not exist.	Does not exist.	Bonus Combinati on 2	Bonus Combinati on 2

Payment start month Number of payments Number of bonus payments Bonus month (1) Bonus month (2) Bonus month (3) Bonus month (4) Bonus month (5) Bonus month (6)	33	Bonus Combination 3	Bonus Combination 3	Does not exist.	Does not exist.	Bonus Combination 3 (Up to two entries for bonus month)	Bonus Combination 3

	Payment start month Number of payments Number of bonus payments Bonus month (1) Bonus amount (1) Bonus month (2) Bonus amount (2) Bonus month (3) Bonus amount (3) Bonus month (4) Bonus amount (4) Bonus month (5) Bonus amount (5) Bonus month (6) Bonus amount (6)	34	Bonus Combination 4	Bonus Combination 4	Bonus Combina tion 2	Bonus Combinati on 2	Bonus Combinati on 4 (Up to two entries for bonus month and amount)	Bonus Combinati on 4
Revolving	(None)	80	Revolving	Revolving	Revolvin g	Revolving	Revolving	Revolving

See Also **CapPaymentDetail** Property

SequenceNumber Property R

Syntax **LONG SequenceNumber;**

Remarks Stores a "sequence number" as the result of each method call. This number needs to be checked by an application to see if it matches with the argument *SequenceNumber* of the originating method.

If the "sequence number" returned from the CAT device is not numeric, the CAT control set this property to zero (0).

This property is initialized to zero (0) by the **Open** method and is updated when an authorization operation successfully completes

SlipNumber Property R

Syntax **BSTR SlipNumber;**

Remarks Stores a "slip number" as the result of each authorization operation.

This property is initialized to NULL by the **Open** method and is updated when an authorization operation successfully completes

TrainingMode Property R/W

Syntax **BOOL TrainingMode;**

Remarks If TRUE, each operation will be run in training mode; otherwise each operation will be run in normal mode.

TrainingMode needs to be explicitly set to FALSE by an application to exit from training mode, because it will not automatically be set to FALSE after the completion of an operation.

This property will be 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.
OPOS_E_ILLEGAL	**CapTrainingMode** is FALSE.

TransactionNumber Property R

Syntax **BSTR TransactionNumber;**

Remarks Stores a “transaction number” as the result of each authorization operation.

This property is initialized to NULL by the **Open** method and is updated when an authorization operation successfully completes

TransactionType Property R

Syntax **LONG TransactionType;**

Remarks Stores a "transaction type" as the result of each authorization operation.

This property is initialized to zero (0) by the **Open** method and is updated when an authorization operation successfully completes.

This property will be set to one of the following values.

Value	Meaning
CAT_TRANSACTION_SALES	Sales
CAT_TRANSACTION_VOID	Cancellation
CAT_TRANSACTION_REFUND	Refund purchase
CAT_TRANSACTION_COMPLETION	Purchase after approval
CAT_TRANSACTION_PRESALES	Pre-authorization
CAT_TRANSACTION_CHECKCARD	Card Check
CAT_TRANSACTION_VOIDPRESALES	Cancel pre-authorization approval

Methods

AccessDailyLog Method

Syntax **LONG AccessDailyLog (LONG** *SequenceNumber***, LONG** *Type*, **LONG** *Timeout***);**

Parameter	**Description**
SequenceNumber	The sequence number to get daily log.
Type	Specify whether the daily log is intermediate total or final total and erase.
Timeout	The maximum waiting time (in milliseconds) until the response is received from the CAT device. OPOS_FOREVER(-1), 0, and positive values can be specified.

Remarks Gets daily log from CAT.

Daily log will be retrieved and stored in **DailyLog** as specified by *SequenceNumber.*

When *Timeout* is OPOS_FOREVER(-1), timeout never occurs and the device waits until it receives response from the CAT.

Application must specify one of the following values for *Type* for daily log type (either intermediate total or adjustment). Legal values depend upon the **CapDailyLog** value.

Value	**Meaning**
CAT_DL_REPORTING	Intermediate total.
CAT_DL_SETTLEMENT	Final total and erase.

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

Value	**Meaning**
OPOS_SUCCESS	Acquisition processing was successful.
OPOS_E_ILLEGAL	Invalid or unsupported *Type* or *Timeout* parameter was specified, or **CapDailyLog** is FALSE.

OPOS_E_TIMEOUT	No response was received from CAT during the specified *Timeout* time in milliseconds.
OPOS_E_EXTENDED	The detail code has been stored in **ResultCodeExtended**.
OPOS_E_BUSY	The CAT device cannot accept any commands now.
Other Values	See **ResultCode**.

See Also **CapDailyLog** Property, **DailyLog** Property

AuthorizeCompletion Method

Syntax **LONG AuthorizeCompletion (LONG** *SequenceNumber*, **CURRENCY** *Amount*, **CURRENCY** *TaxOthers*, **LONG** *Timeout*);

Parameter	Description
SequenceNumber	Sequence number for approval
Amount	Purchase amount for approval
TaxOthers	Tax and other amounts for approval
Timeout	The maximum waiting time (in milliseconds) until the response is received from the CAT device. OPOS_FOREVER(-1), 0 and positive values can be specified.

Remarks Purchase after approval is intended.

Sales after approval for *Amount* and *TaxOthers* is intended as the approval specified by *SequenceNumber.*

When *Timeout* is OPOS_FOREVER(-1), timeout never occurs and the device waits until it receives response from the CAT.

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

Value	Meaning
OPOS_SUCCESS	Approval processing was successful.
OPOS_E_ILLEGAL	Invalid *Timeout* parameter was specified, or **CapAuthorizeCompletion** is FALSE**.**
OPOS_E_TIMEOUT	No response was received from CAT during the specified *Timeout* time in milliseconds.
OPOS_E_EXTENDED	The detail code has been stored in **ResultCodeExtended**.
OPOS_E_BUSY	The CAT device cannot accept any commands now.
Other Values	See **ResultCode.**

.

See Also **CapAuthorizeCompletion** Property

AuthorizePreSales Method

Syntax **LONG AuthorizePreSales (LONG** *SequenceNumber*, **CURRENCY** *Amount*, **CURRENCY** *TaxOthers*, **LONG** *Timeout*)**;**

Parameter	Description
SequenceNumber	Sequence number for approval
Amount	Purchase amount for approval
TaxOthers	Tax and other amounts for approval
Timeout	The maximum waiting time (in milliseconds) until the response is received from the CAT device. OPOS_FOREVER(-1), 0 and positive values can be specified.

Remarks Makes a pre-authorization.

Pre-authorization for *Amount* and *TaxOthers* is made as the approval specified by *SequenceNumber.*

When *Timeout* is OPOS_FOREVER(-1), timeout never occurs and the device waits until it receives response from the CAT.

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

Value	Meaning
OPOS_SUCCESS	Approval processing was successful.
OPOS_E_ILLEGAL	Invalid *Timeout* parameter was specified, or **CapAuthorizePreSales** is FALSE.
OPOS_E_TIMEOUT	No response was received from CAT during the specified *Timeout* time in milliseconds.
OPOS_E_EXTENDED	The detail code has been stored in **ResultCodeExtended**.
OPOS_E_BUSY	The CAT device cannot accept any commands now.
Other Values	See **ResultCode.**

See Also **CapAuthorizePreSales** Property

AuthorizeRefund Method

Syntax **LONG AuthorizeRefund (LONG** *SequenceNumber*, **CURRENCY** *Amount*, **CURRENCY** *TaxOthers*, **LONG** *Timeout***);**

Parameter	Description
SequenceNumber	Sequence number for approval
Amount	Purchase amount for approval
TaxOthers	Tax and other amounts for approval
Timeout	The maximum waiting time (in milliseconds) until the response is received from the CAT device. OPOS_FOREVER(-1), 0 and positive values can be specified.

Remarks Refund purchase approval is intended.

Refund purchase approval for *Amount* and *TaxOthers* is intended as the approval specified by *SequenceNumber.*

When *Timeout* is OPOS_FOREVER(-1), timeout never occurs and the device waits until it receives response from the CAT.

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

Value	Meaning
OPOS_SUCCESS	Approval processing was successful.
OPOS_E_ILLEGAL	Invalid *Timeout* parameter was specified, or **CapAuthorizeRefund** is FALSE.
OPOS_E_TIMEOUT	No response was received from CAT during the specified *Timeout* time in milliseconds.
OPOS_E_EXTENDED	The detail code has been stored in **ResultCodeExtended**.
OPOS_E_BUSY	The CAT device cannot accept any commands now.
Other Values	See **ResultCode.**

See Also **CapAuthorizeRefund** Property

AuthorizeSales Method

Syntax **LONG AuthorizeSales (LONG** *SequenceNumber*, **CURRENCY** *Amount*, **CURRENCY** *TaxOthers*, **LONG** *Timeout*);

Parameter	**Description**
SequenceNumber	Sequence number for approval
Amount	Purchase amount for approval
TaxOthers	Tax and other amounts for approval
Timeout	The maximum waiting time (in milliseconds) until the response is received from the CAT device. OPOS_FOREVER(-1), 0 and positive values can be specified.

Remarks Normal purchase approval is intended.

Normal purchase approval for *Amount* and *TaxOthers* is intended as the approval specified by *SequenceNumber.*

When *Timeout* is OPOS_FOREVER(-1), timeout never occurs and the device waits until it receives response from the CAT.

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

Value	**Meaning**
OPOS_SUCCESS	Approval processing was successful.
OPOS_E_ILLEGAL	Invalid *Timeout* parameter was specified.
OPOS_E_TIMEOUT	No response was received from CAT during the specified *Timeout* time in milliseconds.
OPOS_E_EXTENDED	The detail code has been stored in **ResultCodeExtended**.
OPOS_E_BUSY	The CAT device cannot accept any commands now.
Other Values	See **ResultCode.**

AuthorizeVoid Method

Syntax **LONG AuthorizeVoid (LONG** *SequenceNumber*, **CURRENCY** *Amount*, **CURRENCY** *TaxOthers*, **LONG** *Timeout*);

Parameter	Description
SequenceNumber	Sequence number for approval
Amount	Purchase amount for approval
TaxOthers	Tax and other amounts for approval
Timeout	The maximum waiting time (in milliseconds) until the response is received from the CAT device. OPOS_FOREVER(-1), 0 and positive values can be specified.

Remarks Purchase cancellation approval is intended.

Cancellation approval for *Amount* and *TaxOthers* is intended as the approval specified by *SequenceNumber.*

When *Timeout* is OPOS_FOREVER(-1), timeout never occurs and the device waits until it receives response from the CAT.

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

Value	Meaning
OPOS_SUCCESS	Approval processing was successful.
OPOS_E_ILLEGAL	Invalid *Timeout* parameter was specified, or **CapAuthorizeVoid** is FALSE.
OPOS_E_TIMEOUT	No response was received from CAT during the specified *Timeout* time in milliseconds.
OPOS_E_EXTENDED	The detail code has been stored in **ResultCodeExtended**.
OPOS_E_BUSY	The CAT device cannot accept any commands now.
Other Values	See **ResultCode.**

See Also **CapAuthorizeVoid** Property

AuthorizeVoidPreSales Method

Syntax **LONG AuthorizeVoidPreSales (LONG** *SequenceNumber*, **CURRENCY** *Amount*, **CURRENCY** *TaxOthers*, **LONG** *Timeout*);

Parameter	Description
SequenceNumber	Sequence number for approval
Amount	Purchase amount for approval
TaxOthers	Tax and other amounts for approval
Timeout	The maximum waiting time (in milliseconds) until the response is received from the CAT device. OPOS_FOREVER(-1), 0 and positive values can be specified.

Remarks Pre-authorization cancellation approval is intended.

Pre-authorization cancellation approval for *Amount* and *TaxOthers* is intended as the approval specified by *SequenceNumber.*

When *Timeout* is OPOS_FOREVER(-1), timeout never occurs and the device waits until it receives response from the CAT.

Normal cancellation could be used for CAT control and CAT devices which have not implemented the pre-authorization approval cancellation. Refer to the documentation supplied with CAT device and / or CAT control.

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

Value	Meaning
OPOS_SUCCESS	Approval processing was successful.
OPOS_E_ILLEGAL	Invalid *Timeout* parameter was specified, or **CapAuthorizeVoidPreSales** is FALSE.
OPOS_E_TIMEOUT	No response was received from CAT during the specified *Timeout* time in milliseconds.
OPOS_E_EXTENDED	The detail code has been stored in **ResultCodeExtended**.
OPOS_E_BUSY	The CAT device cannot accept any commands now.
Other Values	See **ResultCode.**

See Also **CapAuthorizeVoidPreSales** Property

CheckCard Method

Syntax **LONG CheckCard (LONG** *SequenceNumber*, **LONG** *Timeout***);**

Parameter	**Description**
SequenceNumber	Sequence number for approval
Timeout	The maximum waiting time (in milliseconds) until the response is received from the CAT device. OPOS_FOREVER(-1), 0 and positive values can be specified.

Remarks Card Check is intended.

Card Check will be made as specified by *SequenceNumber.*

When *Timeout* is OPOS_FOREVER(-1), timeout never occurs and the device waits until it receives response from the CAT.

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

Value	**Meaning**
OPOS_SUCCESS	Approval processing was successful.
OPOS_E_ILLEGAL	Invalid *Timeout* parameter was specified, or **CapCheckCard** is FALSE.
OPOS_E_TIMEOUT	No response was received from CAT during the specified *Timeout* time in milliseconds.
OPOS_E_EXTENDED	The detail code has been stored in **ResultCodeExtended**.
OPOS_E_BUSY	The CAT device cannot accept any commands now.
Other Values	See **ResultCode.**

See Also **CapCheckCard** Property

Events

ErrorEvent Event

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

Parameter	Description
ResultCode	The code which caused the error event. Remarks **ResultCode** for the value.
ResultCodeExtended	The extended code which caused the error event. Remarks the value below for the value.
ErrorLocus	OPOS_EL_OUTPUT is specified. An error occurred during asynchronous action.
pErrorResponse	Pointer to the error event response. Remarks the value below.

If **ResultCode** is OPOS_E_EXTENDED, **ResultCodeExtended** will be set to one of the following values.

Value	Meaning
OPOS_ECAT_CENTERERROR	An error was returned from the approval agency. The detail error code is defined in **CenterResultCode.**
OPOS_ECAT_COMMANDERR	The command sent to CAT is wrong. This error is never returned so long as CAT control is working correctly.
OPOS_ECAT_RESET	CAT was stopped during processing by CAT reset key (stop key) and so on.
OPOS_ECAT_COMMUNICATIONERROR	Communication error has occurred between the approval agency and CAT.

OPOS_ECAT_DAILYLOGOVERFLOW
Daily log was too big to be stored. Keeping daily log has been stopped and the value of **DailyLog** property is uncertain.

The content of the position specified by *pErrorResponse* will be preset to the default value of OPOS_ER_RETRY. An application sets one of the following values.

Value	Meaning
OPOS_ER_RETRY	Retries the asynchronous processing. The error state is exited.
OPOS_ER_CLEAR	Clear the asynchronous processing. The error state is exited.

Remarks Fired when an error is detected while processing an asynchronous authorize group method or the **AccessDailyLog** method. The control' s **State** transitions into the error state.

See Also Status, Result Code, and State Model"

CHAPTER 6

Coin Dispenser

Summary

Properties

Common		*Type*	*Access*	*Initialized After*
AutoDisable	1.2	Boolean	R/W	*Not Supported*
BinaryConversion	1.2	Long	R/W	Open
CapPowerReporting	1.3	Long	R	Open
CheckHealthText	1.0	String	R	Open
Claimed	1.0	Boolean	R	Open
DataCount	1.2	Long	R	*Not Supported*
DataEventEnabled	1.0	Boolean	R/W	*Not Supported*
DeviceEnabled	1.0	Boolean	R/W	Open & Claim
FreezeEvents	1.0	Boolean	R/W	Open
OutputID	1.0	Long	R	*Not Supported*
PowerNotify	1.3	Long	R/W	Open
PowerState	1.3	Long	R	Open
ResultCode	1.0	Long	R	--
ResultCodeExtended	1.0	Long	R	Open
State	1.0	Long	R	--
ControlObjectDescription	1.0	String	R	--
ControlObjectVersion	1.0	Long	R	--
ServiceObjectDescription	1.0	String	R	Open
ServiceObjectVersion	1.0	Long	R	Open
DeviceDescription	1.0	String	R	Open
DeviceName	1.0	String	R	Open

Specific		*Type*	*Access*	*Initialized After*
CapEmptySensor	1.0	Boolean	R	Open
CapJamSensor	1.0	Boolean	R	Open
CapNearEmptySensor	1.0	Boolean	R	Open
DispenserStatus	1.0	Long	R	Open, Claim, & Enable

Methods

Common		*May Use After*
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	*Not Supported*
ClearOutput	1.0	*Not Supported*
DirectIO	1.0	Open

Specific		
DispenseChange	1.0	Open, Claim, & Enable

Events

Name		*May Occur After*
DataEvent	1.0	*Not Supported*
DirectIOEvent	1.0	Open, Claim
ErrorEvent	1.0	*Not Supported*
OutputCompleteEvent	1.0	*Not Supported*
StatusUpdateEvent	1.0	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 *Amount* parameter value of zero was specified, or the *Amount* parameter contained a negative value or a value greater than the device can dispense.
Other Values	See **ResultCode**.

Events

StatusUpdateEvent

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

The *Status* 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.
Power reporting ***StatusUpdateEvent*** *values*	See **StatusUpdateEvent** description on page 68.

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 7

Fiscal Printer

Summary

Properties

Common		*Type*	*Access*	*Initialized After*
AutoDisable	1.3	Boolean	R/W	*Not Supported*
BinaryConversion	1.3	Long	R/W	Open
CheckHealthText	1.3	String	R	Open
Claimed	1.3	Boolean	R	Open
DataCount	1.3	Long	R	*Not Supported*
DataEventEnabled	1.3	Boolean	R/W	*Not Supported*
DeviceEnabled	1.3	Boolean	R/W	Open & Claim
FreezeEvents	1.3	Boolean	R/W	Open
OutputID	1.3	Long	R	Open
PowerState	1.3	Long	R	Open
PowerNotify	1.3	Long	R/W	Open
ResultCode	1.3	Long	R	--
ResultCodeExtended	1.3	Long	R	Open
State	1.3	Long	R	--
ControlObjectDescription	1.3	String	R	--
ControlObjectVersion	1.3	Long	R	--
ServiceObjectDescription	1.3	String	R	Open
ServiceObjectVersion	1.3	Long	R	Open
DeviceDescription	1.3	String	R	Open
DeviceName	1.3	String	R	Open

Specific		*Type*	*Access*	*Initialized After*
CapAdditionalLines	1.3	Boolean	R	Open
CapAmountAdjustment	1.3	Boolean	R	Open
CapAmountNotPaid	1.3	Boolean	R	Open
CapCheckTotal	1.3	Boolean	R	Open
CapCoverSensor (2)	1.3	Boolean	R	Open
CapDoubleWidth	1.3	Boolean	R	Open
CapDuplicateReceipt	1.3	Boolean	R	Open
CapFixedOutput	1.3	Boolean	R	Open
CapHasVatTable	1.3	Boolean	R	Open
CapIndependentHeader	1.3	Boolean	R	Open
CapItemList	1.3	Boolean	R	Open
CapJrnEmptySensor (2)	1.3	Boolean	R	Open
CapJrnNearEndSensor (2)	1.3	Boolean	R	Open
CapJrnPresent (2)	1.3	Boolean	R	Open
CapNonFiscalMode	1.3	Boolean	R	Open
CapOrderAdjustmentFirst	1.3	Boolean	R	Open
CapPercentAdjustment	1.3	Boolean	R	Open
CapPositiveAdjustment	1.3	Boolean	R	Open
CapPowerLossReport	1.3	Boolean	R	Open
CapPredefinedPayment Lines	1.3	Boolean	R	Open
CapReceiptNotPaid	1.3	Boolean	R	Open
CapRecEmptySensor (2)	1.3	Boolean	R	Open
CapRecNearEndSensor (2)	1.3	Boolean	R	Open
CapRecPresent (2)	1.3	Boolean	R	Open
CapRemainingFiscal Memory	1.3	Boolean	R	Open
CapReservedWord	1.3	Boolean	R	Open
CapSetHeader	1.3	Boolean	R	Open
CapSetPOSID	1.3	Boolean	R	Open
CapSetStoreFiscalID	1.3	Boolean	R	Open
CapSetTrailer	1.3	Boolean	R	Open
CapSetVatTable	1.3	Boolean	R	Open

Specific (continued)		*Type*	*Access*	*Initialized After*
CapSlpEmptySensor (2)	1.3	Boolean	R	Open
CapSlpFiscalDocument	1.3	Boolean	R	Open
CapSlpFullSlip (2)	1.3	Boolean	R	Open
CapSlpNearEndSensor (2)	1.3	Boolean	R	Open
CapSlpPresent (2)	1.3	Boolean	R	Open
CapSlpValidation	1.3	Boolean	R	Open
CapSubAmountAdjustment	1.3	Boolean	R	Open
CapSubPercentAdjustment	1.3	Boolean	R	Open
CapSubtotal	1.3	Boolean	R	Open
CapTrainingMode	1.3	Boolean	R	Open
CapValidateJournal	1.3	Boolean	R	Open
CapXReport	1.3	Boolean	R	Open
AmountDecimalPlaces	1.3	Long	R	Open, Claim, & Enable
AsyncMode	1.3	Boolean	R/W	Open
CheckTotal	1.3	Boolean	R/W	Open
CountryCode	1.3	Long	R	Open, Claim, & Enable
CoverOpen (2)	1.3	Boolean	R	Open, Claim, & Enable
DayOpened	1.3	Boolean	R	Open, Claim, & Enable
DescriptionLength	1.3	Long	R	Open
DuplicateReceipt	1.3	Boolean	R/W	Open
ErrorLevel	1.3	Long	R	Open
ErrorOutID	1.3	Long	R	Open, Claim & Enable
ErrorState	1.3	Long	R	Open
ErrorStation	1.3	Long	R	Open
ErrorString	1.3	String	R	Open
FlagWhenIdle	1.3	Boolean	R/W	Open
JrnEmpty (2)	1.3	Boolean	R	Open, Claim, & Enable
JrnNearEnd (2)	1.3	Boolean	R	Open, Claim, & Enable
MessageLength	1.3	Long	R	Open
NumHeaderLines	1.3	Long	R	Open
NumTrailerLines	1.3	Long	R	Open
NumVatRates	1.3	Long	R	Open

PredefinedPaymentLines	1.3	String	R	Open
PrinterState	1.3	Long	R	Open, Claim, & Enable
QuantityDecimalPlaces	1.3	Long	R	Open, Claim, & Enable
QuantityLength	1.3	Long	R	Open, Claim, & Enable
RecEmpty (2)	1.3	Boolean	R	Open, Claim, & Enable
RecNearEnd (2)	1.3	Boolean	R	Open, Claim, & Enable
RemainingFiscalMemory	1.3	Long	R	Open, Claim, & Enable
ReservedWord (1)	1.3	String	R	Open
SlpEmpty (2)	1.3	Boolean	R	Open, Claim, & Enable
SlpNearEnd (2)	1.3	Boolean	R	Open, Claim, & Enable
SlipSelection	1.3	Long	R/W	Open, Claim, & Enable
TrainingModeActive	1.3	Boolean	R	Open, Claim, & Enable

Methods

Common		*May Use After*
Open	1.3	--
Close	1.3	Open
Claim	1.3	Open
Release	1.3	Open & Claim
CheckHealth	1.3	Open, Claim, & Enable
ClearInput	1.3	*Not Supported*
ClearOutput	1.3	Open & Claim
DirectIO	1.3	Open
Specific - Presetting Fiscal		
SetDate	1.3	Open, Claim, & Enable
SetHeaderLine	1.3	Open, Claim, & Enable
SetPOSID (1)	1.3	Open, Claim, & Enable
SetStoreFiscalID	1.3	Open, Claim, & Enable
SetTrailerLine	1.3	Open, Claim, & Enable
SetVatTable	1.3	Open, Claim, & Enable
SetVatValue	1.3	Open, Claim, & Enable
Specific - Fiscal Receipt		
BeginFiscalReceipt	1.3	Open, Claim, & Enable
EndFiscalReceipt	1.3	Open, Claim, & Enable
PrintDuplicateReceipt	1.3	Open, Claim, & Enable
PrintRecItem	1.3	Open, Claim, & Enable
PrintRecItemAdjustment	1.3	Open, Claim, & Enable
PrintRecMessage	1.3	Open, Claim, & Enable
PrintRecNotPaid	1.3	Open, Claim, & Enable
PrintRecRefund	1.3	Open, Claim, & Enable
PrintRecSubtotal	1.3	Open, Claim, & Enable
PrintRecSubtotalAdjustment	1.3	Open, Claim, & Enable
PrintRecTotal	1.3	Open, Claim, & Enable
PrintRecVoid	1.3	Open, Claim, & Enable
PrintRecVoidItem	1.3	Open, Claim, & Enable

Specific (continued)		*May Use After*
Specific - Fiscal Document		
BeginFiscalDocument	1.3	Open, Claim, & Enable
EndFiscalDocument	1.3	Open, Claim, & Enable
PrintFiscalDocumentLine	1.3	Open, Claim, & Enable
Specific - Item Lists		
BeginItemList (1)	1.3	Open, Claim, & Enable
EndItemList (1)	1.3	Open, Claim, & Enable
VerifyItem (1)	1.3	Open, Claim, & Enable
Specific - Fiscal Reports		
PrintPeriodicTotalsReport	1.3	Open, Claim, & Enable
PrintPowerLossReport	1.3	Open, Claim, & Enable
PrintReport	1.3	Open, Claim, & Enable
PrintXReport	1.3	Open, Claim, & Enable
PrintZReport	1.3	Open, Claim, & Enable
Specific - Slip Insertion		
BeginInsertion (2)	1.3	Open, Claim, & Enable
BeginRemoval (2)	1.3	Open, Claim, & Enable
EndInsertion (2)	1.3	Open, Claim, & Enable
EndRemoval (2)	1.3	Open, Claim, & Enable
Specific - Non-Fiscal		
BeginFixedOutput (1)	1.3	Open, Claim, & Enable
BeginNonFiscal	1.3	Open, Claim, & Enable
BeginTraining	1.3	Open, Claim, & Enable
EndFixedOutput (1)	1.3	Open, Claim, & Enable
EndNonFiscal	1.3	Open, Claim, & Enable
EndTraining	1.3	Open, Claim, & Enable
PrintFixedOutput (1)	1.3	Open, Claim, & Enable
PrintNormal	1.3	Open, Claim, & Enable

Specific (continued)		*May Use After*
Specific - Data Requests		
GetData	1.3	Open, Claim, & Enable
GetDate	1.3	Open, Claim, & Enable
GetTotalizer	1.3	Open, Claim, & Enable
GetVatEntry (1)	1.3	Open, Claim, & Enable
Specific - Error Correction		
ClearError	1.3	Open, Claim, & Enable
ResetPrinter	1.3	Open, Claim, & Enable

Events

Name		*May Occur After*
DataEvent	1.3	*Not Supported*
DirectIOEvent	1.3	Open
ErrorEvent	1.3	Open, Claim, & Enable
OutputCompleteEvent	1.3	Open, Claim, & Enable
StatusUpdateEvent	1.3	Open, Claim, & Enable

All methods and properties marked with (1) are specific to at least one particular country and are not required by the fiscal legislation of all countries.

Properties and methods marked with (2) are adapted from the POS Printer device.

General Information

The Fiscal Printer Control’ s OLE programmatic ID is “OPOS.FiscalPrinter”.

This device was added in OPOS Release 1.3.

The fiscal 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.

Since fiscal rules differ between countries, this interface tries to generalize the common requirements at the maximum extent specifications. This interface is based upon the fiscal requirements of the following countries, but it may fit the needs of other countries as well:

- Brazil
- Greece
- Hungary
- Italy
- Poland
- Turkey

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

- **Journal** Used to log transaction information. Must be kept by the store for audit.
- **Receipt** Used to print transaction information. It is mandatory to give a printed fiscal receipt to the customer. 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.

Configuration and initialization of the fiscal memory of the printer are not covered in this specification. These low level operations must be performed by technical assistance personnel.

Device Sharing

The Fiscal 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.

General requirements

Fiscal printers do not simply print text as standard printers do, they are used to monitor and memorize all fiscal information about a sale transaction. A fiscal printer has to accumulate totals, discounts, number of canceled receipts, taxes, etc. In order to do this, it is not sufficient to send unformatted strings of text to the printer; there is a need to separate each individual field in a receipt line item, thus differentiating between descriptions, prices and discounts. Moreover, it is necessary to define different printing commands for each different sale functionality (such as refund, item or void).

Fiscal rules are different among countries. This interface tries to generalize these requirements by summarizing the common requirements. Fiscal law requires that:

- Fiscal receipts must be printed and given to the customer.
- Fiscal printers must be equipped with memory to store daily totals. Each receipt line item must increment totals registers and, in most countries (Greece, Poland, Brazil, Hungary and Turkey) tax registers as well.
- Discounts, canceled items and canceled receipts must increment their associated registers on the printer.
- Fiscal printer must include a clock to store date and time information relative to each single receipt.
- Each fiscal receipt line item is printed both on the receipt and on the journal. (Italy, Greece, Poland)
- After a power failure (or a turn off) the fiscal printer must be in the same state as it was before this event occurred. This implies that care must be taken in managing the fiscal printer status and that power failure events must be managed by the application. In some countries a power failure must be logged and a report must be printed.

Printer Modes

According to fiscal rules, it is possible for a fiscal printer to also offer functionality beyond the required fiscal printing mode. These additional modes are optional and may or may not be present on any particular fiscal printer.

There are three possible printer modes:

- **Fiscal:** This is the only required mode for a fiscal printer. In this mode the application has access to all the methods needed to manage a sale transaction and to print a fiscal receipt. It is assumed that any lines printed to the receipt station while in fiscal mode are also printed on the journal station.
- **Training:** In this mode the printer is used for training purposes (such as cashier training). In this mode the printer will accept fiscal commands but the printer will indicate on each receipt or document that the transaction is not an actual fiscal transaction. The printer will not update any of its internal fiscal registers while in training mode. Such printed receipts are usually marked as "training" receipts by fiscal printers.
  The **CapTrainingMode** property will be set to **TRUE** if the printer supports training mode, **FALSE** otherwise.
- **Non-Fiscal:** In this mode the printer can be used to print simple text on the receipt station (echoed on the journal station) or the slip station. The printer will print some additional lines along with the application requested output to indicate that this output is not of a fiscal nature. Such printed receipts are usually marked as "non-fiscal" receipts by fiscal printers.
  The **CapNonFiscalMode** property will be set to **TRUE** if the printer supports non-fiscal printing, **FALSE** otherwise.

Model

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

- Most methods are always performed synchronously. Synchronous methods will fail if asynchronous output is outstanding.
- The following methods are performed either synchronously or asynchronously, depending on the value of the **AsyncMode** property:

  PrintFiscalDocumentLine
  PrintFixedOutput
  PrintNormal
  PrintRecItem
  PrintRecItemAdjustment
  PrintRecMessage
  PrintRecNotPaid
  PrintRecRefund
  PrintRecSubtotal
  PrintRecSubtotalAdjustment
  PrintRecTotal
  PrintRecVoid
  PrintRecVoidItem

  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 enqueued and delivered. The **ErrorStation** property is set to the station or stations that were printing when the error occurred. The **ErrorLevel**, **ErrorString** and **ErrorState** and **ErrorOutID** 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.
- ♦ The Control 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. **OutputCompleteEvent**s 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.

The printer error reporting model is as follows:

- Most of the fiscal printer error conditions are reported by setting the **ResultCode** to OPOS_E_EXTENDED and then setting **ResultCodeExtended** to one of the following error conditions:

  OPOS_EFPTR_COVER_OPEN
  The printer cover is open.

  OPOS_EFPTR_JRN_EMPTY
  The journal station has run out of paper.

  OPOS_EFPTR_REC_EMPTY
  The receipt station has run out of paper.

  OPOS_EFPTR_SLP_EMPTY
  The slip station has run out of paper.

  OPOS_EFPTR_MISSING_DEVICES:
  Some of the other devices which according to the local fiscal legislation are to be connected are missing. In some countries in order to use a fiscal printer a full set of peripheral devices are to be connected to the POS (such as cash drawer and customer display). In case one of these devices is not present sales are not allowed.

  OPOS_EFPTR_WRONG_STATE
  The requested method could not be executed in the printer' s current state.

OPOS_EFPTR_TECHNICAL_ASSISTANCE
: The printer has encountered a severe error condition. Calling for printer technical assistance is required.

OPOS_EFPTR_CLOCK_ERROR
: The printer’ s internal clock has failed.

OPOS_EFPTR_FISCAL_MEMORY_FULL
: The printer’ s fiscal memory has been exhausted.

OPOS_EFPTR_FISCAL_MEMORY_DISCONNECTED
: The printer’ s fiscal memory has been disconnected.

OPOS_EFPTR_FISCAL_TOTALS_ERROR
: The Grand Total in working memory does not match the one in the EPROM.

OPOS_EFPTR_BAD_ITEM_QUANTITY
: The Quantity parameter is invalid.

OPOS_EFPTR_BAD_ITEM_AMOUNT
: The Amount parameter is invalid.

OPOS_EFPTR_BAD_ITEM_DESCRIPTION
: The Description parameter is either too long, contains illegal characters or contains a reserved word.

OPOS_EFPTR_RECEIPT_TOTAL_OVERFLOW
: The receipt total has overflowed.

OPOS_EFPTR_BAD_VAT
: The Vat parameter is invalid.

OPOS_EFPTR_BAD_PRICE
: The Price parameter is invalid.

OPOS_EFPTR_BAD_DATE
: The date parameter is invalid.

OPOS_EFPTR_NEGATIVE_TOTAL
: The printer’ s computed total or subtotal is less than zero.

OPOS_EFPTR_WORD_NOT_ALLOWED
: The description contains the reserved word.

- 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.

Printer States

As previously described, a fiscal printer is characterized by different printing modes. Moreover, the set of commands that can be executed at a particular moment depends upon the current state of the printer.

The current state of the fiscal printer is kept in the **PrinterState** property.

The fiscal printer has the following states:

- **Monitor:**
  This is a neutral state. From this state it is possible to move to most of the other printer states. After a successful call to the **Claim** method and successful setting of the **DeviceEnabled** property to TRUE the printer should be in this state unless there is a printer error.
- **Fiscal Receipt:**
  The printer is processing a fiscal receipt. All **PrintRec…** methods are available for use while in this state. This state is entered from the **Monitor** state using the **BeginFiscalReceipt** method.
- **Fiscal Receipt Total:**
  The printer has already accepted at least one payment method, but the receipt' s total amount has not yet been tendered. This state is entered from the **Fiscal Receipt** state by use of the **PrintRecTotal** method. The printer remains in this state while the total remains unpaid. This state can left by using the **PrintRecTotal**, **PrintRecNotPaid** or **PrintRecVoid** methods.
- **Fiscal Receipt Ending:**
  The printer has completed the receipt up to the **Total** line. In this state it may be possible to print general messages using the **PrintRecMessage** method if it is supported by the printer. This state is entered from the **Fiscal Receipt** state via the **PrintRecVoid** method or from the **Fiscal Receipt Total** state using either the **PrintRecTotal**, **PrintRecNotPaid** or **PrintRecVoid** methods. This state is exited using the **EndFiscalReceipt** method at which time the printer returns to the **Monitor** state.
- **Fiscal Document:**
  The printer is processing a fiscal document. The printer will accept the **PrintFiscalDocumentLine** method while in this state. This state is entered from the **Monitor** state using the **BeginFiscalDocument** method. This state is exited using the **EndFiscalDocument** method at which time the printer returns to the **Monitor** state.

- **Monitor** and **TrainingModeActive** = TRUE**:**
  The printer is being used for training purposes. All fiscal receipt and document commands are available. This state is entered from the **Monitor** state using the **BeginTraining** method. This state is exited using the **EndTraining** method at which time the printer returns to the **Monitor** state.
- **Fiscal Receipt** and **TrainingModeActive** = TRUE**:**
  The printer is being used for training purposes and a receipt is currently opened. To each line of the receipt special text will be added in order to differentiate it from a fiscal receipt.
- **Fiscal Total** and **TrainingModeActive** = TRUE**:**
  The printer is in training mode and receipt total is being handled.
- **Fiscal ReceiptEnding** and **TrainingModeActive** = TRUE**:**
  The printer is being used for training is in the receipt ending phase.
- **NonFiscal:**
  The printer is printing non-fiscal output on either the receipt (echoed on the journal) or the slip. In this state the printer will accept the **PrintNormal** method. The printer prints a message that indicates that this is non-fiscal output with all application text. This state is entered from the **Monitor** state using the **BeginNonFiscal** method. This state is exited using the **EndNonFiscal** method at which time the printer returns to the **Monitor** state.
- **Fixed:**
  The printer is being used to print fixed, non-fiscal output to one of the printer’ s stations. In this state the printer will accept the **PrintFixedOutput** method. This state is entered from the **Monitor** state using the **BeginFixedOutput** method. This state is exited using the **EndFixedOutput** method at which time the printer returns to the **Monitor** state.
- **ItemList:**
  The printer is currently printing a line item report. In this state the printer will accept the **VerifyItem** method. This state is entered from the **Monitor** state using the **BeginItemList** method. This state is exited using the **EndItemList** method at which time the printer returns to the **Monitor** state.
- **Report:**
  The printer is currently printing one of the supported types of reports. This state is entered from the **Monitor** state using one of the **PrintReport**, **PrintPeriodicTotalsReport**, **PrintPowerLossReport**, **PrintXReport** or **PrintZReport** methods. When the report print completes, the printer automatically returns to **Monitor** state.
- **FiscalSystemBlocked:**
  The printer is no longer operational due to one of the following reasons:

- The printer has been disconnected or has lost power.
- The printer' s fiscal memory has been exhausted.
- The printer' s internal data has become inconsistent.

In this state the printer will only accept methods to print reports and retrieve data. The printer cannot exit this state without the assistance of a technician.

When the application sets the property **DeviceEnabled** to TRUE it also monitors its current state. In a standard situation, the **PrinterState** property is set to FPTR_PS_MONITOR after a successfully setting **DeviceEnabled** to TRUE. This indicates that there was no interrupted operation remaining in the printer.

If the printer is not in the FPTR_PS_MONITOR state, the state reflects the printer's interrupted operation and the **PowerState** property is set to OPOS_PS_OFF. In this situation it is necessary to force the printer to a normal state by calling the **ResetPrinter** method.

This means that a power failure occurred or the last application which accessed the device left it in a not clear state.

Notice that even in this case the **ResultCode** property will be set to OPOS_SUCCESS after setting **DeviceEnabled** to TRUE. It is required that the application check the **PowerState** property and checks for a received **StatusUpdateEvent** with the value OPOS_SUE_POWER_OFF in the Data argument after successfully setting the **DeviceEnabled** property.

Document Printing

Using a fiscal printer' s slip station it may be possible (depending upon the printer' s capabilities and on special fiscal rules) to print the following kinds of documents:

- **Fiscal Documents:**
  In order to print fiscal documents an amount value must be sent to the printer and recorded by it. The **CapSlpFiscalDocument** property will be set to **TRUE** if the printer supports printing fiscal documents, and **FALSE** otherwise. If fiscal documents are supported they may be either full length (if **CapSlpFullSlip** is **TRUE**) or validation (if **CapSlpValidation** is **TRUE**). The actual selection is made using the **SlipSelection** property but only one totalizer is assigned to all the fiscal documents.
- **Non-Fiscal Full Length Documents:**
  Full length slip documents may be printed if **CapSlpFullSlip** is **TRUE** and **SlipSelection** is set to FPTR_SS_FULL_LENGTH.
- **Non-Fiscal Validation Documents:**
  Validation documents may be printed if **CapSlpValidation** is **TRUE** and **SlipSelection** is set to FPTR_SS_VALIDATION.
- **Fixed Text Documents:**
  Fixed text documents may be printed if **CapFixedOutput** is **TRUE**. If fixed text documents are supported they may be either full length (if **CapSlpFullSlip** is **TRUE**) or validation (if **CapSlpValidation** is **TRUE**). The actual selection is made using the **SlipSelection** property.

Ordering of Fiscal Receipt Print Requests

A fiscal receipt is started using the **BeginFiscalReceipt** method. If the **CapIndependentHeader** property is true, then it is up to the application to decide if the fiscal receipt header lines are to be printed at this time or not. Otherwise header lines are printed immediately prior to the first line item inside a fiscal receipt. Printing the header lines at this time will decrease the amount of time required to process the first fiscal receipt print method, but it may result in more receipt voids as well. The **BeginFiscalReceipt** method may only be called if the printer is currently in the Monitor state and this call will change the printer' s current state to Fiscal Receipt.

Before selling the first line item it is possible to exit from the fiscal receipt state by calling the **EndFiscalReceipt** method. If header lines have already been printed, this method will cause also receipt voiding.

Once the first line item has been printed and the printer remains in the Fiscal Receipt state, the following fiscal print methods are available:

PrintRecItem
PrintRecItemAdjustment
PrintRecNotPaid
PrintRecRefund
PrintRecSubtotal
PrintRecSubtotalAdjustment
PrintRecTotal
PrintRecVoid
PrintRecVoidItem

The **PrintRecItem**, **PrintRecItemAdjustment**, **PrintRecRefund**, **PrintRecSubtotal**, **PrintRecSubtotalAdjustment** and **PrintRecVoidItem** will leave the printer in the Fiscal Receipt state. The **PrintRecNotPaid** (only available if the **CapReceiptNotPaid** property is TRUE) and **PrintRecTotal** methods will change the printer's state to either Fiscal Receipt Total or Fiscal Receipt Ending, depending upon whether the entire receipt total has been met. The **PrintRecVoid** method will change the printer's state to Fiscal Receipt Ending.

While in the Fiscal Receipt Total state the following fiscal print methods are available:

PrintRecNotPaid
PrintRecTotal
PrintRecVoid

The **PrintRecNotPaid** (only available if the **CapReceiptNotPaid** property is TRUE) and **PrintRecTotal** methods will either leave the printer in the Fiscal Receipt Total state or change the printer's state to Fiscal Receipt Ending, depending upon whether the entire receipt total has been met. The **PrintRecVoid** method will change the printer's state to Fiscal Receipt Ending.

While in the Fiscal Receipt Ending state the following fiscal methods are available:

PrintRecMessage
EndFiscalReceipt

The **PrintRecMessage** method is only available if the **CapAdditionalLines** property is TRUE and this method will leave the printer in the Fiscal Receipt Ending state. The **EndFiscalReceipt** will cause receipt closing and will then change the printer's state to Monitor.

Be aware that at no time can the printer’ s total for the receipt be negative. If this occurs the printer will generate an error.

Receipt Layouts

The following is an example of a typical receipt layout:

- **Header Lines:**
  Header lines contain all of the information about the store, such as telephone number, address and name of the store. All of these lines are fixed and are defined before selling the first item (using the **SetHeaderLine** method). These lines may either be printed when the **BeginFiscalReceipt** method is called or when the first fiscal receipt method is called.
- **Transaction Lines:**
  All of the lines of a fiscal transaction, such as line items, discounts and surcharges.
- **Total Line:**
  The line containing the transaction total, tender amounts and possibly change due.
- **Trailer Lines:**
  These are fixed promotional messages stored on the printer (using the **SetTrailerLine** method). They are automatically printed when the **EndFiscalReceipt** method is called. Note that the fiscal logotype, date and time and serial number lines are not considered part of the trailer lines. In fact, depending upon fiscal legislation and upon the printer vendor, the relative position of the trailer and the fiscal logotype lines can vary. Information which has to be inserted in the receipt due to fiscal legislation is automatically printed at receipt closure.

Example of a fiscal receipt:

VAT Tables

Some fiscal printers support storing VAT (Value Added Tax) tables in the printer' s memory. Some of these printers will allow the application to set and modify any of the table entries. Others allow only adding new table entries but do not allow existing entries to be modified. Some printers allow the VAT table to bet set only once.

If the printer supports VAT tables, the **CapHasVatTable** property is set to TRUE. If the printer allows the VAT table entries to be set or modified the **CapSetVatTable** property is set to TRUE. The maximum number of different vat rate entries in the VAT table is given by the **NumVatRates** property. VAT tables are set through a two step process. First the application uses the **SetVatValue** method to set each table entry to be sent to the printer. Next, the **SetVatTable** method is called to send the entire VAT table to the printer at one time.

Receipt Duplication

In some countries fiscal legislation can allow printing more than one copy of the same receipt. The **CapDuplicateReceipt** property will be set to TRUE if the printer is capable of printing duplicate receipts. Then, setting the **DuplicateReceipt** to TRUE causes the buffering of all receipt printing commands. **DuplicateReceipt** property is set to FALSE after receipt closing In order to print the receipt again the **PrintDuplicateReceipt** method has to be called.

CURRENCY amounts, percentage amounts, VAT rates, and quantity amounts

- CURRENCY amounts (and also prices) are passed as values with the data type CURRENCY. On a Win32-based platform this is a 64 bit signed long value that implicitly assumes four digits as the fractional part. So, the range supported is from
  -922,337,203,685,477.5808
  to
  +922,337,203,685,477.5807

  The fractional part used in the calculation unit of a Fiscal Printer may differ from the CURRENCY data type. The number of digits in the fractional part is stored in the **AmountDecimalPlaces** property and determined by the Fiscal Printer. The application has to take care that calculations in the application use the same fractional part for amounts.
- If the **CapHasVatTable** property is TRUE, VAT rates are passed using the indexes that were sent to the **SetVatValue** method.
  If the **CapHasVatTable** property is FALSE, VAT rates are passed as amounts with the data type LONG. The number of digits in the fractional part is implicitly assumed to be four.
- Percentage amounts are used in methods which allow also surcharge and/or discount amounts. If the amounts are specified to be a percentage value the value is also passed in a parameter of type CURRENCY.
  On a Win32-based platform the percentage value has then (as given by the CURRENCY data type) four digits in the fractional part.
  It is the percentage (0.0001% to 99.9999%) multiplied by 10000.
- Quantity amounts are passed as values with the data type LONG. The number of digits in the fractional part is stored in the **QuantityDecimalPlaces** property and determined by the Fiscal Printer.

Properties

AmountDecimalPlaces Property

Syntax **LONG AmountDecimalPlaces;**

Remarks Holds the number of decimal digits that the fiscal device uses for calculations.

This property is initialized when the device is enabled.

AsyncMode Property R/W

Syntax **BOOL AsyncMode;**

Remarks If TRUE, then some print methods like **PrintRecItemAdjustment**, **PrintRecItem**, **PrintNormal**, etc. will be performed asynchronously.
If FALSE, they will be performed synchronously.

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

For the complete list of method which are performed either synchronously or asynchronously see Printer States Model on page **187**.

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 Printer States Model (Page **187**)

CapAdditionalLines Property

Syntax **BOOL CapAdditionalLines;**

Remarks If TRUE, then the printer supports the printing of application defined lines on a fiscal receipt between the total line and the end of the fiscal receipt, FALSE otherwise.

If this property is TRUE, then after all totals lines are printed it is possible to print application-defined strings, such as the ones used for fidelity cards. In this case, after the total lines are printed, the **PrinterState** property is set to **ReceiptEnding** and **PrintRecMessage** can be called.

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

CapAmountAdjustment Property

Syntax **BOOL CapAmountAdjustment;**

Remarks If TRUE, then the printer handles fixed amount discounts or fixed amount surcharges on items, FALSE otherwise.

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

CapAmountNotPaid Property

Syntax **BOOL CapAmountNotPaid;**

Remarks If TRUE, then the printer allows the recording of not paid amounts, FALSE otherwise.

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

CapCheckTotal Property

Syntax **BOOL CapCheckTotal;**

Remarks If TRUE, then automatic comparison of the printer' s total and the application' s total can be enabled and disabled. If FALSE, then the automatic comparison cannot be enabled and is always considered disabled.

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.

CapDoubleWidth Property

Syntax **BOOL CapDoubleWidth;**

Remarks If TRUE, then the printer can print double width characters, FALSE otherwise.

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

CapDuplicateReceipt Property

Syntax **BOOL CapDuplicateReceipt;**

Remarks If TRUE, then the printer allows printing more than one copy of the same fiscal receipt, FALSE otherwise..

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

CapFixedOutput Property

Syntax **BOOL CapFixedOutput;**

Remarks If TRUE, then the printer supports fixed format text printing through the **BeginFixedOutput**, **PrintFixedOutput** and **EndFixedOutput** methods, FALSE otherwise.

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

CapHasVatTable Property

Syntax **BOOL CapHasVatTable;**

Remarks If TRUE, then the printer has a tax table, FALSE otherwise.

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

CapIndependentHeader Property

Syntax **BOOL CapIndependentHeader;**

Remarks If TRUE, then the printer supports printing the fiscal receipt header lines before the first fiscal receipt command is processed, FALSE otherwise.

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

CapItemList Property

Syntax **BOOL CapItemList;**

Remarks If TRUE, then the printer can print a report of items of a specified VAT class, FALSE otherwise.

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.

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.

Unlike POS printers, on fiscal printers the application is not able to directly access the journal. The fiscal printer itself prints on the journal if present.

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

CapNonFiscalMode Property

Syntax **BOOL CapNonFiscalMode;**

Remarks If TRUE, then the printer allows printing in non-fiscal mode, FALSE otherwise.

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

CapOrderAdjustmentFirst Property

Syntax **BOOL CapOrderAdjustmentFirst;**

Remarks This property defines the usage of **PrintRecItem** and **PrintRecItemAdjustment**

If FALSE, the application has to call **PrintRecItem** first and then call **PrintRecItemAdjustment** to give a discount or a surcharge for a single article.

If TRUE, the application has to call **PrintRecItemAdjustment** first and then call **PrintRecItem** .

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

CapPercentAdjustment Property

Syntax **BOOL CapPercentAdjustment;**

Remarks If TRUE, then the printer handles percentage discounts or percentage surcharges on items, FALSE otherwise.

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

CapPositiveAdjustment Property

Syntax **BOOL CapPositiveAdjustment;**

Remarks This property defines abilities of the **PrintRecItemAdjustment**

If it is TRUE then it is possible to apply surcharges, otherwise it is false.

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

CapPowerLossReport Property

Syntax **BOOL CapPowerLossReport;**

Remarks If TRUE, then the printer can print a power loss report using the **PrintPowerLossReport** method, FALSE otherwise.

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

CapPredefinedPaymentLines Property

Syntax **BOOL CapPredefinedPaymentLines;**

Remarks If TRUE, the printer can store and print predefined payment descriptions, FALSE otherwise.

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

CapReceiptNotPaid Property

Syntax **BOOL CapReceiptNotPaid;**

Remarks If TRUE, then the printer supports using the **PrintRecNotPaid** method to specify a part of the receipt total that is not paid, FALSE otherwise.

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.

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.

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.

CapRemainingFiscalMemory Property

Syntax **BOOL CapRemainingFiscalMemory;**

Remarks If TRUE, then the printer supports using the **RemainingFiscalMemory** property to show the amount of Fiscal Memory remaining If FALSE, the printer does not support reporting the Fiscal Memory status of the printer.

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

CapReservedWord Property

Syntax **BOOL CapReservedWord;**

Remarks If TRUE, then the printer prints a reserved word (for example, "TOTALE") before printing the total amount, FALSE otherwise.

If TRUE, the reserved word is stored in the **ReservedWord** property. This reserved word may not be printed using any fiscal print method.

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

CapSetHeader Property

Syntax **BOOL CapSetHeader;**

Remarks If TRUE, then it is possible to use the **SetHeaderLine** method to initialize the contents of a particular line of the receipt header, FALSE otherwise.

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

CapSetPOSID Property

Syntax **BOOL CapSetPOSID;**

Remarks If TRUE, then it is possible to use the **SetPOSID** method to initialize the values of POSID and CashierID, FALSE otherwise.

These values are printed on each fiscal receipt.

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

CapSetStoreFiscalID Property

Syntax **BOOL CapSetStoreFiscalID;**

Remarks If TRUE, then it is possible to use the **SetStoreFiscalID** method to set up the Fiscal ID number which will be printed on each fiscal receipt, FALSE otherwise.

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

CapSetTrailer Property

Syntax **BOOL CapSetTrailer;**

Remarks If TRUE, then it is possible to use the **SetTrailerLine** method to initialize the contents of a particular line of the receipt trailer, FALSE otherwise.

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

CapSetVatTable Property

Syntax **BOOL CapSetVatTable;**

Remarks If TRUE, then it is possible to use the **SetVatValue** and **SetVatTable** methods to modify the contents of the printer’ s VAT table, FALSE otherwise.

Some printers may not allow existing VAT table entries to be modified. Only new entries may be set on these printers.

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.

CapSlpFiscalDocument Property

Syntax **BOOL CapSlpFiscalDocument;**

Remarks If TRUE, then the printer allows fiscal printing to the slip station, FALSE otherwise.

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

CapSlpFullSlip Property

Syntax **BOOL CapSlpFullSlip;**

Remarks If TRUE, then the printer supports printing full length forms on the slip station, FALSE otherwise.

It is possible to choose between full slip and validation documents by setting the **SlipSelection** property.

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 printer has a slip station, FALSE otherwise.

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

CapSlpValidation Property

Syntax **BOOL CapSlpValidation;**

Remarks If TRUE, then the printer supports printing validation information on the slip station, FALSE otherwise.

It is possible to choose between full slip and validation documents by setting the **SlipSelection** property.

In some countries, when printing non fiscal validations using the slip station a limited number of lines could be printed.

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

CapSubAmountAdjustment Property

Syntax **BOOL CapSubAmountAdjustment;**

Remarks If TRUE, then the printer handles fixed amount discounts on the subtotal, FALSE otherwise.

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

CapSubPercentAdjustment Property

Syntax **BOOL CapSubPercentAdjustment;**

Remarks If TRUE, then the printer handles percentage discounts on the subtotal, FALSE otherwise.

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

CapSubtotal Property

Syntax **BOOL CapSubtotal;**

Remarks If TRUE, then it is possible to use the **PrintRecSubtotal** method to print the current subtotal, FALSE otherwise.

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

CapTrainingMode Property

Syntax **BOOL CapTrainingMode;**

Remarks If TRUE, then the printer supports a training mode, FALSE otherwise.

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

CapValidateJournal Property

Syntax **BOOL CapValidateJournal;**

Remarks If TRUE, then it is possible to use the **PrintNormal** method to print a validation string on the journal station, FALSE otherwise.

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

CapXReport Property

Syntax **BOOL CapXReport;**

Remarks If TRUE, then it is possible to use the **PrintXReport** method to print an X report, FALSE otherwise.

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

CheckTotal Property R/W

Syntax **BOOL CheckTotal;**

Remarks If TRUE, automatic comparison between the fiscal printer' s total and the application' s total is enabled. If FALSE, automatic comparison is disabled.

This property is only valid if **CapCheckTotal** is TRUE.

This property is initialized to TRUE 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.
OPOS_E_ILLEGAL	Setting this property is not valid for this service object (see **CapCheckTotal**).

CountryCode Property

Syntax **LONG CountryCode;**

Remarks Holds a value identifying which countries are supported by this Service Object. It can contain any of the following values logically ORed together:

Value	Meaning
FPTR_CC_BRAZIL	The printer supports Brazil' s fiscal rules.
FPTR_CC_GREECE	The printer supports Greece' s fiscal rules.
FPTR_CC_HUNGARY	The printer supports Hungary' s fiscal rules.
FPTR_CC_ITALY	The printer supports Italy' s fiscal rules.
FPTR_CC_POLAND	The printer supports Poland' s fiscal rules.
FPTR_CC_TURKEY	The printer supports Turkey' s fiscal rules.

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

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.

DayOpened Property

Syntax **BOOL DayOpened;**

Remarks If TRUE, then the fiscal day has been started on the printer, FALSE otherwise.

The Fiscal Day of the printer can be either opened or not opened. The **DayOpened** property reflects whether or not the printer considers its Fiscal Day to be opened or not.

Some methods may only be called while the Fiscal Day is not yet opened (**DayOpened** is FALSE). Methods that can be called after the Fiscal Day is opened change from country to country. Usually all the configuration methods are to be called only before the Fiscal Day is opened.

Depending on fiscal legislation, some of the following methods may be allowed only if the printer has not yet begun its Fiscal Day:

SetDate
SetHeaderLine
SetPOSID
SetStoreFiscalID
SetTrailerLine
SetVatTable
SetVatValue

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

DescriptionLength Property

Syntax **LONG DescriptionLength;**

Remarks Holds the maximum number of characters that may be passed as a description parameter.

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

DuplicateReceipt Property

Syntax **BOOL DuplicateReceipt;**

Remarks If this property is set to TRUE all the printing commands inside a fiscal receipt will be buffered and they can be printed again via the **PrintDuplicateReceipt** method.

ErrorLevel Property

Syntax **LONG ErrorLevel;**

Remarks The severity of the error condition.

Values are:

Value	**Meaning**
FPTR_EL_NONE	No error condition is present.
FPTR_EL_RECOVERABLE	A recoverable error has occurred. (Example: Out of paper.)
FPTR_EL_FATAL	A non-recoverable error has occurred. (Example: Internal printer failure.)
FPTR_EL_BLOCKED	A severe hardware failure which can be resolved only by technicians. (Example: Fiscal memory failure.). This error can not be recovered.

This property is set by the Control just before delivering an **ErrorEvent**. When the error is cleared, then the property is changed to FPTR_EL_NONE.

ErrorOutID Property

Syntax **LONG ErrorOutID;**

Remarks The identifier of the output in the queue which raised an error event, when using asynchronous printing.

This property is set just before an **ErrorEvent** is delivered.

ErrorState Property

Syntax **LONG ErrorState;**

Remarks Holds the current state of the printer when firing an error event for an asynchronous output.

This property is set just before an **ErrorEvent** is delivered.

See the **PrinterState** property on page 220 for a list of values.

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: FPTR_S_JOURNAL, FPTR_S_RECEIPT, FPTR_S_SLIP, FPTR_S_JOURNAL_RECEIPT.

This property is set just before an **ErrorEvent** is delivered.

ErrorString Property

Syntax **BSTR ErrorString;**

Remarks A vendor-supplied description of the current error.

This property is set by the Control just before delivering 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 delivered.

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.

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 **JrnNearEnd** Property

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

MessageLength Property

Syntax **LONG MessageLength;**

Remarks Holds the maximum number of characters that may be passed as a message line in the method **PrintRecMessage**. The value may change in different modes of the fiscal printer. For example in the mode "Fiscal Receipt" the number of characters may be bigger than in the mode "Fiscal Receipt Total".

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

NumHeaderLines Property

Syntax **LONG NumHeaderLines;**

Remarks Contains the maximum number of header lines that can be printed for each fiscal receipt. Header lines usually contain information like store address, store name, store Fiscal ID. Each header line is set using the **SetHeaderLine** method and remains set even after the printer is switched off. Header lines are automatically printed when a fiscal receipt is initiated using the **BeginFiscalReceipt** method or when the first line item inside a receipt is sold.

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

NumTrailerLines Property

Syntax **LONG NumTrailerLines;**

Remarks Contains the maximum number of trailer lines that can be printed for each fiscal receipt. Trailer lines are usually promotional messages. Each trailer line is set using the **SetTrailerLine** method and remains set even after the printer is switched off. Trailer lines are automatically printed either after the last **PrintRecTotal** or when a fiscal receipt is closed using the **EndFiscalReceipt** method.

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

NumVatRates Property

Syntax **LONG NumVatRates;**

Remarks Contains the maximum number of vat rates that can be entered into the printer’ s Vat table.

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

PredefinedPaymentLines Property

Syntax **BSTR PredefinedPaymentLines;**

Remarks If **CapPredefinedPaymentLines** is TRUE, only predefined payment lines are allowed. The value of this property is the list of all possible words to be used as indexes of the predefined payment lines (for example, "a,b,c,d,z"). Those indexes are used in the **PrintRecTotal** method for the *description* parameter.

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

PrinterState Property

Syntax **LONG PrinterState;**

Remarks Holds the printer' s current operational state. This property controls which methods are currently legal.

Values are:

Value	Meaning
FPTR_PS_MONITOR	If **TrainingModeActive** property is FALSE: The printer is currently not in a specific operational mode. In this state the printer will accept any of the **Begin…** methods as well as the **Set…** methods. If **TrainingModeActive** property is TRUE: The printer is currently being used for training purposes. In this state the printer will accept any of the **PrintRec…** methods or the **EndTraining** method.
FPTR_PS_FISCAL_RECEIPT	If **TrainingModeActive** property is FALSE: The printer is currently processing a fiscal receipt. In this state the printer will accept any of the **PrintRec…** methods. If **TrainingModeActive** property is TRUE: The printer is currently being used for training purposes and a fiscal receipt is currently opened.
FPTR_PS_FISCAL_RECEIPT_TOTAL	If **TrainingModeActive** property is FALSE: The printer has already accepted at least one payment, but the total has not been completely paid. In this state the printer will accept either the **PrintRecTotal** or **PrintRecNotPaid** methods. If **TrainingModeActive** property is TRUE: The printer is currently being used for training purposes and the printer has already accepted at least one payment, but the total has not been completely paid.

FPTR_PS_FISCAL_RECEIPT_ENDING	If **TrainingModeActive** property is FALSE: The printer has completed the receipt up to the total line. In this state the printer will accept either the **PrintRecMessage** or **EndFiscalReceipt** methods. If **TrainingModeActive** property is TRUE: The printer is currently being used for training purposes and a fiscal receipt is going to be closed.
FPTR_PS_FISCAL_DOCUMENT	The printer is currently processing a fiscal slip. In this state the printer will accept either the **PrintFiscalDocumentLine** or **EndFiscalDocument** methods.
FPTR_PS_FIXED_OUTPUT	The printer is currently processing fixed text output to one or more stations. In this state the printer will accept either the **PrintFixedOutput** or **EndFixedOutput** methods.
FPTR_PS_ITEM_LIST	The printer is currently processing an item list report. In this state the printer will accept either the **VerifyItem** or **EndItemList** methods.
FPTR_PS_NONFISCAL	The printer is currently processing non-fiscal output to one or more stations. In this state the printer will accept either the **PrintNormal** or **EndNonFiscal** methods.
FPTR_PS_LOCKED	The printer has encountered a non-recoverable hardware problem. A printer technician must be contacted to exit this state.
FPTR_PS_REPORT	The printer is currently processing a fiscal report. In this state the printer will not accept any methods until the report has completed.

There are a few methods that are accepted in any state except FPTR_PS_LOCKED. These are **BeginInsertion**, **EndInsertion**, **BeginRemoval**, **EndRemoval**, **GetDate**, **GetData**, **GetTotalizer**, **GetVatEntry**, **ResetPrinter** and **ClearOutput**.

For more information, see the discussion of Printer States on page **190**.

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

QuantityDecimalPlaces Property

Syntax **LONG QuantityDecimalPlaces;**

Remarks Holds the number of decimal digits in the fractional part that should be assumed to be in any quantity parameter passed to this Service Object.

This property is initialized to 0 (zero) by the **Open** method.

QuantityLength Property

Syntax **LONG QuantityLength;**

Remarks Holds the maximum number of digits that may be passed as a quantity parameter, including both the whole and fractional parts.

This property is initialized by the **Open** 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

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

RemainingFiscalMemory Property

Syntax **LONG RemainingFiscalMemory;**

Remarks Holds the remaining counter of Fiscal Memory.

This property is initialized and kept current while the device is enabled and may be updated by **PrintZReport** method.

See Also **CapRemainingFiscalMemory** Property

ReservedWord Property

Syntax **BSTR ReservedWord;**

Remarks Holds the string that is automatically printed with the total when the **PrintRecTotal** method is called. This word may not occur in any string that is passed into any fiscal output methods.

This property is only valid if **CapReservedWord** is TRUE.

This property is initialized by the **Open** 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

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

SlipSelection Property R/W

Syntax **LONG SlipSelection;**

Remarks Selects the kind of document to be printed on the slip station.

Values are:

Value	Meaning
FPTR_SS_FULL_LENGTH	Print full length documents.
FPTR_SS_VALIDATION	Print validation documents.

The value of **SlipSelection** is initialized to FPTR_SS_FULL_LENGTH by the **Claim** 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 slip type was specified.

TrainingModeActive Property

Syntax **BOOL TrainingModeActive;**

Remarks Holds the current printer's operational state concerning the training mode. Training mode allows all fiscal commands, but each receipt is marked as non-fiscal and no internal printer registers are updated with any data while in training mode. Some countries' fiscal rules require that all blank characters on a training mode receipt are printed as some other character. Italy, for example, requires that all training mode receipts print a ? instead of a blank.

Values are:

Value	**Meaning**
TRUE	The printer is currently in training mode. That means no data are written into the EPROM of the fiscal printer.
FALSE	The printer is currently in normal mode. All printed receipts will also update the fiscal memory..

Methods

BeginFiscalDocument Method

Syntax **LONG BeginFiscalDocument (LONG** *DocumentAmount***);**

Parameter	**Description**
DocumentAmount	Amount of document to be stored by the printer.

Remarks Called to initiate fiscal printing to the slip station.

This method is only supported if **CapSlpFiscalDocument** is TRUE.

The slip paper must be inserted into the slip station using **Begin/EndInsertion** before calling this method.

Each fiscal line will be printed using the **PrintFiscalDocumentLine** method.

If this method is successful, the **PrinterState** property will be changed to FPTR_PS_FISCAL_DOCUMENT.

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.
OPOS_E_ILLEGAL	One of the following errors occurred: • The slip station does not exist (see the **CapSlpPresent** property). • The printer does not support fiscal output to the slip station (see the **CapSlpFiscalDocument** property).

OPOS_E_EXTENDED:

ResultCodeExtended = OPOS_EFPTR_WRONG_STATE:
The printer’ s current state does not allow this state transition.

ResultCodeExtended = OPOS_EFPTR_SLP_EMPTY:
There is no paper in the slip station.

ResultCodeExtended = OPOS_EFPTR_BAD_ITEM_AMOUNT:
The *DocumentAmount* parameter is invalid.

Other Values	See **ResultCode**.

See Also **EndFiscalDocument** Method, **PrintFiscalDocumentLine** Method
AmountDecimalPlaces Property

BeginFiscalReceipt Method

Syntax **LONG BeginFiscalReceipt (BOOL** *PrintHeader***);**

Parameter	Description
PrintHeader	Indicates if the header lines are to be printed at this time.

Remarks Called to initiate fiscal printing to the receipt station.

If *PrintHeader* and the **CapIndependentHeader** property are both TRUE all defined header lines will be printed before control is returned. Otherwise header lines will be printed when the first item is sold.

If this method is successful, the **PrinterState** property will be changed to FPTR_PS_FISCAL_RECEIPT.

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.
OPOS_E_EXTENDED:	
ResultCodeExtended = OPOS_EFPTR_WRONG_STATE:	The printer' s current state does not allow this state transition.
Other Values	See **ResultCode**.

See Also **EndFiscalReceipt** Method, **PrintRec…** Methods, **CapIndependentHeader** Property

BeginFixedOutput Method

Syntax **LONG BeginFixedOutput (LONG** *Station*, **LONG** *DocumentType***);**

Parameter	Description
Station	The printer station to be used. May be either FPTR_S_RECEIPT or FPTR_S_SLIP.
DocumentType	Identifier of a document stored in the printer.

Remarks Called to initiate non-fiscal fixed text printing on a printer station.
This method is only supported if **CapFixedOutput** is TRUE.

If the *Station* parameter is FPTR_S_SLIP, the slip paper must be inserted into the slip station using **Begin/EndInsertion** before calling this method.

Each fixed output will be printed using the **PrintFixedOutput** method. If this method is successful, the **PrinterState** property will be changed to FPTR_PS_FIXED_OUTPUT. The **EndFixedOutput** method ends fixed output modality and resets **PrinterState**.

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.
OPOS_E_ILLEGAL	One of the following errors occurred: • The slip station does not exist (see the **CapSlpPresent** property). • The printer does not support fixed output (see the **CapFixedOutput** property). • The *Station* parameter is invalid. • The *DocumentType* is invalid.

OPOS_E_EXTENDED:

ResultCodeExtended = OPOS_EFPTR_WRONG_STATE: The printer’ s current state does not allow this state transition.

ResultCodeExtended = OPOS_EFPTR_SLP_EMPTY: There is no paper in the slip station.

Other Values	See **ResultCode**.

See Also **EndFixedOutput** Method, **PrintFixedOutput** Method

BeginInsertion Method

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

The *Timeout* parameter gives the number of milliseconds before failing the method.
If zero, the method tries to begin insertion mode, then returns the appropriate status immediately.
If OPOS_FOREVER (-1), the method tries to begin insertion mode, then waits as long as needed until either the form is inserted or an error occurs.

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.
Other Values	See **ResultCode**.

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

BeginItemList Method

Syntax **LONG BeginItemList (LONG** *VatID***);**

Parameter	Description
VatID	Vat identifier for reporting.

Remarks Called to initiate a validation report of items belonging to a particular VAT class.

This method is only supported if **CapItemList** is TRUE.

If this method is successful, the **PrinterState** property will be changed to FPTR_PS_ITEM_LIST.

After this method only **VerifyItem** and **EndItemList** methods may be 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_BUSY	Cannot perform while output is in progress.
OPOS_E_ILLEGAL	One of the following errors occurred: • The printer does not support an item list report (see the **CapItemList** property). • The printer does not support VAT tables (see the **CapHasVatTable** property).
OPOS_E_EXTENDED:	
ResultCodeExtended = OPOS_EFPTR_WRONG_STATE:	The printer’ s current state does not allow this state transition.
ResultCodeExtended = OPOS_EFPTR_BAD_VAT:	The *VatID* parameter is invalid.
Other Values	See **ResultCode**.

See Also **EndItemList** Method, **VerifyItem** Method

BeginNonFiscal Method

Syntax **LONG BeginNonFiscal ();**

Remarks Called to initiate non-fiscal operations on the printer.

This method is only supported if **CapNonFiscalMode** is TRUE.

Output in this mode is accomplished using the **PrintNormal** method.

This method can be successfully called only if the current value of the **PrinterState** property is FPTR_PS_MONITOR.

If this method is successful, the **PrinterState** property will be changed to FPTR_PS_NONFISCAL.

In order to stop non fiscal modality **EndNonFiscal** method should be 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_BUSY	Cannot perform while output is in progress.
OPOS_E_ILLEGAL	One of the following errors occurred: • The printer does not support non-fiscal output (see the **CapNonFiscalMode** property).
OPOS_E_EXTENDED:	
ResultCodeExtended =	OPOS_EFPTR_WRONG_STATE: The printer' s current state does not allow this state transition.
Other Values	See **ResultCode**.

See Also **EndNonFiscal** Method, **PrintNormal** Method

BeginRemoval Method

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

The *Timeout* property gives the number of milliseconds before failing the method.
If zero, the method tries to begin removal mode, then returns the appropriate status immediately.
If OPOS_FOREVER (-1), the method tries to begin removal mode, then waits as long as needed until either the form is removed or an error occurs.

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.
Other Values	See **ResultCode**.

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

BeginTraining Method

Syntax **LONG BeginTraining ();**

Remarks Called to initiate training operations.

This method is only supported if **CapTrainingMode** is TRUE.

Output in this mode is accomplished using the **PrintRec…** methods in order to print a receipt or other methods to print reports.

This method can be successfully called only if the current value of the **PrinterState** property is FPTR_PS_MONITOR.

If this method is successful, the **TrainingModeActive** property will be changed to 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_BUSY	Cannot perform while output is in progress.
OPOS_E_ILLEGAL	One of the following errors occurred: • The printer does not support training mode (see the **CapTrainingMode** property).
OPOS_E_EXTENDED:	
ResultCodeExtended = OPOS_EFPTR_WRONG_STATE:	The printer' s current state does not allow this state transition.
Other Values	See **ResultCode**.

See Also **EndTraining** Method, **PrintRec…** Methods

ClearError Method

Syntax **LONG ClearError ();**

Remarks Called to clear all printer error conditions.

This method is always performed synchronously.

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

Value	Meaning
OPOS_SUCCESS	The method was successful.
OPOS_E_FAILURE	Error recovery failed.
Other Values	See **ResultCode**.

EndFiscalDocument Method

Syntax **LONG EndFiscalDocument ();**

Remarks Called to terminate fiscal printing to the slip station.

This method is only supported if **CapSlpFiscalDocument** is TRUE.

If this method is successful, the **PrinterState** property will be changed to FPTR_PS_MONITOR.

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.
OPOS_E_ILLEGAL	The printer does not support fiscal output to the slip station (see the **CapSlpFiscalDocument** property).

OPOS_E_EXTENDED:

ResultCodeExtended = OPOS_EFPTR_WRONG_STATE:
The printer is not currently in the Fiscal Document state.

Other Values See **ResultCode**.

See Also **BeginFiscalDocument** Method, **PrintFiscalDocumentLine** Method

EndFiscalReceipt Method

Syntax **LONG EndFiscalReceipt (BOOL** *PrintHeader***);**

Parameter	**Description**
PrintHeader	Indicates if the header lines are to be printed at this time.

Remarks Called to terminate fiscal printing to the receipt station.

If *PrintHeader* is FALSE, this method will close the current fiscal receipt, cut it, and print the trailer lines and fiscal logotype, if they were not already printed after the total lines. All functions carried out by this method will be completed before this call returns.

If this method is successful, the **PrinterState** property will be changed to FPTR_PS_MONITOR.

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.
OPOS_E_EXTENDED:	
ResultCodeExtended = OPOS_EFPTR_WRONG_STATE:	The printer is not currently in the Fiscal Receipt Ending state.
Other Values	See **ResultCode**.

See Also **BeginFiscalReceipt** Method, **PrintRec…** Methods

EndFixedOutput Method

Syntax **LONG EndFixedOutput ();**

Remarks Called to terminate non-fiscal fixed text printing on a printer station.

This method is only supported if **CapFixedOutput** is TRUE.

If this method is successful, the **PrinterState** property will be changed to FPTR_PS_MONITOR.

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.
OPOS_E_ILLEGAL	The printer does not support fixed output (see the **CapFixedOutput** property).

OPOS_E_EXTENDED:

ResultCodeExtended = OPOS_EFPTR_WRONG_STATE:
The printer is not currently in the Fixed Output state.

Other Values	See **ResultCode**.

See Also **BeginFixedOutput** Method, **PrintFixedOutput** 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_EFPTR_SLP_EMPTY 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	**ResultCodeExtended** = OPOS_EFPTR_COVER_OPEN: The device was taken out of insertion mode while the printer cover was open. **ResultCodeExtended** = OPOS_EFPTR_SLP_EMPTY: The device was taken out of insertion mode without a form being inserted.
Other Values	See **ResultCode**.

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

EndItemList Method

Syntax **LONG EndItemList ();**

Remarks Called to terminate a validation report of items belonging to a particular VAT class.

This method is only supported if **CapItemList** is TRUE and **CapHasVatTable** is TRUE.

This method is paired with the **BeginItemList** method.

This method can be successfully called only if current value of PrinterState property is equal to FPTR_PS_ITEM_LIST.

If this method is successful, the **PrinterState** property will be changed to FPTR_PS_MONITOR.

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.
OPOS_E_ILLEGAL	One of the following errors occurred: • The printer does not support fixed output (see the **CapItemList** property). • The printer does not support VAT tables (see the **CapHasVatTable** property).
OPOS_E_EXTENDED:	
ResultCodeExtended =	OPOS_EFPTR_WRONG_STATE: The printer' s current state does not allow this state transition.
Other Values	See **ResultCode**.

See Also **BeginItemList** Method, **VerifyItem** Method

EndNonFiscal Method

Syntax **LONG EndNonFiscal ();**

Remarks Called to terminate non-fiscal operations on one printer station.

This method is only supported if **CapNonFiscalMode** is TRUE.

If this method is successful, the **PrinterState** property will be changed to FPTR_PS_MONITOR.

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.
OPOS_E_ILLEGAL	The printer does not support non-fiscal output (see the **CapNonFiscalMode** property).

OPOS_E_EXTENDED:

ResultCodeExtended = OPOS_EFPTR_WRONG_STATE:
The printer is not currently in the Non-Fiscal state.

Other Values	See **ResultCode**.

See Also **BeginNonFiscal** Method, **PrintNormal** 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_EFPTR_SLP_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_EFPTR_SLP_FORM: The device was taken out of removal mode while a form was still present.
Other Values	See **ResultCode**.

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

EndTraining Method

Syntax **LONG EndTraining ();**

Remarks Called to terminate training operations on either the receipt or slip station.

This method is only supported if **CapTrainingMode** is TRUE.

If this method is successful, the **TrainingModeActive** property will be changed to 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_BUSY	Cannot perform while output is in progress.
OPOS_E_ILLEGAL	The printer does not support training mode (see the **CapTrainingMode** property).

OPOS_E_EXTENDED:

ResultCodeExtended = OPOS_EFPTR_WRONG_STATE:
The printer is not currently in the Training state.

Other Values	See **ResultCode**.

See Also **BeginTraining** Method, **PrintRec…** Methods

GetData Method

Syntax **LONG GetData (LONG** *DataItem*, **LONG*** *OptArgs*, **BSTR*** *Data***);**

Parameter	Description
DataItem	The specific data item to retrieve.
OptArgs	For some countries, this additional argument may be needed. Consult the Service Object vendor's documentation for details.*Data* Character string to hold the data retrieved.

The *DataItem* parameter values are:

Value	Meaning
FPTR_GD_CURRENT_TOTAL	Get the current receipt total.
FPTR_GD_DAILY_TOTAL	Get the daily total.
FPTR_GD_RECEIPT_NUMBER	Get the number of fiscal receipts printed.
FPTR_GD_REFUND	Get the current total of refunds.
FPTR_GD_NOT_PAID	Get the current total of not paid receipts.
FPTR_GD_MID_VOID	Get the total number of voided receipts.
FPTR_GD_Z_REPORT	Get the Z report number.
FPTR_GD_GRAND_TOTAL	Get the printer’ s grand total.
FPTR_GD_PRINTER_ID	Get the printer’ s fiscal ID.
FPTR_GD_FIRMWARE	Get the printer’ s firmware release number.
FPTR_GD_RESTART	Get the printer’ s restart count

Remarks Called to retrieve data from the printer’ s fiscal module.

The data is returned in a string because some of the fields, such as the grand total, might overflow a 4-byte integer.

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 *DataItem* specified is invalid.
OPOS_E_BUSY	Cannot perform while output is in progress.
Other Values	See **ResultCode**.

GetDate Method

Syntax **LONG GetDate (BSTR*** *Date***);**

Parameter	Description
Date	Date and time returned as a string.

Remarks Called to get the printer’ s date and time.

The date and time are returned as a string in the format “ddmmyyyyhhmm”, where:

dd	day of the month (1 - 31)
mm	month (1 - 12)
yyyy	year (1997-)
hh	hour (0-23)
mm	minutes (0-59)

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	Retrieval of the date and time is not valid at this time.
Other Values	See **ResultCode**.

GetTotalizer Method

Syntax **LONG GetTotalizer (LONG** *VatID*, **LONG** *OptArgs*, **BSTR*** *Data***);**

Parameter	Description
VatID	VAT identifier of the required totalizer.
OptArgs	For some countries, this additional argument may be needed. Consult the Service Object vendor's documentation for details.
Data	Totalizer returned as a string.

Remarks Called to get the totalizer associated with the given VAT rate.

If **CapSetVatTable** is false then only one totalizer is present.

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 *VatID* parameters is invalid.
Other Values	See **ResultCode**.

GetVatEntry Method

Syntax **LONG GetVatEntry (LONG** *VatID*, **LONG** *OptArgs*, **LONG*** *VatRate***);**

Parameter	Description
VatID	VAT identifier of the required rate.
OptArgs	For some countries, this additional argument may be needed. Consult the Service Object vendor's documentation for details.
VatRate	Pointer to the rate associated with the VAT identifier.

Remarks Called to get the rate associated with a given VAT identifier.

This method is only supported if **CapSetVatTable** 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 *VatID* parameters is invalid.
Other Values	See **ResultCode**.

PrintDuplicateReceipt Method

Syntax **LONG PrintDuplicateReceipt ();**

Remarks Called to print a duplicate of a buffered transaction.

This method is only supported if **CapDuplicateReceipt** is TRUE.

This method will succeed if both the **CapDuplicateReceipt** and **DuplicateReceipt** properties are TRUE.

This method resets the **DuplicateReceipt** property to 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_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: • The printer does not support duplicate receipts (see the **CapDuplicateReceipt** property). • There is no buffered transaction to print (see **DuplicateReceipt** property).

OPOS_E_EXTENDED:

ResultCodeExtended = OPOS_EFPTR_WRONG_STATE:
The printer is not currently in the Monitor state.

ResultCodeExtended = OPOS_EFPTR_JRN_EMPTY:
The journal station is out of paper.

ResultCodeExtended = OPOS_EFPTR_REC_EMPTY:
The receipt station is out of paper.

PrintFiscalDocumentLine Method

Syntax **LONG PrintFiscalDocumentLine (BSTR** *DocumentLine***);**

Parameter	Description
DocumentLine	String to be printed on the fiscal slip.

Remarks Called to print a line of fiscal text to the slip station.

This method is only supported if **CapSlpFiscalDocument** is TRUE.

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_BUSY	Cannot perform while output is in progress. (Can only be returned if **AsyncMode** is FALSE.)
OPOS_E_ILLEGAL	The printer does not support fiscal documents (see the **CapSlpFiscalDocument** property).

OPOS_E_EXTENDED:

ResultCodeExtended = OPOS_EFPTR_WRONG_STATE:
The printer is not currently in the Fiscal Document state.

ResultCodeExtended = OPOS_EFPTR_COVER_OPEN:
The printer cover is open.
(Can only be returned if **AsyncMode** is FALSE.)

ResultCodeExtended = OPOS_EFPTR_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 **BeginFiscalDocument** Method, **EndFiscalDocument** Method

PrintFixedOutput Method

Syntax **LONG PrintFixedOutput (LONG** *DocumentType*, **LONG** *LineNumber*, **BSTR** *Data***);**

Parameter	**Description**
DocumentType	Identifier of a document stored in the printer
LineNumber	Number of the line in the document to print.
Data	String parameter for placement in printed line.

Remarks Called to print a line of a fixed document to the print station specified in the **BeginFixedOutput** method. Each call prints a single line from a document by merging the stored text with the parameter *Data*. Within a document lines must be printed sequentially. Some lines are optional and some lines are required, such as the first and last lines.

This method is only supported if **CapFixedOutput** is TRUE.

The printer state is set to FPTR_PS_FIXED_OUTPUT

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_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: • The printer does not support fixed output (see the **CapFixedOutput** property). • The *LineNumber* is invalid.
OPOS_E_EXTENDED:	
ResultCodeExtended = OPOS_EFPTR_WRONG_STATE:	The printer is not currently in the Fixed Output state.
ResultCodeExtended = OPOS_EFPTR_COVER_OPEN:	The printer cover is open. (Can only be returned if **AsyncMode** is FALSE.)
ResultCodeExtended = OPOS_EFPTR_JRN_EMPTY:	The journal station is out of paper. (Can only be returned if **AsyncMode** is FALSE.)
ResultCodeExtended = OPOS_EFPTR_REC_EMPTY:	The receipt station was specified but is out of paper. (Can only be returned if **AsyncMode** is FALSE.)
ResultCodeExtended = OPOS_EFPTR_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 **BeginFixedOutput** Method, **EndFixedOutput** Method

PrintNormal Method

Syntax **LONG PrintNormal (LONG** *Station*, **BSTR** *Data***);**

Parameter	**Description**
Station	The printer station to be used. May be FPTR_S_RECEIPT, FPTR_S_JOURNAL, FPTR_S_JOURNAL_RECEIPT or FPTR_S_SLIP.
Data	The characters to be printed, consisting mostly of printable characters.

This method performs non-fiscal printing. Escape sequences, carriage returns (13 decimal), and line feeds (10 decimal) are available on some printers, but in many cases these are not supported.
The format of this data depends upon the value of the **BinaryConversion** property.

Remarks Called 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)	If a Carriage Return immediately precedes a Line Feed, or if the line buffer is empty, then it is ignored. Otherwise, the line buffer is printed and the printer does not feed to the next print line. On some printers, print without feed may be directly supported. 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. If the printer does not support either of these features, then Carriage Return acts like a Line Feed.

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 *Station* does not exist. (See the **CapSlpPresent** property.)
OPOS_E_BUSY	Cannot perform while output is in progress. (Can only be returned if **AsyncMode** is FALSE.)

OPOS_E_EXTENDED;

ResultCodeExtended = OPOS_EFPTR_WRONG_STATE:
The printer is not currently in the Non-Fiscal state.

ResultCodeExtended = OPOS_EFPTR_COVER_OPEN:
The printer cover is open.
(Can only be returned if **AsyncMode** is FALSE.)

ResultCodeExtended = OPOS_EFPTR_JRN_EMPTY:
The journal station was specified but is out of paper.
(Can only be returned if **AsyncMode** is FALSE.)

ResultCodeExtended = OPOS_EFPTR_REC_EMPTY:
The receipt station was specified but is out of paper.
(Can only be returned if **AsyncMode** is FALSE.)

ResultCodeExtended = OPOS_EFPTR_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 **BeginNonFiscal** Method, **EndNonFiscal** Method, **AsyncMode** property

PrintPeriodicTotalsReport Method

Syntax **LONG PrintPeriodicTotalsReport (BSTR** *Date1*, **BSTR** *Date2***);**

Parameter	**Description**
Date1	Starting date of report to print.
Date2	Ending date of report to print.

Remarks Called to print a report of totals for a range of dates on the receipt.

This method is always performed synchronously.

The dates are strings in the format "ddmmyyyyhhmm", where:

dd	day of the month (1 - 31)
mm	month (1 - 12)
yyyy	year (1997-)
hh	hour (0-23)
mm	minutes (0-59)

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
OPOS_E_EXTENDED:	
ResultCodeExtended = OPOS_EFPTR_WRONG_STATE:	The printer' s current state does not allow this state transition.
ResultCodeExtended = OPOS_EFPTR_JRN_EMPTY:	The journal station is out of paper.
ResultCodeExtended = OPOS_EFPTR_REC_EMPTY:	The receipt station is out of paper.
ResultCodeExtended = OPOS_EFPTR_BAD_DATE:	One of the date parameters is invalid.
Other Values	See **ResultCode**.

PrintPowerLossReport Method

Syntax **LONG PrintPowerLossReport ();**

Remarks Called to print on the receipt a report of a power failure that resulted in a loss of data stored in the CMOS of the printer.

This method is only supported if **CapPowerLossReport** is TRUE.

This method is always performed synchronously.

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.
OPOS_E_ILLEGAL	The printer does not support power loss reports (see the **CapPowerLossReport** property).

OPOS_E_EXTENDED:

ResultCodeExtended = OPOS_EFPTR_WRONG_STATE:
The printer' s current state does not allow this state transition.

ResultCodeExtended = OPOS_EFPTR_COVER_OPEN:
The printer cover is open.

ResultCodeExtended = OPOS_EFPTR_JRN_EMPTY:
The journal station is out of paper.

ResultCodeExtended = OPOS_EFPTR_REC_EMPTY:
The receipt station is out of paper.

Other Values	See **ResultCode**.

PrintRecItem Method

Syntax **LONG PrintRecItem (BSTR** *Description***, CURRENCY** *Price***, LONG** *Quantity***, LONG** *VatInfo***, CURRENCY** *UnitPrice***, BSTR** *UnitName***)**

Parameter	**Description**
Description	Text describing the item sold.
Price	Price of the line item.
Quantity	Number of items. If zero, a single item is assumed.
VatInfo	VAT rate identifier or amount. If not used a zero is to be transferred.
UnitPrice	Price of each item. If not used a zero is to be transferred.
UnitName	Name of the unit i.e. “kg” or “ltr” or “pcs”. If not used an empty string (““) is to be transferred

Remarks Called to print a receipt item for a sold item. If the *Quantity* parameter is 0, then a single item quantity will be assumed.

Minimum parameters are *Description* and *Price* or *Description*, *Price*, *Quantity*, and *UnitPrice*. Most countries require *Quantity* and *VatInfo* and some countries also require *UnitPrice* and *UnitName*.

VatInfo contains a VAT table identifier if **CapHasVatTable** is TRUE. Otherwise it contains a VAT amount.

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_BUSY	Cannot perform while output is in progress. (Can only be returned if **AsyncMode** is FALSE.)
OPOS_E_EXTENDED:	

ResultCodeExtended = OPOS_EFPTR_WRONG_STATE:
The printer is not currently in the Fiscal Receipt state.

ResultCodeExtended = OPOS_EFPTR_COVER_OPEN:
The printer cover is open.
(Can only be returned if **AsyncMode** is FALSE.)

ResultCodeExtended = OPOS_EFPTR_JRN_EMPTY:
The journal station is out of paper.
(Can only be returned if **AsyncMode** is FALSE.)

ResultCodeExtended = OPOS_EFPTR_REC_EMPTY:
The receipt station is out of paper.
(Can only be returned if **AsyncMode** is FALSE.)

ResultCodeExtended = OPOS_EFPTR_BAD_ITEM_QUANTITY:
The quantity is invalid.
(Can only be returned if **AsyncMode** is FALSE.)

ResultCodeExtended = OPOS_EFPTR_BAD_PRICE:
The unit price is invalid.
(Can only be returned if **AsyncMode** is FALSE.)

ResultCodeExtended = OPOS_EFPTR_BAD_ITEM_DESCRIPTION:
The discount description is too long or contains a reserved word.
(Can only be returned if **AsyncMode** is FALSE.)

ResultCodeExtended = OPOS_EFPTR_BAD_VAT:
The VAT parameter is invalid.
(Can only be returned if **AsyncMode** is FALSE.)

ResultCodeExtended = OPOS_EFPTR_RECEIPT_TOTAL_OVERFLOW:
The receipt total has overflowed.
(Can only be returned if **AsyncMode** is FALSE.)

Other Values	See **ResultCode**.

See Also **BeginFiscalReceipt** Method, **EndFiscalReceipt** Method, **PrintRec…** Methods, **AmountDecimalPlaces** Property

PrintRecItemAdjustment Method

Syntax **LONG PrintRecItemAdjustment (LONG** *AdjustmentType*, **BSTR** *Description*, **CURRENCY** *Amount*, **LONG** *VatInfo***);**

Parameter	Description
AdjustmentType	Type of discount. See below for values.
Description	Text describing the discount.
Amount	Amount of the discount.
VatInfo	VAT rate identifier or amount.

AdjustmentType can have the following values:

Value	Meaning
FPTR_AT_AMOUNT_DISCOUNT	Fixed amount discount. The *Amount* parameter contains a currency value.
FPTR_AT_AMOUNT_SURCHARGE	Fixed amount surcharge. The *Amount* parameter contains a currency value.
FPTR_AT_PERCENTAGE_DISCOUNT	Percentage discount. The *Amount* parameter contains a percentage value.
FPTR_AT_PERCENTAGE_SURCHARGE	Percentage surcharge. The *Amount* parameter contains a percentage value.

Remarks

Called to apply and print a discount or a surcharge to the last receipt item sold. This discount may be either a fixed currency amount or a percentage amount relating to the last item.

If **CapOrderAdjustmentFirst** is true, the method must be called before the corresponding **PrintRecItem** method.

If **CapOrderAdjustmentFirst** is false, the method must be called after the **PrintRecItem.**

This discount/surcharge may be either a fixed currency amount or a percentage amount relating to the last item.

If the discount amount is greater than the receipt subtotal, an error occurs since the subtotal can never be negative.

In many countries discount operations cause the printing of a fixed line of text expressing the kind of operation that has been perform

VatInfo contains a VAT table identifier if **CapHasVatTable** is TRUE. Otherwise it contains a VAT amount.

Fixed amount discounts/surcharges are only supported if **CapAmountAdjustment** is TRUE.

Percentage discounts are only supported if **CapPercentAdjustment** is TRUE.

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_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: • The printer does not support fixed amount adjustments (see the **CapAmountAdjustment** property). • The printer does not support percentage discounts (see the **CapPercentAdjustment** property). • The *AdjustmentType* parameter is invalid.

OPOS_E_EXTENDED:

ResultCodeExtended = OPOS_EFPTR_WRONG_STATE:
The printer is not currently in the Fiscal Receipt state.

ResultCodeExtended = OPOS_EFPTR_COVER_OPEN:
The printer cover is open.
(Can only be returned if **AsyncMode** is FALSE.)

ResultCodeExtended = OPOS_EFPTR_JRN_EMPTY:
The journal station is out of paper.
(Can only be returned if **AsyncMode** is FALSE.)

ResultCodeExtended = OPOS_EFPTR_REC_EMPTY:
The receipt station is out of paper.
(Can only be returned if **AsyncMode** is FALSE.)

ResultCodeExtended = OPOS_EFPTR_BAD_ITEM_AMOUNT:
The discount amount is invalid.
(Can only be returned if **AsyncMode** is FALSE.)

ResultCodeExtended = OPOS_EFPTR_BAD_ITEM_DESCRIPTION:
The discount description is too long or contains a reserved word.
(Can only be returned if **AsyncMode** is FALSE.)

ResultCodeExtended = OPOS_EFPTR_BAD_VAT:
The VAT parameter is invalid.
(Can only be returned if **AsyncMode** is FALSE.)

Other Values See **ResultCode**.

See Also **BeginFiscalReceipt** Method, **EndFiscalReceipt** Method, **PrintRec…** Methods, **AmountDecimalPlaces** Property

PrintRecMessage Method

Syntax **LONG PrintRecMessage (BSTR** *Message***);**

Parameter	**Description**
Message	Text message to print.

Remarks Called to print a message on the fiscal receipt. The length of an individual message is limited to the number of characters given in the **MessageLength** property.

This method is only supported if **CapAdditionalLines** is TRUE.

This method is only supported when the printer is in the Fiscal Receipt Ending state.

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_BUSY	Cannot perform while output is in progress. (Can only be returned if **AsyncMode** is FALSE.)
OPOS_E_EXTENDED:	
ResultCodeExtended = OPOS_EFPTR_WRONG_STATE:	The printer is not currently in the Fiscal Receipt Ending state.
ResultCodeExtended = OPOS_EFPTR_COVER_OPEN:	The printer cover is open. (Can only be returned if **AsyncMode** is FALSE.)
ResultCodeExtended = OPOS_EFPTR_JRN_EMPTY:	The journal station is out of paper. (Can only be returned if **AsyncMode** is FALSE.)
ResultCodeExtended = OPOS_EFPTR_REC_EMPTY:	The receipt station is out of paper. (Can only be returned if **AsyncMode** is FALSE.)
ResultCodeExtended = OPOS_EFPTR_BAD_ITEM_DESCRIPTION:	The message is too long or contains a reserved word. (Can only be returned if **AsyncMode** is FALSE.)
Other Values	See **ResultCode**.

See Also **BeginFiscalReceipt** Method, **EndFiscalReceipt** Method, **PrintRec…** Methods, **MessageLength** property, **CapAdditionalLines** property

PrintRecNotPaid Method

Syntax **LONG PrintRecNotPaid (BSTR** *Description*, **CURRENCY** *Amount***);**

Parameter	Description
Description	Text describing the not paid amount.
Amount	Amount not paid.

Remarks Called to indicate that part of the receipt' s total was not paid.

Some fixed text, along with the *Description*, will be printed on the receipt and journal to indicate that part of the receipt total has not been paid.

This method is only supported if **CapAmountNotPaid** is TRUE.

This method is performed synchronously if **AsyncMode** is FALSE, and asynchronously if **AsyncMode** is TRUE.

If this method is successful, the **PrinterState** property will be changed to either FPTR_PS_FISCAL_RECEIPT_TOTAL or FPTR_PS_FISCAL_RECEIPT_ENDING depending upon whether the entire receipt total is now accounted for or not.

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_EXTENDED:	
ResultCodeExtended = OPOS_EFPTR_WRONG_STATE:	The printer is not currently in either the Fiscal Receipt or Fiscal Receipt Total state.
ResultCodeExtended = OPOS_EFPTR_COVER_OPEN:	The printer cover is open. (Can only be returned if **AsyncMode** is FALSE.)
ResultCodeExtended = OPOS_EFPTR_JRN_EMPTY:	The journal station is out of paper. (Can only be returned if **AsyncMode** is FALSE.)
ResultCodeExtended = OPOS_EFPTR_REC_EMPTY:	The receipt station is out of paper. (Can only be returned if **AsyncMode** is FALSE.)
ResultCodeExtended = OPOS_EFPTR_BAD_ITEM_DESCRIPTION:	The description is too long or contains a reserved word. (Can only be returned if **AsyncMode** is FALSE.)
ResultCodeExtended = OPOS_EFPTR_BAD_ITEM_AMOUNT:	The amount is invalid. (Can only be returned if **AsyncMode** is FALSE.)
Other Values	See **ResultCode**.

See Also **BeginFiscalReceipt** Method, **EndFiscalReceipt** Method, **PrintRec…** Methods, **AmountDecimalPlaces** Property

PrintRecRefund Method

Syntax **LONG PrintRecRefund (BSTR** *Description*, **CURRENCY** *Amount*, **LONG** *VatInfo***);**

Parameter	Description
Description	Text describing the refund.
Amount	Amount of the refund.
VatInfo	VAT rate identifier or amount.

Remarks Called to process a refund. The *Amount* is positive, but it is printed as a negative number and the totals registers are decremented.

Some fixed text, along with the *Description*, will be printed on the receipt and journal to indicate that a refund has occurred.

VatInfo contains a VAT table identifier if **CapHasVatTable** is TRUE. Otherwise it contains a VAT amount.

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_BUSY	Cannot perform while output is in progress. (Can only be returned if **AsyncMode** is FALSE.)
OPOS_E_EXTENDED:	
ResultCodeExtended = OPOS_EFPTR_WRONG_STATE:	The printer is not currently in the Fiscal Receipt state.
ResultCodeExtended = OPOS_EFPTR_COVER_OPEN:	The printer cover is open. (Can only be returned if **AsyncMode** is FALSE.)
ResultCodeExtended = OPOS_EFPTR_JRN_EMPTY:	The journal station is out of paper. (Can only be returned if **AsyncMode** is FALSE.)
ResultCodeExtended = OPOS_EFPTR_REC_EMPTY:	The receipt station is out of paper. (Can only be returned if **AsyncMode** is FALSE.)
ResultCodeExtended = OPOS_EFPTR_BAD_ITEM_DESCRIPTION:	The description is too long or contains a reserved word. (Can only be returned if **AsyncMode** is FALSE.)
ResultCodeExtended = OPOS_EFPTR_BAD_ITEM_AMOUNT:	The amount is invalid. (Can only be returned if **AsyncMode** is FALSE.)
ResultCodeExtended = OPOS_EFPTR_BAD_VAT:	The VAT information is invalid. (Can only be returned if **AsyncMode** is FALSE.)
Other Values	See **ResultCode**.

See Also **BeginFiscalReceipt** Method, **EndFiscalReceipt** Method, **PrintRec…** Methods, **AmountDecimalPlaces** Property

PrintRecSubtotal Method

Syntax **LONG PrintRecSubtotal (CURRENCY** *Amount***);**

Parameter	**Description**
Amount	Amount of the subtotal.

Remarks Called to check and print the current receipt subtotal. If **CapCheckTotal** is TRUE, the *Amount* is compared to the subtotal calculated by the printer. If the subtotals match, the subtotal is printed on both the receipt and journal. If the results do not match, the receipt is automatically canceled. If **CapCheckTotal** is FALSE, then the subtotal is printed on the receipt and journal and the parameter is never compared to the subtotal computed by the printer.

This method is performed synchronously if **AsyncMode** is FALSE, and asynchronously if **AsyncMode** is TRUE.

If this method compares the application' s subtotal with the printer' s subtotal and they do not match, the **PrinterState** property will be changed to FPTR_PS_FISCAL_RECEIPT_ENDING.

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_EXTENDED:	
ResultCodeExtended = OPOS_EFPTR_WRONG_STATE:	The printer is not currently in the Fiscal Receipt state.
ResultCodeExtended = OPOS_EFPTR_COVER_OPEN:	The printer cover is open. (Can only be returned if **AsyncMode** is FALSE.)
ResultCodeExtended = OPOS_EFPTR_JRN_EMPTY:	The journal station is out of paper. (Can only be returned if **AsyncMode** is FALSE.)
ResultCodeExtended = OPOS_EFPTR_REC_EMPTY:	The receipt station is out of paper. (Can only be returned if **AsyncMode** is FALSE.)
ResultCodeExtended = OPOS_EFPTR_BAD_ITEM_AMOUNT:	The subtotal from the application does not match the subtotal computed by the printer. (Can only be returned if **AsyncMode** is FALSE.)
ResultCodeExtended = OPOS_EFPTR_NEGATIVE_TOTAL:	The total computed by the printer is less than zero. (Can only be returned if **AsyncMode** is FALSE.)
Other Values	See **ResultCode**.

See Also **BeginFiscalReceipt** Method, **EndFiscalReceipt** Method, **PrintRec…** Methods, **AmountDecimalPlaces** Property

PrintRecSubtotalAdjustment Method

Syntax **LONG PrintRecSubtotalAdjustment (LONG** *AdjustmentType*, **BSTR** *Description*, **CURRENCY** *Amount*)**;**

Parameter	Description
AdjustmentType	Type of discount. See below for values.
Description	Text describing the discount.
Amount	Amount of the discount.

AdjustmentType can have the following values:

Value	Meaning
FPTR_AT_AMOUNT_DISCOUNT	Fixed amount discount. The *Amount* parameter contains a currency value.
FPTR_AT_AMOUNT_SURCHARGE	Fixed amount surcharge. The *Amount* parameter contains a currency value.
FPTR_AT_PERCENTAGE_DISCOUNT	Percentage discount. The *Amount* parameter contains a percentage value.
FPTR_AT_PERCENTAGE_SURCHARGE	Percentage surcharge. The *Amount* parameter contains a percentage value.

Remarks Called to apply and print a discount/surcharge to the current receipt subtotal. This discount/surcharge may be either a fixed currency amount or a percentage amount relating to the current receipt subtotal.

If the discount/surcharge amount is greater than the receipt subtotal, an error occurs since the subtotal can never be negative.

In many countries discount/surcharge operations cause the printing of a fixed line of text expressing the kind of operation that has been performed.

Fixed amount discounts are only supported if **CapSubAmountAdjustment** is TRUE.

Percentage discounts are only supported if **CapSubPercentAdjustment** is TRUE.

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_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: • The printer does not support fixed amount discounts (see the **CapSubAmountAdjustment** property). • The printer does not support percentage discounts (see the **CapSubPercentAdjustment** property). • The *AdjustmentType* parameter is invalid.

OPOS_E_EXTENDED:

ResultCodeExtended = OPOS_EFPTR_WRONG_STATE:
The printer is not currently in the Fiscal Receipt state.

ResultCodeExtended = OPOS_EFPTR_COVER_OPEN:
The printer cover is open.
(Can only be returned if **AsyncMode** is FALSE.)

ResultCodeExtended = OPOS_EFPTR_JRN_EMPTY:
The journal station is out of paper.
(Can only be returned if **AsyncMode** is FALSE.)

ResultCodeExtended = OPOS_EFPTR_REC_EMPTY:
The receipt station is out of paper.
(Can only be returned if **AsyncMode** is FALSE.)

ResultCodeExtended = OPOS_EFPTR_BAD_ITEM_AMOUNT:
The discount amount is invalid.
(Can only be returned if **AsyncMode** is FALSE.)

ResultCodeExtended = OPOS_EFPTR_BAD_ITEM_DESCRIPTION:
The discount description is too long or contains a reserved word.
(Can only be returned if **AsyncMode** is FALSE.)

Other Values See **ResultCode**.

See Also **BeginFiscalReceipt** Method, **EndFiscalReceipt** Method, **PrintRec…** Methods, **AmountDecimalPlaces** Property

PrintRecTotal Method

Syntax **LONG PrintRecTotal (CURRENCY** *Total*, **CURRENCY** *Payment*, **BSTR** *Description***);**

Parameter	**Description**
Total	Application computed receipt total.
Payment	Amount of payment tendered.
Description	Text description of the payment or the index of a predefined payment description.

Remarks Called to check and print the current receipt total and to tender a payment. If **CapCheckTotal** is TRUE, the *Total* is compared to the total calculated by the printer. If the totals match, the total is printed on both the receipt and journal along with some fixed text. If the results do not match, the receipt is automatically canceled. If **CapCheckTotal** is FALSE, then the total is printed on the receipt and journal and the parameter is never compared to the total computed by the printer.

If **CapPredefinedPaymentLines** is TRUE, then the *Description* parameter contains the index of one of the printer' s predefined payment descriptions. The index is typically a single character of the alphabet. The set of allowed values for this index is to be described in the description of the service object and stored in the **PredefinedPaymentLines** property.

If *Payment* = *Total*, a line containing the *Description* and *Payment* is printed. The **PrinterState** property will be set to FPTR_PS_FISCAL_RECEIPT_ENDING.

If *Payment* > *Total*, a line containing the *Description* and *Payment* is printed followed by a second line containing the change due. The **PrinterState** property will be set to FPTR_PS_FISCAL_RECEIPT_ENDING.

If *Payment* < *Total*, a line containing the *Description* and *Payment* is printed. Since the entire receipt total has not yet been tendered, the **PrinterState** property will be set to FPTR_PS_FISCAL_RECEIPT_TOTAL.

If **CapAdditionalLines** property is FALSE, then receipt trailer lines, fiscal logotype and receipt cut are executed after the last total line, whenever receipt' s total became equal to the payment from the application. Otherwise these lines are printed calling the **EndFiscalReceipt** method.

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_BUSY	Cannot perform while output is in progress. (Can only be returned if **AsyncMode** is FALSE.)

OPOS_E_EXTENDED:

ResultCodeExtended = OPOS_EFPTR_WRONG_STATE:
The printer is not currently in the Fiscal Receipt state.

ResultCodeExtended = OPOS_EFPTR_COVER_OPEN:
The printer cover is open.
(Can only be returned if **AsyncMode** is FALSE.)

ResultCodeExtended = OPOS_EFPTR_JRN_EMPTY:
The journal station is out of paper.
(Can only be returned if **AsyncMode** is FALSE.)

ResultCodeExtended = OPOS_EFPTR_REC_EMPTY:
The receipt station is out of paper.
(Can only be returned if **AsyncMode** is FALSE.)

ResultCodeExtended = OPOS_EFPTR_BAD_ITEM_AMOUNT:
One of the following errors occurred:

- The application computed total does not match the printer computed total.
- The *Total* parameter is invalid.
- The *Payment* parameter is invalid
  (Can only be returned if **AsyncMode** is FALSE.)

ResultCodeExtended = OPOS_EFPTR_BAD_ITEM_DESCRIPTION:
The description is too long or contains a reserved word.
(Can only be returned if **AsyncMode** is FALSE.)

ResultCodeExtended = OPOS_EFPTR_NEGATIVE_TOTAL:
The total computed by the printer is less than zero.
(Can only be returned if **AsyncMode** is FALSE.)

ResultCodeExtended = OPOS_EFPTR_WORD_NOT_ALLOWED:
The description contains the reserved word

Other Values See **ResultCode**.

See Also **BeginFiscalReceipt** Method, **EndFiscalReceipt** Method, **PrintRec…** Methods, **PredefinedPaymentLines** property, **AmountDecimalPlaces** Property

PrintRecVoid Method

Syntax **LONG PrintRecVoid (BSTR** *Description***);**

Parameter	Description
Description	Text describing the void.

Remarks Called to cancel the current receipt. The receipt is annulled but it is not physically canceled from the printer’ s fiscal memory since fiscal receipts are printed with an increasing serial number and totals are accumulated in registers. When a receipt is canceled, its subtotal is subtracted from the totals registers, but it is added to the canceled receipt register.

Some fixed text, along with the *Description*, will be printed on the receipt and journal to indicate that the receipt has been canceled.

This method is performed synchronously if **AsyncMode** is FALSE, and asynchronously if **AsyncMode** is TRUE.

If this method is successful, the **PrinterState** property will be changed to FPTR_PS_FISCAL_RECEIPT_ENDING.

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_EXTENDED:	
ResultCodeExtended = OPOS_EFPTR_WRONG_STATE:	The printer is not currently in the Fiscal Receipt state.
ResultCodeExtended = OPOS_EFPTR_COVER_OPEN:	The printer cover is open. (Can only be returned if **AsyncMode** is FALSE.)
ResultCodeExtended = OPOS_EFPTR_JRN_EMPTY:	The journal station is out of paper. (Can only be returned if **AsyncMode** is FALSE.)
ResultCodeExtended = OPOS_EFPTR_REC_EMPTY:	The receipt station is out of paper. (Can only be returned if **AsyncMode** is FALSE.)
ResultCodeExtended = OPOS_EFPTR_BAD_ITEM_DESCRIPTION:	The description is too long or contains a reserved word. (Can only be returned if **AsyncMode** is FALSE.)
Other Values	See **ResultCode**.

See Also **BeginFiscalReceipt** Method, **EndFiscalReceipt** Method, **PrintRec…** Methods

PrintRecVoidItem Method

Syntax **LONG PrintRecVoidItem (BSTR** *Description*, **CURRENCY** *Amount*, **LONG** *Quantity*, **LONG** *AdjustmentType*, **CURRENCY** *Adjustment*, **LONG** *VatInfo***);**

Parameter	**Description**
Description	Text description of the item void.
Amount	Amount of item to be voided.
Quantity	Quantity of item to be voided.
AdjustmentType	Type of discount. See below for values.
Adjustment	Amount of the discount/surcharge
VatInfo	VAT rate identifier or amount.

AdjustmentType can have the following values:

Value	**Meaning**
FPTR_AT_AMOUNT_DISCOUNT	Fixed amount discount. The *Adjustment* parameter contains a currency value.
FPTR_AT_AMOUNT_SURCHARGE	Fixed amount surcharge. The *Adjustment* parameter contains a currency value.
FPTR_AT_PERCENTAGE_DISCOUNT	Percentage discount. The *Adjustment* parameter contains a percentage value.
FPTR_AT_PERCENTAGE_SURCHARGE	Percentage surcharge. The *Adjustment* parameter contains a percentage value.

Remarks Called to cancel an item that has been added to the receipt and print a void description. *Amount* is a positive number, it will be printed as a negative and will be decremented from the totals registers.

VatInfo contains a VAT table identifier if **CapHasVatTable** is TRUE. Otherwise it contains a VAT amount.

Fixed amount discounts/surcharges are only supported if **CapAmountAdjustment** is TRUE.

Percentage discounts are only supported if **CapPercentAdjustment** is TRUE.

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_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: • The printer does not support fixed amount adjustments (see the **CapAmountAdjustment** property). • The printer does not support percentage discounts (see the **CapPercentAdjustment** property). • The *AdjustmentType* parameter is invalid.

OPOS_E_EXTENDED:

ResultCodeExtended = OPOS_EFPTR_WRONG_STATE:
The printer is not currently in the Fiscal Receipt state.

ResultCodeExtended = OPOS_EFPTR_COVER_OPEN:
The printer cover is open.
(Can only be returned if **AsyncMode** is FALSE.)

ResultCodeExtended = OPOS_EFPTR_JRN_EMPTY:
The journal station is out of paper.
(Can only be returned if **AsyncMode** is FALSE.)

ResultCodeExtended = OPOS_EFPTR_REC_EMPTY:
The receipt station is out of paper.
(Can only be returned if **AsyncMode** is FALSE.)

ResultCodeExtended = OPOS_EFPTR_BAD_ITEM_AMOUNT:
The amount is invalid.
(Can only be returned if **AsyncMode** is FALSE.)

ResultCodeExtended = OPOS_EFPTR_BAD_ITEM_QUANTITY:
The quantity is invalid.
(Can only be returned if **AsyncMode** is FALSE.)

ResultCodeExtended = OPOS_EFPTR_BAD_VAT:
The VAT information is invalid.
(Can only be returned if **AsyncMode** is FALSE.)

ResultCodeExtended = OPOS_EFPTR_BAD_ITEM_DESCRIPTION:
The description is too long or contains a reserved word.
(Can only be returned if **AsyncMode** is FALSE.)

ResultCodeExtended = OPOS_EFPTR_NEGATIVE_TOTAL:
The total computed by the printer is less than zero.
(Can only be returned if **AsyncMode** is FALSE.)

Other Values See **ResultCode**.

See Also **BeginFiscalReceipt** Method, **EndFiscalReceipt** Method, **PrintRec…** Methods, **AmountDecimalPlaces** Property

PrintReport Method

Syntax **LONG PrintReport (LONG** *ReportType***, BSTR** *StartNum***, BSTR** *EndNum***);**

Parameter	Description
ReportType	The kind of report to print.
StartNum	ASCII string identifying the starting record in printer memory from which to begin printing
EndNum	ASCII string identifying the final record in printer memory at which printing is to end. See *ReportType* table below to find out the exact meaning of this parameter.

ReportType can have the following values:

Value	Meaning
FPTR_RT_ORDINAL	Prints a report between two Z report. If both *StartNum* and *EndNum* are valid and *EndNum* > *StartNum*, then a report of the period between *StartNum* and *EndNum* will be printed. If *StartNum* is valid and *EndNum* is 0, then a report of relating only to *StartNum* will be printed.
FPTR_RT_DATE	Prints a report between two dates. The dates are strings in the format “ddmmyyyyhhmm”, where: dd day of the month (01 - 31) mm month (01 - 12) yyyy year (1997- ...) hh hour (00-23) mm minutes (00-59)

Remarks Called to print a report of the fiscal EPROM contents on the receipt that occurred between two end points.

This method is always performed synchronously.

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.
OPOS_E_ILLEGAL	One of the following errors occurred:
	* The *ReportType* parameter is invalid.
	* One or both of *StartNum* and *EndNum* are invalid.
	* *StartNum* > *EndNum.*

OPOS_E_EXTENDED:

ResultCodeExtended = OPOS_EFPTR_WRONG_STATE:
The printer's current state does not allow this state transition.

ResultCodeExtended = OPOS_EFPTR_COVER_OPEN:
The printer cover is open.

ResultCodeExtended = OPOS_EFPTR_JRN_EMPTY:
The journal station is out of paper.

ResultCodeExtended = OPOS_EFPTR_REC_EMPTY:
The receipt station is out of paper.

Other Values	See **ResultCode**.

PrintXReport Method

Syntax **LONG PrintXReport ();**

Remarks Called to print on the receipt a report of all the daily fiscal activities. No data will be written to the fiscal EPROM as a result of this method invocation.

This method is only supported if **CapXReport** is TRUE.

This method is always performed synchronously.

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.
OPOS_E_ILLEGAL	The printer does not support X reports (see the **CapXReport** property).

OPOS_E_EXTENDED:

ResultCodeExtended = OPOS_EFPTR_WRONG_STATE:
The printer' s current state does not allow this state transition.

ResultCodeExtended = OPOS_EFPTR_COVER_OPEN:
The printer cover is open.

ResultCodeExtended = OPOS_EFPTR_JRN_EMPTY:
The journal station is out of paper.

ResultCodeExtended = OPOS_EFPTR_REC_EMPTY:
The receipt station is out of paper.

Other Values	See **ResultCode**.

PrintZReport Method

Syntax **LONG PrintZReport ();**

Remarks Called to print on the receipt a report of all the daily fiscal activities. Data will be written to the fiscal EPROM as a result of this method invocation.

This method is always performed synchronously.

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.
OPOS_E_EXTENDED:	
ResultCodeExtended = OPOS_EFPTR_WRONG_STATE:	The printer' s current state does not allow this state transition.
ResultCodeExtended = OPOS_EFPTR_COVER_OPEN:	The printer cover is open.
ResultCodeExtended = OPOS_EFPTR_JRN_EMPTY:	The journal station is out of paper.
ResultCodeExtended = OPOS_EFPTR_REC_EMPTY:	The receipt station is out of paper.
Other Values	See **ResultCode**.

ResetPrinter Method

Syntax **LONG ResetPrinter ();**

Remarks Called to force the printer to return to Monitor state. This forces any interrupted operations to be canceled and closed. This method must be invoked when the printer is not in a Monitor state after a successful call to the **Claim** method and successful setting of the **DeviceEnabled** property to TRUE. This typically happens if a power failures occurs during a fiscal operation.

Calling this method does not close the printer, i.e. does not force a Z report to be printed.

The Service Object will handle this command as follows:

- If the printer was in either Fiscal Receipt, Fiscal Receipt Total or Fiscal Receipt Ending state, the receipt will be ended without updating any registers.
- If the printer was in a non-fiscal state, the printer will exit that state.
- If the printer was in the training state, the printer will exit the training state.

This method is always performed synchronously.

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

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

SetDate Method

Syntax **LONG SetDate (BSTR** *Date***);**

Parameter	Description
Date	Date and time as a string.

Remarks Called to set the printer’ s date and time.

The date and time is passed as a string in the format “ddmmyyyyhhmm”, where:

dd	day of the month (1 - 31)
mm	month (1 - 12)
yyyy	year (1997-)
hh	hour (0-23)
mm	minutes (0-59)

This method can only be called while **DayOpened** 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 printer has already begun the fiscal day (see the **DayOpened** property).
OPOS_E_EXTENDED:	
ResultCodeExtended =	OPOS_EFPTR_BAD_DATE: One of the date parameters is invalid.
Other Values	See **ResultCode**.

SetHeaderLine Method

Syntax **LONG SetHeaderLine (LONG** *LineNumber*, **BSTR** *Text*, **BOOL** *DoubleWidth***);**

Parameter	Description
LineNumber	Line number of the header line to set.
Text	Text to which to set the header line.
DoubleWidth	Print this line in double wide characters.

Remarks Called to set one of the fiscal receipt header lines. The text set by this method will be stored by the printer and retained across power losses.

LineNumber must be between 1 and the value of the **NumHeaderLines** property.

If *Text* is an empty string ("“), then the header line is unset and will not be printed.

DoubleWidth characters will be printed if the printer supports them. See the **CapDoubleWidth** property to determine if they are supported.

This method is only supported if **CapSetHeader** is TRUE.

This method can only be called while **DayOpened** 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	One of the following errors occurred: • The printer has already begun the fiscal day (see the **DayOpened** property). • The *LineNumber* parameter was invalid.

OPOS_E_EXTENDED:

ResultCodeExtended = OPOS_EFPTR_BAD_ITEM_DESCRIPTION:
The *Text* parameter is too long or contains a reserved word.
(Can only be returned if **AsyncMode** is FALSE.)

Other Values	See **ResultCode**.

SetPOSID Method

Syntax **LONG SetPOSID (BSTR** *POSID*, **BSTR** *CashierID***);**

Parameter	Description
POSID	Identifier for the POS system.
CashierID	Identifier of the current cashier.

Remarks Called to set the POS and cashier identifiers. These values will be printed when each fiscal receipt is closed.

This method is only supported if **CapSetPOSID** is TRUE.

This method can only be called while **DayOpened** 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	One of the following error occurred: • The printer does not support setting the POS identifier (see the **CapSetPOSID** property). • The printer has already begun the fiscal day (see the **DayOpened** property). • Either the *POSID* or *CashierID* parameter is invalid.
Other Values	See **ResultCode**.

SetStoreFiscalID Method

Syntax **LONG SetStoreFiscalID (BSTR *ID*);**

Parameter	Description
ID	Fiscal identifier.

Remarks Called to set the store fiscal ID. This value is retained by the printer even after power failures. This ID is automatically printed by the printer after the fiscal receipt header lines.

This method is only supported if **CapSetStoreFiscalID** is TRUE.

This method can only be called while **DayOpened** 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	One of the following errors occurred: • The printer does not support setting the store fiscal identifier (see the **CapSetStoreFiscalID** property). • The printer has already begun the fiscal day (see the **DayOpened** property). • The *ID* parameter was invalid.
Other Values	See **ResultCode**.

SetTrailerLine Method

Syntax **LONG SetTrailerLine (LONG** *LineNumber*, **BSTR** *Text*, **BOOL** *DoubleWidth***);**

Parameter	Description
LineNumber	Line number of the trailer line to set.
Text	Text to which to set the trailer line.
DoubleWidth	Print this line in double wide characters.

Remarks Called to set one of the fiscal receipt trailer lines. The text set by this method will be stored by the printer and retained across power losses.

LineNumber must be between 1 and the value of the **NumTrailerLines** property.

If *Text* is an empty string (" "), then the trailer line is unset and will not be printed.

DoubleWidth characters will be printed if the printer supports them. See the **CapDoubleWidth** property to determine if they are supported.

This method is only supported if **CapSetTrailer** is TRUE.

This method can only be called while **DayOpened** 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	One of the following errors occurred: • The printer has already begun the fiscal day (see the **DayOpened** property). • The *LineNumber* parameter was invalid.
OPOS_E_EXTENDED:	
ResultCodeExtended = OPOS_EFPTR_BAD_ITEM_DESCRIPTION:	The *Text* parameter is too long or contains a reserved word. (Can only be returned if **AsyncMode** is FALSE.)
Other Values	See **ResultCode**.

SetVatTable Method

Syntax **LONG SetVatTable ();**

Remarks Called to send the VAT table built inside the Service Object to the printer. The VAT table is built one entry at a time using the **SetVatValue** method.

This method is only supported if **CapHasVatTable** is TRUE.

This method can only be called while **DayOpened** 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 printer has already begun the fiscal day (see the **DayOpened** property).
Other Values	See **ResultCode**.

See Also **SetVatValue** Method

SetVatValue Method

Syntax **LONG SetVatValue (LONG** *VatID*, **BSTR** *VatValue***);**

Parameter	Description
VatID	Index of the VAT table entry to set.
VatValue	Tax value as a percentage.

Remarks Called to set the value of a specific VAT class in the VAT table. The VAT table is built one entry at a time in the Service Object using this method. The entire table is then sent to the printer at one time using the **SetVatTable** method.

This method is only supported if **CapHasVatTable** is TRUE.

This method can only be called while **DayOpened** 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	One of the following errors occurred: • The printer does not support VAT tables (see the **CapHasVatTable** property). • The printer has already begun the fiscal day (see the **DayOpened** property). • The printer does not support changing an existing VAT value.
Other Values	See **ResultCode**.

See Also **SetVatTable** Method

VerifyItem Method

Syntax **LONG VerifyItem (BSTR** *ItemName*, **LONG** *VatID***);**

Parameter	Description
ItemName	Item to be verified.
VatID	VAT identifier of the item.

Remarks Called to compare *ItemName* and its *VatID* with the values stored in the printer.

This method is only supported if **CapHasVatTable** is TRUE.

This method can only be called while the printer is in the Item List state.

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 printer does not support VAT tables (see the **CapHasVatTable** property).
OPOS_E_EXTENDED:	
ResultCodeExtended = OPOS_EFPTR_WRONG_STATE:	The printer is not currently in the Item List state.
ResultCodeExtended = OPOS_EFPTR_BAD_ITEM_DESCRIPTION:	The item name is too long or contains a reserved word. (Can only be returned if **AsyncMode** is FALSE.)
ResultCodeExtended = OPOS_EFPTR_BAD_VAT:	The VAT parameter is invalid. (Can only be returned if **AsyncMode** is FALSE.)
Other Values	See **ResultCode**.

See Also **SetVatTable** Method

Events

ErrorEvent Event

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

Parameter	**Description**
ResultCode	Result code causing the error event. See **ResultCode** for values.
ResultCodeExtended	Extended result code causing the error event. See values below.
ErrorLocus	Set to OPOS_EL_OUTPUT: Error occurred while processing asynchronous output.
pErrorResponse	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_EFPTR_COVER_OPEN	The printer cover is open.
OPOS_EFPTR_JRN_EMPTY	The journal station is out of paper.
OPOS_EFPTR_REC_EMPTY	The receipt station is out of paper.
OPOS_EFPTR_SLP_EMPTY	A form is not inserted in the slip station.
OPOS_EFPTR_WRONG_STATE	The requested method could not be executed in the printer' s current state.
OPOS_EFPTR_TECHNICAL_ASSISTANCE	The printer has encountered a severe error condition. Calling for printer technical assistance is required.
OPOS_EFPTR_CLOCK_ERROR	The printer' s internal clock has failed.
OPOS_EFPTR_FISCAL_MEMORY_FULL	The printer' s fiscal memory has been exhausted.
OPOS_EFPTR_FISCAL_MEMORY_DISCONNECTED	The printer' s fiscal memory has been disconnected.
OPOS_EFPTR_FISCAL_TOTALS_ERROR	The Grand Total in working memory does not match the one in the EPROM.
OPOS_EFPTR_BAD_ITEM_QUANTITY	The Quantity parameter is invalid.
OPOS_EFPTR_BAD_ITEM_AMOUNT	The Amount parameter is invalid.
OPOS_EFPTR_BAD_ITEM_DESCRIPTION	The Description parameters is either to long, contains illegal characters or contains the reserved word.

OPOS_EFPTR_RECEIPT_TOTAL_OVERFLOW	The receipt total has overflowed.
OPOS_EFPTR_BAD_VAT	The Vat parameter is invalid.
OPOS_EFPTR_BAD_PRICE	The Price parameter is invalid.
OPOS_EFPTR_NEGATIVE_TOTAL	The printer' s computed total or subtotal is less than zero.
OPOS_EFPTR_MISSING_DEVICES	Some of the other devices which according to the local fiscal legislation are to be connected has been disconnected. In some countries in order to use a fiscal printer a full set of peripheral devices are to be connected to the POS (such as cash drawer and customer display). In case one of these devices is not present sales are not allowed.

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 transitions into the error state.

See Also Printer Error Model (Page **188**)

StatusUpdateEvent Event

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

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

Value	Meaning
FPTR_SUE_COVER_OPEN	Printer cover is open.
FPTR_SUE_COVER_OK	Printer cover is closed.
FPTR_SUE_JRN_EMPTY	No journal paper.
FPTR_SUE_JRN_NEAREMPTY	Journal paper is low.
FPTR_SUE_JRN_PAPEROK	Journal paper is ready.
FPTR_SUE_REC_EMPTY	No receipt paper.
FPTR_SUE_REC_NEAREMPTY	Receipt paper is low.
FPTR_SUE_REC_PAPEROK	Receipt paper is ready.
FPTR_SUE_SLP_EMPTY	No slip form.
FPTR_SUE_SLP_NEAREMPTY	Almost at the bottom of the slip form.
FPTR_SUE_SLP_PAPEROK	Slip form is inserted.
FPTR_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 delivering the event.

*Power reporting **StatusUpdateEvent** values*

See **StatusUpdateEvent** description on page 68.

Remarks Fired when a significant status event has occurred.

C H A P T E R 8

Hard Totals

Summary

Properties

Common		*Type*	*Access*	*Initialized After*
AutoDisable	1.2	Boolean	R/W	*Not Supported*
BinaryConversion	1.2	Long	R/W	Open
CapPowerReporting	1.3	Long	R	Open
CheckHealthText	1.0	String	R	Open
Claimed	1.0	Boolean	R	Open
DataCount	1.2	Long	R	*Not Supported*
DataEventEnabled	1.0	Boolean	R/W	*Not Supported*
DeviceEnabled	1.0	Boolean	R/W	Open
FreezeEvents	1.0	Boolean	R/W	Open
OutputID	1.0	Long	R	*Not Supported*
PowerNotify	1.3	Long	R/W	Open
PowerState	1.3	Long	R	Open
ResultCode	1.0	Long	R	--
ResultCodeExtended	1.0	Long	R	Open
State	1.0	Long	R	--
ControlObjectDescription	1.0	String	R	--
ControlObjectVersion	1.0	Long	R	--
ServiceObjectDescription	1.0	String	R	Open
ServiceObjectVersion	1.0	Long	R	Open
DeviceDescription	1.0	String	R	Open
DeviceName	1.0	String	R	Open

Specific		*Type*	*Access*	*Initialized After*
CapErrorDetection	1.0	Boolean	R	Open
CapSingleFile	1.0	Boolean	R	Open
CapTransactions	1.0	Boolean	R	Open
FreeData	1.0	Long	R	Open & Enable
TotalsSize	1.0	Long	R	Open & Enable
NumberOfFiles	1.0	Long	R	Open & Enable
TransactionInProgress	1.0	Boolean	R	Open

Methods

Common		*May Use After*
Open	1.0	--
Close	1.0	Open
Claim	1.0	Open
Release	1.0	Open & Claim
CheckHealth	1.0	Open & Enable; *Note 1*
ClearInput	1.0	*Not Supported*
ClearOutput	1.0	*Not Supported*
DirectIO	1.0	Open
Specific		
ClaimFile	1.0	Open & Enable; *Note 2*
ReleaseFile	1.0	Open & Enable
Read	1.0	Open & Enable; *Note 2*
Write	1.0	Open & Enable; *Note 2*
SetAll	1.0	Open & Enable; *Note 2*
ValidateData	1.0	Open & Enable; *Note 2*
RecalculateValidationData	1.0	Open & Enable; *Note 2*
Create	1.0	Open & Enable; *Note 1*
Find	1.0	Open & Enable; *Note 1*
FindByIndex	1.0	Open & Enable; *Note 1*
Delete	1.0	Open & Enable; *Note 2*
Rename	1.0	Open & Enable; *Note 2*
BeginTrans	1.0	Open & Enable
CommitTrans	1.0	Open & Enable
Rollback	1.0	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

Name		*May Occur After*
DataEvent	1.0	*Not Supported*
DirectIOEvent	1.0	Open, Claim
ErrorEvent	1.0	*Not Supported*
OutputCompleteEvent	1.0	*Not Supported*
StatusUpdateEvent	1.3	Open, Claim, & Enable

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.
  - ♦ **SetAll**: Set all the data in a totals file to a value.
  - ♦ **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 **Write**s 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 **Create**d and **Delete**d. 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.
Other Values	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, then returns the appropriate status immediately.
If OPOS_FOREVER (-1), the method waits as long as needed until exclusive access is satisfied.

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_ILLEGAL	An invalid *Timeout* parameter was specified.
OPOS_E_TIMEOUT	Another application has exclusive access to the device or one or more of its files and did not relinquish control before *Timeout* milliseconds expired.

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

ClaimFile Method

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

Parameter	**Description**
HTotalsFile	Handle to the totals file that is to be claimed.
Timeout	The time in milliseconds to wait for the file to become available. If zero, the method attempts to claim the file, then returns the appropriate status immediately. If OPOS_FOREVER (-1), the method waits as long as needed until exclusive access is satisfied.

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, or an invalid *Timeout* parameter was specified.
OPOS_E_TIMEOUT	The *Timeout* 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.
Other Values	See **ResultCode**.

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

Create Method

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

Parameter	**Description**
FileName	The name to be assigned to the file. Must be no longer than 10 characters. All displayable characters – characters ≥ 20-hex – are valid.
pHTotalsFile	Pointer to the handle of the newly created totals file. Set by the method.
Size	The length of the file in bytes. Once created, the file size cannot be changed – totals files are fixed-length files.
ErrorDetection	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 *FileName* is too long or contains invalid characters.
OPOS_E_EXISTS	*FileName* already exists.
OPOS_E_EXTENDED	**ResultCodeExtended** = OPOS_ETOT_NOROOM: There is insufficient room in the totals area to create the file.

Other Values	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 *FileName* is too long or contains invalid characters.
OPOS_E_NOEXIST	*FileName* was not found.
Other Values	See **ResultCode**.

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

Find Method

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

Parameter	Description
FileName	The totals file name to be located.
pHTotalsFile	Pointer to the handle of the totals file. Set by the method.
pSize	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 *FileName* is too long or contains invalid characters.
OPOS_E_NOEXIST	*FileName* was not found.
Other Values	See **ResultCode**.

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

FindByIndex Method

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

Parameter	Description
Index	The index of the totals file name to be found.
pFileName	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 *Index* is greater than the largest file index that is currently defined.
Other Values	See **ResultCode**.

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

Read Method

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

Parameter	Description
HTotalsFile	Totals file handle returned from a **Create** or **Find** method.
pData	Pointer to the data buffer in which the totals data will be placed. The format of this data depends upon the value of the **BinaryConversion** property. See page 37.
Offset	Starting offset for the data to be read.
Count	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.
Other Values	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 Service Object or by this file.
Other Values	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**
HTotalsFile	Handle of the totals file to be renamed.
FileName	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 *FileName* is too long or contains invalid characters, or the **CapSingleFile** property is TRUE.
OPOS_E_EXISTS	*FileName* already exists.
Other Values	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:

Value	Meaning
OPOS_SUCCESS	The method was successful.
OPOS_E_ILLEGAL	Transactions are not supported by this device, or no transaction is in progress.
Other Values	See **ResultCode**.

See Also **BeginTrans** Method; **CommitTrans** Method

SetAll Method

Syntax **LONG SetAll (LONG** *HTotalsFile***, LONG** *Value***);**

Parameter	Description
HTotalsFile	Handle of a totals file.
Value	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:

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.
Other Values	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 Service Object or by this file.
Other Values	See **ResultCode**.

Write Method

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

Parameter	**Description**
HTotalsFile	Totals file handle returned from a **Create** or **Find** method.
Data	Data buffer containing the totals data to be written. The format of this data depends upon the value of the **BinaryConversion** property. See page 37.
Offset	Starting offset for the data to be written.
Count	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.
Other Values	See **ResultCode**.

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

CHAPTER 9

Keylock

Summary

Properties

Common		*Type*	*Access*	*Initialized After*
AutoDisable	1.2	Boolean	R/W	*Not Supported*
BinaryConversion	1.2	Long	R/W	Open
CapPowerReporting	1.3	Long	R	Open
CheckHealthText	1.0	String	R	Open
Claimed	1.0	Boolean	R	Open
DataCount	1.2	Long	R	*Not Supported*
DataEventEnabled	1.0	Boolean	R/W	*Not Supported*
DeviceEnabled	1.0	Boolean	R/W	Open
FreezeEvents	1.0	Boolean	R/W	Open
OutputID	1.0	Long	R	*Not Supported*
PowerNotify	1.3	Long	R/W	Open
PowerState	1.3	Long	R	Open
ResultCode	1.0	Long	R	--
ResultCodeExtended	1.0	Long	R	Open
State	1.0	Long	R	--
ControlObjectDescription	1.0	String	R	--
ControlObjectVersion	1.0	Long	R	--
ServiceObjectDescription	1.0	String	R	Open
ServiceObjectVersion	1.0	Long	R	Open
DeviceDescription	1.0	String	R	Open
DeviceName	1.0	String	R	Open

Specific		*Type*	*Access*	*Initialized After*
KeyPosition	1.0	Long	R	Open & Enable
PositionCount	1.0	Long	R	Open

Methods

Common		*May Use After*
Open	1.0	--
Close	1.0	Open
Claim	1.0	Open
Release	1.0	Open & Claim
CheckHealth	1.0	Open & Enable
ClearInput	1.0	*Not Supported*
ClearOutput	1.0	*Not Supported*
DirectIO	1.0	Open
Specific		
WaitForKeylockChange	1.0	Open & Enable

Events

Name		*May Occur After*
DataEvent	1.0	*Not Supported*
DirectIOEvent	1.0	Open
ErrorEvent	1.0	*Not Supported*
OutputCompleteEvent	1.0	*Not Supported*
StatusUpdateEvent	1.0	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).
Other Values	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
KeyPosition	Requested keylock position. See values below.
Timeout	Maximum number of milliseconds to wait for the keylock before returning control back to the application. If zero, the method then returns the appropriate status immediately. If OPOS_FOREVER (-1), the method waits as long as needed until the requested key position is satisfied or an error occurs.

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).
Other Values	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 *KeyPosition* is LOCK_KP_ANY, then the keylock position has changed.
OPOS_E_ILLEGAL	An invalid parameter value was specified.
OPOS_E_TIMEOUT	The timeout period expired before the requested keylock positioning occurred.
Other Values	See **ResultCode**.

Events

StatusUpdateEvent Event

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

The *Status* 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).
Other Values	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.

*Power reporting **StatusUpdateEvent** values*
See **StatusUpdateEvent** description on page 68.

Remarks Fired when the keylock position changes.

C H A P T E R   1 0

Line Display

Summary

Properties

Common		*Type*	*Access*	*Initialized After*
AutoDisable	1.2	Boolean	R/W	*Not Supported*
BinaryConversion	1.2	Long	R/W	Open
CapPowerReporting	1.3	Long	R	Open
CheckHealthText	1.0	String	R	Open
Claimed	1.0	Boolean	R	Open
DataCount	1.2	Long	R	*Not Supported*
DataEventEnabled	1.0	Boolean	R/W	*Not Supported*
DeviceEnabled	1.0	Boolean	R/W	Open & Claim
FreezeEvents	1.0	Boolean	R/W	Open
OutputID	1.0	Long	R	*Not Supported*
PowerNotify	1.3	Long	R/W	Open
PowerState	1.3	Long	R	Open
ResultCode	1.0	Long	R	--
ResultCodeExtended	1.0	Long	R	Open
State	1.0	Long	R	--
ControlObjectDescription	1.0	String	R	--
ControlObjectVersion	1.0	Long	R	--
ServiceObjectDescription	1.0	String	R	Open
ServiceObjectVersion	1.0	Long	R	Open
DeviceDescription	1.0	String	R	Open
DeviceName	1.0	String	R	Open

Specific		*Type*	*Access*	*Initialized After*
CapBlink	1.0	Long	R	Open
CapBrightness	1.0	Boolean	R	Open
CapCharacterSet	1.0	Long	R	Open
CapDescriptors	1.0	Boolean	R	Open
CapHMarquee	1.0	Boolean	R	Open
CapICharWait	1.0	Boolean	R	Open
CapVMarquee	1.0	Boolean	R	Open
DeviceWindows	1.0	Long	R	Open
DeviceRows	1.0	Long	R	Open
DeviceColumns	1.0	Long	R	Open
DeviceDescriptors	1.0	Long	R	Open
DeviceBrightness	1.0	Long	R/W	Open, Claim, & Enable
CharacterSet	1.0	Long	R/W	Open, Claim, & Enable
CharacterSetList	1.0	String	R	Open
CurrentWindow	1.0	Long	R/W	Open
Rows	1.0	Long	R	Open
Columns	1.0	Long	R	Open
CursorRow	1.0	Long	R/W	Open
CursorColumn	1.0	Long	R/W	Open
CursorUpdate	1.0	Boolean	R/W	Open
MarqueeType	1.0	Long	R/W	Open
MarqueeFormat	1.0	Long	R/W	Open
MarqueeUnitWait	1.0	Long	R/W	Open
MarqueeRepeatWait	1.0	Long	R/W	Open
InterCharacterWait	1.0	Long	R/W	Open

Methods

Common		*May Use After*
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	*Not Supported*
ClearOutput	1.0	*Not Supported*
DirectIO	1.0	Open
Specific		
DisplayText	1.0	Open, Claim, & Enable
DisplayTextAt	1.0	Open, Claim, & Enable
ClearText	1.0	Open, Claim, & Enable
ScrollText	1.0	Open, Claim, & Enable
SetDescriptor	1.0	Open, Claim, & Enable
ClearDescriptors	1.0	Open, Claim, & Enable
CreateWindow	1.0	Open, Claim, & Enable
DestroyWindow	1.0	Open, Claim, & Enable
RefreshWindow	1.0	Open, Claim, & Enable

Events

Name		*May Occur After*
DataEvent	1.0	*Not Supported*
DirectIOEvent	1.0	Open, Claim
ErrorEvent	1.0	*Not Supported*
OutputCompleteEvent	1.0	*Not Supported*
StatusUpdateEvent	1.3	Open, Claim, & Enable

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

- ***<u>Immediate Mode</u>***
  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.
- ***<u>Teletype Mode</u>***
  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.
- ***<u>Marquee Mode</u>***
  In effect when **MarqueeType** is not DISP_MT_NONE.

  The window must be bigger than the viewport.

  A marquee is typically initialized after entering <u>Marquee Init Mode</u> by setting **MarqueeType** to DISP_MT_INIT, then calling **ClearText**, **DisplayText**, and **DisplayTextAt** methods. Then, when **MarqueeType** is changed to an "on" value, <u>Marquee On Mode</u> 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.
Other Values	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. The wait time is the specified number of milliseconds. (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.
OPOS_E_ILLEGAL	An illegal value was specified.

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:

Marquee Type	Window		Viewport
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 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:

Marquee Type	Window		Viewport
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 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.
OPOS_E_ILLEGAL	An illegal value was specified.

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 *Height* 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 *Height* 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 *Width* 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 *Width* 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.
OPOS_E_ILLEGAL	An illegal value was specified.

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.
Other Values	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.
Other Values	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**
ViewportRow	The viewport' s start device row.
ViewportColumn	The viewport' s start device column.
ViewportHeight	The number of device rows in the viewport.
ViewportWidth	The number of device columns in the viewport.
WindowHeight	The number of rows in the window.
WindowWidth	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 *WindowHeight*.
Columns	Set to *WindowWidth*.
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.
Other Values	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.
Other Values	See **ResultCode**.

See Also **CreateWindow** Method; **CurrentWindow** Property

DisplayText Method

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

Parameter	**Description**
Data	The string of characters to display. The format of this data depends upon the value of the **BinaryConversion** property. See page 37.
Attribute	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	*Attribute* is illegal, or the display is in Marquee On Mode.
Other Values	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
Row	The start row for the text.
Column	The start column for the text.
Data	The string of characters to display. The format of this data depends upon the value of the **BinaryConversion** property. See page 37.
Attribute	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	*Row* or *Column* are out or range, *Attribute* is illegal, or in Marquee On Mode.
Other Values	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 redisplays 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	*Window* is larger than **DeviceWindows** or has not been created, or in Marquee On Mode.
Other Values	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	*Direction* is illegal, or in Teletype Mode or Marquee Mode.
Other Values	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.
Other Values	See **ResultCode**.

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

CHAPTER 11

MICR - Magnetic Ink Character Recognition Reader

Summary

Properties

Common		***Type***	***Access***	***Initialized After***
AutoDisable	1.2	Boolean	R/W	Open
BinaryConversion	1.2	Long	R/W	Open
CapPowerReporting	1.3	Long	R	Open
CheckHealthText	1.0	String	R	Open
Claimed	1.0	Boolean	R	Open
DataCount	1.2	Long	R	Open
DataEventEnabled	1.0	Boolean	R/W	Open
DeviceEnabled	1.0	Boolean	R/W	Open & Claim
FreezeEvents	1.0	Boolean	R/W	Open
OutputID	1.0	Long	R	*Not Supported*
PowerNotify	1.3	Long	R/W	Open
PowerState	1.3	Long	R	Open
ResultCode	1.0	Long	R	--
ResultCodeExtended	1.0	Long	R	Open
State	1.0	Long	R	--
ControlObjectDescription	1.0	String	R	--
ControlObjectVersion	1.0	Long	R	--
ServiceObjectDescription	1.0	String	R	Open
ServiceObjectVersion	1.0	Long	R	Open
DeviceDescription	1.0	String	R	Open
DeviceName	1.0	String	R	Open

Specific		*Type*	*Access*	*Initialized After*
CapValidationDevice	1.0	Boolean	R	Open
RawData	1.0	String	R	Open
AccountNumber	1.0	String	R	Open
Amount	1.0	String	R	Open
BankNumber	1.0	String	R	Open
EPC	1.0	String	R	Open
SerialNumber	1.0	String	R	Open
TransitNumber	1.0	String	R	Open
CheckType	1.0	Long	R	Open
CountryCode	1.0	Long	R	Open

Methods

Common		*May Use After*
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	Open & Claim
ClearOutput	1.0	*Not Supported*
DirectIO	1.0	Open
Specific		
BeginInsertion	1.0	Open, Claim, & Enable
EndInsertion	1.0	Open, Claim, & Enable
BeginRemoval	1.0	Open, Claim, & Enable
EndRemoval	1.0	Open, Claim, & Enable

Events

Name		*May Occur After*
DataEvent	1.0	Open, Claim, & Enable
DirectIOEvent	1.0	Open, Claim
ErrorEvent	1.0	Open, Claim, & Enable
OutputCompleteEvent	1.0	*Not Supported*
StatusUpdateEvent	1.3	Open, Claim, & Enable

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 “APPENDIX B
OPOS Registry Usage”, page 683.)

Model

The MICR Control follows the general "Input Model" (page 24). 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 calls the **BeginInsertion** method, specifying a timeout value. This results in the device being made ready to have a check inserted. The method either returns a success status if the check is inserted before the timeout limit was expired, or a timeout status is 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 calls the **EndInsertion** method, which results in the MICR device being taken out of check insertion mode and the check, if present, actually being read.

    - ♦ If the check is successfully read by the Control, it enqueues a **DataEvent**.
    - ♦ If the **AutoDisable** property is TRUE, then the control automatically disables itself when a **DataEvent** is enqueued.
    - ♦ An enqueued **DataEvent** can be delivered to the application when the **DataEventEnabled** property is TRUE. Just before delivering this event, the Control copies the data into properties, and disables further data events by setting the **DataEventEnabled** property to FALSE. This causes subsequent input data to be enqueued by 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.
    - ♦ An **ErrorEvent** (or events) are enqueued if the Control encounters an error while reading the check, and is delivered to the application when the **DataEventEnabled** property is TRUE.
    - ♦ The **DataCount** property may be read to obtain the number of **DataEvent**s enqueued by the Control.
    - ♦ All input enqueued by the Control may be deleted by calling the **ClearInput** method.
- After processing a **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 positions 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 either returns a success status if the check is removed, or timeout if the check is not 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 enqueue 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 character 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 A string containing the account number parsed from the most recently read MICR data.

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.

If the account number cannot be identified successfully, the string will be empty (“”).

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

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

Amount Property

Syntax **BSTR Amount;**

Remarks A string containing the amount field parsed from the most recently read MICR data.

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.

If the amount is not present, the string will be empty (“”).

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

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 4 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 though 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 384). 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. If zero, the method tries to begin insertion mode, then returns the appropriate status immediately.
If OPOS_FOREVER (-1), the method tries to begin insertion mode, then waits as long as needed until either the check is inserted or an error occurs.

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_ILLEGAL	An invalid *Timeout* parameter was specified.
OPOS_E_TIMEOUT	The specified time has elapsed without the check being properly inserted.
Other Values	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.
If zero, the method tries to begin removal mode, then returns the appropriate status immediately.
If OPOS_FOREVER (-1), the method tries to begin removal mode, then waits as long as needed until either the check is removed or an error occurs.

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_ILLEGAL	An invalid *Timeout* parameter was specified.
OPOS_E_TIMEOUT	The specified time has elapsed without the check being properly removed.
Other Values	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.
Other Values	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.
Other Values	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 delivering 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
ResultCode	Result code causing the error event. See **ResultCode** for values.
ResultCodeExtended	Extended result code causing the error event. See **ResultCodeExtended** for values.
ErrorLocus	Location of the error. See values below.
pErrorResponse	Pointer to the error event response. See values below.

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

Value	Meaning
OPOS_EL_INPUT	Error occurred while gathering or processing event-driven input. No input data is available.
OPOS_EL_INPUT_DATA	Error occurred while gathering or processing event-driven input, and some previously buffered data is available.

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 deliver additional **DataEvent**s as directed by the **DataEventEnabled** property. When all input has been delivered and the **DataEventEnabled** property is again set to TRUE, then another **ErrorEvent** is delivered 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 delivered until the **DataEventEnabled** property is TRUE, so that proper application sequencing occurs.

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

C H A P T E R   1 2

MSR - Magnetic Stripe Reader

Summary

Properties

Common		*Type*	*Access*	*Initialized After*
AutoDisable	1.2	Boolean	R/W	Open
BinaryConversion	1.2	Long	R/W	Open
CapPowerReporting	1.3	Long	R	Open
CheckHealthText	1.0	String	R	Open
Claimed	1.0	Boolean	R	Open
DataCount	1.2	Long	R	Open
DataEventEnabled	1.0	Boolean	R/W	Open
DeviceEnabled	1.0	Boolean	R/W	Open & Claim
FreezeEvents	1.0	Boolean	R/W	Open
OutputID	1.0	Long	R	*Not Supported*
PowerNotify	1.3	Long	R/W	Open
PowerState	1.3	Long	R	Open
ResultCode	1.0	Long	R	--
ResultCodeExtended	1.0	Long	R	Open
State	1.0	Long	R	--
ControlObjectDescription	1.0	String	R	--
ControlObjectVersion	1.0	Long	R	--
ServiceObjectDescription	1.0	String	R	Open
ServiceObjectVersion	1.0	Long	R	Open
DeviceDescription	1.0	String	R	Open
DeviceName	1.0	String	R	Open

Specific		***Type***	***Access***	***Initialized After***
CapISO	1.0	Boolean	R	Open
CapJISOne	1.0	Boolean	R	Open
CapJISTwo	1.0	Boolean	R	Open
TracksToRead	1.0	Long	R/W	Open
DecodeData	1.0	Boolean	R/W	Open
ParseDecodeData	1.0	Boolean	R/W	Open
ErrorReportingType	1.2	Long	R/W	Open
Track1Data	1.0	String	R	Open
Track2Data	1.0	String	R	Open
Track3Data	1.0	String	R	Open
AccountNumber	1.0	String	R	Open
ExpirationDate	1.0	String	R	Open
Title	1.0	String	R	Open
FirstName	1.0	String	R	Open
MiddleInitial	1.0	String	R	Open
Surname	1.0	String	R	Open
Suffix	1.0	String	R	Open
ServiceCode	1.0	String	R	Open
Track1DiscretionaryData	1.0	String	R	Open
Track2DiscretionaryData	1.0	String	R	Open

Methods

Common		***May Use After***
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	Open & Claim
ClearOutput	1.0	*Not Supported*
DirectIO	1.0	Open

Events

Name		***May Occur After***
DataEvent	1.0	Open, Claim, & Enable
DirectIOEvent	1.0	Open, Claim
ErrorEvent	1.0	Open, Claim, & Enable
OutputCompleteEvent	1.0	*Not Supported*
StatusUpdateEvent	1.3	Open, Claim, & Enable

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

Four 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 **ErrorReportingType** property controls the type of handling that occurs when a track containing invalid data is read.

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

- When input is received by the Control, it enqueues a **DataEvent**.
- If the **AutoDisable** property is TRUE, then the control automatically disables itself when a **DataEvent** is enqueued.
- An enqueued **DataEvent** can be delivered to the application when the **DataEventEnabled** property is TRUE. Just before delivering this event, the Control copies the data into properties, and disables further data events by setting the **DataEventEnabled** property to FALSE. This causes subsequent input data to be enqueued by 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.
- An **ErrorEvent** (or events) are enqueued if the Control encounters an error while gathering or processing input, and is delivered to the application when the **DataEventEnabled** property is TRUE.
- The **DataCount** property may be read to obtain the number of **DataEvent**s enqueued by the Control.
- All input enqueued by the Control 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 FALSE, the **Track1Data**, **Track2Data**, and **Track3Data** properties contain the original encoded bit sequence, known as "raw format".

If TRUE, each byte of track data contained within the **Track1Data**, **Track2Data**, and **Track3Data** properties is mapped from its raw format 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	Raw Bytes	Decoded Bytes
ISO	Track 1	6-Bit	0x00 - 0x3F	0x20 - 0x5F
	Track 2	4-Bit	0x00 - 0x0F	0x30 - 0x3F
	Track 3	4-Bit	0x00 - 0x0F	0x30 - 0x3F
JIS-I	Track 1	6-Bit	0x00 - 0x3F	0x20 - 0x5F
	Track 1	7-Bit	0x00 - 0x7F	Unchanged
	Track 2	4-Bit	0x00 - 0x0F	0x30 - 0x3F
	Track 3	4-Bit	0x00 - 0x0F	0x30 - 0x3F
	Track 3	7-Bit	0x00 - 0x7F	Unchanged
JIS-II	JIS Track on Front of Card	7-Bit	0x00 - 0x7F	Unchanged

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

ErrorReportingType Property R/W *Added in Release 1.2*

Syntax **LONG ErrorReportingType;**

Remarks An error is reported by an **ErrorEvent** when a card is swiped, and one or more of the tracks specified by the **TracksToRead** property contains data with errors.

When the **ErrorEvent** is fired to the application, two types of error reporting are supported:

- Card level: A general error status is given, with no data returned.
  This level should be used when a simple pass/fail of the card data is sufficient.
- Track level: The Control can return an extended status with a separate status for each of the tracks. Also, for those tracks that contain valid data or no data, the track' s properties are updated as with a **DataEvent**. For those tracks that contain invalid data, the track' s properties are set to empty.
  This level should be used when the application may be able to utilize a successfully read track or tracks when another of the tracks contains errors.
  For example, suppose **TracksToRead** is MSR_TR_1_2_3, and a swiped card contains good track 1 and 2 data, but track 3 contains "random noise" that is flagged as an error by the MSR. With track level error reporting, the **ErrorEvent** sets the track 1 and 2 properties with the valid data, sets the track 3 properties to empty, and returns an error code indicating the status of each track.

Value	Meaning
MSR_ERT_CARD	Report errors at a card level.
MSR_ERT_TRACK	Report errors at a track level.

This property is initialized to MSR_ERT_CARD by the **Open** method, which is the functionality supported prior to Release 1.2.

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 value was specified.

See Also **ErrorEvent**

ExpirationDate Property

Syntax **BSTR ExpirationDate;**

Remarks The expiration date obtained from the most recently swiped card, as four ASCII decimal characters in the form YYMM. For example, February 1998 is "9802" and August 2018 is "1808".

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.

ParseDecodeData Property R/W

Syntax **BOOL ParseDecodeData;**
BOOL ParseDecodedData; (Synonym for **ParseDecodeData**.[5])

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

[5] An MSR Control Object must support the property **ParseDecodeData**. In addition, due to a documentation error in OPOS APG Releases 1.1 and earlier, it is recommended that the property **ParseDecodedData** also be supported, and that it refer to the same 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.

This property contains track data between but not including the start and end sentinels.

If **DecodeData** is TRUE, then the data returned by this property has been decoded from "raw" format. The data 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.

This property contains track data between but not including the start and end sentinels.

If **DecodeData** is TRUE, then the data returned by this property has been decoded from "raw" format. It 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.

This property contains track data between but not including the start and end sentinels.

If **DecodeData** is TRUE, then the data returned by this property has been decoded from "raw" format.

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 delivering the event, the swiped data is placed into **Track1Data**, **Track2Data**, and **Track3Data**. If **DecodeData** is TRUE, then this track data is decoded. If **ParseDecodeData** 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
ResultCode	Result code causing the error event. See values below.
ResultCodeExtended	Extended result code causing the error event. See values below.
ErrorLocus	Location of the error. See values below.
pErrorResponse	Pointer to the error event response. See values below.

If the **ErrorReportingType** property is MSR_ERT_CARD, then 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.
Other Values	See **ResultCode**.

If the **ErrorReportingType** property is MSR_ERT_TRACK, then the *ResultCode* parameter may be one of the following:

Value	Meaning
OPOS_E_EXTENDED	**ResultCodeExtended** = Track-level status, broken down as follows:

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

Each of the track status bytes may be one of the following:

OPOS_SUCCESS	No error.
OPOS_EMSR_START	Start sentinel error.
OPOS_EMSR_END	End sentinel error.
OPOS_EMSR_PARITY	Parity error.
OPOS_EMSR_LRC	LRC error.
OPOS_E_FAILURE	Other or general error.

Other Values	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 deliver additional **DataEvent**s as directed by the **DataEventEnabled** property. When all input has been delivered and the **DataEventEnabled** property is again set to TRUE, then another **ErrorEvent** is delivered 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 delivered until the **DataEventEnabled** property is TRUE, so that proper application sequencing occurs.

If the **ErrorReportingType** property is MSR_ERT_CARD, then the track that caused the fault cannot be determined, and the track data properties are not changed.

If the **ErrorReportingType** property is MSR_ERT_TRACK, then the *ResultCode* and *ResultCodeExtended* parameters may indicate the track-level status. Also, the track data properties are updated as with **DataEvent**, with the properties for the track or tracks in error set to empty strings. Unlike **DataEvent**, individual track lengths are not reported. However, the application can determine their lengths by getting the length of each of the **Track*x*Data** properties. Also, since this is an **ErrorEvent** (even though it is reporting partial data), the **DataCount** property is not incremented and the Control remains enabled, regardless of the **AutoDisable** property value.

See Also “Status, Result Code, and State Model”; **ErrorReportingType** Property

C H A P T E R   1 3

PIN Pad

Summary

Properties

Common		*Type*	*Access*	*Initialized After*
AutoDisable	1.3	Boolean	R/W	*Not Supported*
BinaryConversion	1.3	Long	R/W	*Not Supported*
CapPowerReporting	1.3	Long	R	Open
CheckHealthText	1.3	String	R	Open
Claimed	1.3	Boolean	R	Open
DataCount	1.3	Long	R	Open
DataEventEnabled	1.3	Boolean	R/W	Open
DeviceEnabled	1.3	Boolean	R/W	Open & Claim
FreezeEvents	1.3	Boolean	R/W	Open
OutputID	1.3	Long	R	*Not Supported*
PowerNotify	1.3	Long	R/W	Open
PowerState	1.3	Long	R	Open
ResultCode	1.3	Long	R	--
ResultCodeExtended	1.3	Long	R	Open
State	1.3	Long	R	--
ControlObjectDescription	1.3	String	R	--
ControlObjectVersion	1.3	Long	R	--
ServiceObjectDescription	1.3	String	R	Open
ServiceObjectVersion	1.3	Long	R	Open
DeviceDescription	1.3	String	R	Open
DeviceName	1.3	String	R	Open

Specific		*Type*	*Access*	*Initialized After*
CapMACCalculation	1.3	Boolean	R	Open
CapDisplay	1.3	Long	R	Open
CapLanguage	1.3	Long	R	Open
CapKeyboard	1.3	Boolean	R	Open
CapTone	1.3	Boolean	R	Open
AvailablePromptsList	1.3	String	R	Open
Prompt	1.3	Long	R/W	Open
AvailableLanguagesList	1.3	String	R	Open
PromptLanguage	1.3	Long	R/W	Open
AccountNumber	1.3	String	R/W	Open
Amount	1.3	Currency	R/W	Open
MerchantID	1.3	String	R/W	Open
TerminalID	1.3	String	R/W	Open
Track1Data	1.3	String	R/W	Open
Track2Data	1.3	String	R/W	Open
Track3Data	1.3	String	R/W	Open
TransactionType	1.3	String	R/W	Open
MinimumPINLength	1.3	Long	R/W	Open
MaximumPINLength	1.3	Long	R/W	Open
PINEntryEnabled	1.3	Boolean	R	Open
EncryptedPIN	1.3	String	R	Open
AdditionalSecurity Information	1.3	String	R	Open

Methods

Common		*May Use After*
Open	1.3	--
Close	1.3	Open
Claim	1.3	Open
Release	1.3	Open & Claim
CheckHealth	1.3	Open, Claim, & Enable
ClearInput	1.3	Open, Claim, & Enable
ClearOutput	1.3	*Not Supported*
DirectIO	1.3	Open
Specific		
BeginEFTTransaction	1.3	Open, Claim, & Enable
EndEFTTransaction	1.3	BeginEFTTransaction
EnablePINEntry	1.3	BeginEFTTransaction
ComputeMAC	1.3	BeginEFTTransaction
VerifyMAC	1.3	BeginEFTTransaction
UpdateKey	1.3	BeginEFTTransaction

Events

Name		*May Occur After*
DataEvent	1.3	Open, Claim, & Enable
DirectIOEvent	1.3	Open, Claim
ErrorEvent	1.3	Open, Claim, & Enable
OutputCompleteEvent	1.3	*Not Supported*
StatusUpdateEvent	1.3	Open, Claim, & Enable

General Information

The Pinpad Control’ s OLE programmatic ID is “OPOS.PINPad”.

This device was added in OPOS Release 1.3.

A Pinpad

- Provides a mechanism for customers to perform PIN Entry
- Acts as a cryptographic engine for communicating with an EFT Transaction Host.

A Pinpad will perform these functions by implementing one or more Pinpad Management Systems. A Pinpad Management System defines the manner in which the Pinpad will perform functions such as PIN Encryption, Message Authentication Code calculation, and Key Updating. Examples of Pinpad Management Systems include: Master-Session, DUKPT, APACS40, HGEPOS, and AS2805, along with many others.

Capabilities

The Pinpad Control has the following minimal capability:

- Accept a PIN Entry at its keyboard and provide an Encrypted PIN to the application.

The Pinpad Control may have the following additional capabilities:

- Compute Message Authentication Codes.
- Perform Key Updating in accordance with the selected Pinpad Management System.
- Support multiple Pinpad Management Systems.
- Allow use of the Pinpad Keyboard, Display, & Tone Generator for application usage. If one or more of these features are available, then the Application opens and uses the associated POS Keyboard, Line Display, or Tone Indicator Control Objects.

Features Not Supported

This specification does not include support for the following:

- Initial Key Loading. This operation usually requires downloading at least one key in the clear and must be done in a secure location (typically either the factory or at a Financial Institution). Thus, support for initial key loading is outside the scope of this specification. However, this specification does include support for updating keys while a Pinpad unit is installed at a retail site.
- Full EFT functionality. This specification addresses the functionality of a Pinpad that is used solely as a peripheral device by an Electronic Funds Transfer application. It specifically does not define the functionality of an Electronic Funds Transfer application that might execute within an intelligent Pinpad. This specification does not include support for applications in which the Pinpad Application determines that a message needs to be transmitted to the EFT Transaction Host. *Consequently, this specification will not apply in Canada, Germany, Netherlands, and possibly other countries. It also does not apply to Pinpads in which the vendor has chosen to provide EFT Functionality in the Pinpad.*
- Smartcard Reader. Some Pinpad devices will include a Smartcard reader. Support for this device may be included in a future revision of this specification.

Model

A Pinpad performs encryption functions under control of a Pinpad Management System. Some Pinpads will support multiple Pinpad Management Systems. Some Pinpad Management Systems support multiple keys (sets) for different EFT Transaction Hosts. Thus, for each EFT transaction, the application will need to select the Pinpad Management System and EFT Transaction Host to be used. Depending on the Pinpad Management System, one or more EFT transaction parameters will need to be provided to the Pinpad for use in the encryption functions. The application should set the value of **ALL** EFT Transaction parameter properties to enable easier migration to EFT Transaction Hosts that require a different Pinpad Management System.

After opening, claiming, and enabling the Pinpad Control, the application should use the following general scenario for each EFT Transaction.

- The application must set the EFT transaction parameters (**AccountNumber**, **Amount**, **MerchantID**, **TerminalID**, **Track1Data**, **Track2Data**, **Track3Data** and **TransactionType** properties) and then perform a **BeginEFTTransaction** method. This will initialize the Service Object and Pinpad for performing the encryption functions for the EFT transaction.
- If PIN Entry is required, call the **EnablePINEntry** method. Then set the **DataEventEnabled** property and wait for the **DataEvent** event.
- If Message Authentication Codes are required, use the **ComputeMAC** and **VerifyMAC** methods as needed.
- Perform an **EndEFTTransaction** method to notify the Control that all operations for the EFT transaction have been completed.

This specification supports 2 models of how the display on the Pinpad is used. The **CapDisplay** property indicates which model the Pinpad device supports.

- In one model, the Application has complete control of the text that is to be displayed. For this model, there is an associated OPOS Line Display Control that is used by the Application to interact with the display.
- In the other model, the Application cannot supply the text to be displayed. Instead, it can only select from a list of pre-defined messages to be displayed. For this model, there is a set of Pinpad properties that are used to control the display.

Device Sharing

The Pinpad 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 R/W

Syntax **BSTR AccountNumber;**

Remarks The account number to be used for the current EFT transaction. The application must set this property before calling the **BeginEFTTransaction** method. Any attempt to change this property after the **BeginEFTTransaction** method has been called will result in a value of OPOS_E_ILLEGAL being stored into **ResultCode**.

AdditionalSecurityInformation Property

Syntax **BSTR AdditionalSecurityInformation;**

Remarks This property may contain additional security/encryption information after a **DataEvent** event. This property will be formatted as a Hex-ASCII string. The information content and internal format of this string will vary among Pinpad Management Systems. For example, if the Pinpad Management System is DUKPT, then this property will contain the "Pinpad sequence number". If the PIN Entry was canceled, this property will contain the empty string.

Amount Property R/W

Syntax **CURRENCY Amount;**

Remarks The amount of the current EFT transaction. The application must set this property before calling the **BeginEFTTransaction** method. Any attempt to change this property after the **BeginEFTTransaction** method has been called will result in a value of OPOS_E_ILLEGAL being stored into **ResultCode**.

AvailableLanguagesList Property

Syntax **BSTR AvailableLanguagesList;**

Remarks This property is a comma separated string of the languages supported by the pre-defined prompts in the Pinpad. Languages are numeric values and are Microsoft Language Ids. If **CapLanguage** = PPAD_LANG_NONE, then this property will be the empty string.

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

AvailablePromptsList Property

Syntax **BSTR AvailablePromptsList;**

Remarks This property is a comma-separated string of supported values for the **Prompt** property.

Value	Meaning
PPAD_MSG_ENTERPIN	The user should enter his pin number on the Pinpad.
PPAD_MSG_PLEASEWAIT	The system is processing. The user should wait.
PPAD_MSG_ENTERVALIDPIN	The pin that was entered is not correct. The user should enter the correct pin number.
PPAD_MSG_RETRIESEXCEEDED	The user has failed to enter the correct pin number and the maximum number of attempts has been exceeded.
PPAD_MSG_APPROVED	The request has been approved.
PPAD_MSG_DECLINED	The EFT Transaction Host has declined to perform the requested function.
PPAD_MSG_CANCELED	The request is canceled.
PPAD_MSG_AMOUNTOK	The customer should enter Yes/No to approve the amount.
PPAD_MSG_NOTREADY	Pinpad is not ready for use by customer.
PPAD_MSG_IDLE	The System is Idle.
PPAD_MSG_SLIDE_CARD	The user should slide their card through the integrated MSR.
PPAD_MSG_INSERTCARD	The customer should insert their (smart)card.

PPAD_MSG_SELECTCARDTYPE	The customer should select the card type (typically credit or debit).

Values 1000 and above are reserved for OEM defined values.

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

CapDisplay Property

Syntax **LONG CapDisplay;**

Remarks Defines the operations that the Application may perform on the Pinpad display.

Value	Meaning
PPAD_DISP_UNRESTRICTED	The application can use the Pinpad display in an unrestricted manner to display messages. In this case, an associated Line Display Control Object is the interface to the Pinpad display. The Application must call Line Display methods to manipulate the display.
PPAD_DISP_PINRESTRICTED	The Application can use the Pinpad display in an unrestricted manner except during PIN Entry. The Pinpad will display a pre-defined message during PIN Entry. If an attempt is made to use the associated Line Display Control Object while PIN Entry is enabled, the Line Display Control will return a result of OPOS_E_BUSY.
PPAD_DISP_RESTRICTED_LIST	The Application cannot specify the text of messages to display. It can only select from a list of pre-defined messages. There is no associated Line Display Control Object.
PPAD_DISP_RESTRICTED_ORDER	The application cannot specify the text of messages to display. It can only select from a list of pre-defined messages. The selections must occur in a pre-defined acceptable order. There is no associated Line Display Control object.

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

CapLanguage Property

Syntax **LONG CapLanguage;**

Remarks Defines the capabilities that the application has to select the language of pre-defined messages (e.g. English, French, Arabic).

Value	Meaning
PPAD_LANG_NONE	The Pinpad supports no pre-defined prompt messages. The property will be set to this value if **CapDisplay** = PPAD_DISP_UNRESTRICTED. Any attempt to set the value of the **PromptLanguage** property will cause the **ResultCode** property to have a value of OPOS_E_ILLEGAL.
PPAD_LANG_ONE	The Pinpad supports pre-defined prompt messages in one language. Any attempt to set the value of the **PromptLanguage** property to other than the default value will cause the **ResultCode** property to have a value of OPOS_E_ILLEGAL.
PPAD_LANG_PINRESTRICTED	The Pinpad cannot change prompt languages during PIN Entry. The application must set the desired value into the **PromptLanguage** property before calling **EnablePINEntry**. Any attempt to set the value of the **PromptLanguage** while **PINEntryEnabled** is TRUE will cause the **ResultCode** property to have a value of OPOS_E_BUSY.
PPAD_LANG_UNRESTRICTED	The application can change the language of pre-defined prompt messages at anytime. The currently displayed message will change immediately.

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

CapMACCalculation Property

Syntax **BOOL CapMACCalculation;**

Remarks If TRUE, the Pinpad supports MAC calculation.

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

CapKeyboard Property

Syntax **BOOL CapKeyboard;**

Remarks Defines whether the application can obtain input from the Pinpad keyboard.

If TRUE, the application can use the Pinpad to obtain input. The application will use an associated POS Keyboard Control object as the interface to the Pinpad keyboard. Note that the associated POS Keyboard Control is effectively disabled while **PINEntryEnabled** is TRUE.

If FALSE, the application cannot obtain input directly from the Pinpad keyboard.

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

CapTone Property

Syntax **BOOL CapTone;**

Remarks If TRUE, the Pinpad has a Tone Indicator. The Tone Indicator may be accessed by use of an associated Tone Indicator Control. If FALSE, there is no Tone Indicator.

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

EncryptedPIN Property

Syntax **BSTR EncryptedPIN;**

Remarks This property will contain the value of the Encrypted PIN after a **DataEvent** event. This property will be formatted as a 16 byte Hex-ASCII string. If the PIN Entry was canceled, this property will contain the empty string.

MaximumPINLength Property R/W

Syntax **LONG MaximumPINLength;**

Remarks The application should set this property to the maximum acceptable number of digits in a PIN. This property must be set by the application before the **EnablePINEntry** method is executed This property will be set to a default value by the **Open** method. Note that in some implementations, this value cannot be changed by the application

MerchantID Property R/W

Syntax **BSTR MerchantID;**

Remarks The Merchant ID, as it is known to the EFT Transaction Host. The application must set this property before calling the **BeginEFTTransaction** method. Any attempt to change this property after the **BeginEFTTransaction** method has been called will result in a value of OPOS_E_ILLEGAL being stored into **ResultCode**.

MinimumPINLength Property R/W

Syntax **LONG MinimumPINLength;**

Remarks The application should set this property to the minimum acceptable number of digits in a PIN. This property must be set by the application before the **EnablePINEntry** method is executed. This property will be set to a default value by the **Open** method. Note that in some implementations, this value cannot be changed by the application.

PINEntryEnabled Property

Syntax **BOOL PINEntryEnabled;**

Remarks The Pinpad Control object sets this property to TRUE when an **EnablePINEntry** method is executed. It will be set to FALSE when the user has completed the PIN Entry operation or an **EndEFTTransaction** is executed.

Prompt Property R/W

Syntax **LONG Prompt;**

Remarks This property identifies a pre-defined message to be displayed on the Pinpad. This property is used if **CapDisplay** has a value of PPAD_DISP_RESTRICTED_LIST or PPAD_DISP_RESTRICTED_ORDER. It is also used during PIN Entry if **CapDisplay** has a value of PPAD_DISP_PINRESTRICTED. The **AvailablePromptsList** property lists the values for this property that the Service Object will accept.

This property is initialized 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	One of the following occurred: • An attempt was made to set the property to a value that is not supported by the Pinpad Service object • An attempt was made to select prompt messages in an unacceptable order **(CapDisplay** = PPAD_DISP_RESTRICTED_ORDER)
Other Values	See **ResultCode**.

See Also **PromptLanguage**

PromptLanguage Property R/W

Syntax **LONG PromptLanguage;**

Remarks This property specifies the language of the message to be displayed (as specified by the **Prompt** property). This property is used if the **Prompt** property is being used. The exact effect of changing this property depends on the value of the **CapLanguage** property.

The values for this property are MS Windows Language IDs. The property is initialized to a default value 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 attempt was made to set the property to a value that is not supported by the Pinpad Service object.
Other Values	See **ResultCode**.

See Also **CapLanguage, AvailableLanguagesList**

TerminalID Property R/W

Syntax **BSTR TerminalID;**

Remarks The terminal ID, as it is known to the EFT Transaction Host. The application must set this property before calling the **BeginEFTTransaction** method. Any attempt to change this property after the **BeginEFTTransaction** method has been called will result in a value of OPOS_E_ILLEGAL being stored into **ResultCode**.

Track1Data Property R/W

Syntax **BSTR Track1Data;**

Remarks Contains either the track 1 data from the previous card swipe or an empty string. An empty string indicates that the track was not physically read. The application must set this property before calling the **BeginEFTTransaction** method Any attempt to change this property after the **BeginEFTTransaction** method has been called will result in a value of OPOS_E_ILLEGAL being stored into **ResultCode**.

Track2Data Property R/W

Syntax **BSTR Track2Data;**

Remarks Contains either the track 2 data from the previous card swipe or an empty string. An empty string indicates that the track was not physically read. The application must set this property before calling the **BeginEFTTransaction** method Any attempt to change this property after the **BeginEFTTransaction** method has been called will result in a value of OPOS_E_ILLEGAL being stored into **ResultCode**.

Track3Data Property R/W

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 physically read. The application must set this property before calling the **BeginEFTTransaction** method Any attempt to change this property after the **BeginEFTTransaction** method has been called will result in a value of OPOS_E_ILLEGAL being stored into **ResultCode**.

TransactionType Property R/W

Syntax **LONG TransactionType;**

Remarks The type of the current EFT transaction. The application must set this property before calling the **BeginEFTTransaction** method. Any attempt to change this property after the **BeginEFTTransaction** method has been called will result in a value of OPOS_E_ILLEGAL being stored into **ResultCode**.

TransactionType can have one of the following values:

Value	Meaning
PPAD_TRANS_DEBIT	Debit (decrease) the specified account
PPAD_TRANS_CREDIT	Credit (increase) the specified account.
PPAD_TRANS_INQ	(Balance) Inquiry
PPAD_TRANS_RECONCILE	Reconciliation/Settlement
PPAD_TRANS_ADMIN	Administrative Transaction

Methods

BeginEFTTransaction Method

Syntax **LONG BeginEFTTransaction (BSTR** *PINPadSystem,* **LONG** *TransactionHost***);**

Parameter	Description
PINPadSystem	Name of the desired Pinpad Management System. See below for the Pinpad Management System names defined by this standard. The Service Object implementer may define names for other Pinpad Management systems.
TransactionHost	Identifies the particular EFT Transaction Host to be used for this transaction.

The defined *PINPadSystem* parameter values are:

Value	Meaning
"M/S"	Master/Session. (USA, Latin America)
"DUKPT"	Derived Unique Key Per Transaction (USA, Latin America)
"APACS40"	Standard 40 (UK and other countries)
"AS2805"	Australian Standard 2805
"HGEPOS"	(Italian)

Remarks This method must be called by the application to inform the Pinpad Control of the beginning of an EFT Transaction. The Pinpad Control will perform initialization functions (such as computing session keys). No other Pinpad functions can be performed until this 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_NOSERVICE	The requested Pinpad Management System is not supported by the service object.

OPOS_E_ILLEGAL	The requested EFT Transaction Host is an illegal value for the selected Pinpad Management System.
OPOS_E_BUSY	The Pinpad is already performing an EFT transaction.
Other Values	See **ResultCode**.

ComputeMAC Method

Syntax **LONG ComputeMAC (BSTR** *InMsg*, **BSTR*** *pOutMsg***);**

Parameter	Description
InMsg	The message that the Application intends to send to an EFT Transaction Host. The format of this data depends upon the value of the **BinaryConversion** property. See page 37.
pOutMsg	Pointer to the result of applying the MAC calculation to *InMsg*. This output parameter will contain a reformatted message that may actually be transmitted to an EFT Transaction Host. The format of this data depends upon the value of the **BinaryConversion** property. See page 37.

Remarks This method is called by the application to have the Pinpad compute a MAC value and append it to the designated message. Depending on the selected Pinpad Management System, the Pinpad may also insert other fields into the message. Note that the **ComputeMAC** method cannot be used while Pinpad input (PIN Entry) is enabled.

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_DISABLED	A **BeginEFTTransaction** method has not been performed.
OPOS_E_BUSY	**PINEntryEnabled** is TRUE. The Pinpad cannot perform a MAC calculation during PIN Entry.
Other Values	See **ResultCode**.

EnablePINEntry Method

Syntax **LONG EnablePINEntry ();**

Remarks This method is called by the application to enable PIN Entry at the Pinpad device. When this method is called, the **PINEntryEnabled** property will be changed to TRUE. If the Pinpad uses pre-defined prompts for PIN Entry, then the value of the **Prompt** property will be changed to PPAD_MSG_ENTERPIN.

When the user has completed the PIN entry operation (either by entering their PIN or by hitting Cancel), the **PINEntryEnabled** property will be changed to FALSE. A **DataEvent** event will be fired to provide the encrypted PIN to the application when **DataEventEnabled** is set to TRUE. Note that any data entered at the Pinpad while **PINEntryEnabled** is TRUE will be supplied in encrypted form to this Control Object and will NOT be provided to any associated Keyboard Control Object.

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_DISABLED	A **BeginEFTTransaction** method has not been performed.
Other Values	See **ResultCode**.

EndEFTTransaction Method

Syntax **LONG EndEFTTransaction (LONG** *CompletionCode***);**

CompletionCode is one of the following values:

Value	Meaning
PPAD_EFT_NORMAL	The EFT transaction completed normally. Note that this does not mean that the EFT transaction was approved. It merely means that the proper sequence of messages was transmitted and received.
PPAD_EFT_ABNORMAL	The proper sequence of messages was not transmitted & received.

Remarks This method must be called by the application to inform the Pinpad Control of the end of an EFT Transaction. The Pinpad Control will perform termination functions (such as computing next transaction keys).

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

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

UpdateKey Method

Syntax **LONG UpdateKey (LONG** *KeyNum*, **BSTR** *Key***);**

Parameter	Description
KeyNum	A key number.
Key	A Hex-ASCII value for a new key.

Remarks This method is used to provide a new encryption key to the Pinpad. It is used only for those Pinpad Management Systems in which new key values are sent to the terminal as a field in standard messages from the EFT Transaction Host.

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

Value	Meaning
OPOS_SUCCESS	The specified key has been updated was successful.
OPOS_E_BUSY	The Pinpad cannot accept a new key at this time.
OPOS_E_ILLEGAL	One of the following conditions occurred: • The selected Pinpad Management System does not support this function • *KeyNum* specifies an unacceptable key number. • *Key* contains a bad key (not Hex-ASCII or wrong length or bad parity).
Other Values	See **ResultCode**.

VerifyMAC Method

Syntax **BOOL VerifyMAC (BSTR** *Message***);**

Message contains a message received from an EFT Transaction Host.

Remarks This method is called by the application to have the Pinpad verify the MAC value in a message received from an EFT Transaction Host. This method returns TRUE if it can verify the message; otherwise, it returns FALSE. Note that the **VerifyMAC** method cannot be used while PIN Entry is enabled.

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_DISABLED	A **BeginEFTTransaction** method has not been performed.
OPOS_E_BUSY	**PINEntryEnabled** is TRUE. The Pinpad cannot perform a MAC verification during PIN Entry.
Other Values	See **ResultCode**.

Events

DataEvent Event

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

The *Status* parameter is one of the following values:.

Value	Meaning
PPAD_SUCCESS	PIN Entry has occurred and values have been stored into the **EncryptedPIN** and **AdditionalSecurityInformation** properties.
PPAD_CANCEL	The user hit the cancel button on the Pinpad.
PPAD_TIMEOUT	A timeout condition occurred in the Pinpad. (Not all Pinpads will report this condition)

Remarks Fired to indicate the completion of a PIN Entry operation.

ErrorEvent Event

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

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

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

Value	Meaning
OPOS_E_EXTENDED	**ResultCodeExtended** = PPAD_BAD_KEY: An Encryption Key is corrupted or missing.
Other Values	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.

The contents at the location pointed to by the *pErrorResponse* parameter are preset to a default value, based on the *ErrorLocus*:

Value	Meaning
OPOS_ER_CLEAR	Clear the buffered input data. The error state is exited. Default when locus is OPOS_EL_INPUT.

Remarks Fired when an error is detected while trying to perform a PIN encryption function. The Pinpad service object may optionally provide more detailed diagnostic information via a **CheckHealth** or **DirectIO** mechanism.

C H A P T E R   1 4

POS Keyboard

Summary

Properties

Common		*Type*	*Access*	*Initialized After*
AutoDisable	1.2	Boolean	R/W	Open
BinaryConversion	1.2	Long	R/W	Open
CapPowerReporting	1.3	Long	R	Open
CheckHealthText	1.1	String	R	Open
Claimed	1.1	Boolean	R	Open
DataCount	1.2	Long	R	Open
DataEventEnabled	1.1	Boolean	R/W	Open
DeviceEnabled	1.1	Boolean	R/W	Open & Claim
FreezeEvents	1.1	Boolean	R/W	Open
OutputID	1.1	Long	R	*Not Supported*
PowerNotify	1.3	Long	R/W	Open
PowerState	1.3	Long	R	Open
ResultCode	1.1	Long	R	--
ResultCodeExtended	1.1	Long	R	Open
State	1.1	Long	R	--
ControlObjectDescription	1.1	String	R	--
ControlObjectVersion	1.1	Long	R	--
ServiceObjectDescription	1.1	String	R	Open
ServiceObjectVersion	1.1	Long	R	Open
DeviceDescription	1.1	String	R	Open
DeviceName	1.1	String	R	Open

Specific		*Type*	*Access*	*Initialized After*
CapKeyUp	1.2	Boolean	R	Open
EventTypes	1.2	Long	R/W	Open
POSKeyData	1.1	Long	R	Open
POSKeyEventType	1.2	Long	R	Open

Methods

Common		*May Use After*
Open	1.1	--
Close	1.1	Open
Claim	1.1	Open
Release	1.1	Open & Claim
CheckHealth	1.1	Open, Claim, & Enable
ClearInput	1.1	Open & Claim
ClearOutput	1.1	*Not Supported*
DirectIO	1.1	Open

Events

Name		*May Occur After*
DataEvent	1.1	Open, Claim, & Enable
DirectIOEvent	1.1	Open, Claim
ErrorEvent	1.1	Open, Claim, & Enable
OutputCompleteEvent	1.1	*Not Supported*
StatusUpdateEvent	1.3	Open, Claim, & Enable

General Information

The POS Keyboard Control' s OLE programmatic ID is "OPOS.POSKeyboard".

This device was added in OPOS Release 1.1.

Capabilities

The POS Keyboard Control has the following capability:

- Reads keys from a POS keyboard. A POS keyboard may be an auxiliary keyboard, or it may be a virtual keyboard consisting of some or all of the keys on the system keyboard.[6]

[6] OPOS 1.1 defined a POS Keyboard as a secondary key entry device, separate from the primary keyboard. OPOS 1.2 expanded this definition.

Model

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

- When input is received by the Control, it enqueues a **DataEvent**.
- If the **AutoDisable** property is TRUE, then the control automatically disables itself when a **DataEvent** is enqueued.
- An enqueued **DataEvent** can be delivered to the application when the **DataEventEnabled** property is TRUE. Just before delivering this event, the Control copies the data into properties, and disables further data events by setting the **DataEventEnabled** property to FALSE. This causes subsequent input data to be enqueued by 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.
- An **ErrorEvent** (or events) are enqueued if the Control encounters an error while gathering or processing input, and is delivered to the application when the **DataEventEnabled** property is TRUE.
- The **DataCount** property may be read to obtain the number of **DataEvent**s enqueued by the Control.
- All input enqueued by the Control may be deleted by calling the **ClearInput** method.

Keyboard Translation

The POS Keyboard Control must supply a method for translating its internal key codes into user-defined codes which are returned by the data events. Note that this translation ***must*** be end-user configurable.

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

CapKeyUp Property *Added in Release 1.2*

Syntax **LONG CapKeyUp;**

Remarks If TRUE, then the Control is able to generate both key down and key up events, depending upon the setting of the **EventTypes**.

If FALSE, then the Control is only able to generate the key down event.

EventTypes Property R/W *Added in Release 1.2*

Syntax **LONG EventTypes;**

Remarks Select the type of events that the application wants to receive.

Values are:

Value	Meaning
KBD_ET_DOWN	Generate key down events.
KBD_ET_DOWN_UP	Generate key down and key up events.

Release 1.1: Only key down events can be delivered.
Release 1.2 and later: Key down and key up events can be delivered.

This property is initialized to KBD_ET_DOWN 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 property value was used, or the Control does not support the selected value.

POSKeyData Property

Syntax **LONG POSKeyData;**

Remarks The value of the key from the last **DataEvent**.

The Application may treat this value as device independent, assuming that the system installer has configured the POSKeyboard Service Object to translate internal key codes to the codes expected by the Application. Such configuration is inherently Service Object-specific.

Release 1.1: **POSKeyData** was defined as a logical key code in the upper 16 bits and a scan code in the lower 16 bits, where the values need not match a standard PC keyboard's values.

Release 1.2 and later: Added the requirement for an end-user configurable translation into arbitrary keycodes.

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

POSKeyEventType Property *Added in Release 1.2*

Syntax **LONG POSKeyEventType;**

Remarks This property holds the type of the last keyboard event: Is the key being pressed or released?

Values are:

Value	Meaning
KBD_KET_KEYDOWN	The key in **POSKeyData** was pressed.
KBD_KET_KEYUP	The key in **POSKeyData** was released.

This property is set by the Control just before delivering 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 logical key number is placed in the **POSKeyData** property and the event type is placed in the **POSKeyEventType** property before this event is delivered.

ErrorEvent Event

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

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

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

Value	Meaning
OPOS_EL_INPUT	Error occurred while gathering or processing event-driven input. No input data is available.
OPOS_EL_INPUT_DATA	Error occurred while gathering or processing event-driven input, and some previously buffered data is available.

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 deliver additional **DataEvent**s as directed by the **DataEventEnabled** property. When all input has been delivered and the **DataEventEnabled** property is again set to TRUE, then another **ErrorEvent** is delivered 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 delivered until the **DataEventEnabled** property is TRUE, so that proper application sequencing occurs.

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

C H A P T E R   1 5

POS Printer

Summary

Properties

Common		*Type*	*Access*	*Initialized After*
AutoDisable	1.2	Boolean	R/W	*Not Supported*
BinaryConversion	1.2	Long	R/W	Open
CapPowerReporting	1.3	Long	R	Open
CheckHealthText	1.0	String	R	Open
Claimed	1.0	Boolean	R	Open
DataCount	1.2	Long	R	*Not Supported*
DataEventEnabled	1.0	Boolean	R/W	*Not Supported*
DeviceEnabled	1.0	Boolean	R/W	Open & Claim
FreezeEvents	1.0	Boolean	R/W	Open
OutputID	1.0	Long	R	Open
PowerNotify	1.3	Long	R/W	Open
PowerState	1.3	Long	R	Open
ResultCode	1.0	Long	R	--
ResultCodeExtended	1.0	Long	R	Open
State	1.0	Long	R	--
ControlObjectDescription	1.0	String	R	--
ControlObjectVersion	1.0	Long	R	--
ServiceObjectDescription	1.0	String	R	Open
ServiceObjectVersion	1.0	Long	R	Open
DeviceDescription	1.0	String	R	Open
DeviceName	1.0	String	R	Open

Specific		*Type*	*Access*	*Initialized After*
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
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

Specific (continued)		*Type*	*Access*	*Initialized After*
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

Specific (continued)		*Type*	*Access*	*Initialized After*
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

Specific (continued)		*Type*	*Access*	*Initialized After*
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		***May Use After***
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	*Not Supported*
ClearOutput	1.0	Open & Claim
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		***May Occur After***
DataEvent	1.0	*Not Supported*
DirectIOEvent	1.0	Open, Claim
ErrorEvent	1.0	Open, Claim, & Enable
OutputCompleteEvent	1.0	Open, Claim, & Enable
StatusUpdateEvent	1.0	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 enqueued and delivered. The **ErrorStation** property is set to the station or stations that were printing when the error occurred. ***Release 1.1 and later:*** 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.
- ♦ The Control 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. **OutputCompleteEvent**s 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.

- ***Release 1.1 and later -- Transaction Mode***

  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. Sequences that do not begin with ESC “|” are passed through to the printer. Also, sequences that begin with ESC “|” but which are not valid OPOS escape sequences are passed through to the printer.

To determine if 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_TWO_RECEIPT_JOURNAL and PTR_S_JOURNAL_RECEIPT station parameters.

If FALSE, the application should print to only one of the stations at a time, and minimize transitions between the stations. Non-concurrent printing may be required for reasons such as:

- Higher likelihood of error, such as greater chance of paper jams when moving between the stations.
- Higher performance when each station is printed separately.

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_TWO_SLIP_JOURNAL and PTR_S_JOURNAL_SLIP station parameters.

If FALSE, the application must use the sequence **BeginInsertion/EndInsertion** followed by print requests to the Slip followed by **BeginRemoval/EndRemoval** before printing on the Journal. Non-concurrent printing may be required for reasons such as:

- Physical constraints, such as the Slip form being placed in front of the Journal station.
- Higher likelihood of error, such as greater chance of paper jams when moving between the stations.
- Higher performance when each station is printed separately.

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_TWO_SLIP_RECEIPT and PTR_S_RECEIPT_SLIP station parameters.

If FALSE, the application must use the sequence **BeginInsertion/EndInsertion** followed by print requests to the Slip followed by **BeginRemoval/EndRemoval** before printing on the Receipt. Non-concurrent printing may be required for reasons such as:

- Physical constraints, such as the Slip form being placed in front of the Receipt station.
- Higher likelihood of error, such as greater chance of paper jams when moving between the stations.
- Higher performance when each station is printed separately.

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.
Other Values	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 delivering 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,
PTR_S_RECEIPT_SLIP,
PTR_TWO_RECEIPT_JOURNAL, PTR_TWO_SLIP_JOURNAL,
PTR_TWO_SLIP_RECEIPT.

This property is set just before an **ErrorEvent** is delivered.

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 delivering 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 **JrnNearEnd** 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.

Setting **JrnLetterQuality** may also update **JrnLineWidth**, **JrnLineHeight**, and **JrnLineSpacing** if **MapMode** is PTR_MM_DOTS. (See the footnote at **MapMode**.)

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. (For example, if set to 36 and the printer can print either 30 or 40 characters per line, then the Service Object should select the character size "40" and print up to 36 characters on each line.)

If the character width cannot be supported, then an error is returned. (For example, if set to 42 and the printer can print either 30 or 40 characters per line, then the Service Object cannot support the request.)

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 spacing of each single-high print line, including both the printed line height plus the whitespace between each pair of lines. Depending upon the printer and the current line spacing, a multi-high print line might exceed this value. Line spacing is 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.[7]
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.

[7] From the OPOS POS Printer perspective, the exact definition of a " dot" is not significant. It is a Printer/Service Object unit used to express various metrics. For example, some printers define a " half-dot" that is used in high-density graphics printing, and perhaps in text printing. An OPOS POS Printer Service Object may handle this case in one of these ways:

(a) Consistently define a " dot" as the printer' s smallest physical size, that is, a half-dot.

(b) If the Service Object changes bitmap graphics printing density based on the **XxxLetterQuality** setting, then alter the size of a dot to match the bitmap density (that is, a physical printer dot when FALSE and a half-dot when TRUE). Note that this choice should not be used if the printer' s text metrics are based on half-dot sizes, since accurate values for the metrics may not then be possible.

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.

Setting **RecLetterQuality** may also update **RecLineWidth**, **RecLineHeight**, and **RecLineSpacing** if **MapMode** is PTR_MM_DOTS. (See the footnote at **MapMode**.)

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. (For example, if set to 36 and the printer can print either 30 or 40 characters per line, then the Service Object should select the character size "40" and print up to 36 characters on each line.)

If the character width cannot be supported, then an error is returned. (For example, if set to 42 and the printer can print either 30 or 40 characters per line, then the Service Object cannot support the request.)

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 The spacing of each single-high print line, including both the printed line height plus the whitespace between each pair of lines. Depending upon the printer and the current line spacing, a multi-high print line might exceed this value. Line spacing is 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 **RecLineChars** or **RecLineHeight** are changed, **RecLineSpacing** is updated to the default line spacing for the selected width or height.

The value of **RecLineSpacing** 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.

RecLinesToPaperCut Property

Syntax **LONG RecLinesToPaperCut;**

Remarks Holds the number of lines that must be advanced before the receipt paper is cut.

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.

Changing the properties **RecLineChars**, **RecLineHeight**, and **RecLineSpacing** may cause this property to change.

This property is initialized when the device is first enabled following the **Open** method.

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 90º to the right (clockwise).
PTR_RP_LEFT90	Rotate printing 90º to the left (counter-clockwise).
PTR_RP_ROTATE180	Rotate printing 180º, 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.

Setting **SlpLetterQuality** may also update **SlpLineWidth**, **SlpLineHeight**, and **SlpLineSpacing** if **MapMode** is PTR_MM_DOTS. (See the footnote at **MapMode**.)

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. (The Service Object should print the requested characters in the column positions closest to the side of the slip table at which the slip is aligned. For example, if the operator inserts the slip with the right edge against the table side, and if **SlpLineChars** is set to 36 and the printer prints 60 characters per line, then the Service Object should add 24 spaces at the left margin, and print the characters in columns 25 through 60.)

If the character width cannot be supported, then an error is returned. (For example, if set to 65 and the printer can print 60 characters per line, then the Service Object cannot support the request.)

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 spacing of each single-high print line, including both the printed line height plus the whitespace between each pair of lines. Depending upon the printer and the current line spacing, a multi-high print line might exceed this value. Line spacing is 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.
If zero, the method tries to begin insertion mode, then returns the appropriate status immediately.
If OPOS_FOREVER (-1), the method tries to begin insertion mode, then waits as long as needed until either the form is inserted or an error occurs.

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) or an invalid *Timeout* parameter was specified..
OPOS_E_TIMEOUT	The specified time has elapsed without the form being properly inserted.
Other Values	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.
If zero, the method tries to begin removal mode, then returns the appropriate status immediately.
If OPOS_FOREVER (-1), the method tries to begin removal mode, then waits as long as needed until either the form is removed or an error occurs.

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) or an invalid *Timeout* parameter was specified..
OPOS_E_TIMEOUT	The specified time has elapsed without the form being properly removed.
Other Values	See **ResultCode**.

See Also **BeginInsertion** Method; **EndInsertion** Method; **EndRemoval** 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	**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 is out of paper. (Can only be returned if **AsyncMode** is FALSE.)
Other Values	See **ResultCode**.

See Also "Data Characters and Escape Sequences"

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_SLP_EMPTY 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	**ResultCodeExtended** = OPOS_EPTR_COVER_OPEN: The device was taken out of insertion mode while the printer cover was open. **ResultCodeExtended** = OPOS_EPTR_SLP_EMPTY: The device was taken out of insertion mode without a form being inserted.
Other Values	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_SLP_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.
Other Values	See **ResultCode**.

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

PrintBarCode Method

Syntax **LONG PrintBarCode (LONG** *Station*, **BSTR** *Data*, **LONG** *Symbology*, **LONG** *Height*, **LONG** *Width*, **LONG** *Alignment*, **LONG** *TextPosition***);**

Parameter	Description
Station	The printer station to be used. May be either PTR_S_RECEIPT or PTR_S_SLIP.
Data	Character string to be bar coded. The format of this data depends upon the value of the **BinaryConversion** property. See page 37.
Symbology	Bar code symbol type to use. See values below.
Height	Bar code height. Expressed in the unit of measure given by **MapMode**.
Width	Bar code width. Expressed in the unit of measure given by **MapMode**.
Alignment	Placement of the bar code. See values below.
TextPosition	Placement of the readable character string. See values below.

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.
Other Values	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.

The *Symbology* parameter values for this release are:

Value	**Meaning**
One Dimensional Symbologies	
PTR_BCS_UPCA	UPC-A
PTR_BCS_UPCA_S	UPC-A with supplemental barcode
PTR_BCS_UPCE	UPC-E
PTR_BCS_UPCE_S	UPC-E with supplemental barcode
PTR_BCS_UPCD1	UPC-D1
PTR_BCS_UPCD2	UPC-D2
PTR_BCS_UPCD3	UPC-D3
PTR_BCS_UPCD4	UPC-D4
PTR_BCS_UPCD5	UPC-D5
PTR_BCS_EAN8	EAN 8 (= JAN 8)

PTR_BCS_JAN8	JAN 8 (= EAN 8)
PTR_BCS_EAN8_S	EAN 8 with supplemental barcode
PTR_BCS_EAN13	EAN 13 (= JAN 13)
PTR_BCS_JAN13	JAN 13 (= EAN 13)
PTR_BCS_EAN13_S	EAN 13 with supplemental barcode
PTR_BCS_EAN128	EAN-128
PTR_BCS_TF	Standard (or discrete) 2 of 5
PTR_BCS_ITF	Interleaved 2 of 5
PTR_BCS_Codabar	Codabar
PTR_BCS_Code39	Code 39
PTR_BCS_Code93	Code 93
PTR_BCS_Code128	Code 128
PTR_BCS_OCRA	OCR “A”
PTR_BCS_OCRB	OCR “B”

Two Dimensional Symbologies

PTR_BCS_PDF417	PDF 417
PTR_BCS_MAXICODE	MAXICODE

Special Cases

PTR_BCS_OTHER	If a Service Object defines additional symbologies, they will be greater or equal to this value.

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: • *Station* does not exist • *Station* does not support bar code printing • *Height* or *Width* are zero or too big • *Symbology* is not supported • *Alignment* is invalid or too big • *TextPosition* 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**
Station	The printer station to be used. May be either PTR_S_RECEIPT or PTR_S_SLIP.
FileName	Name of Windows bitmap file. The file must be in uncompressed format.
Width	Printed width of the bitmap to be performed. See values below.
Alignment	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.
Other Values	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.
Other Values	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: • *Station* does not exist • *Station* does not support bitmap printing • *Width* is too big • *Alignment* is invalid or too big
OPOS_E_NOEXIST	*FileName* 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).
The format of this data depends upon the value of the **BinaryConversion** property. See page 37.

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)	If a Carriage Return immediately precedes a Line Feed, or if the line buffer is empty, then it is ignored. Otherwise, the line buffer is printed and the printer does not feed to the next print line. On some printers, print without feed may be directly supported. 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. If the printer does not support either of these features, then Carriage Return acts like a Line Feed. 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.

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 *Station* does not exist. (See the **CapJrnPresent**, **CapRecPresent**, and **CapSlpPresent** properties.)
OPOS_E_EXTENDED	**ResultCodeExtended** = OPOS_EPTR_COVER_OPEN: The printer cover is open.
	ResultCodeExtended = OPOS_EPTR_JRN_EMPTY: The journal station was specified but is out of paper.
	ResultCodeExtended = OPOS_EPTR_REC_EMPTY: The receipt station was specified but is out of paper.
	ResultCodeExtended = OPOS_EPTR_SLP_EMPTY: The slip station was specified, but a form is not inserted.
Other Values	See **ResultCode**.

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).
The format of this data depends upon the value of the **BinaryConversion** property. See page 37.

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)	If a Carriage Return immediately precedes a Line Feed, or if the line buffer is empty, then it is ignored. Otherwise, the line buffer is printed and the printer does not feed to the next print line. On some printers, print without feed may be directly supported. 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. If the printer does not support either of these features, then Carriage Return acts like a Line Feed. 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.

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 *Station* 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	**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 **PrintImmediate** Method; **PrintTwoNormal** Method

PrintTwoNormal Method

Syntax **LONG PrintTwoNormal (LONG** *Stations*, **BSTR** *Data1*, **BSTR** *Data2***);**

Parameter	**Description**
Stations	The printer stations to be used.

OPOS Release 1.3 and later:
Select one of the following:

Stations Parameter	First Station	Second Station
PTR_TWO_RECEIPT_JOURNAL	Receipt	Journal
PTR_TWO_SLIP_JOURNAL	Slip	Journal
PTR_TWO_SLIP_RECEIPT	Slip	Receipt

OPOS Release 1.0 - 1.2:
Select one of the following:

PTR_S_JOURNAL_RECEIPT,
PTR_S_JOURNAL_SLIP, or
PTR_S_RECEIPT_SLIP.

Data1 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.
The format of this data depends upon the value of the **BinaryConversion** property. See page 37.

Data2 The characters to be printed on the second station. (Restrictions are the same as *Data1*.)
If this string is the empty string (“”), then print the same data as *Data1*. On some printers, using this format may give additional increased print performance.
The format of this data depends upon the value of the **BinaryConversion** property. See page 37.

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.

Release 1.0 – 1.2

Documentation releases prior to 1.3 were not sufficiently clear as to the meaning of "first" and "second" station, so implementations varied between the following:

- Assign stations based on order within the constants. For example, PTR_S_JOURNAL_RECEIPT prints *Data1* on the journal and *Data2* on the receipt.
- Assign stations based upon physical device characteristics or internal print order.

Due to this inconsistency, the application should use the new constants if the Control Object and Service Object versions indicate Release 1.3 or later.

Release 1.3 and later

Service Objects for Release 1.3 or later should support both sets of constants. The vendor should define and document the behavior of the obsolete constants.

The sequence of stations in the constants does not imply the physical printing sequence on the stations. The physical sequence depends on the printer, and may be different based on bi-directional printing, multiple print heads, and so on.

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 S*tations* 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**
Station	The printer station to be used. May be PTR_S_RECEIPT or PTR_S_SLIP.
Rotation	Direction of rotation. See values below.

Value	**Meaning**
PTR_RP_RIGHT90	Rotate printing 90º to the right (clockwise).
PTR_RP_LEFT90	Rotate printing 90º to the left (counter-clockwise).
PTR_RP_ROTATE180	Rotate printing 180º, 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 *Station* does not exist (see the **CapJrnPresent**, **CapRecPresent**, and **CapSlpPresent** properties), or the *Station* 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	**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**.

See Also “Data Characters and Escape Sequences”

SetBitmap Method

Syntax **LONG SetBitmap (LONG** *BitmapNumber*, **LONG** *Station*, **BSTR** *FileName*, **LONG** *Width***, LONG** *Alignment***);**

Parameter	**Description**
BitmapNumber	The number to be assigned to this bitmap. Two bitmaps, numbered 1 and 2, may be set.
Station	The printer station to be used. May be either PTR_S_RECEIPT or PTR_S_SLIP.
FileName	Name of Windows bitmap file. The file must be in uncompressed format. If set to an empty string (“”), then the bitmap is unset.
Width	Printed width of the bitmap to be performed. See **PrintBitmap** for values.
Alignment	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.

Release 1.0 – 1.1

Only 2 bitmaps may be set, and each bitmap number may only be used for one station at a time.

Release 1.2 and later

The application may set bitmaps numbered 1 and 2 for each of the two valid *Station*s. If desired, the same bitmap *FileName* may be set to the same *BitmapNumber* for each station, so that the same print bitmap escape sequence may be used for either station.

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: • *BitmapNumber* is invalid • *Station* does not exist • *Station* does not support bitmap printing • *Width* is too big • *Alignment* is invalid or too big
OPOS_E_NOEXIST	*FileName* 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.
Other Values	See **ResultCode**.

See Also "Data Characters and Escape Sequences"; **PrintBitmap** Method

SetLogo Method

Syntax **LONG SetLogo (LONG** *Location*, **BSTR** *Data***);**

Parameter	Description
Location	The logo to be set. May be PTR_L_TOP or PTR_L_BOTTOM.
Data	The characters that produce the logo. May consist of printable characters, escape sequences, carriage returns (13 decimal), and line feeds (10 decimal). The format of this data depends upon the value of the **BinaryConversion** property. See page 37.

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 *Location* was specified.
Other Values	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
Station	The printer station to be used. May be PTR_S_JOURNAL, PTR_S_RECEIPT, or PTR_S_SLIP.
Control	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 *Station* 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 and *Control* is PTR_TP_NORMAL.)
OPOS_E_EXTENDED	**ResultCodeExtended** = OPOS_EPTR_COVER_OPEN: The printer cover is open. (Can only be returned if **AsyncMode** is FALSE and *Control* is PTR_TP_NORMAL.) **ResultCodeExtended** = OPOS_EPTR_JRN_EMPTY: The journal station was specified but is out of paper. (Can only be returned if **AsyncMode** is FALSE and *Control* is PTR_TP_NORMAL.) **ResultCodeExtended** = OPOS_EPTR_REC_EMPTY: The receipt station was specified but is out of paper. (Can only be returned if **AsyncMode** is FALSE and *Control* is PTR_TP_NORMAL.) **ResultCodeExtended** = OPOS_EPTR_SLP_EMPTY: The slip station was specified, but a form is not inserted. (Can only be returned if **AsyncMode** is FALSE and *Control* is PTR_TP_NORMAL.)
Other Values	See **ResultCode**.

ValidateData Method *Added in Release 1.1*

Syntax **LONG ValidateData (LONG** *Station*, **BSTR** *Data***);**

Parameter	Description
Station	The printer station to be used. May be PTR_S_JOURNAL, PTR_S_RECEIPT, or PTR_S_SLIP.
Data	The data to be validated. May include printable data and escape sequences. The format of this data depends upon the value of the **BinaryConversion** property. See page 37.

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.
Underline	The thickness ‘ #’ is not precisely supported: Control will select the closest 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**
*data1***CR***data2***LF**	(Where **CR** is a Carriage Return and **LF** is a Line Feed) In order to print data *data1* and remain on the same line, the Service Object will print with a line advance, then perform a reverse line feed. The data *data2* will then overprint *data1*.

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**
*data1***CR***data2***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 *data1* will print on one line, and the data *data2* 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**
ResultCode	Result code causing the error event. See **ResultCode** for values.
ResultCodeExtended	Extended result code causing the error event. See values below.
ErrorLocus	Set to OPOS_EL_OUTPUT: Error occurred while processing asynchronous output.
pErrorResponse	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 (LONG** *Status***);**

The *Status* 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 delivering the event.

*Power reporting **StatusUpdateEvent** values*

See **StatusUpdateEvent** description on page 68.

Remarks Fired when a significant status change has occurred.

CHAPTER 16

Remote Order Display

Summary

Properties

Common		*Type*	*Access*	*Initialized After*
AutoDisable	1.3	Boolean	R/W	*Not Supported*
BinaryConversion	1.3	Long	R/W	Open
CapPowerReporting	1.3	Long	R	Open
CheckHealthText	1.3	String	R	Open
Claimed	1.3	Boolean	R	Open
DataCount	1.3	Long	R	Open
DataEventEnabled	1.3	Boolean	R/W	Open
DeviceEnabled	1.3	Boolean	R/W	Open; Claim
FreezeEvents	1.3	Boolean	R/W	Open
OutputID	1.3	Long	R	Open
PowerNotify	1.3	Long	R/W	Open
PowerState	1.3	Long	R	Open
ResultCode	1.3	Long	R	--
ResultCodeExtended	1.3	Long	R	Open
State	1.3	Long	R	--
ControlObjectDescription	1.3	String	R	--
ControlObjectVersion	1.3	Long	R	--
ServiceObjectDescription	1.3	String	R	Open
ServiceObjectVersion	1.3	Long	R	Open
DeviceDescription	1.3	String	R	Open
DeviceName	1.3	String	R	Open

Properties (continued)

Specific		*Type*	*Access*	*Initialized After*
CapTransaction	1.3	Boolean	R	Open
AsyncMode	1.3	Boolean	R/W	Open, Claim, & Enable
EventType	1.3	Long	R/W	Open
SystemClocks	1.3	Long	R	Open. Claim, Enable
SystemVideoSaveBuffers	1.3	Long	R	Open, Claim, & Enable
Timeout	1.3	Long	R/W	Open
UnitsOnline	1.3	Long	R	Open, Claim, & Enable
CurrentUnitID	1.3	Long	R/W	Open, Claim, & Enable
CapSelectCharacterSet	1.3	Boolean	R	Open, Claim, & Enable (*)
CapTone	1.3	Boolean	R	Open, Claim, & Enable (*)
CapTouch	1.3	Boolean	R	Open, Claim, & Enable (*)
AutoToneDuration	1.3	Long	R/W	Open, Claim, & Enable
AutoToneFrequency	1.3	Long	R/W	Open, Claim, & Enable
CharacterSet	1.3	Long	R	Open, Claim, & Enable
CharacterSetList	1.3	String	R	Open, Claim, & Enable
Clocks	1.3	Long	R	Open, Claim, & Enable (*)
VideoDataCount	1.3	Long	R	Open, Claim, & Enable (*)
VideoMode	1.3	Long	R/W	Open, Claim, & Enable (*)
VideoModesList	1.3	String	R	Open, Claim, & Enable (*)
VideoSaveBuffers	1.3	Long	R	Open, Claim, & Enable (*)
ErrorUnits	1.3	Long	R	Open
ErrorString	1.3	String	R	Open
EventUnitID	1.3	Long	R	Open, Claim
EventUnits	1.3	Long	R	Open, Claim
EventString	1.3	String	R	Open, Claim

Methods

Common		*Prerequisites*
Open	1.3	None
Close	1.3	Open
Claim	1.3	Open
Release	1.3	Open, Claim
CheckHealth	1.3	Open, Claim, & Enable
ClearInput	1.3	Open, Claim
ClearOutput	1.3	Open, Claim
DirectIO	1.3	Open
Specific		
ControlClock	1.3	Open, Claim, & Enable
ControlCursor	1.3	Open, Claim, & Enable
FreeVideoRegion	1.3	Open, Claim, & Enable
ResetVideo	1.3	Open, Claim, & Enable
SelectChararacterSet	1.3	Open, Claim, & Enable
SetCursor	1.3	Open, Claim, & Enable
ClearVideo	1.3	Open, Claim, & Enable
ClearVideoRegion	1.3	Open, Claim, & Enable
CopyVideoRegion	1.3	Open, Claim, & Enable
DisplayData	1.3	Open; Claim; Enable
DrawBox	1.3	Open, Claim, & Enable
RestoreVideoRegion	1.3	Open, Claim, & Enable
SaveVideoRegion	1.3	Open, Claim, & Enable
UpdateVideoRegion Attribute	1.3	Open, Claim, & Enable
VideoSound	1.3	Open, Claim, & Enable
TransactionDisplay	1.3	Open, Claim, & Enable

Events

Name		*May Occur After*
DataEvent	1.3	Open, Claim, & Enable
DirectIOEvent	1.3	Open, Claim
ErrorEvent	1.3	Open, Claim, & Enable
OutputCompleteEvent	1.3	Open, Claim, & Enable
StatusUpdateEvent	1.3	Open, Claim, & Enable

General Information

The Remote Order Display Control' s OLE Programmatic ID is "OPOS.RemoteOrderDisplay".

This device was added in OPOS Release 1.3.

Capabilities

The Remote Order Display Control has the following minimal set of capabilities:

- Supports color or monochrome text character displays.
- Supports 8 foreground colors (or gray scale on monochrome display) with the option of using the intensity attribute.
- Supports 8 background colors (or gray scale on monochrome display) with the option of using only a blinking attribute.
- The individual event types can be disabled such that the application only receives a subset of data events if requested.
- Supports video region buffering.
- Supports cursor functions.
- Supports clock functions.
- Supports resetting a video unit to power on state.

The Remote Order Display Control may also have the following additional capabilities:

- Supports multiple video displays each with possibly different video modes.
- Supports touch video input for a touch screen display unit.
- Supports video enunciator output with frequency and duration.
- Supports tactile feedback via an automatic tone when a video display unit is touched (for touch screen only).
- Supports downloading alternate character sets to one or many video units.
- Support transaction mode display output to one or many video units.

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

- Support for graphical displays, where the video display is addressable by individual pixels or dots. The addition of this support is under investigation for future revisions.

Model

The general model of a remote order display:

- The remote order display device class is a subsystem of video units. The initial targeted environment is food service, to display order preparation and fulfillment information. Remote order displays are often used in conjunction with bump bars.

  The subsystem can support up to 32 video units.

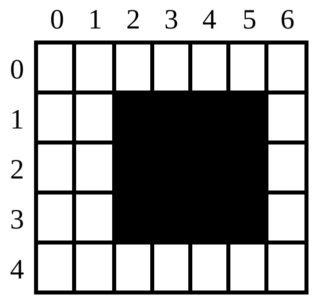
  One Application on one PC or POS Terminal will typically manage and control the entire subsystem of video units. If Applications on the same or other PCs and POS Terminals will need to access the subsystem, then this Application must act as a subsystem server and expose interfaces to other Applications.

- All specific methods are broadcast methods. This means that the method can apply to one unit, a selection of units or all online units. The *Units* parameter is a **Long**, with each bit identifying an individual video unit. (One or more of the constants ROD_UID_1 through ROD_UID_32 are bitwise ORed to form the bitmask.) The service object will attempt to satisfy the method for all units specified by the *Units* parameter. If an error is received from one or more units, the **ErrorUnits** property is updated with the appropriate units in error. The **ErrorString** property is updated with a description of the error or errors received. The method will then return with the corresponding OPOS error. In the case where two or more units encounter different errors, the service object should determine the most severe OPOS error to return.
- The common methods **CheckHealth**, **ClearInput**, and **ClearOutput** are not broadcast methods and use the unit ID specified by the **CurrentUnitID** property. (One of the constants ROD_UID_1 through ROD_UID_32 are selected.) See the description of these common methods to understand how the current unit ID property is used.
- When the current unit ID property is set by the application, all the corresponding properties are updated to reflect the settings for that unit.

  If the current unit ID property is set to a unit ID that is not online, the dependent properties will contain non-initialized values.

  The **CurrentUnitID** uniquely represents a single video unit. The definitions range from ROD_UID_1 to ROD_UID_32. These definitions are also used to create the bitwise parameter, *Units,* used in the broadcast methods. See the Examples section below for usage.

- The rows and columns are numbered beginning with (0,0) at the top-left corner of the video display. The dimensions are defined by the height and width parameters. The region depicted below would have the parameters *Row = 1, Column = 2, Height = 3, and Width = 4*.



All position parameters are expressed in text characters.

- The VGA-like *Attribute* parameter, that is used in various methods, is a **Long**. Bits 7-0 define the text attribute and bits 31-8 are reserved and must be 0, otherwise an OPOS_E_ILLEGAL error will be returned. The following table defines bits 7-0:

Bit 7	**Bit 6**	**Bit 5**	**Bit 4**	**Bit 3**	**Bit 2**	**Bit 1**	**Bit 0**
Blinking	Background Color			Intensity	Foreground Color		

If a foreground or background color is requested, but the service object does not support that color, it chooses the best fit from the colors supported.

The following constants may be used, with up to one constant selected from each category:

♦ Blinking: ROD_ATTR_BLINK

♦ Background Color: ROD_ATTR_BG_*color*, where *color* is replaced by BLACK, BLUE, GREEN, CYAN, RED, MAGENTA, BROWN, or GRAY

♦ Intensity: ROD_ATTR_INTENSITY

♦ Foreground Color: ROD_ATTR_FG_*color*, where *color* is replaced by BLACK, BLUE, GREEN, CYAN, RED, MAGENTA, BROWN, or GRAY

See the examples section below for usage.

Input – Touch Video

The Remote Order Display Control follows the general "Input Model" for event-driven input with some differences:

- When input is received by the Control, it enqueues a **DataEvent**.
- This device does not support the **AutoDisable** property, so the control will not automatically disable itself when a **DataEvent** is enqueued.
- An enqueued **DataEvent** is delivered to the application when the **DataEventEnabled** property is TRUE and other event delivery requirements are met. Just before delivering this event, the Control copies the data into properties, and disables further data events by setting the **DataEventEnabled** property to FALSE. This causes subsequent input data to be enqueued by 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.
- An **ErrorEvent** (or events) are enqueued if the Control encounters an error while gathering or processing input, and is delivered to the application when the **DataEventEnabled** property is TRUE and other event delivery requirements are met.
- The **VideoDataCount** property may be read to obtain the number of video **DataEvent**s for a specific unit ID enqueued by the Control. The **DataCount** property can be read to obtain the total number of data events enqueued by the Control.
- Input enqueued by the Control may be deleted by calling the **ClearInput** method. See **ClearInput** method description for more details.

Output – Video and Tone

The Remote Order Display Control follows the general "Output Model", with some enhancements:

- The following methods are always performed synchronously: **ControlClock**, **ControlCursor**, **SelectChararacterSet**, **ResetVideo**, and **SetCursor**. These methods will fail if asynchronous output is outstanding. The following method is also always performed synchronously but without regard to outstanding asynchronous output: **FreeVideoRegion**.
- The following methods are performed either synchronously or asynchronously, depending on the value of the **AsyncMode** property: **ClearVideo**, **ClearVideoRegion**, **CopyVideoRegion**, **DisplayData**, **DrawBox**, **RestoreVideoRegion**, **SaveVideoRegion**, **TransactionDisplay**, **UpdateVideoRegionAttribute**, and **VideoSound**. When **AsyncMode** is FALSE, then these methods operate 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 updates the **EventUnits** property and fires an **OutputCompleteEvent**. A parameter of this event contains the output ID of the completed request.

    Asynchronous display methods will <u>not</u> return an error status due to a display problem, such as communications failure. These errors will only be reported by an **ErrorEvent**. An error status is returned only if the display 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 enqueued and delivered. The **EventUnits** property is set to the unit or units in error. The **EventString** property is also set. *<u>Note</u>: **ErrorEvent** updates **EventUnits** and **EventString**. If an error is reported by a broadcast method, then **ErrorUnits** and **ErrorString** are set instead.*

    The event handler may call synchronous display methods (but not asynchronous methods), then can either retry the outstanding output or clear it.
  - ♦ The Control guarantees that asynchronous output is performed on a first-in first-out basis.

- ♦ All unit output buffered by the Control may be deleted by setting the **CurrentUnitID** property and calling the **ClearOutput** method. **OutputCompleteEvent**s will not be fired for cleared output. This method also stops any output that may be in progress (when possible).

- The Remote Order Display Control device may support transaction mode. A transaction is a sequence of display operations that are sent to a video unit as a single unit. Display operations which may be included in a transaction are **ClearVideo**, **ClearVideoRegion**, **CopyVideoRegion**, **DisplayData**, **DrawBox**, **RestoreVideoRegion**, **SaveVideoRegion**, and **UpdateVideoRegionAttribute**. During a transaction, the display operations are first validated. If valid, they are added to the transaction but not displayed yet. Once the application has added as many operations as required, then the transaction display method is called.

  If the transaction is displayed synchronously, then the returned status indicates either that the entire transaction displayed successfully or that an error occurred during the display. If the transaction is displayed asynchronously, then the asynchronous display rules listed above are followed. If an error occurs and the Error Event handler causes a retry, the entire transaction is retried.

Examples

Set up an attribute variable and initializes it for various uses.

```
 'Standard white foreground on black background
lAttribute = ROD_ATTR_BG_BLACK | ROD_ATTR_FG_GRAY

 'Turn Blinking on
lAttribute = lAttribute | ROD_ATTR_BLINK
```

Draws a box with a solid border on unit ID 1 and unit ID 4. The box is located at the top left corner (0,0) with a height of 80 and a width of 25.

```
ROD.DrawBox( ROD_UID_1 | ROD_UID_4, 0, 0, 80, 25, lAttribute,
   ROD_BDR_SOLID )
```

Device Sharing

The remote order 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 many remote order display specific properties.
- The application must claim and enable the device before calling methods that manipulate the device.
- When a **Claim** method is called again, settable device characteristics are restored to their condition at **Release**. Examples of restored characteristics are character set, video mode, and tone frequency. Region memory buffers, clock and cursor settings are considered state characteristics and are not restored.
- See the "Summary" table for precise usage prerequisites.

Properties

AsyncMode Property R/W

Syntax **BOOL AsyncMode;**

Remarks If TRUE, then the **ClearVideo**, **ClearVideoRegion**, **CopyVideoRegion**, **DisplayData**, **DrawBox**, **RestoreVideoRegion**, **SaveVideoRegion**, **TransactionDisplay**, **UpdateVideoRegionAttribute**, and **VideoSound** methods will be performed asynchronously.
If FALSE, they will be performed 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.

AutoToneDuration Property R/W

Syntax **LONG AutoToneDuration;**

Remarks Sets the duration (in milliseconds) of the automatic tone for the video unit specified by the **CurrentUnitID** property.

This property is initialized to the default value for each online video unit 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 illegal value was specified. The **ErrorString** property is updated before return.

See Also **CurrentUnitID** Property

AutoToneFrequency Property R/W

Syntax **LONG AutoToneFrequency;**

Remarks Sets the frequency (in Hertz) of the automatic tone for the video unit specified by the **CurrentUnitID** property.

This property is initialized to the default value for each online video unit 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 illegal value was specified. The **ErrorString** property is updated before return.

See Also **CurrentUnitID** Property

CapSelectCharacterSet Property

Syntax **BOOL CapSelectCharacterSet;**

Remarks If TRUE, the video unit specified by the **CurrentUnitID** property may be loaded with an alternate, user supplied character set; otherwise it is FALSE.

This property is initialized for each video unit online when the device is first enabled following the **Open** method.

See Also **CurrentUnitID** Property

CapTone Property

Syntax **BOOL CapTone;**

Remarks If TRUE, the video unit specified by the **CurrentUnitID** property supports an enunciator; otherwise it is FALSE

This property is initialized for each video unit online when the device is first enabled following the **Open** method.

See Also **CurrentUnitID** Property

CapTouch Property

Syntax **BOOL CapTouch;**

Remarks If TRUE, the video unit specified by the **CurrentUnitID** property supports the ROD_DE_TOUCH_UP, ROD_DE_TOUCH_DOWN, and ROD_DE_TOUCH_MOVE event types; otherwise it is FALSE.

This property is initialized for each video unit online when the device is first enabled following the **Open** method.

See Also **CurrentUnitID** Property; **DataEvent** Event

CapTransaction Property

Syntax **BOOL CapTransaction;**

Remarks If TRUE, then transactions are supported by each video unit;
otherwise it is FALSE.

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

CharacterSet Property

Syntax **LONG CharacterSet;**

Remarks Contains the character set for displaying characters for the video unit specified by the **CurrentUnitID** property. When **CapSelectCharacterSet** is TRUE, this property can be set with one of the character set numbers found in the **CharacterSetList** property.

This property is initialized to the default video character set used by each video unit online when the device is first enabled following the **Open** method.

This is updated during the **SelectCharacterSet** method.

See Also **CurrentUnitID** Property; **CharacterSetList** Property; **CapSelectCharacterSet** Property, **SelectCharacterSet** Method

CharacterSetList Property

Syntax **BSTR CharacterSetList;**

Remarks A string of character set numbers for the video unit specified by the **CurrentUnitID** property.

If **CapSelectCharacterSet** is TRUE, this property is initialized for each video unit online when the device is first enabled following the **Open** method; otherwise, this property is initialized with the string "[Error]".

The character set number string consists of an ASCII numeric set of numbers, separated by commas.

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

The character set number 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.
ROD_CS_ASCII	The ASCII character set, supporting the ASCII characters between 20-hex and 7F-hex. The value of this constant is 998.
ROD_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.

See Also **CurrentUnitID** Property; **CharacterSet** Property; **CapSelectCharacterSet** Property, **SelectCharacterSet** Method

Clocks Property

Syntax **LONG Clocks;**

Remarks Indicates the number of clocks the video unit, specified by the **CurrentUnitID** property, can support.

This property is initialized for each online video unit when the device is first enabled following the **Open** method.

See Also **CurrentUnitID** Property

CurrentUnitID Property R/W

Syntax **LONG CurrentUnitID;**

Remarks Selects the current video unit ID. Up to 32 units are allowed on one remote order display device. The unit ID definitions range from ROD_UID_1 to ROD_UID_32.

The following properties and methods apply only to the selected video unit ID:

- Properties: **AutoToneDuration, AutoToneFrequency, CapSelectCharacterSet, CapTone, CapTouch, CharacterSet, CharacterSetList, Clocks, VideoDataCount, VideoMode, VideoModesList, VideoSaveBuffers.**

  Setting **CurrentUnitID** will update these properties to the current values for the specified unit.
- Methods: **CheckHealth, ClearInput, ClearOutput.**

This property is initialized to ROD_UID_1 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 illegal unit id was specified. The **ErrorString** property is updated before return.

DataCount Property (Common)

Syntax **LONG DataCount;**

Remarks Indicates the total number of **DataEvent**s enqueued at the control. All units online are included in this value. The number of enqueued events for a specific unit ID is stored in the **VideoDataCount** property.

The application may interrogate **DataCount** to determine whether additional input is enqueued from a device, but has not yet been delivered because of other application processing, freezing of events, or other causes.

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

See Also "Input Model"; **VideoDataCount** Property; **DataEvent** Event

ErrorString Property

Syntax **BSTR ErrorString;**

Remarks When an error occurs for any method that acts on a bitwise set of video units, the **ErrorString** is set to a description of the error which occurred to the unit(s) specified by the **ErrorUnits** property.

If an error occurs during processing of an asynchronous request, the **ErrorEvent** updates the property **EventString** instead.

This property is initialized to an empty string by the **Open** method.

See Also **ErrorUnits** Property

ErrorUnits Property

Syntax **LONG ErrorUnits;**

Remarks When an error occurs for any method that acts on a bitwise set of video units, the **ErrorUnits** will contain a bitwise mask of the unit(s) that encountered an error.

If an error occurs during processing of an asynchronous request, the **ErrorEvent** updates the property **EventUnits** instead.

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

See Also **ErrorString** Property

EventString Property

Syntax **BSTR EventString;**

Remarks When an **ErrorEvent** is delivered, this property is set to a description of the error which occurred to the unit(s) specified by the **EventUnits** property.

This property is initialized to an empty string by the **Open** method.

See Also **EventUnits** Property; **ErrorEvent**

EventType Property R/W

Syntax **LONG EventType;**

Remarks A bitwise mask that is used to selectively indicate which event types are to be fired by the **DataEvent**, for all video units online. See the **DataEvent** description for event type definitions.

For example if the ROD_DE_TOUCH_MOVE event is not desired:

```
ROD.EventType = ROD_DE_TOUCH_UP | ROD_DE_TOUCH_DOWN
```

This property is initialized to all defined event types 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 illegal event type value was specified. The **ErrorString** property is updated before return.

See Also **DataEvent** Event

EventUnitID Property

Syntax **LONG EventUnitID;**

Remarks Just before the Control delivers a **DataEvent** to the Application, it sets this property to the video unit ID causing the event. The unit ID definitions range from ROD_UID_1 to ROD_UID_32.

See Also **DataEvent**

EventUnits Property

Syntax **LONG EventUnits;**

Remarks When an **OutputCompleteEvent,** output **ErrorEvent**, or **StatusUpdateEvent** is fired, the **EventUnits** property will contain a bitwise mask of the unit(s).

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

See Also **OutputCompleteEvent**, **ErrorEvent, StatusUpdateEvent**

SystemClocks Property

Syntax **LONG SystemClocks;**

Remarks Indicates the total number of clocks the remote order display device can support at one time.

This property is initialized when the device is first enabled following the **Open** method.

See Also **Clocks** Property

SystemVideoSaveBuffers Property

Syntax **LONG SystemVideoSaveBuffers;**

Remarks Indicates the total number of video save buffers the remote order display device can support at one time.

This property is initialized when the device is first enabled following the **Open** method.

See Also **VideoSaveBuffers** Property

Timeout Property R/W

Syntax **LONG Timeout;**

Remarks Timeout value in milliseconds used by the remote order display device to complete all output methods supported. If the device cannot successfully complete an output method within the timeout value, then the method returns a failure status if **AsyncMode** is FALSE, or enqueues an **ErrorEvent** if **AsyncMode** is TRUE.

This property is initialized to a Service Object dependent default timeout 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 illegal timeout value was specified. The **ErrorString** property is updated before return.

See Also **AsyncMode** Property

UnitsOnline Property

Syntax **LONG UnitsOnline;**

Remarks Bitwise mask indicating the video units online, where zero or more of the unit constants ROD_UID_1 (bit 0 on) through ROD_UID_32 (bit 31 on) are bitwise ORed.

This property is initialized when the device is first enabled following the **Open** method. This property is updated as changes are detected, such as before a **StatusUpdateEvent** is fired and during the **CheckHealth** method.

See Also **CheckHealth** Method**; StatusUpdateEvent** Event

VideoDataCount Property

Syntax **LONG VideoDataCount;**

Remarks Indicates the number of **DataEvent**s enqueued for the video unit specified by the **CurrentUnitID** property.

The application may interrogate **VideoDataCount** to determine whether additional input is enqueued by a video unit, but has not yet been delivered because of other application processing, freezing of events, or other causes.

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

See Also **CurrentUnitID** Property; **DataEvent** Event

VideoMode Property R/W

Syntax **LONG VideoMode;**

Remarks Indicates the video *ModeId* selected for the video unit specified by the **CurrentUnitID** property. The *ModeId* represents one of the selections in the **VideoModesList** property.

This property is initialized to the Service Object dependent default video *ModeId* used by each video unit online 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	The desired video mode is not supported. The **ErrorString** property is updated before return.
OPOS_E_FAILURE	An error occurred while communicating with the video unit specified by the **CurrentUnitID** property. The **ErrorString** property is updated before return.

See Also **CurrentUnitID** Property; **VideoModesList** Property

VideoModesList Property

Syntax **BSTR VideoModesList;**

Remarks The video modes supported for the video unit specified by the **CurrentUnitID** property. The video modes are listed in a comma delineated string with the following format:
<*ModeId*>:<*Height*>x<Width>x<*NumberOfColors*><M|C>.
The *ModeId* values are determined by the remote order display system.
M = Monochrome (and gray scales) and C = Color.

For example, if the string is "1:40x25x16C,2:80x25x16C", then the video unit supports two video modes, *ModeId* 1 and *ModeId* 2. *ModeId* 1 has 40 rows, 25 columns, 16 colors, and is Color. *ModeId* 2 has 80 rows, 25 columns, 16 colors, and is Color.

The *ModeId* is used to initialize the **VideoMode** property for each video unit online.

This property is initialized to the video modes list supported by each video unit online when the device is first enabled following the **Open** method.

See Also **CurrentUnitID** Property; **VideoMode** Property

VideoSaveBuffers Property

Syntax **LONG VideoSaveBuffers;**

Remarks Indicates the number of save buffers for the video unit specified by the **CurrentUnitID** property. This property should be consulted when using the **SaveVideoRegion, RestoreVideoRegion** and **FreeVideoRegion** methods. When set to 0, this indicates that buffering for the selected unit is not supported. When **VideoSaveBuffers** is greater than 0, the remote order display device can save at minimum one entire video screen for the selected video unit.

This property is initialized for each video unit online when the device is first enabled following the **Open** method.

See Also **CurrentUnitID** Property; **SaveVideoRegion** Method**; RestoreVideoRegion** Method; **FreeVideoRegion** Method

Methods

CheckHealth Method (Common)

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 displayed on the video.
OPOS_CH_INTERACTIVE	Perform an interactive test of the device. The Service Object will typically display a modal dialog box to present test options and results.

Remarks When OPOS_CH_INTERNAL or OPOS_CH_EXTERNAL level is requested, the method will check the health of the unit specified by the **CurrentUnitID** property. When the current unit ID property is set to a unit that is not currently online, the device will attempt to check the health of the video unit and report a communication error if necessary. The OPOS_CH_INTERACTIVE health check operation is up to the service object designer.

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

The **UnitsOnline** property will be updated with any changes before returning to the application.

The **CheckHealth** method is always synchronous.

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

Value	Meaning
OPOS_SUCCESS	Indicates that the health check 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_FAILURE	An error occurred while communicating with the video unit specified by the **CurrentUnitID** property.
Other Values	See **ResultCode**.

See Also **CurrentUnitID** Property**; UnitsOnline** Property

ClearInput Method (Common)

Syntax **LONG ClearInput ();**

Remarks Called to clear the device input that has been buffered for the unit specified by the **CurrentUnitID** property.

Any data events that are enqueued – usually waiting for **DataEventEnabled** to be set to TRUE and **FreezeEvents** to be set to FALSE – are also cleared.

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	The device is claimed by another process.
OPOS_E_NOTCLAIMED	The device must be claimed before this method can be used.

See Also **CurrentUnitID** Property; "Input Model"

ClearOutput Method (Common)

Syntax **LONG ClearOutput ();**

Remarks Called to clear all outputs that have been buffered for the unit specified by the **CurrentUnitID** property, including video and tone outputs.

Any output complete and output error events that are enqueued – usually waiting for **DataEventEnabled** to be set to TRUE and **FreezeEvents** to be set to FALSE – are also cleared.

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	The device is claimed by another process.
OPOS_E_NOTCLAIMED	The device must be claimed before this method can be used.

See Also **CurrentUnitID** Property; "Output Model"

ClearVideo Method

Syntax **LONG ClearVideo (LONG** *Units*, **LONG** *Attribute***);**

Parameter	Description
Units	Bitwise mask indicating which video unit(s) to operate on.
Attribute	See "Model" discussion in the General Information section.

Remarks This method will clear the entire display area for the video unit(s) specified by the *Units* parameter. The display area will be cleared using the attribute placed in the *Attribute* parameter.

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	One of the following errors occurred: • *Attribute* is illegal. • *Units* is zero or a non-existent unit was specified.
OPOS_E_FAILURE	An error occurred while communicating with one of the video units specified by *Units.* The **ErrorUnits** and **ErrorString** properties are updated before return. (Can only be returned if **AsyncMode** is FALSE.)
Other Values	See **ResultCode**.

See Also **AsyncMode** Property; "Model" discussion

ClearVideoRegion Method

Syntax **LONG ClearVideoRegion (LONG** *Units,* **LONG** *Row*, **LONG** *Column*, **LONG** *Height*, **LONG** *Width*, **LONG** *Attribute***);**

Parameter	Description
Units	Bitwise mask indicating which video unit(s) to operate on.
Row	The region's start row.
Column	The region's start column.
Height	The number of rows in the region.
Width	The number of columns in the region.
Attribute	See "Model" discussion in the General Information section.

Remarks This method will clear the specified video region for the video unit(s) specified by the *Units* parameter. The display area will be cleared using the attribute placed in the *Attribute* parameter.

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	One of the following errors occurred: • *Row*, *Column*, *Height*, or *Width* are out of range. • *Attribute* is illegal. • *Units* is zero or a non-existent unit was specified. The **ErrorUnits** and **ErrorString** properties may be updated before return.
OPOS_E_FAILURE	An error occurred while communicating with one of the video units specified by *Units*. The **ErrorUnits** and **ErrorString** properties are updated before return. (Can only be returned if **AsyncMode** is FALSE.)
Other Values	See **ResultCode**.

See Also **AsyncMode** Property; **ErrorString** Property; **ErrorUnits** Property; “Model” discussion

ControlClock Method

Syntax **LONG ControlClock (LONG** *Units*, **LONG** *Function*, **LONG** *ClockId*,
LONG *Hour*, **LONG** *Min*, **LONG** *Sec*, **LONG** *Row*, **LONG** *Column*,
LONG *Attribute*, **LONG** *Mode***);**

Parameter	**Description**
Units	Bitwise mask indicating which video unit(s) to operate on.
Function	The requested clock command. See values below.
ClockId	Clock identification number. The valid values can be from 1 **- Clocks**. When the *Function* parameter is ROD_CLK_PAUSE, ROD_CLK_RESUME, or ROD_CLK_STOP then *ClockId* can be ROD_CLK_ALL to specify all clocks started on the specified video unit(s).
Hour	The initial hours for the clock display.
Min	The initial minutes for the clock display.
Sec	The initial seconds for the clock display.
Row	The clock’ s row.
Column	The clock’ s start column.
Attribute	See “Model” discussion in the General Information section.
Mode	The type of clock to display. See values below.

The *Function* parameter values are:

Value	Meaning
ROD_CLK_START	Starts a clock display assigned to the given *ClockId*.
ROD_CLK_PAUSE	Temporarily stops a clock from updating the display until a ROD_CLK_RESUME requested.
ROD_CLK_RESUME	Resumes a clock that was previously paused, such that display updates continue.
ROD_CLK_STOP	Permanently stops the clock from updating the display and the *ClockId* becomes free.
ROD_CLK_MOVE	Moves an instantiated clock to a new position.

The *Mode* parameter values are:

Value	Meaning
ROD_CLK_SHORT	Displays a clock with "M:SS" format.
ROD_CLK_NORMAL	Displays a clock with "MM:SS" format.
ROD_CLK_12_LONG	Displays a 12 hour clock with "HH:MM:SS" format.
ROD_CLK_24_LONG	Displays a 24 hour clock with "HH:MM:SS" format.

Remarks This method will carryout the clock command requested in the *Function* parameter on the video unit(s) specified by the *Units* parameter. The clock will be displayed in the requested *Mode* format at the location found in the *Row* and *Column* parameters.

The clock will start at the specified *Hour, Min,* and *Sec,* time values and will be updated every second until a ROD_CLK_PAUSE or ROD_CLK_STOP is requested for this *ClockId*.

When a ROD_CLK_PAUSE, ROD_CLK_RESUME, or ROD_CLK_STOP command is issued, the *Hour*, *Min, Sec, Left*, *Top*, *Attribute*, and *Mode* parameters are ignored. During a ROD_CLK_PAUSE command, the clock display updates are suspended. During a ROD_CLK_RESUME command, the clock updates continue.

If a ROD_CLK_PAUSE, ROD_CLK_RESUME, ROD_CLK_STOP or ROD_CLK_MOVE command is requested on an uninitialized *ClockId* for any of the video units specified by the *Units* parameter, an OPOS_EROD_BADCLK is returned. If a ROD_CLK_RESUME command is requested without doing a ROD_CLK_PAUSE, this has no effect and no error is returned.

When a ROD_CLK_MOVE command is issued, the clock is moved to the new location found in the *Row* and *Column* parameters. The *Hour*, *Min*, *Sec*, *Attribute* and *Mode* parameters are ignored for this command function.

Generally a video unit can support the number of clocks specified by the **Clocks** property. However, the ROD_CLK_START command will return OPOS_EROD_NOCLOCKS if it exceeds the number of **SystemClocks** even though the **Clocks** property may indicated the unit can support more clocks than allocated for that unit.

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: • *ClockId*, *Hour*, *Min*, *Sec*, *Row*, or *Column* are out of range. • *Function*, *Attribute* or *Mode* is illegal. • *Units* is zero or a non-existent unit was specified. The **ErrorUnits** and **ErrorString** properties are updated before return.
OPOS_E_EXTENDED	**ResultCodeExtended** = OPOS_EROD_BADCLK: A ROD_CLK_PAUSE, ROD_CLK_RESUME, ROD_CLK_START, ROD_CLK_MOVE command was requested and the specified *ClockId* has not been initialized by the ROD_CLK_START command. **ResultCodeExtended** = OPOS_EROD_NOCLOCKS: The ROD_CLK_START failed because the number of **SystemClocks** has been reached. The **ErrorUnits** and **ErrorString** properties are updated before return.
OPOS_E_FAILURE	An error occurred while communicating with one of the video units specified by the *Units* parameter. The **ErrorUnits** and **ErrorString** properties are updated before return.
OPOS_E_BUSY	A ROD_CLK_START command was requested but the specified *ClockId* is in use. The **ErrorUnits** and **ErrorString** properties are updated before return.
Other Values	See **ResultCode**.

See Also **Clocks** Property; **ErrorString** Property; **ErrorUnits** Property; "Model" discussion

ControlCursor Method

Syntax **LONG ControlCursor (LONG** *Units*, **LONG** *Function***);**

Parameter	Description
Units	Bitwise mask indicating which video unit(s) to operate on.
Function	The cursor command, indicating the type of cursor to display. See values below.

Value	Meaning
ROD_CRS_LINE	Enable a solid underscore line.
ROD_CRS_LINE_BLINK	Enable a blinking solid underscore cursor.
ROD_CRS_BLOCK	Enable a solid block cursor.
ROD_CRS_BLOCK_BLINK	Enable a blinking solid block cursor.
ROD_CRS_OFF	Disable cursor.

Remarks This method will enable or disable the cursor depending on the *Function* parameter, for the video unit(s) specified by the *Units* parameter.

When the *Function* is ROD_CRS_OFF, the cursor is disabled, otherwise the cursor is enabled as the requested cursor type. If the video unit cannot support the requested cursor type, the service object will use the next closest cursor type.

The cursor attribute is taken from the current cursor location.

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: • *Function* is illegal. • *Units* is zero or a non-existent unit was specified.
OPOS_E_FAILURE	An error occurred communicating with one of the video units specified by *Units*. The **ErrorUnits** and **ErrorString** properties are updated before return.
Other Values	See **ResultCode**.

See Also **ErrorString** Property; **ErrorUnits** Property

CopyVideoRegion Method

Syntax **LONG CopyVideoRegion (LONG** *Units*, **LONG** *Row*, **LONG** *Column*, **LONG** *Height*, **LONG** *Width*, **LONG** *TargetRow*, **LONG** *TargetColumn***);**

Parameter	Description
Units	Bitwise mask indicating which video unit(s) to operate on.
Row	The region’ s start row.
Column	The region’ s start column.
Height	The number of rows in the region.
Width	The number of columns in the region.
TargetRow	The start row of the target location.
TargetColumn	The start column of the target location.

Remarks This method will copy a region of the display area to a new location on the display area for the video unit(s) specified by the *Units* parameter. The source area is defined by the *Row*, *Column*, *Height*, and *Width* parameters. The top-left corner of the target location is defined by the *TargetRow* and *TargetColumn* parameters. If the ranges overlap the copy is done such that all original data is preserved.

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	One of the following errors occurred: • *Row*, *Column*, *Height*, *Width*, *TargetRow*, or *TargetColumn* are out of range. • *Units* is zero or a non-existent unit was specified. The **ErrorUnits** and **ErrorString** properties are updated before return.
OPOS_E_FAILURE	An error occurred while communicating with one of the video units specified by *Units.* The **ErrorUnits** and **ErrorString** properties are updated before return. (Can only be returned if **AsyncMode** is FALSE.)
Other Values	See **ResultCode**.

See Also **AsyncMode** Property; **ErrorString** Property; **ErrorUnits** Property; "Model" discussion

DisplayData Method

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

Parameter	Description
Units	Bitwise mask indicating which video unit(s) to operate on.
Row	The start row for the text.
Column	The start column for the text.
Attribute	The video attribute. See "Model" discussion in the General Information section.
Data	The string of characters to display. The format of this data depends upon the value of the **BinaryConversion** property. See page 31.

Remarks The characters in *Data* are processed beginning at the location specified by *Row* and *Column*, and continue in succeeding columns on the video unit(s) specified by the *Units* parameter. Any characters that extend beyond the last column will be discarded.

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	One of the following errors occurred: • *Row* or *Column* parameters are out of range. • *Attribute* is illegal.*Units* is zero or a non-existent unit was specified. The **ErrorUnits** and **ErrorString** properties are updated before return.
OPOS_E_FAILURE	An error occurred while communicating with one of the video units specified by *Units.* The **ErrorUnits** and **ErrorString** properties are updated before return. (Can only be returned if **AsyncMode** is FALSE.)
Other Values	See **ResultCode**.

See Also **AsyncMode** Property; **ErrorString** Property; **ErrorUnits** Property; “Model” discussion

DrawBox Method

Syntax **LONG DrawBox (LONG** *Units*, **LONG** *Row*, **LONG** *Column*, **LONG** *Height*, **LONG** *Width*, **LONG** *Attribute*, **LONG** *BorderType*);

Parameter	Description
Units	Bitwise mask indicating which video unit(s) to operate on.
Row	The box' s start row.
Column	The box' s start column.
Height	The number of rows in the box.
Width	The number of columns in the box.
Attribute	The video attribute. See "Model" discussion in the General Information section.
BorderType	The border type to be drawn. Can be any printable character or a defined border type. See values below.

Value	Meaning
ROD_BDR_SINGLE	A single line border.
ROD_BDR_DOUBLE	A double line border.
ROD_BDR_SOLID	A solid block border.

Remarks This method will draw a box on the video units(s) specified by the *Units* parameter.

The remote order display will attempt to draw a box with the border type specified. If the character set does not support the chosen border type, the service object will choose the best fit from the given character set.

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	One of the following errors occurred: • *Row*, *Column*, *Height*, or *Width* are out of range. • *Attribute* or *BorderType* are illegal. • *Units* is zero or a non-existent unit was specified. The **ErrorUnits** and **ErrorString** properties are updated before return.
OPOS_E_FAILURE	An error occurred while communicating with one of the displays specified by *Units*. The **ErrorUnits** and **ErrorString** properties are updated before return.
Other Values	See **ResultCode**.

See Also **AsyncMode** Property; **ErrorString** Property; **ErrorUnits** Property; "Model" discussion

FreeVideoRegion Method

Syntax **LONG FreeVideoRegion (LONG** *Units,* **LONG** *BufferId***);**

Parameter	Description
Units	Bitwise mask indicating which video unit(s) to operate on.
BufferId	Number identifying the video buffer to free. Valid values range from 1 to the **VideoSaveBuffers** property for a selected unit(s).

Remarks This method will free any buffer memory allocated for the video unit(s) specified by the *Units* parameter. The number of video buffers supported is stored in the **VideoSaveBuffers** property for each video unit online. If the *BufferId* was never used in a previous **SaveVideoRegion** method, no action is take and OPOS_SUCCESS is returned.

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: • *BufferId* is out of range. • *Units* is zero or a non-existent unit was specified. The **ErrorUnits** and **ErrorString** properties are updated before return.
OPOS_E_FAILURE	An error occurred communicating with one of the video units specified by *Units*. The **ErrorUnits** and **ErrorString** properties are updated before return.
Other Values	See **ResultCode**.

See Also **ErrorString** Property; **ErrorUnits** Property; **VideoSaveBuffers** Property; **SaveVideoRegion** Method

ResetVideo Method

Syntax **LONG ResetVideo (LONG** *Units***);**

Units is a bitwise mask indicating which video unit(s) to operate on.

Remarks Sets the video unit(s) specified by the *Units* parameter to a power on state. All internal service object buffers and clocks associated with the unit(s) are released. All settable characteristics are set to default 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_FAILURE	An error occurred while communicating with one of the video units specified by *Units*. The **ErrorUnits** and **ErrorString** properties are updated before return.
Other Values	See **ResultCode**.

See Also **ErrorString** Property; **ErrorUnits** Property

RestoreVideoRegion Method

Syntax **LONG RestoreVideoRegion (LONG** *Units*, **LONG** *TargetRow*, **LONG** *TargetColumn*, **LONG** *BufferId***);**

Parameter	Description
Units	Bitwise mask indicating which video unit(s) to operate on.
TargetRow	The start row of the target location.
TargetColumn	The start column of the target location.
BufferId	Number identifying the source video buffer to use. Valid values range from 1 to the **VideoSaveBuffers** property for the selected unit(s).

Remarks This method will restore a previously saved video region of the display area from the requested *BufferId* for the video unit(s) specified by the *Units* parameter. A region can be saved using the **SaveVideoRegion** method. The number of video buffers supported is stored in the **VideoSaveBuffers** property for each video unit online. The target location is defined by the *TargetRow* and *TargetColumn* parameters. This method doesn’ t free the memory after restoring, therefore, this method can be used to copy a video region to multiple locations on the display. Use the **FreeVideoRegion** method to free any memory allocated for a video buffer.

If the *BufferId* does not contain a previously saved video region for the *Units* selected, an OPOS_EROD_NOREGION error is returned.

Video regions cannot be restored between video units. For example, the **SaveVideoRegion** method is called with *Units* = 0000 1000 and *BufferId* = 1. This will save a video region for the Unit Id 4, in to Buffer 1 for that unit. If **RestoreVideoRegion** is called with *Units* = 0000 0100 and *BufferId* = 1 with the intention of restoring the previously saved buffer to Unit Id 3, the return status could either be OPOS_EROD_NOREGION or an unwanted region is restored.

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	One of the following errors occurred: • *BufferId*, *TargetRow*, or *TargetColumn* are out of range. • *Units* is zero or a non-existent unit was specified. The **ErrorUnits** and **ErrorString** properties are updated before return.
OPOS_E_EXTENDED	**ResultCodeExtended** = OPOS_EROD_NOREGION: The *BufferId* does not contain a previously saved video region.
OPOS_E_FAILURE	An error occurred while communicating with one of the video units specified by *Units*. The **ErrorUnits** and **ErrorString** properties are updated before return. (Can only be returned if **AsyncMode** is FALSE.)
Other Values	See **ResultCode**.

See Also **AsyncMode** Property; **ErrorString** Property; **ErrorUnits** Property; **VideoSaveBuffers** Property; **SaveVideoRegion** Method

SaveVideoRegion Method

Syntax **LONG SaveVideoRegion (LONG** *Units*, **LONG** *Row*, **LONG** *Column*, **LONG** *Height*, **LONG** *Width*, **LONG** *BufferId***);**

Parameter	Description
Units	Bitwise mask indicating which video unit(s) to operate on.
Row	The start row of the region to save.
Column	The start column of the region to save.
Height	The number of rows in the region to save.
Width	The number of columns in the region to save.
BufferId	Number identifying the video buffer to use. Valid values range from 1 to the **VideoSaveBuffers** property for a selected unit(s).

Remarks This method will save the specified video region of the display area to one of the provided video buffers for the video unit(s) specified by the *Units* parameter. The number of video buffers supported is stored in the **VideoSaveBuffers** property for each video unit online. However, an OPOS_EROD_NOBUFFERS error will be returned if the requested buffer exceeds the number of **SystemVideoSaveBuffers** even though the **VideoSaveBuffers** property may indicated the unit can support more save buffers than currently allocated for that unit.

If **VideoSaveBuffers** is greater than 0, the service object will be able to support at minimum one entire video screen. This does not guarantee that the service object can save an entire video screen in each supported buffer for a single unit. An OPOS_EROD_NOROOM error is returned when all the buffer memory has been allocated for a specific unit.

The source area is defined by the *Row*, *Column*, *Height*, and *Width* parameters. The video region can be restored to the screen by calling the **RestoreVideoRegion** method. If **SaveVideoRegion** is called twice with the same *BufferId*, the previous video data is lost, and any allocated memory is returned to the system.

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	One of the following errors occurred: • *BufferId*, *Row*, *Column*, *Height*, or *Width*, are out of range. • *Units* is zero or a non-existent unit was specified. The **ErrorUnits** and **ErrorString** properties are updated before return.
OPOS_E_EXTENDED	**ResultCodeExtended** = OPOS_EROD_NOBUFFERS: Requested buffer exceeds the number of **SystemVideoSaveBuffers**. **ResultCodeExtended** = OPOS_EROD_NOROOM: All the buffer memory has been allocated for a specific unit. The **ErrorUnits** and **ErrorString** properties are updated before return.
OPOS_E_FAILURE	An error occurred while communicating with one of the video units specified by *Units*. The **ErrorUnits** and **ErrorString** properties are updated before return. (Can only be returned if **AsyncMode** is FALSE.)
Other Values	See **ResultCode**.

See Also **AsyncMode** Property; **ErrorString** Property; **ErrorUnits** Property; **SystemVideoSaveBuffers** Property; **VideoSaveBuffers** Property; **RestoreVideoRegion** Method

SelectChararacterSet Method

Syntax **LONG SelectChararacterSet (LONG** *Units*, **LONG** *CharacterSet***);**

Parameter	Description

Units	Bitwise mask indicating which video unit(s) to operate on.
CharacterSet	Contains the character set for displaying characters. Values are:

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

Remarks Selects a compatible character set for the video unit(s) specified by the *Units* parameter.

The **CharacterSet** property is updated for each video unit id for which a new character set is is 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_ILLEGAL	One of the following errors occurred: • Value in *CharacterSet* is not supported or the unit(s) does not support the **CapSelectCharacterSet** capability. • *Units* is zero or a non-existent unit was specified. The **ErrorUnits** and **ErrorString** properties are updated before return.
OPOS_E_FAILURE	An error occurred while communicating with one of the video units specified by *Units.* The **ErrorUnits** and **ErrorString** properties are updated before return.

Other Values	See **ResultCode**.

See Also **ErrorString** Property; **ErrorUnits** Property; **CapSelectCharacterSet** Property; **CharacterSet** Property

SetCursor Method

Syntax **LONG SetCursor (LONG** *Units*, **LONG** *Row*, **LONG** *Column***);**

Parameter	Description
Units	Bitwise mask indicating which video unit(s) to operate on.
Row	Row to place the cursor on.
Column	Column to place the cursor on.

Remarks This method will update the cursor position on the video unit(s) specified by the *Units* parameter.

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: • *Row* or *Column* positions are out of range. • *Units* is zero or a non-existent unit was specified. The **ErrorUnits** and **ErrorString** properties are updated before return.
OPOS_E_FAILURE	An error occurred while communicating with one of the video units specified by *Units*. The **ErrorUnits** and **ErrorString** properties are updated before return.
Other Values	See **ResultCode**.

See Also **ErrorString** Property; **ErrorUnits** Property

TransactionDisplay Method

Syntax **LONG TransactionDisplay (LONG** *Units*, **LONG** *Function***);**

Parameter	**Description**
Units	Bitwise mask indicating which video unit(s) to operate on.
Function	Transaction control function. Valid values are:

Value	**Meaning**
ROD_TD_TRANSACTION	Begin a transaction.
ROD_TD_NORMAL	End a transaction by displaying the buffered data.

Remarks Enters or exits transaction mode for the video unit(s) specified by the *Units* parameter.

If *Function* is ROD_TD_TRANSACTION, then transaction mode is entered. Subsequent calls to **ClearVideo**, **ClearVideoRegion**, **CopyVideoRegion**, **DisplayData**, **DrawBox**, **RestoreVideoRegion**, **SaveVideoRegion**, and **UpdateVideoRegionAttribute** will buffer the display data (either at the video unit or the Service Object, depending on the display capabilities) until **TransactionDisplay** is called with the *Function* parameter set to ROD_TD_NORMAL. (In this case, the display methods only validate the method parameters and buffer the data – they do not initiate displaying. 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 *Function* is ROD_TD_NORMAL, then transaction mode is exited. If some data was buffered by calls to the methods **ClearVideo**, **ClearVideoRegion**, **CopyVideoRegion**, **DisplayData**, **DrawBox**, **RestoreVideoRegion**, **SaveVideoRegion**, and **UpdateVideoRegionAttribute**, then the buffered data is displayed. 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 for the unit specified by the **CurrentUnitID** property. Any buffered print lines are also cleared.

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: • When **CapTransaction** is FALSE, this method is not supported. • *Function* parameter is illegal. • *Units* is zero or a non-existent unit was specified. The **ErrorUnits** and **ErrorString** properties are updated before return.
OPOS_E_BUSY	Cannot perform while output is in progress for one of the video units specified by *Units*. The **ErrorUnits** and **ErrorString** properties are updated before return. (Can only be returned if **AsyncMode** is FALSE and *Function* is ROD_TD_NORMAL)
OPOS_E_FAILURE	An error occurred communicating with one of the video units specified by *Units*. The **ErrorUnits** and **ErrorString** properties are updated before return. (Can only be returned if **AsyncMode** is FALSE and *Function* is ROD_TD_NORMAL)
Other Values	See **ResultCode**.

UpdateVideoRegionAttribute Method

Syntax **LONG UpdateVideoRegionAttribute (LONG** *Units***, LONG** *Function***, LONG** *Row***, LONG** *Column*, **LONG** *Height*, **LONG** *Width*, **LONG** *Attribute***);**

Parameter	**Description**
Units	Bitwise mask indicating which video unit(s) to operate on.
Function	The attribute command. See values below.
Row	The region' s start row.
Column	The region' s start column.
Height	The number of rows in the region.
Width	The number of columns in the region.
Attribute	See "Model" discussion in the General Information section.

The *Function* parameter values are:

Value	**Meaning**
ROD_UA_SET	Set the region with the new attribute.
ROD_UA_INTENSITY_ON	Turn on foreground intensity in the region.
ROD_UA_INTENSITY_OFF	Turn off foreground intensity in the region.
ROD_UA_REVERSE_ON	Reverse video the region.
ROD_UA_REVERSE_OFF	Remove reverse video from the region.
ROD_UA_BLINK_ON	Turn on blinking in the region.
ROD_UA_BLINK_OFF	Turn off blinking in the region.

Remarks This method will modify the attribute on the video unit(s) specified by the *Units* parameter in the region defined by the *Row*, *Column*, *Height*, and *Width* parameters. When the *Function* parameter is ROD_UA_SET, the region' s attributes will be replaced with the new value in the *Attribute* parameter; otherwise the *Attribute* parameter is ignored and the region' s attributes will be modified.

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	One of the following errors occurred: • *Row*, *Column*, *Height,* or *Width* positions are out of range. • *Attribute* or *Function* is illegal. • *Units* is zero or a non-existent unit was specified. The **ErrorUnits** and **ErrorString** properties are updated before return.
OPOS_E_FAILURE	An error occurred while communicating with one of the video units specified by *Units.* The **ErrorUnits** and **ErrorString** properties are updated before return. (Can only be returned if **AsyncMode** is FALSE.)
Other Values	See **ResultCode**.

See Also **AsyncMode** Property; **ErrorString** Property; **ErrorUnits** Property; “Model” discussion

VideoSound Method

Syntax **LONG VideoSound (LONG** *Units***, LONG** *Frequency,* **LONG** *Duration,* **LONG** *NumberOfCycles,* **LONG** *InterSoundWait***);**

Parameter	**Description**
Units	Bitwise mask indicating which video unit(s) to operate on.
Frequency	Tone frequency in Hertz.
Duration	Tone duration in milliseconds.
NumberOfCycles	If OPOS_FOREVER, then start tone sounding and, repeat continuously. Else perform the specified number of cycles.
InterSoundWait	When *NumberOfCycles* is not one, then pause for *InterSoundWait* milliseconds before repeating the tone cycle (before playing the tone again)

Remarks Sound the video enunciator for the video(s) specified by the *Units* parameter.

This method is performed synchronously if **AsyncMode** is FALSE, and asynchronously if **AsyncMode** is TRUE.

The duration of a video tone cycle is:

Duration parameter +
InterSoundWait parameter (except on the last tone cycle)

After the video has started an asynchronous sound, then the ClearOutput method will stop the sound. (When an *InterSoundWait* value of OPOS_FOREVER was used to start the sound, then the application must use **ClearOutput** to stop the continuous sounding of tones.)

If the **CapTone** property is FALSE for the selected unit(s), an OPOS_E_ILLEGAL is returned.

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: • *NumberOfCycles* is neither a positive, non-zero value nor OPOS_FOREVER. • *NumberOfCycles* is OPOS_FOREVER when **AsyncMode** is FALSE. • A negative *InterSoundWait* was specified. • *Units* is zero or a non-existent unit was specified. • A unit in *Units* does not support the **CapTone** capability. The **ErrorUnits** and **ErrorString** properties may be updated before return.
OPOS_E_FAILURE	An error occurred while communicating with one of the video units specified by the *Units* parameter. The **ErrorUnits** and **ErrorString** properties are updated before return. (Can only be returned if **AsyncMode** is FALSE.)
Other Values	See **ResultCode**.

See Also **AsyncMode** Property; **ErrorString** Property; **ErrorUnits** Property; **CapTone** Property; **ClearOutput** Method

Events

DataEvent Event

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

The *Status* parameter is divided into four bytes. The diagram below indicates how the parameter *Status* is divided:

High Word		Low Word(Event Type)
High Byte	Low Byte	
Row	Column	ROD_DE_TOUCH_UP ROD_DE_TOUCH_DOWN ROD_DE_TOUCH_MOVE

Remarks Fired to indicate input data from a video touch unit to the application. The low word contains the Event Type. The high word contains additional data depending on the Event Type. When the Event Type is ROD_DE_TOUCH_UP, ROD_DE_TOUCH_DOWN, or ROD_DE_TOUCH_MOVE, the high word indicates where the touch occurred. The low byte contains the Column position and the high byte contains the Row position, with valid values ranging from 0-255.

Data events can be filtered at the remote order display device by setting the **EventTypes** property.

The **EventUnitID** property is updated before delivering the event.

See Also "Input Model"; **EventUnitID** Property; **DataEventEnabled** Property; **FreezeEvents** Property

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.

The **EventUnits** property is updated before delivering the event.

See Also "Output Model"; **EventUnits** Property

StatusUpdateEvent Event

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

The *Status* parameter reports a change in the power state of a video unit.

Remarks Fired when the remote order display device detects a power state change.

Deviation from the standard **StatusUpdateEvent** (see page 68):

- Before delivering the event, the **EventUnits** property is set to the units for which the new power state applies.
- When the remote order display device is enabled, then the Control will fire a **StatusUpdateEvent** to specify the bitmask of online units.
- While the remote order display device is enabled, a **StatusUpdateEvent** is fired when the power state of one or more units change. If more than one unit changes state at the same time, the Service Object may choose to either fire multiple events or to coalesce the information into a minimal number of events applying to **EventUnits**.

See Also **EventUnits** Property

ErrorEvent Event

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

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

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

Value	**Meaning**
OPOS_EL_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 the value 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 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 deliver additional **DataEvent**s as directed by the **DataEventEnabled** property. When all input has been delivered and the **DataEventEnabled** property is again set to TRUE, then another **ErrorEvent** is delivered with locus OPOS_EL_INPUT.
Default when locus is OPOS_EL_INPUT_DATA.

Remarks Fired when an error is detected while trying to read remote order display data.

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

The **EventUnits** and **EventString** properties are updated before return.

See Also "Status, Result Code, and State Model"; **DataEventEnabled** Property; **EventUnits** Property; **EventString** Property

C H A P T E R   1 7

Scale

Summary

Properties

Common		*Type*	*Access*	*Initialized After*
AutoDisable	1.3	Boolean	R/W	Open
BinaryConversion	1.2	Long	R/W	Open
CapPowerReporting	1.3	Long	R	Open
CheckHealthText	1.0	String	R	Open
Claimed	1.0	Boolean	R	Open
DataCount	1.3	Long	R	Open
DataEventEnabled	1.3	Boolean	R/W	Open
DeviceEnabled	1.0	Boolean	R/W	Open & Claim
FreezeEvents	1.0	Boolean	R/W	Open
OutputID	1.0	Long	R	*Not Supported*
PowerNotify	1.3	Long	R/W	Open
PowerState	1.3	Long	R	Open
ResultCode	1.0	Long	R	--
ResultCodeExtended	1.0	Long	R	Open
State	1.0	Long	R	--
ControlObjectDescription	1.0	String	R	--
ControlObjectVersion	1.0	Long	R	--
ServiceObjectDescription	1.0	String	R	Open
ServiceObjectVersion	1.0	Long	R	Open
DeviceDescription	1.0	String	R	Open
DeviceName	1.0	String	R	Open

Specific		*Type*	*Access*	*Initialized After*
CapDisplay	1.2	Boolean	R	Open
CapDisplayText	1.3	Boolean	R	Open
CapPriceCalculating	1.3	Boolean	R	Open
CapTareWeight	1.3	Boolean	R	Open
CapZeroScale	1.3	Boolean	R	Open
AsyncMode	1.3	Boolean	R/W	Open
MaxDisplayTextChars	1.3	Long	R	Open
MaximumWeight	1.0	Long	R	Open
SalesPrice	1.3	Currency	R	Open, Claim, & Enable
TareWeight	1.3	Long	R/W	Open, Claim, & Enable
UnitPrice	1.3	Currency	R/W	Open, Claim, & Enable
WeightUnit	1.0	Long	R	Open

Methods

Common		*May Use After*
Open	1.0	--
Close	1.0	Open
Claim	1.0	Open
Release	1.0	Open & Claim
CheckHealth	1.0	Open, Claim, & Enable
ClearInput	1.3	Open & Claim
ClearOutput	1.0	*Not Supported*
DirectIO	1.0	Open
Specific		
DisplayText	1.3	Open, Claim, & Enable
ReadWeight	1.0	Open, Claim, & Enable
ZeroScale	1.3	Open, Claim, & Enable

Events

Name		*May Occur After*
DataEvent	1.3	Open, Claim, & Enable
DirectIOEvent	1.0	Open, Claim
ErrorEvent	1.3	Open, Claim, & Enable
OutputCompleteEvent	1.0	*Not Supported*
StatusUpdateEvent	1.3	Open, Claim, & Enable

General Information

The Scale Control’ s OLE programmatic ID is “OPOS.Scale”.

Capabilities

The scale has the following capability:

- Provides item weight to the application. The measure of weight may be in grams, kilograms, ounces, or pounds, depending upon the scale device.

The scale may have the following additional capabilities:

- Includes an integrated display with the current weight, or with the current weight plus Application-specified text.
- Performs price calculations (weight X unit price) and returns the sale price. (This feature is mostly used in Europe at this time.)
- Supports Application setting of tare weight.
- Supports Application zeroing of the scale.

Model

The general model of a scale is:

- A scale returns the weight of an item placed on its weighing surface.
- The primary scale method is **ReadWeight**. By default, it is performed synchronously. It returns after reading data from the scale; the weight is returned in the location pointed to by the method parameter *pWeightData*. If an error occurs or if the timeout elapses, the **ReadWeight** method returns with an error code.

- ***OPOS Release 1.3 and later – Asynchronous Input***

  If the property **AsyncMode** is TRUE when **ReadWeight** is called, then the method is performed asynchronously. It initiates event driven input and returns immediately. The timeout parameter specifies the maximum time the application wants to wait for a settled weight. Additional points are:

  - ♦ If an error occurs while initiating event driven input (such as the device is offline), then an error code is returned by **ReadWeight**. Otherwise, **ReadWeight** returns a success status to the Application, and scale processing continues asynchronously …
  - ♦ If a settled weight is received, then a **DataEvent** is enqueued containing the weight data in the *Status* parameter.
  - ♦ If a scale error occurs (including a timeout with no settled weight), then an **ErrorEvent** is enqueued with an error code. The Application event handler may retry the weighing process by setting the response parameter (pointed to by *pErrorResponse*) to OPOS_ER_RETRY.
  - ♦ Only one asynchronous call to **ReadWeight** can be in progress at a time. Nesting of asynchronous scale operations is illegal.
  - ♦ An asynchronous scale operation may be cancelled with the **ClearInput** method.

For price-calculating scales, the Application must set the property **UnitPrice** before calling **ReadWeight**. After a weight is read (and just before the **DataEvent** is delivered to the Application, for asynchronous mode), the Control sets the property **SalesPrice** to the calculated price of the item.

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

AsyncMode Property R/W *Added in Release 1.3*

Syntax **BOOL AsyncMode;**

Remarks If TRUE, then the **ReadWeight** method will be performed asynchronously.
If FALSE, this methods will be performed 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.

See Also **ReadWeight** Method

CapDisplay Property *Added in Release 1.2*

Syntax **BOOL CapDisplay;**

Remarks Set to TRUE if the scale includes an integrated display that shows the current weight;
otherwise it is FALSE, indicating that the application may need to show the current weight on another display.

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

CapDisplayText Property *Added in Release 1.3*

Syntax **BOOL CapDisplayText;**

Remarks Set to TRUE if the scale includes an integrated display that shows the current weight and can also show a text that describes the item being weighed. Otherwise FALSE, indicating that extra text cannot be shown on the display.

If TRUE, then **CapDisplay** must also be TRUE.

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

See Also **MaxDisplayTextChars** Property

CapPriceCalculating Property *Added in Release 1.3*

Syntax **BOOL CapPriceCalculating;**

Remarks Set to TRUE if the scale can calculate prices. Otherwise FALSE, indicating that the scale only returns a weight.

For price calculating scales the calculation unit is in the scale rather than in the data-receiving terminal. For price-calculating scales the **UnitPrice** property is to be set before calling the **ReadWeight** method

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

See Also **ReadWeight** Method, **WeightUnit** Property,
UnitPrice Property, **SalesPrice** Property

CapTareWeight Property *Added in Release 1.3*

Syntax **BOOL CapTareWeight;**

Remarks Set to TRUE if the scale includes setting a tare value. Otherwise FALSE, indicating that the scale does not support tare values.

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

See Also **TareWeight** Property

CapZeroScale Property *Added in Release 1.3*

Syntax **BOOL CapZeroScale;**

Remarks Set to TRUE if the Application can set the scale weight to zero. Otherwise FALSE, indicating that the scale does not support programmatic zeroing.

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

See Also **ZeroScale** Method

MaxDisplayTextChars Property *Added in Release 1.3*

Syntax **LONG MaxDisplayTextChars;**

Remarks The number of characters that may be displayed on an integrated display for the text which describes an article.

If the capability **CapDisplayText** is FALSE, then the device does not support text displaying and **MaxDisplayTextChars** is always zero.

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

See Also **CapDisplayText** Property

MaximumWeight Property

Syntax **LONG MaximumWeight;**

Remarks Holds the maximum weight measurement possible from the scale. The measurement unit is available via the **WeightUnit** property.

MaximumWeight 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.

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

See Also **WeightUnit** Property

SalesPrice Property *Added in Release 1.3*

Syntax **CURRENCY SalesPrice;**

Remarks The sales price read from the scale for price calculating scales. For price-calculating scales the scale calculates this value during the process of weighing by multiplying the **UnitPrice** property by the acquired weight.

This property is set by the control before the **ReadWeight** method returns (in synchronous use) or the **DataEvent** is delivered by the control (in asynchronous use).

If the capability **CapPriceCalculating** is FALSE then the device is not a price-calculating scale and **SalesPrice** is always zero.

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

See Also **ReadWeight** Method**, WeightUnit** Property**, CapPriceCalculating** Property**, UnitPrice** Property

TareWeight Property R/W *Added in Release 1.3*

Syntax **LONG TareWeight;**

Remarks Holds the tare weight of scale data. The weight in **TareWeight** property has an assumed fractional part of three digits. For example, an actual value of 12345 represents 12.345, and an actual value of 5 represents 0.005. The measured unit is specified in the **WeightUnit** property. If the capability **CapTareWeight** is FALSE then the device does not support setting of a tare value and **TareWeight** is always zero.

Tare weight is not included in the item weight returned by the **ReadWeight** method.

This property is initialized by the **Open** method to the scale' s default tare weight (usually zero)

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	**CapTareWeight** is FALSE or an invalid tare value was specified.
Other Values	See **ResultCode**.

See Also **CapTareWeight** Property**, ReadWeight** Method**, WeightUnit** Property

UnitPrice Property R/W *Added in Release 1.3*

Syntax **CURRENCY UnitPrice;**

Remarks Holds the unit price of the article to be weighed. For price calculating scales this property is to be set before starting the process of weighing. The scale itself calculates during weighing the property **SalesPrice** by multiplying the **UnitPrice** with the *pWeightData* parameter of the **ReadWeight** method. So, this property contains only a factor.

If the capability **CapPriceCalculating** is FALSE then the scale is not a price-calculating scale. In this case, setting of a unit price is not supported and **UnitPrice** is always zero.

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

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	**CapPriceCalculating** is FALSE or an invalid price was specified.
Other Values	See **ResultCode**.

See Also **ReadWeight** Method**, WeightUnit** Property**, CapPriceCalculating** Property**, SalesPrice** Property

WeightUnit Property

Syntax **LONG WeightUnit;**
LONG WeightUnits; (Synonym for **WeightUnit**.[8])

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.

[8] A Scale Control Object must support the property **WeightUnit**. In addition, due to a documentation error in OPOS APG Releases 1.1 and earlier, it is recommended that the property **WeightUnits** also be supported, and that it refer to the same property.

Methods

DisplayText Method *Added in Release 1.3*

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

Parameter	**Description**
Data	The string of characters to display. The format of this data depends upon the value of the **BinaryConversion** property. See page **37**.

Remarks Call this method to update the text shown on the integrated display. Calling this method with an empty string (“”) will clear the display.

If the capability **CapDisplayText** is FALSE, then the device does not support text displaying and **DisplayText** 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_ILLEGAL	An invalid text was specified – the text contains more characters than allowed in **MaxDisplayTextChars,** or displaying text is not allowed.
Other Values	See **ResultCode**.

See Also **CapDisplay** Property**, CapDisplayText** Property**,**
MaxDisplayTextChars Property**,**

ReadWeight Method

Syntax **LONG ReadWeight (LONG*** *pWeightData*, **LONG** *Timeout*);

Parameter	Description
pWeightData	If **AsyncMode** is FALSE, points to where the weight is returned; else must be zero.
Timeout	The number of milliseconds to wait for a settled weight before failing the method. If zero, the method attempts to read the scale weight, then returns the appropriate status immediately. If OPOS_FOREVER (-1), the method waits as long as needed until a weight is successfully read or an error occurs.

Remarks Call to read a weight from the scale.

Release 1.0 – 1.2

The weighing process is performed synchronously and the method will return after finishing the weighing process. The weight is returned at *pWeightData*,

Release 1.3 and later

If **AsyncMode** is FALSE, then **ReadWeight** operates synchronously, as with earlier releases.

If **AsyncMode** is TRUE, the weighing process is performed asynchronously. The method will initiate a read, then return immediately. If the method returns a success status, the weighing process is started and a **DataEvent** containing the weight in its *Status* parameter indicates its completion.

The weight 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_ILLEGAL	An invalid *Timeout* parameter was specified.

OPOS_E_TIMEOUT	A stable non-zero weight was not available before *Timeout* milliseconds elapsed (only if **AsyncMode** is FALSE).
OPOS_E_EXTENDED	**ResultCodeExtended** = OPOS_ESCAL_OVERWEIGHT: The weight was over **MaximumWeight**.
Other Values	See **ResultCode**.

See Also **UnitPrice** Property**, WeightUnit** Property**, CapPriceCalculating** Property**, SalesPrice** Property, **TareWeight** Property

ZeroScale Method *Added in Release 1.3*

Syntax **LONG ZeroScale ();**

Remarks Call to set the current scale weight to zero. It may be used for initial calibration, or to account for tare weight on the scale.

May be called only if the property **CapZeroScale** 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	Scale zeroing is not supported.
Other Values	See **ResultCode**.

See Also **CapZeroScale** Property

Events

DataEvent Event

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

The *Status* parameter contains the weight.

Remarks Fired to present input data from the device to the application after an asynchronous **ReadWeight** was initiated.

If the scale is a price-calculating scale, the unit price is placed in the **UnitPrice** property and the calculated sales price is placed in the **SalesPrice** property before this event is delivered.

ErrorEvent Event

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

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

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

Value	**Meaning**
OPOS_EL_INPUT	Error occurred while gathering or processing event-driven input. No input data is available.
OPOS_EL_INPUT_DATA	Error occurred while gathering or processing event-driven input, and some previously buffered data is available.

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 deliver additional **DataEvent**s as directed by the **DataEventEnabled** property. When all input has been delivered and the **DataEventEnabled** property is again set to TRUE, then another **ErrorEvent** is delivered with locus OPOS_EL_INPUT. Default when locus is OPOS_EL_INPUT_DATA.

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

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

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

C H A P T E R   1 8

Scanner (Bar Code Reader)

Summary

Properties

Common		*Type*	*Access*	*Initialized After*
AutoDisable	1.2	Boolean	R/W	Open
BinaryConversion	1.2	Long	R/W	Open
CapPowerReporting	1.3	Long	R	Open
CheckHealthText	1.0	String	R	Open
Claimed	1.0	Boolean	R	Open
DataCount	1.2	Long	R	Open
DataEventEnabled	1.0	Boolean	R/W	Open
DeviceEnabled	1.0	Boolean	R/W	Open & Claim
FreezeEvents	1.0	Boolean	R/W	Open
OutputID	1.0	Long	R	*Not Supported*
PowerNotify	1.3	Long	R/W	Open
PowerState	1.3	Long	R	Open
ResultCode	1.0	Long	R	--
ResultCodeExtended	1.0	Long	R	Open
State	1.0	Long	R	--
ControlObjectDescription	1.0	String	R	--
ControlObjectVersion	1.0	Long	R	--
ServiceObjectDescription	1.0	String	R	Open
ServiceObjectVersion	1.0	Long	R	Open
DeviceDescription	1.0	String	R	Open
DeviceName	1.0	String	R	Open

Specific		*Type*	*Access*	*Initialized After*
DecodeData	1.2	Boolean	R/W	Open
ScanData	1.0	String	R	Open
ScanDataLabel	1.2	String	R	Open
ScanDataType	1.2	Long	R	Open

Methods

Common		*May Use After*
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	Open & Claim
ClearOutput	1.0	*Not Supported*
DirectIO	1.0	Open

Events

Name		*May Occur After*
DataEvent	1.0	Open, Claim, & Enable
DirectIOEvent	1.0	Open, Claim
ErrorEvent	1.0	Open, Claim, & Enable
OutputCompleteEvent	1.0	*Not Supported*
StatusUpdateEvent	1.3	Open, Claim, & Enable

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 enqueues a **DataEvent**.
- If the **AutoDisable** property is TRUE, then the control automatically disables itself when a **DataEvent** is enqueued.
- An enqueued **DataEvent** can be delivered to the application when the **DataEventEnabled** property is TRUE. Just before delivering this event, the Control copies the data into properties, and disables further data events by setting the **DataEventEnabled** property to FALSE. This causes subsequent input data to be enqueued by 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.
- An **ErrorEvent** (or events) are enqueued if the Control encounters an error while gathering or processing input, and is delivered to the application when the **DataEventEnabled** property is TRUE.
- The **DataCount** property may be read to obtain the number of **DataEvent**s enqueued by the Control.
- All input enqueued by the Control may be deleted by calling the **ClearInput** method.

Scanned data is placed into the property **ScanData**. If the application sets the property **DecodeData** to TRUE, then the data is decoded into **ScanDataLabel** and **ScanDataType**.

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

DecodeData Property R/W *Added in Release 1.2*

Syntax **BOOL DecodeData;**

Remarks If TRUE, then the Control will decode **ScanData** into the properties **ScanDataLabel** and **ScanDataType**.

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.

ScanData Property

Syntax **BSTR ScanData;**

Remarks The data read from the scanner.
The format of this data depends upon the value of the **BinaryConversion** property. See page 37.

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 (for example, 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.)

Some merchandise may be marked with a supplemental barcode. This barcode is typically placed to the right of the main barcode, and consists of an additional two or five characters of information. If the scanner reads merchandise that contains both main and supplemental barcodes, the supplemental characters are appended to the main characters, and the result is delivered to the application as one label. (Note that a scanner may support configuration that enables or disables the reading of supplemental codes.)

Some merchandise may be marked with multiple labels, sometimes called multi-symbol labels or tiered labels. These barcodes are typically arranged vertically, and may be of the same or different symbology. If the scanner reads merchandise that contains multiple labels, each barcode is delivered to the application as a separate label. 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 multiple label barcode has been read based upon the data returned. (Note that a scanner may or may not support reading of multiple labels.)

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

ScanDataLabel Property *Added in Release 1.2*

Syntax **BSTR ScanDataLabel;**

Remarks The decoded bar code label.
The format of this data depends upon the value of the **BinaryConversion** property. See page 37.

When the property **DecodeData** is FALSE, **ScanDataLabel** is set to the empty string (“”).

When the property **DecodeData** is TRUE, the Control decodes **ScanData** into **ScanDataLabel** as follows:

- Scanner-generated symbology characters are removed, if present.
- If the label type contains a readable check digit (such as with UPC-A and EAN-13), then it must be present in **ScanDataLabel**. If the scanner does not return the check digit to the Service Object, then it is to be calculated and included.
- For variable length bar codes, the length identification is removed, if present.

For example, the EAN-13 barcode which appears printed as "5 018374 827715" on a label may be received from the scanner and placed into **ScanData** as the following:

Received from scanner	**ScanData**	**Comment**
5018374827715	5018374827715	Complete barcode only
501837482771<CR>	501837482771	Without check digit with carriage return
F5018374827715<CR>	F5018374827715	With scanner-dependent symbology character and carriage return
*<STX>*F5018374827715*<ETX>*	F5018374827715	With header, symbology character, and trailer

For each of these cases (and any other variations), **ScanDataLabel** must always be set to the string "5018374827715", and **ScanDataType** must be set to SCAN_SDT_EAN13.

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

ScanDataType Property *Added in Release 1.2*

Syntax **LONG ScanDataType ;**

Remarks The decoded bar code label type.

When the property **DecodeData** is FALSE, **ScanDataType** is set to SCAN_SDT_UNKNOWN.

When the property **DecodeData** is TRUE, the Control tries to determine the scan label type. The Scanner Control header file (OposScan.h) defines several symbologies with constant names beginning with SCAN_SDT.

The following label types are defined in this release:

Value	Label Type
One Dimensional Symbologies	
SCAN_SDT_UPCA	UPC-A
SCAN_SDT_UPCA_S	UPC-A with supplemental barcode
SCAN_SDT_UPCE	UPC-E
SCAN_SDT_UPCE_S	UPC-E with supplemental barcode
SCAN_SDT_UPCD1	UPC-D1
SCAN_SDT_UPCD2	UPC-D2
SCAN_SDT_UPCD3	UPC-D3
SCAN_SDT_UPCD4	UPC-D4
SCAN_SDT_UPCD5	UPC-D5
SCAN_SDT_EAN8	EAN 8 (= JAN 8)
SCAN_SDT_JAN8	JAN 8 (= EAN 8)
SCAN_SDT_EAN8_S	EAN 8 with supplemental barcode
SCAN_SDT_EAN13	EAN 13 (= JAN 13)
SCAN_SDT_JAN13	JAN 13 (= EAN 13)
SCAN_SDT_EAN13_S	EAN 13 with supplemental barcode
SCAN_SDT_EAN128	EAN-128

SCAN_SDT_TF	Standard (or discrete) 2 of 5
SCAN_SDT_ITF	Interleaved 2 of 5
SCAN_SDT_Codabar	Codabar
SCAN_SDT_Code39	Code 39
SCAN_SDT_Code93	Code 93
SCAN_SDT_Code128	Code 128
SCAN_SDT_OCRA	OCR “A”
SCAN_SDT_OCRB	OCR “B”
Two Dimensional Symbologies	
SCAN_SDT_PDF417	PDF 417
SCAN_SDT_MAXICODE	MAXICODE
Special Cases	
SCAN_SDT_OTHER	If greater or equal to this type, then the Service Object has returned a non-OPOS defined symbology.
SCAN_SDT_UNKNOWN	The Service Object cannot determine the barcode symbology. **ScanDataLabel** may not be properly formatted for the actual barcode type.

This property is set by the Control just before delivering 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, ScanDataLabel,** and **ScanDataType** properties before this event is delivered.

ErrorEvent Event

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

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

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

Value	**Meaning**
OPOS_EL_INPUT	Error occurred while gathering or processing event-driven input. No input data is available.
OPOS_EL_INPUT_DATA	Error occurred while gathering or processing event-driven input, and some previously buffered data is available.

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 deliver additional **DataEvent**s as directed by the **DataEventEnabled** property. When all input has been delivered and the **DataEventEnabled** property is again set to TRUE, then another **ErrorEvent** is delivered 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 delivered until the **DataEventEnabled** property is TRUE, so that proper application sequencing occurs.

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

C H A P T E R   1 9

Signature Capture

Summary

Properties

Common		*Type*	*Access*	*Initialized After*
AutoDisable	1.2	Boolean	R/W	Open
BinaryConversion	1.2	Long	R/W	Open
CapPowerReporting	1.3	Long	R	Open
CheckHealthText	1.0	String	R	Open
Claimed	1.0	Boolean	R	Open
DataCount	1.2	Long	R	Open
DataEventEnabled	1.0	Boolean	R/W	Open
DeviceEnabled	1.0	Boolean	R/W	Open & Claim
FreezeEvents	1.0	Boolean	R/W	Open
OutputID	1.0	Long	R	*Not Supported*
PowerNotify	1.3	Long	R/W	Open
PowerState	1.3	Long	R	Open
ResultCode	1.0	Long	R	--
ResultCodeExtended	1.0	Long	R	Open
State	1.0	Long	R	--
ControlObjectDescription	1.0	String	R	--
ControlObjectVersion	1.0	Long	R	--
ServiceObjectDescription	1.0	String	R	Open
ServiceObjectVersion	1.0	Long	R	Open
DeviceDescription	1.0	String	R	Open
DeviceName	1.0	String	R	Open

Specific		*Type*	*Access*	*Initialized After*
CapDisplay	1.0	Boolean	R	Open
CapRealTimeData	1.2	Boolean	R	Open
CapUserTerminated	1.0	Boolean	R	Open
MaximumX	1.0	Long	R	Open
MaximumY	1.0	Long	R	Open
RawData	1.0	String	R	Open, Claim, & Enable
RealTimeDataEnabled	1.2	Boolean	R/W	Open
TotalPoints	1.0	Long	R	Open, Claim, & Enable
PointArray	1.0	String	R	Open, Claim, & Enable

Methods

Common		*May Use After*
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	Open & Claim
ClearOutput	1.0	*Not Supported*
DirectIO	1.0	Open
Specific		
BeginCapture	1.0	Open, Claim, & Enable
EndCapture	1.0	Open, Claim, & Enable

Events

Name		*May Occur After*
DataEvent	1.0	Open, Claim, & Enable
DirectIOEvent	1.0	Open, Claim
ErrorEvent	1.0	Open, Claim, & Enable
OutputCompleteEvent	1.0	*Not Supported*
StatusUpdateEvent	1.3	Open, Claim, & Enable

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 line points are presented to the application by a **DataEvent** with a single array of line points

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.
- Returns the signature in “real time” as it is entered on the device. If this capability is true and has been enabled by application by setting the **RealTimeDataEnabled** property to TRUE, then a series of **DataEvent**s are generated, each with an array of one or more line points representing a partial signature.

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 is capable of supplying signature data in real time as the signature is entered (**CapRealTimeData** is set to TRUE), and if the **RealTimeDataEnabled** property is set to TRUE, the signature is presented to the application as a series of partial signature data events until the signature capture is terminated.
- 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:

- When input is received by the Control, it enqueues a **DataEvent**.
- If the **AutoDisable** property is TRUE, then the control automatically disables itself when a **DataEvent** is enqueued.
- An enqueued **DataEvent** can be delivered to the application when the **DataEventEnabled** property is TRUE. Just before delivering this event, the Control copies the data into properties, and disables further data events by setting the **DataEventEnabled** property to FALSE. This causes subsequent input data to be enqueued by 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.
- An **ErrorEvent** (or events) are enqueued if the Control encounters an error while gathering or processing input, and is delivered to the application when the **DataEventEnabled** property is TRUE.
- The **DataCount** property may be read to obtain the number of **DataEvent**s enqueued by the Control.
- All input enqueued by the Control may be deleted by calling the **ClearInput** method.

Deviations from the Input Model are:

- The capture of signature data begins when the **BeginCapture** method is called.
- If signature capture is terminated by calling **EndCapture**, then no **DataEvent** is fired.

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.

CapRealTimeData Property *Added in Release 1.2*

Syntax **BOOL CapRealTimeData;**

Remarks Set to TRUE if the device is able to supply signature data as the signature is being captured ("real time");
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 **CapDisplay** 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.

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).
The format of this data depends upon the value of the **BinaryConversion** property. See page 37.

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

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

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

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

This property is set by the Control just before delivering the **DataEvent** or by the **EndCapture** method.

See Also **RawData** Property

RawData Property

Syntax **BSTR RawData;**

Remarks Contains the signature captured from the device in a device-specific format. The format of this data depends upon the value of the **BinaryConversion** property. See page 37.

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 delivering the **DataEvent** or by the **EndCapture** method.

See Also **TotalPoints** Property; **PointArray** Property

RealTimeDataEnabled Property R/W *Added in Release 1.2*

Syntax **BOOL RealTimeDataEnabled;**

Remarks When **CapRealTimeData** is TRUE and this property is set to TRUE, a series of partial signature data events is fired as the signature is captured until signature capture is terminated.

Else, the captured signature is fired as a single data event when signature capture is terminated.

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_ILLEGAL	Cannot set to TRUE because **CapRealTimeData** is FALSE.

See Also "General OLE for Retail POS Control Model"

TotalPoints Property

Syntax **LONG TotalPoints;**

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

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

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

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 “APPENDIX B
OPOS Registry Usage”, page 683.)

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	*FormName* was not found.
Other Values	See **ResultCode**.

EndCapture Method

Syntax **LONG EndCapture ();**

Remarks Call to stop capturing a signature.

Terminates signature capture.

If the **RealTimeDataEnabled** property is FALSE:
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 (“”).

If the **RealTimeDataEnabled** property is TRUE:
If there are signature points remaining which have not been delivered to the application by a **DataEvent**, then the remaining signature is placed into the properties **TotalPoints**, **PointArray**, and **RawData**. If no signature was captured or all signature points have been delivered to the application, 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.
Other Values	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
ResultCode	Result code causing the error event. See **ResultCode** for values.
ResultCodeExtended	Extended result code causing the error event. See **ResultCodeExtended** for values.
ErrorLocus	Location of the error. See values below.
pErrorResponse	Pointer to the error event response. See values below.

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

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

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 deliver additional **DataEvent**s as directed by the **DataEventEnabled** property. When all input has been delivered and the **DataEventEnabled** property is again set to TRUE, then another **ErrorEvent** is delivered 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 delivered 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"

CHAPTER 20

Tone Indicator

Summary

Properties

Common		*Type*	*Access*	*Initialized After*
AutoDisable	1.2	Boolean	R/W	*Not Supported*
BinaryConversion	1.2	Long	R/W	Open
CapPowerReporting	1.3	Long	R	Open
CheckHealthText	1.2	String	R	Open
Claimed	1.2	Boolean	R	Open
DataCount	1.2	Long	R	*Not Supported*
DataEventEnabled	1.2	Boolean	R/W	*Not Supported*
DeviceEnabled	1.2	Boolean	R/W	Open
FreezeEvents	1.2	Boolean	R/W	Open
OutputID	1.2	Long	R	Open
PowerNotify	1.3	Long	R/W	Open
PowerState	1.3	Long	R	Open
ResultCode	1.2	Long	R	--
ResultCodeExtended	1.2	Long	R	Open
State	1.2	Long	R	--
ControlObjectDescription	1.2	String	R	--
ControlObjectVersion	1.2	Long	R	--
ServiceObjectDescription	1.2	String	R	Open
ServiceObjectVersion	1.2	Long	R	Open
DeviceDescription	1.2	String	R	Open
DeviceName	1.2	String	R	Open

Specific		***Type***	***Access***	***Initialized After***
AsyncMode	1.2	Boolean	R/W	Open & Enable
CapPitch	1.2	Boolean	R	Open
CapVolume	1.2	Boolean	R	Open
Tone1Pitch	1.2	Long	R/W	Open & Enable
Tone1Volume	1.2	Long	R/W	Open & Enable
Tone1Duration	1.2	Long	R/W	Open & Enable
Tone2Pitch	1.2	Long	R/W	Open & Enable
Tone2Volume	1.2	Long	R/W	Open & Enable
Tone2Duration	1.2	Long	R/W	Open & Enable
InterToneWait	1.2	Long	R/W	Open & Enable

Methods

Common		***May Use After***
Open	1.2	--
Close	1.2	Open
Claim	1.2	Open
Release	1.2	Open & Claim
CheckHealth	1.2	Open & Enable; *Note*
ClearInput	1.2	*Not Supported*
ClearOutput	1.2	Open
DirectIO	1.2	Open
Specific		
Sound	1.2	Open & Enable; *Note*
SoundImmediate	1.2	Open & Enable; *Note*

Note: Also requires that no other application has claimed the tone indicator.

Events

Name		***May Occur After***
DataEvent	1.2	*Not Supported*
DirectIOEvent	1.2	Open
ErrorEvent	1.2	Open & Enable
OutputCompleteEvent	1.2	Open & Enable
StatusUpdateEvent	1.3	Open & Enable

General Information

The Tone Indicator Control’ s OLE programmatic ID is “OPOS.ToneIndicator”.

Capabilities

The Tone Indicator Control has the following capabilities:

- Sound a tone device, which may be the PC system speaker or another hardware device. In many cases the PC speaker will not be available or will be in a position that is inaudible to the operator.
- Sound a two-tone indicator, providing simple pitch and volume control.
- Provide a synchronous one-shot (play once while waiting) indicator, similar to the Win32 Beep function.

Model

The Tone Indicator device is for use when the POS hardware platform provides such capabilities external to the PC standard speaker. Many POS systems have such devices, for example the ICL 92R keyboard, so that an indicator is always present at the point of sale.

This device supports a two-tone sound so that “*siren*” tones can be produced. The indicator is in general also started asynchronously so applications may perform other functions while waiting for the user to acknowledge the tone. There are also options to start the tone asynchronously with no count, so it runs forever, and be stopped when running.

When the indicator is started asynchronously then an **OutputCompleteEvent** is fired when all the tones have been played. This allows the application to know that the tone has stopped. For example when the cash drawer is opened the tone could be started, quietly for a given number of cycles. If the cash drawer is closed then the tone is stopped explicitly by the application, if not then the **OutputCompleteEvent** allows us to alter the prompt to the operator and possibly restart the tone a little louder.

The Tone Indicator follows the general output model. Asynchronous output is handled 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.

  The **Sound** method will not return an error status due to a hardware problem. These errors will only be reported by an **ErrorEvent**. An error status is returned only if the Control is claimed by another application, is not enabled, a parameter is invalid, or the request cannot be enqueued. The first three 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 Control 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. **OutputCompleteEvent**s will not be fired for cleared output. This method also stops any output that may be in progress (when possible).

Examples

Set up an asynchronous two-tone indicator and sounds it 100 times. Each tone is sounded for 750 milliseconds at 50% volume, with no pause between each tone.

```
Indicator.Tone1Pitch = 500
Indicator.Tone1Volume = 50
Indicator.Tone1Duration = 750
Indicator.Tone2Pitch = 800
Indicator.Tone2Volume = 50
Indicator.Tone2Duration = 750
Indicator.InterToneWait = 0

Indicator.AsyncMode = True
Indicator.Sound 100, 0
```

Start a synchronous indicator. This has a simple alternating beep, 500 milliseconds on and 500 milliseconds off.

```
Indicator.Tone1Pitch = 500
Indicator.Tone1Volume = 50
Indicator.Tone1Duration = 500
Indicator.Tone2Pitch = 0              ' turn off second tone
Indicator.InterToneWait = 0           ' no wait after tone-1

Indicator.AsyncMode = False
Indicator.Sound 100, 500
```

The following example will cause an error, as it defines both tones to be zero.

```
Indicator.Tone1Pitch = 0              ' turn off first tone
Indicator.Tone2Pitch = 0              ' turn off second tone

Indicator.Sound 100, 0
```

The indicator **Sound** method can also be used to start an indefinite duration tone. If the *NumberOfCycles* parameter is specified to be OPOS_FOREVER then the tone is started and must be stopped explicitly.

```
Indicator.Tone1Pitch = 500
Indicator.Tone1Volume = 50
Indicator.Tone1Duration = 500
Indicator.Tone2Pitch = 0              ' turn off second tone
Indicator.InterToneWait = 0           ' no wait after tone-1

Indicator.AsyncMode = True
Indicator.Sound OPOS_FOREVER, 500
```

To stop an outstanding tone you have to use the **ClearOutput** or **SoundImmediate** method.

```
...
Indicator.AsyncMode = True
Indicator.Sound OPOS_FOREVER, 500
...
Indicator.ClearOutput
  or
Indicator.SoundImmediate
```

There is also a **SoundImmediate** method which causes both tones to be sounded once with their **InterToneWait.** The tones are sounded synchronously. This imitates a more normal **Beep** function such as that provided by the Win32 API.

```
Indicator.Tone1Pitch = 500
Indicator.Tone1Volume = 50
Indicator.Tone1Duration = 500
Indicator.Tone2Pitch = 0              ' turn off second tone
Indicator.InterToneWait = 0           ' no wait after tone-1

Indicator.AsyncMode = True
Indicator.Sound 1, 0                  ' asynchronous beep
Indicator.SoundImmediate              ' synchronous beep
```

Device Sharing

The Tone Indicator 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 tone indicator, then only that application may call the **Sound** and **SoundImmediate** methods. Use of this feature will effectively restrict the tone indicator to the main POS application if that application claims the device at startup.
- The application that initiates asynchronous sounds is the only one that receives the corresponding **OutputCompleteEvent**s or **ErrorEvent**s.
- See the “Summary” table for precise usage prerequisites.

Properties

AsyncMode Property R/W

Syntax **BOOL AsyncMode;**

Remarks If TRUE, then the **Sound** method will be performed asynchronously.
If FALSE, tones are generated 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.

CapPitch Property

Syntax **BOOL CapPitch;**

Remarks If TRUE, then the hardware tone generator has the ability to vary the pitch of the tone;
otherwise it is FALSE.

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

CapVolume Property

Syntax **BOOL CapVolume;**

Remarks If TRUE, then the hardware tone generator has the ability to vary the volume of the tone;
otherwise it is FALSE.

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

InterToneWait Property R/W

Syntax **LONG InterToneWait;**

Remarks The number of milliseconds of silence between tone-1 and tone-2.
If a gap is required after tone-2 but before a repeat of tone-1, then set the **Sound** parameter *InterSoundWait*.

This property is initialized to zero 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.
OPOS_E_ILLEGAL	An illegal value was specified.

Tone1Duration Property R/W

Syntax **LONG Tone1Duration;**

Remarks The duration of the first tone in milliseconds. A value of zero or less will cause this tone not to sound.

This property is initialized to zero 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.
OPOS_E_ILLEGAL	An illegal value was specified.

Tone1Pitch Property R/W

Syntax **LONG Tone1Pitch;**

Remarks The pitch or frequency of the first tone in hertz. A value of zero or less will cause this tone not to sound.

If the device does not support user-defined pitch (**CapPitch** is FALSE), then any value greater than zero indicates that the tone indicator uses its default value.

This property is initialized to zero 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.
OPOS_E_ILLEGAL	An illegal value was specified.

Tone1Volume Property R/W

Syntax **LONG Tone1Volume;**

Remarks The volume of the first tone in percent of the device's capability, where 0 (or less) is silent and 100 (or more) is maximum.

If the device does not support user-defined volume (**CapVolume** is FALSE), then any value greater than zero indicates that the tone indicator uses its default value.

This property is initialized to 100 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.
OPOS_E_ILLEGAL	An illegal value was specified.

Tone2Duration Property R/W

Syntax **LONG Tone2Duration;**

Remarks The duration of the second tone in milliseconds. A value of zero or less will cause this tone not to sound.

This property is initialized to zero 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.
OPOS_E_ILLEGAL	An illegal value was specified.

Tone2Pitch Property R/W

Syntax **LONG Tone2Pitch;**

Remarks The pitch or frequency of the second tone in hertz. A value of zero or less will cause this tone not to sound.

If the device does not support user-defined pitch (**CapPitch** is FALSE), then any value greater than zero indicates that the tone indicator uses its default value.

This property is initialized to zero 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.
OPOS_E_ILLEGAL	An illegal value was specified.

Tone2Volume Property R/W

Syntax **LONG Tone2Volume;**

Remarks The volume of the second tone in percent of the device's capability, where 0 (or less) is silent and 100 (or more) is maximum.

If the device does not support user-defined volume (**CapVolume** is FALSE), then any value greater than zero indicates that the tone indicator uses its default value.

This property is initialized to 100 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.
OPOS_E_ILLEGAL	An illegal value was specified.

Methods

Sound Method

Syntax **LONG Sound (LONG** *NumberOfCycles,* **LONG** *InterSoundWait***);**

Parameter	**Description**
NumberOfCycles	If OPOS_FOREVER, then start the indicator sounding, and repeat continuously. Else perform the specified number of cycles.
InterSoundWait	When *NumberOfCycles* is not one, then pause for *InterSoundWait* milliseconds before repeating the tone cycle (before playing tone-1 again).

Remarks Sound the indicator, or start it sounding asynchronously.

This method is performed synchronously if **AsyncMode** is FALSE, and asynchronously if **AsyncMode** is TRUE.

The duration of an indicator cycle is:

Tone1Duration property +
InterToneWait property +
Tone2Duration property +
InterSoundWait parameter (except on the last tone cycle)

After the tone indicator has started an asynchronous sound, then the sound may be stopped by using one of the following methods. (When an *InterSoundWait* value of OPOS_FOREVER was used to start the sound, then the application must use one of these to stop the continuous sounding of the tones.)

- **ClearOutput**
- **SoundImmediate**

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 indicator was sounded or has been started.
OPOS_E_ILLEGAL	One of the following errors occurred:

- *NumberOfCycles* is neither a positive, non-zero value nor OPOS_FOREVER.
- *NumberOfCycles* is OPOS_FOREVER when **AsyncMode** is FALSE.
- A negative *InterSoundWait* was specified

Other Values	See **ResultCode**.

SoundImmediate Method

Syntax **LONG SoundImmediate ();**

Remarks Sounds the hardware tone generator once, synchronously. Both tone-1 and tone-2 are sounded, with their **InterToneWait**.

If asynchronous output is outstanding, then it is terminated before playing the immediate sound (as if **ClearOutput** were called). **SoundImmediate** is primarily intended for use in exception conditions when asynchronous output is outstanding, such as within an error event handler.

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 indicator was sounded or has been started.
Other Values	See **ResultCode**.

APPENDIX 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 1000*xxx*, where *xxx* is a vendor-specific build number. The **ServiceObjectVersion** for a compliant Service Object is 1000*xxx*, 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 **Release**d, then re**Claim**ed, 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	*BeepFrequency* 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.

Section	Change
POS Keyboard	New device: Add information in several locations, plus POS Keyboard chapter and header file.
Second Page	Remove CompuServe reference.
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.

Release 1.2

Release 1.2 adds additional device classes, plus additional APIs based on requirements from various OPOS-US, OPOS-Japan, and OPOS-Europe members.

Release 1.2 is a superset of Release 1.1.

Section	Change
Cash Changer	New device: Add information in several locations, plus Cash Changer chapter and header file.
Tone Indicator	New device: Add information in several locations, plus Tone Indicator chapter and header file.
Several places	When a method has a *Timeout* parameter, added the constant OPOS_FOREVER as a value, and noted that OPOS_E_ILLEGAL can be returned.
First Two Pages	Update company names. Update copyright notices. Update web reference.
Introduction **How an Application Uses an OPOS Control** and **Device Sharing Model**	Explicitly state that a control may be simultaneously opened by many applications, but may be restricted in its functionality based on the **Claim** method.
Introduction **Events**	Add this section.
Introduction **Input Model**	Clarify the handling of error conditions. Add usage of **AutoDisable** and **DataCount**. Clarify the Error state exit conditions. Clarify when **ClearInput** is legal.
Introduction **Output Model**	Clarify the Error state conditions.
Introduction **Result Code Model**	Clarify the setting of **ResultCodeExtended.**

Common **BinaryConversion**, **AutoDisable**, and **DataCount** properties	Add these new properties. Throughout document, add to Summary sections for each device class. Throughout document, specify the BString properties and method parameters that are affected by **BinaryConversion**.
Common **ControlObjectVersion** and **ServiceObjectVersion** properties	Add compliance information when versions don' t match.
Common **FreezeEvents** property	Clarify **FreezeEvents** role in delaying event firing.
Common **ResultCodeExtended** property	Clarify the setting of **ResultCodeExtended.**
Common **ClearInput** and **ClearOutput** methods	Correct return value information: May return one of three statuses.
Common **Open** method	Correct return value information: **ResultCode** may not match method return value.
Common **Release** method	Correct **DeviceEnabled** side effects: Only exclusive use devices are disabled during the **Release**.
Common **StatusUpdateEvent** event	Clarify the initial firing of events at device enable.
MICR **BankNumber**	Correct definition to digits 4-8 of the **TransitNumber**.
MSR **ErrorReportingType**	Add this new property.
MSR **ParseDecodeData**	Clarify inconsistency: Both **ParseDecodeData** and **ParseDecodedData** were used for this property.
MSR **ErrorEvent**	Update for track level error notification.
POS Keyboard General Information	Clarify the type of keyboards that may be a POS Keyboard.

POS Keyboard **POSKeyData** property	
	Update definition of this property: A logical key value..
POS Keyboard **CapKeyUp**, **EventTypes**, and **POSKeyEventType** properties	
	Add these new properties.
POS Printer Escape Sequences	
	Clarify that escape sequences that are not OPOS sequences are passed through to the printer.
POS Printer **CapConcurrentXxxYyy**	
	Clarify the interpretation of a FALSE value.
POS Printer **XxxLineSpacing**	
	Clarify that line spacing includes the printed line height. Could have been interpreted as only the whitespace between each pair of lines.
POS Printer **PrintBarCode**	
	Add list of symbologies.
POS Printer **MapMode** and **XxxLetterQuality**	
	Clarified legal handling of **MapMode** when the printer supports half-dots. Clarified potential impact on metrics when **XxxLetterQuality** is changed and **MapMode** is dots.
POS Printer **SetBitmap**	Extend the bitmap number usage to allow the same bitmap to be used for both receipt and slip.
POS Printer **TransactionPrint**	
	Clarify when Busy and Extended statuses may be returned.
POS Printer **ValidateData**	
	Add "Underline" to the Illegal status section.
Scale Model	Correct to state the weight unit is defined by the device, and not settable by the application.
Scale **CapDisplay**	Add this new property.
Scale **WeightUnit**	Clarify inconsistency: Both **WeightUnit** and **WeightUnits** were used for this property.
Scanner **ScanDataLabel** and **ScanDataType**	
	Add these new properties.

Signature Capture "Real Time" feature	Add the new properties **CapRealTimeData** and **RealTimeDataEnabled**. Update various sections for real time operation.
Change History **Release 1.1**	Remove the compliance requirements for 1.1 Control Objects. This information was corrected and added to the common **ControlObjectVersion** and **ServiceObjectVersion** properties.
Opos.h header file	Add OPOS_FOREVER constant. Add **BinaryConversion** values.
OposMsr.h header file	Add **ErrorReportingType** values.
OposKbd.h header file	Add **EventTypes** values.
OposPtr.h header file	Remove PTR_RP_NORMAL_ASYNC. Add symbologies to match scanner.
OposScan.h header file	Add symbologies for **ScanDataType**.
Technical Details "Event Handlers"	Delete section. Much of the information was inaccurate, and the rest was merged into the new "Events" section in the first chapter.
Throughout	Correct various editing errors.

Release 1.3

Release 1.3 adds additional device classes, a few additional APIs, and some corrections.

Release 1.3 is a superset of Release 1.2.

Section	Change
First Two Pages	Update copyright notices. Update web reference.
General	Modify the use of the term event "firing." Use "enqueue" and "deliver" appropriately to describe event firing.
Bump Bar	New device: Add information in several locations, plus Bump Bar chapter and header file.
Fiscal Printer	New device: Add information in several locations, plus Fiscal Printer chapter and header file.
PIN Pad	New device: Add information in several locations, plus PIN Pad chapter and header file.
Remote Order Display	New device: Add information in several locations, plus Remote Order Display chapter and header file.
Several places	Relax **ErrorEvent** "retry" response to allow its use with some input devices.
Introduction **Events**	Clarify effect of the top event being blocked.
Introduction **Input Model**	Add details concerning enqueuing and delivery of **ErrorEvent**s. Add description of asynchronous input.
Introduction **Device Power Reporting Model**	Add this section.
Introduction **OPOS Control Descriptions**	Add CURRENCY data type.

Common **CapPowerReporting, PowerNotify, PowerState** properties

Add these properties here, plus…
Add to the Summary section of each device.

Common **ResultCode** property

Generalize the meaning of OPOS_E_BUSY.

Common **StatusUpdateEvent**

Add power state reporting information.

Change parameter name from *Data* to *Status.*

Every Device	Add power reporting properties to Summary section. Add **StatusUpdateEvent** support (if previously not reported. Add power reporting reference to existing **StatusUpdateEvent** descriptions.
MSR **DecodeData**	Add "raw format" description and column to track data table.
MSR **ExpirationDate**	Specify the format.
MSR **Track*x*Data**	Specify that data excludes the sentinels and LRC. Add that decoding occurs when **DecodeData** is TRUE.
MSR **ErrorEvent**	Clarify that **DataCount** and **AutoDisable** are not relevant for MSR error events.

POSPrinter ***Xxx*LineChars**

Add implementation recommendations.

POSPrinter **PrintTwoNormal**

Clarify the meaning of the *Stations* parameter, including the addition of new constants.

Scale

Add the following features:

- Asynchronous input. Property **AsyncMode.** Method **ClearInput,** updates to **ReadWeight**. Events **DataEvent** and **ErrorEvent.**
- Display of text. Properties **CapDisplayText**, **MaxDisplayTextChars.** Method **DisplayText.**
- Price calculation. Properties **CapPriceCalculating, SalesPrice, UnitPrice.**

	• Tare weight. Properties **CapTareWeight, TareWeight.** • Scale zeroing. Property **CapZeroScale.** Method **ZeroScale.**
Tone Indicator **Summary** and **General Information**' s **Device Sharing**	Consistently specify that Tone Indicator is a sharable device.
Opos.h header file	Add **CapPowerReporting, PowerState,** and **PowerNotify** properties. Add **StatusUpdateEvent** power reporting values.
OposPtr.h header file	Add new **PrintTwoNormal** station constants.
Throughout	Correct some editing errors.

Release 1.4

Release 1.4 adds one additional device class.

Release 1.4 is a super set of Release 1.3.

Section	Change
CAT	Added new device class, Credit Authorization Terminal which includes CAT chapter and header file. This device class was added at the request of OPOS-J and is used primarily in Japan. No other revisions were made to the version 1.3 of the OPOS specification.

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:

BumpBar
CashChanger
CashDrawer
CAT
CoinDispenser

FiscalPrinter
HardTotals
Keylock
LineDisplay
MICR
MSR
PINPad
POSKeyboard
POSPrinter
RemoteOrderDisplay
Scale
Scanner
SignatureCapture
ToneIndicator

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 ID[9] 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.

Value – Required	**Data**
(Default)	Service Object’ s OLE Programmatic ID.

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 – Recommended	**Data**
Service	Filename of the Service Object.

[9] 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.

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

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
|
|→  \SOFTWARE
|    |
|    |→  \OLEforRetail
↓    |    |
     |    |→  \ServiceOPOS
     ↓    |    |
          |    |→  \POSPrinter                                        Device Class Key
          |    |    |
          |    |    |→  \NCR7156=NCR.Ptr7156.1                        Device Name Key
          |    |    |        Service=C:\OPOS\NCR\PTR7156.DLL
          |    |    |        Description=NCR 7156 Serial Printer
          |    |    |        Version=1.0.12
          |    |    |        ...Service Object-specific values. Might include:
          |    |    |        Port=COM3
          |    |    |        BaudRate=9600
          |    |    |
          |    |    |→  \Epson950=Epson.PtrTMU950.1                   Device Name Key
          |    |    |        Service=TMU950.EXE
          |    |    |        Description=Epson TM-U950 Printer
          |    |    |        Version=1.0.7
          |    |    |        ...Service Object-specific values could go here.
          |    |    |
          |    |    |→  PSI.Ptr.1=NCR7156                          Logical Device Name
          |    |    |
          |    |    |→  PSI.Ptr.2=Epson950                         Logical Device Name
          |    |
          |    |→  \CashDrawer                                        Device Class Key
          |         |
          |         |→  \EpsonCash=Epson.CD.1                         Device Name Key
          |         |        Service=TMU950.EXE
          |         |        Description=Epson Cash Drawer Kickout on TM-U950
          |         |        Version=1.0.7
          |         |        ...Service Object-specific values. Might include:
          |         |        Uses=POSPrinter\Epson950
          |         |
          |         |→  PSI.CD.1=EpsonCash                         Logical Device Name
          |
          |→  \ServiceInfo
               |
               |→  \EPSON
               |        InstallDir=C:\OPOS\EPSON
               |        InstallDate=1995/11/13
               ↓
```

APPENDIX C

OPOS Application Header Files

The header files are listed in alphabetical order. The mapping of device class name to header file name is as follows:

– General –	Opos.h
Bump Bar	OposBb.h
Cash Changer	OposChan.h
Cash Drawer	OposCash.h
CAT	OposCat.h
Coin Dispenser	OposCoin.h
Fiscal Printer	OposFptr.h
Hard Totals	OposTot.h
Keylock	OposLock.h
Line Display	OposDisp.h
MICR	OposMicr.h
MSR	OposMsr.h
PIN Pad	OposPpad.h
POS Keyboard	OposKbd.h
POS Printer	OposPtr.h
Remote Order Display	OposRod.h
Scale	OposScal.h
Scanner	OposScan.h
Signature Capture	OposSig.h
Tone Indicator	OposTone.h

Opos.h : Main OPOS Header File

```
//////////////////////////////////////////////////////////////////////
//
// Opos.h
//
//   General header file for OPOS Applications.
//
// Modification history
// ------------------------------------------------------------------
// 95-12-08 OPOS Release 1.0                         CRM
// 97-06-04 OPOS Release 1.2                         CRM
//   Add OPOS_FOREVER.
//   Add BinaryConversion values.
// 98-03-06 OPOS Release 1.3                         CRM
//   Add CapPowerReporting, PowerState, and PowerNotify values.
//   Add power reporting values for StatusUpdateEvent.
//
//////////////////////////////////////////////////////////////////////

#if !defined(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;


//////////////////////////////////////////////////////////////////////
// OPOS "BinaryConversion" Property Constants
//////////////////////////////////////////////////////////////////////

const LONG OPOS_BC_NONE         = 0;
const LONG OPOS_BC_NIBBLE       = 1;
const LONG OPOS_BC_DECIMAL      = 2;


//////////////////////////////////////////////////////////////////////
// "CheckHealth" Method: "Level" Parameter Constants
//////////////////////////////////////////////////////////////////////

const LONG OPOS_CH_INTERNAL     = 1;
const LONG OPOS_CH_EXTERNAL     = 2;
const LONG OPOS_CH_INTERACTIVE  = 3;


//////////////////////////////////////////////////////////////////////
// OPOS "CapPowerReporting", "PowerState", "PowerNotify ”Property
//   Constants
//////////////////////////////////////////////////////////////////////

const LONG OPOS_PR_NONE         = 0;
const LONG OPOS_PR_STANDARD     = 1;
const LONG OPOS_PR_ADVANCED     = 2;

const LONG OPOS_PN_DISABLED     = 0;
const LONG OPOS_PN_ENABLED      = 1;

const LONG OPOS_PS_UNKNOWN      = 2000;
const LONG OPOS_PS_ONLINE       = 2001;
const LONG OPOS_PS_OFF          = 2002;
const LONG OPOS_PS_OFFLINE      = 2003;
const LONG OPOS_PS_OFF_OFFLINE  = 2004;


//////////////////////////////////////////////////////////////////////
// "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;
```

```
//////////////////////////////////////////////////////////////////////
// "StatusUpdateEvent" Event: Common "Status" Constants
//////////////////////////////////////////////////////////////////////

const LONG OPOS_SUE_POWER_ONLINE      = 2001;
const LONG OPOS_SUE_POWER_OFF         = 2002;
const LONG OPOS_SUE_POWER_OFFLINE     = 2003;
const LONG OPOS_SUE_POWER_OFF_OFFLINE = 2004;


//////////////////////////////////////////////////////////////////////
// General Constants
//////////////////////////////////////////////////////////////////////

const LONG OPOS_FOREVER         = -1;


#endif                // !defined(OPOS_H)
```

OposBb.h: Bump Bar Header File

```
/////////////////////////////////////////////////////////////////////
//
// OposBb.h
//
//   Bump Bar header file for OPOS Applications.
//
// Modification history
// ------------------------------------------------------------------
// 98-03-06 OPOS Release 1.3                              BB
//
/////////////////////////////////////////////////////////////////////

#if !defined(OPOSBB_H)
#define      OPOSBB_H


#include "Opos.h"


/////////////////////////////////////////////////////////////////////
// "CurrentUnitID" and "UnitsOnline" Properties
//  and "Units" Parameter Constants
/////////////////////////////////////////////////////////////////////

#define BB_UID(Unit) (1 << (Unit-1))

const LONG BB_UID_1            = BB_UID(1);
const LONG BB_UID_2            = BB_UID(2);
const LONG BB_UID_3            = BB_UID(3);
const LONG BB_UID_4            = BB_UID(4);
const LONG BB_UID_5            = BB_UID(5);
const LONG BB_UID_6            = BB_UID(6);
const LONG BB_UID_7            = BB_UID(7);
const LONG BB_UID_8            = BB_UID(8);
const LONG BB_UID_9            = BB_UID(9);
const LONG BB_UID_10            = BB_UID(10);
const LONG BB_UID_11            = BB_UID(11);
const LONG BB_UID_12            = BB_UID(12);
const LONG BB_UID_13            = BB_UID(13);
const LONG BB_UID_14            = BB_UID(14);
const LONG BB_UID_15            = BB_UID(15);
const LONG BB_UID_16            = BB_UID(16);
const LONG BB_UID_17            = BB_UID(17);
const LONG BB_UID_18            = BB_UID(18);
const LONG BB_UID_19            = BB_UID(19);
const LONG BB_UID_20            = BB_UID(20);
const LONG BB_UID_21            = BB_UID(21);
const LONG BB_UID_22            = BB_UID(22);
const LONG BB_UID_23            = BB_UID(23);
const LONG BB_UID_24            = BB_UID(24);
const LONG BB_UID_25            = BB_UID(25);
const LONG BB_UID_26            = BB_UID(26);
const LONG BB_UID_27            = BB_UID(27);
const LONG BB_UID_28          = BB_UID(28);
const LONG BB_UID_29          = BB_UID(29);
const LONG BB_UID_30          = BB_UID(30);
const LONG BB_UID_31          = BB_UID(31);
const LONG BB_UID_32          = BB_UID(32);


/////////////////////////////////////////////////////////////////////
// "DataEvent" Event: "Status" Parameter Constants
/////////////////////////////////////////////////////////////////////

const LONG BB_DE_KEY           = 0x01;


#endif                  // !defined(OPOSBB_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
// 98-03-06 OPOS Release 1.3                                  CRM
//
/////////////////////////////////////////////////////////////////////

#if !defined(OPOSCASH_H)
#define      OPOSCASH_H


#include "Opos.h"


/////////////////////////////////////////////////////////////////////
// "StatusUpdateEvent" Event Constants
/////////////////////////////////////////////////////////////////////

const LONG CASH_SUE_DRAWERCLOSED = 0;
const LONG CASH_SUE_DRAWEROPEN   = 1;


#endif                  // !defined(OPOSCASH_H)
```

OposCat.h : CAT Header File

```
/////////////////////////////////////////////////////////////////////
//
// OposCAT.h
//
//   CAT header file for OPOS Applications.
//
// Modification history
// ------------------------------------------------------------------
// 98-06-01 OPOS Release 1.4                              OPOS-J
//
//
//
/////////////////////////////////////////////////////////////////////

#if !defined(OPOSCAT_H)
#define      OPOSCAT_H


#include "Opos.h"


/////////////////////////////////////////////////////////////////////
// Payment Condition Constants
/////////////////////////////////////////////////////////////////////

const LONG CAT_PAYMENT_LUMP                  = 10;
const LONG CAT_PAYMENT_BONUS_1               = 21;
const LONG CAT_PAYMENT_BONUS_2               = 22;
const LONG CAT_PAYMENT_BONUS_3               = 23;
const LONG CAT_PAYMENT_BONUS_4               = 24;
const LONG CAT_PAYMENT_BONUS_5               = 25;
const LONG CAT_PAYMENT_INSTALLMENT_1         = 61;
const LONG CAT_PAYMENT_INSTALLMENT_2         = 62;
const LONG CAT_PAYMENT_INSTALLMENT_3         = 63;
const LONG CAT_PAYMENT_BONUS_COMBINATION_1  = 31;
const LONG CAT_PAYMENT_BONUS_COMBINATION_2  = 32;
const LONG CAT_PAYMENT_BONUS_COMBINATION_3  = 33;
const LONG CAT_PAYMENT_BONUS_COMBINATION_4  = 34;
const LONG CAT_PAYMENT_REVOLVING             = 80;


/////////////////////////////////////////////////////////////////////
// Transaction Type Constants
/////////////////////////////////////////////////////////////////////

const LONG CAT_TRANSACTION_SALES             = 10;
const LONG CAT_TRANSACTION_VOID              = 20;
const LONG CAT_TRANSACTION_REFUND            = 21;
const LONG CAT_TRANSACTION_VOIDPRESALES      = 29;
const LONG CAT_TRANSACTION_COMPLETION        = 30;
const LONG CAT_TRANSACTION_PRESALES          = 40;
const LONG CAT_TRANSACTION_CHECKCARD         = 41;
```

```
/////////////////////////////////////////////////////////////////////
// ResultCodeExtended Constants
/////////////////////////////////////////////////////////////////////

const LONG OPOS_ECAT_CENTERERROR           = 01;
const LONG OPOS_ECAT_COMMANDERROR          = 90;
const LONG OPOS_ECAT_RESET                 = 91;
const LONG OPOS_ECAT_COMMUNICATIONERROR    = 92;
const LONG OPOS_ECAT_DAILYLOGOVERFLOW      = 200;


/////////////////////////////////////////////////////////////////////
// "Daily Log" Property  & Argument Constants
/////////////////////////////////////////////////////////////////////

const LONG CAT_DL_NONE                  = 0;  //None of them
const LONG CAT_DL_REPORTING             = 1;  //Only Reporting
const LONG CAT_DL_SETTLEMENT            = 2;  //Only Settlement
const LONG CAT_DL_REPORTING_SETTLEMENT  = 3;  //Both of them



#endif              // !defined(OPOSCAT_H)
```

OposChan.h : Cash Changer Header File

```
//////////////////////////////////////////////////////////////////////
//
// OposChan.h
//
//   Cash Changer header file for OPOS Applications.
//
// Modification history
// ------------------------------------------------------------------
// 97-06-04 OPOS Release 1.2                               CRM
//
//////////////////////////////////////////////////////////////////////

#if !defined(OPOSCHAN_H)
#define      OPOSCHAN_H


#include "Opos.h"


//////////////////////////////////////////////////////////////////////
// "DeviceStatus" and "FullStatus" Property Constants
// "StatusUpdateEvent" Event Constants
//////////////////////////////////////////////////////////////////////

const LONG CHAN_STATUS_OK        =  0; // DeviceStatus, FullStatus

const LONG CHAN_STATUS_EMPTY     = 11; // DeviceStatus, StatusUpdateEvent
const LONG CHAN_STATUS_NEAREMPTY= 12; // DeviceStatus, StatusUpdateEvent
const LONG CHAN_STATUS_EMPTYOK   = 13; // StatusUpdateEvent

const LONG CHAN_STATUS_FULL      = 21; // FullStatus, StatusUpdateEvent
const LONG CHAN_STATUS_NEARFULL = 22; // FullStatus, StatusUpdateEvent
const LONG CHAN_STATUS_FULLOK    = 23; // StatusUpdateEvent

const LONG CHAN_STATUS_JAM       = 31; // DeviceStatus, StatusUpdateEvent
const LONG CHAN_STATUS_JAMOK     = 32; // StatusUpdateEvent

const LONG CHAN_STATUS_ASYNC     = 91; // StatusUpdateEvent


//////////////////////////////////////////////////////////////////////
// "ResultCodeExtended" Property Constants for Cash Changer
//////////////////////////////////////////////////////////////////////

const LONG OPOS_ECHAN_OVERDISPENSE = 1 + OPOSERREXT;


#endif                 // !defined(OPOSCHAN_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)
```

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)
```

OposFptr.h : Fiscal Printer Header File

```
/////////////////////////////////////////////////////////////////////
//
// OposFptr.h
//
//   Fiscal Printer header file for OPOS Applications.
//
// Modification history
// ------------------------------------------------------------------
// 98-03-06 OPOS Release 1.3                              PDU
//
/////////////////////////////////////////////////////////////////////

#if !defined(OPOSFPTR_H)
#define      OPOSFPTR_H


#include "Opos.h"


/////////////////////////////////////////////////////////////////////
// Fiscal Printer Station Constants
/////////////////////////////////////////////////////////////////////

const LONG FPTR_S_JOURNAL                  = 1;
const LONG FPTR_S_RECEIPT                  = 2;
const LONG FPTR_S_SLIP                     = 4;

const LONG FPTR_S_JOURNAL_RECEIPT = FPTR_S_JOURNAL | FPTR_S_RECEIPT;


/////////////////////////////////////////////////////////////////////
// "CountryCode" Property Constants
/////////////////////////////////////////////////////////////////////

const LONG FPTR_CC_BRAZIL                  =  1;
const LONG FPTR_CC_GREECE                  =  2;
const LONG FPTR_CC_HUNGARY                 =  3;
const LONG FPTR_CC_ITALY                   =  4;
const LONG FPTR_CC_POLAND                  =  5;
const LONG FPTR_CC_TURKEY                  =  6;


/////////////////////////////////////////////////////////////////////
// "ErrorLevel" Property Constants
/////////////////////////////////////////////////////////////////////

const LONG FPTR_EL_NONE                    =  1;
const LONG FPTR_EL_RECOVERABLE             =  2;
const LONG FPTR_EL_FATAL                   =  3;
const LONG FPTR_EL_BLOCKED                 =  4;


/////////////////////////////////////////////////////////////////////
```

```
// "ErrorState", "PrinterState" Property Constants
//////////////////////////////////////////////////////////////////////

const LONG FPTR_PS_MONITOR                  =  1;
const LONG FPTR_PS_FISCAL_RECEIPT           =  2;
const LONG FPTR_PS_FISCAL_RECEIPT_TOTAL     =  3;
const LONG FPTR_PS_FISCAL_RECEIPT_ENDING    =  4;
const LONG FPTR_PS_FISCAL_DOCUMENT          =  5;
const LONG FPTR_PS_FIXED_OUTPUT             =  6;
const LONG FPTR_PS_ITEM_LIST                =  7;
const LONG FPTR_PS_LOCKED                   =  8;
const LONG FPTR_PS_NONFISCAL                =  9;
const LONG FPTR_PS_REPORT                   = 10;


//////////////////////////////////////////////////////////////////////
// "SlipSelection" Property Constants
//////////////////////////////////////////////////////////////////////

const LONG FPTR_SS_FULL_LENGTH              =  1;
const LONG FPTR_SS_VALIDATION               =  2;


//////////////////////////////////////////////////////////////////////
// "GetData" Method Constants
//////////////////////////////////////////////////////////////////////

const LONG FPTR_GD_CURRENT_TOTAL            =  1;
const LONG FPTR_GD_DAILY_TOTAL              =  2;
const LONG FPTR_GD_RECEIPT_NUMBER           =  3;
const LONG FPTR_GD_REFUND                   =  4;
const LONG FPTR_GD_NOT_PAID                 =  5;
const LONG FPTR_GD_MID_VOID                 =  6;
const LONG FPTR_GD_Z_REPORT                 =  7;
const LONG FPTR_GD_GRAND_TOTAL              =  8;
const LONG FPTR_GD_PRINTER_ID               =  9;
const LONG FPTR_GD_FIRMWARE                 = 10;
const LONG FPTR_GD_RESTART                  = 11;


//////////////////////////////////////////////////////////////////////
// "AdjustmentType" arguments in diverse methods
//////////////////////////////////////////////////////////////////////

const LONG FPTR_AT_AMOUNT_DISCOUNT          =  1;
const LONG FPTR_AT_AMOUNT_SURCHARGE         =  2;
const LONG FPTR_AT_PERCENTAGE_DISCOUNT      =  3;
const LONG FPTR_AT_PERCENTAGE_SURCHARGE     =  4;


//////////////////////////////////////////////////////////////////////
// "ReportType" argument in "PrintReport" method
//////////////////////////////////////////////////////////////////////

const LONG FPTR_RT_ORDINAL                  =  1;
const LONG FPTR_RT_DATE                     =  2;
```

```
//////////////////////////////////////////////////////////////////////
// "StatusUpdateEvent" Event: "Data" Parameter Constants
//////////////////////////////////////////////////////////////////////

const LONG FPTR_SUE_COVER_OPEN          =  11;
const LONG FPTR_SUE_COVER_OK            =  12;

const LONG FPTR_SUE_JRN_EMPTY           =  21;
const LONG FPTR_SUE_JRN_NEAREMPTY       =  22;
const LONG FPTR_SUE_JRN_PAPEROK         =  23;

const LONG FPTR_SUE_REC_EMPTY           =  24;
const LONG FPTR_SUE_REC_NEAREMPTY       =  25;
const LONG FPTR_SUE_REC_PAPEROK         =  26;

const LONG FPTR_SUE_SLP_EMPTY           =  27;
const LONG FPTR_SUE_SLP_NEAREMPTY       =  28;
const LONG FPTR_SUE_SLP_PAPEROK         =  29;

const LONG FPTR_SUE_IDLE                =1001;


//////////////////////////////////////////////////////////////////////
// "ResultCodeExtended" Property Constants for Fiscal Printer
//////////////////////////////////////////////////////////////////////

const LONG OPOS_EFPTR_COVER_OPEN = 1 + OPOSERREXT; // (Several)
const LONG OPOS_EFPTR_JRN_EMPTY  = 2 + OPOSERREXT; // (Several)
const LONG OPOS_EFPTR_REC_EMPTY  = 3 + OPOSERREXT; // (Several)
const LONG OPOS_EFPTR_SLP_EMPTY  = 4 + OPOSERREXT; // (Several)
const LONG OPOS_EFPTR_SLP_FORM   = 5 + OPOSERREXT; // EndRemoval
const LONG OPOS_EFPTR_MISSING_DEVICES          =
        6 + OPOSERREXT; // (Several)
const LONG OPOS_EFPTR_WRONG_STATE              =
        7 + OPOSERREXT; // (Several)
const LONG OPOS_EFPTR_TECHNICAL_ASSISTANCE     =
        8 + OPOSERREXT; // (Several)
const LONG OPOS_EFPTR_CLOCK_ERROR              =
        9 + OPOSERREXT; // (Several)
const LONG OPOS_EFPTR_FISCAL_MEMORY_FULL       =
        10 + OPOSERREXT; // (Several)
const LONG OPOS_EFPTR_FISCAL_MEMORY_DISCONNECTED =
        11 + OPOSERREXT; // (Several)
const LONG OPOS_EFPTR_FISCAL_TOTALS_ERROR      =
        12 + OPOSERREXT; // (Several)
const LONG OPOS_EFPTR_BAD_ITEM_QUANTITY        =
        13 + OPOSERREXT; // (Several)
const LONG OPOS_EFPTR_BAD_ITEM_AMOUNT          =
        14 + OPOSERREXT; // (Several)
const LONG OPOS_EFPTR_BAD_ITEM_DESCRIPTION     =
        15 + OPOSERREXT; // (Several)
const LONG OPOS_EFPTR_RECEIPT_TOTAL_OVERFLOW   =
        16 + OPOSERREXT; // (Several)
const LONG OPOS_EFPTR_BAD_VAT                  =
        17 + OPOSERREXT; // (Several)
const LONG OPOS_EFPTR_BAD_PRICE                =
          18 + OPOSERREXT; // (Several)
const LONG OPOS_EFPTR_BAD_DATE                 =
          19 + OPOSERREXT; // (Several)
const LONG OPOS_EFPTR_NEGATIVE_TOTAL           =
          20 + OPOSERREXT; // (Several)
const LONG OPOS_EFPTR_WORD_NOT_ALLOWED         =
          21 + OPOSERREXT; // (Several)


#endif                // !defined(OPOSFPTR_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
// 97-06-04 OPOS Release 1.2                               CRM
//   Add "EventTypes" and "POSKeyEventType" values.
//
/////////////////////////////////////////////////////////////////////

#if !defined(OPOSKBD_H)
#define     OPOSKBD_H


#include "Opos.h"


/////////////////////////////////////////////////////////////////////
// "EventTypes" Property Constants
/////////////////////////////////////////////////////////////////////

const LONG KBD_ET_DOWN          =   1;
const LONG KBD_ET_DOWN_UP       =   2;


/////////////////////////////////////////////////////////////////////
// "POSKeyEventType" Property Constants
/////////////////////////////////////////////////////////////////////

const LONG KBD_KET_KEYDOWN      =   1;
const LONG KBD_KET_KEYUP        =   2;


#endif                // !defined(OPOSKBD_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
// 97-06-04 OPOS Release 1.2                              CRM
//   Add ErrorReportingType values.
//
/////////////////////////////////////////////////////////////////////

#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;


/////////////////////////////////////////////////////////////////////
// "ErrorReportingType" Property Constants
/////////////////////////////////////////////////////////////////////

const LONG MSR_ERT_CARD         = 0;
const LONG MSR_ERT_TRACK        = 1;


/////////////////////////////////////////////////////////////////////
// "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)
```

OposPpad.h : PIN Pad Header File

```
/////////////////////////////////////////////////////////////////////
//
// OposPpad.h
//
//   PIN Pad header file for OPOS Applications.
//
// Modification history
// -----------------------------------------------------------------
// 98-03-06 OPOS Release 1.3                              JDB
//
/////////////////////////////////////////////////////////////////////

#if !defined(OPOSPPAD_H)
#define      OPOSPPAD_H


#include "Opos.h"


/////////////////////////////////////////////////////////////////////
// "CapDisplay" Property Constants
/////////////////////////////////////////////////////////////////////

const LONG PPAD_DISP_UNRESTRICTED          = 1;
const LONG PPAD_DISP_PINRESTRICTED         = 2;
const LONG PPAD_DISP_RESTRICTED_LIST       = 3;
const LONG PPAD_DISP_RESTRICTED_ORDER      = 4;


/////////////////////////////////////////////////////////////////////
// "AvailablePromptsList" and "Prompt" Property Constants
/////////////////////////////////////////////////////////////////////

const LONG PPAD_MSG_ENTERPIN               = 1;
const LONG PPAD_MSG_PLEASEWAIT             = 2;
const LONG PPAD_MSG_ENTERVALIDPIN          = 3;
const LONG PPAD_MSG_RETRIESEXCEEDED        = 4;
const LONG PPAD_MSG_APPROVED               = 5;
const LONG PPAD_MSG_DECLINED               = 6;
const LONG PPAD_MSG_CANCELED               = 7;
const LONG PPAD_MSG_AMOUNTOK               = 8;
const LONG PPAD_MSG_NOTREADY               = 9;
const LONG PPAD_MSG_IDLE                   = 10;
const LONG PPAD_MSG_SLIDE_CARD             = 11;
const LONG PPAD_MSG_INSERTCARD             = 12;
const LONG PPAD_MSG_SELECTCARDTYPE         = 13;


/////////////////////////////////////////////////////////////////////
// "CapLanguage" Property Constants
/////////////////////////////////////////////////////////////////////

const LONG PPAD_LANG_NONE                  = 1;
const LONG PPAD_LANG_ONE                = 2;
const LONG PPAD_LANG_PINRESTRICTED      = 3;
const LONG PPAD_LANG_UNRESTRICTED       = 4;

//////////////////////////////////////////////////////////////////////
// "TransactionType" Property Constants
//////////////////////////////////////////////////////////////////////

const LONG PPAD_TRANS_DEBIT             = 1;
const LONG PPAD_TRANS_CREDIT            = 2;
const LONG PPAD_TRANS_INQ               = 3;
const LONG PPAD_TRANS_RECONCILE         = 4;
const LONG PPAD_TRANS_ADMIN             = 5;


//////////////////////////////////////////////////////////////////////
// "EndEFTTransaction" Method Completion Code Constants
//////////////////////////////////////////////////////////////////////

const LONG PPAD_EFT_NORMAL              = 1;
const LONG PPAD_EFT_ABNORMAL            = 2;


//////////////////////////////////////////////////////////////////////
// "DataEvent" Event Status Constants
//////////////////////////////////////////////////////////////////////
const LONG PPAD_SUCCESS                 = 1;
const LONG PPAD_CANCEL                  = 2;


#endif          // !defined(OPOSPPAD_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.
// 97-06-04 OPOS Release 1.2                              CRM
//   Remove PTR_RP_NORMAL_ASYNC.
//   Add more barcode symbologies.
// 98-03-06 OPOS Release 1.3                              CRM
//   Add more PrintTwoNormal constants.
//
///////////////////////////////////////////////////////////////////////

#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    ;

const LONG PTR_TWO_RECEIPT_JOURNAL  = 0x8000 + PTR_S_JOURNAL_RECEIPT;
const LONG PTR_TWO_SLIP_JOURNAL     = 0x8000 + PTR_S_JOURNAL_SLIP   ;
const LONG PTR_TWO_SLIP_RECEIPT     = 0x8000 + PTR_S_RECEIPT_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;


/////////////////////////////////////////////////////////////////////
// "MapMode" 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;


/////////////////////////////////////////////////////////////////////
// "CutPaper" Method Constant
/////////////////////////////////////////////////////////////////////

const LONG PTR_CP_FULLCUT       = 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 8
const LONG PTR_BCS_EAN8         = 103;  // = JAN 8 (added in 1.2)
const LONG PTR_BCS_JAN13        = 104;  // = EAN 13
const LONG PTR_BCS_EAN13        = 104;  // = JAN 13 (added in 1.2)
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
//      (The following were added in Release 1.2)
const LONG PTR_BCS_UPCA_S       = 111;  // UPC-A with supplemental
                                        //   barcode
const LONG PTR_BCS_UPCE_S       = 112;  // UPC-E with supplemental
                                        //   barcode
const LONG PTR_BCS_UPCD1        = 113;  // UPC-D1
const LONG PTR_BCS_UPCD2        = 114;  // UPC-D2
const LONG PTR_BCS_UPCD3        = 115;  // UPC-D3
const LONG PTR_BCS_UPCD4        = 116;  // UPC-D4
const LONG PTR_BCS_UPCD5        = 117;  // UPC-D5
const LONG PTR_BCS_EAN8_S       = 118;  // EAN 8 with supplemental
                                        //   barcode
const LONG PTR_BCS_EAN13_S      = 119;  // EAN 13 with supplemental
                                        //   barcode
const LONG PTR_BCS_EAN128       = 120;  // EAN 128
const LONG PTR_BCS_OCRA         = 121;  // OCR "A"
const LONG PTR_BCS_OCRB         = 122;  // OCR "B"


//     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;


//////////////////////////////////////////////////////////////////////
// "RotatePrint" Method: "Rotation" Parameter Constants
// "RotateSpecial" Property Constants
//////////////////////////////////////////////////////////////////////

const LONG PTR_RP_NORMAL        = 0x0001;
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)
```

OposRod.h : Remote Order Display Header File

```
/////////////////////////////////////////////////////////////////////
//
// OposRod.h
//
//   Remote Order Display header file for OPOS Applications.
//
// Modification history
// ------------------------------------------------------------------
// 98-03-06 OPOS Release 1.3                                 BB
//
/////////////////////////////////////////////////////////////////////

#if !defined(OPOSROD_H)
#define      OPOSROD_H


#include "Opos.h"


/////////////////////////////////////////////////////////////////////
// "CurrentUnitID" and "UnitsOnline" Properties
//  and "Units" Parameter Constants
/////////////////////////////////////////////////////////////////////

#define ROD_UID(Unit) (1 << (Unit-1))

const LONG ROD_UID_1        = ROD_UID(1);
const LONG ROD_UID_2        = ROD_UID(2);
const LONG ROD_UID_3        = ROD_UID(3);
const LONG ROD_UID_4        = ROD_UID(4);
const LONG ROD_UID_5        = ROD_UID(5);
const LONG ROD_UID_6        = ROD_UID(6);
const LONG ROD_UID_7        = ROD_UID(7);
const LONG ROD_UID_8        = ROD_UID(8);
const LONG ROD_UID_9        = ROD_UID(9);
const LONG ROD_UID_10       = ROD_UID(10);
const LONG ROD_UID_11       = ROD_UID(11);
const LONG ROD_UID_12       = ROD_UID(12);
const LONG ROD_UID_13       = ROD_UID(13);
const LONG ROD_UID_14       = ROD_UID(14);
const LONG ROD_UID_15       = ROD_UID(15);
const LONG ROD_UID_16       = ROD_UID(16);
const LONG ROD_UID_17       = ROD_UID(17);
const LONG ROD_UID_18       = ROD_UID(18);
const LONG ROD_UID_19       = ROD_UID(19);
const LONG ROD_UID_20       = ROD_UID(20);
const LONG ROD_UID_21       = ROD_UID(21);
const LONG ROD_UID_22       = ROD_UID(22);
const LONG ROD_UID_23       = ROD_UID(23);
const LONG ROD_UID_24       = ROD_UID(24);
const LONG ROD_UID_25       = ROD_UID(25);
const LONG ROD_UID_26       = ROD_UID(26);
const LONG ROD_UID_27       = ROD_UID(27);
const LONG ROD_UID_28       = ROD_UID(28);
const LONG ROD_UID_29       = ROD_UID(29);
const LONG ROD_UID_30       = ROD_UID(30);
const LONG ROD_UID_31       = ROD_UID(31);
const LONG ROD_UID_32       = ROD_UID(32);


/////////////////////////////////////////////////////////////////////
// Broadcast Methods: "Attribute" Parameter Constants
/////////////////////////////////////////////////////////////////////

const LONG ROD_ATTR_BLINK       = 0x80;

const LONG ROD_ATTR_BG_BLACK    = 0x00;
const LONG ROD_ATTR_BG_BLUE     = 0x10;
const LONG ROD_ATTR_BG_GREEN    = 0x20;
const LONG ROD_ATTR_BG_CYAN     = 0x30;
const LONG ROD_ATTR_BG_RED      = 0x40;
const LONG ROD_ATTR_BG_MAGENTA  = 0x50;
const LONG ROD_ATTR_BG_BROWN    = 0x60;
const LONG ROD_ATTR_BG_GRAY     = 0x70;

const LONG ROD_ATTR_INTENSITY   = 0x08;

const LONG ROD_ATTR_FG_BLACK    = 0x00;
const LONG ROD_ATTR_FG_BLUE     = 0x01;
const LONG ROD_ATTR_FG_GREEN    = 0x02;
const LONG ROD_ATTR_FG_CYAN     = 0x03;
const LONG ROD_ATTR_FG_RED      = 0x04;
const LONG ROD_ATTR_FG_MAGENTA  = 0x05;
const LONG ROD_ATTR_FG_BROWN    = 0x06;
const LONG ROD_ATTR_FG_GRAY     = 0x07;


/////////////////////////////////////////////////////////////////////
// "DrawBox" Method: "BorderType" Parameter Constants
/////////////////////////////////////////////////////////////////////

const LONG ROD_BDR_SINGLE       = 1;
const LONG ROD_BDR_DOUBLE       = 2;
const LONG ROD_BDR_SOLID        = 3;


/////////////////////////////////////////////////////////////////////
// "ControlClock" Method: "Function" Parameter Constants
/////////////////////////////////////////////////////////////////////

const LONG ROD_CLK_START        = 1;
const LONG ROD_CLK_PAUSE        = 2;
const LONG ROD_CLK_RESUME       = 3;
const LONG ROD_CLK_MOVE         = 4;
const LONG ROD_CLK_STOP         = 5;


/////////////////////////////////////////////////////////////////////
// "ControlCursor" Method: "Function" Parameter Constants
/////////////////////////////////////////////////////////////////////
```

```
const LONG ROD_CRS_LINE         = 1;
const LONG ROD_CRS_LINE_BLINK   = 2;
const LONG ROD_CRS_BLOCK        = 3;
const LONG ROD_CRS_BLOCK_BLINK  = 4;
const LONG ROD_CRS_OFF          = 5;


//////////////////////////////////////////////////////////////////////
// "SelectChararacterSet" Method: "CharacterSet" Parameter Constants
//////////////////////////////////////////////////////////////////////

const LONG ROD_CS_ASCII         = 998;
const LONG ROD_CS_WINDOWS       = 999;


//////////////////////////////////////////////////////////////////////
// "TransactionDisplay" Method: "Function" Parameter Constants
//////////////////////////////////////////////////////////////////////

const LONG ROD_TD_TRANSACTION   = 11;
const LONG ROD_TD_NORMAL        = 12;


//////////////////////////////////////////////////////////////////////
// "UpdateVideoRegionAttribute" Method: "Function" Parameter Constants
//////////////////////////////////////////////////////////////////////

const LONG ROD_UA_SET           = 1;
const LONG ROD_UA_INTENSITY_ON  = 2;
const LONG ROD_UA_INTENSITY_OFF = 3;
const LONG ROD_UA_REVERSE_ON    = 4;
const LONG ROD_UA_REVERSE_OFF   = 5;
const LONG ROD_UA_BLINK_ON      = 6;
const LONG ROD_UA_BLINK_OFF     = 7;


//////////////////////////////////////////////////////////////////////
// "EventTypes" Property and "DataEvent" Event: "Status" Parameter Constants
//////////////////////////////////////////////////////////////////////

const LONG ROD_DE_TOUCH_UP      = 0x01;
const LONG ROD_DE_TOUCH_DOWN    = 0x02;
const LONG ROD_DE_TOUCH_MOVE    = 0x04;


//////////////////////////////////////////////////////////////////////
// "ResultCodeExtended" Property Constants for Remote Order Display
//////////////////////////////////////////////////////////////////////

const LONG OPOS_EROD_BADCLK     = 1 + OPOSERREXT; // ControlClock
const LONG OPOS_EROD_NOCLOCKS   = 2 + OPOSERREXT; // ControlClock
const LONG OPOS_EROD_NOREGION   = 3 + OPOSERREXT; // RestoreVideo
                                     //   Region
const LONG OPOS_EROD_NOBUFFERS  = 4 + OPOSERREXT; // SaveVideoRegion
const LONG OPOS_EROD_NOROOM     = 5 + OPOSERREXT; // SaveVideoRegion
```

```
#endif                  // !defined(OPOSROD_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
// 97-06-04 OPOS Release 1.2                              CRM
//   Add "ScanDataType" values.
//
/////////////////////////////////////////////////////////////////////

#if !defined(OPOSSCAN_H)
#define      OPOSSCAN_H


#include "Opos.h"


/////////////////////////////////////////////////////////////////////
// "ScanDataType" Property Constants
/////////////////////////////////////////////////////////////////////

// One dimensional symbologies
const LONG SCAN_SDT_UPCA        = 101;  // Digits
const LONG SCAN_SDT_UPCE        = 102;  // Digits
const LONG SCAN_SDT_JAN8        = 103;  // = EAN 8
const LONG SCAN_SDT_EAN8        = 103;  // = JAN 8 (added in 1.2)
const LONG SCAN_SDT_JAN13       = 104;  // = EAN 13
const LONG SCAN_SDT_EAN13       = 104;  // = JAN 13 (added in 1.2)
const LONG SCAN_SDT_TF          = 105;  // (Discrete 2 of 5) Digits
const LONG SCAN_SDT_ITF         = 106;  // (Interleaved 2 of 5) Digits
const LONG SCAN_SDT_Codabar     = 107;  // Digits, -, $, :, /, ., +;
                                        //   4 start/stop characters
                                        //   (a, b, c, d)
const LONG SCAN_SDT_Code39      = 108;  // Alpha, Digits, Space, -, .,
                                        //   $, /, +, %; start/stop (*)
                                        // Also has Full ASCII feature
const LONG SCAN_SDT_Code93      = 109;  // Same characters as Code 39
const LONG SCAN_SDT_Code128     = 110;  // 128 data characters

const LONG SCAN_SDT_UPCA_S      = 111;  // UPC-A with supplemental
                                        //   barcode
const LONG SCAN_SDT_UPCE_S      = 112;  // UPC-E with supplemental
                                        //   barcode
const LONG SCAN_SDT_UPCD1       = 113;  // UPC-D1
const LONG SCAN_SDT_UPCD2       = 114;  // UPC-D2
const LONG SCAN_SDT_UPCD3       = 115;  // UPC-D3
const LONG SCAN_SDT_UPCD4       = 116;  // UPC-D4
const LONG SCAN_SDT_UPCD5       = 117;  // UPC-D5
const LONG SCAN_SDT_EAN8_S      = 118;  // EAN 8 with supplemental
```

```
                                //   barcode
const LONG SCAN_SDT_EAN13_S     = 119;  // EAN 13 with supplemental
                                //   barcode
const LONG SCAN_SDT_EAN128      = 120;  // EAN 128
const LONG SCAN_SDT_OCRA        = 121;  // OCR "A"
const LONG SCAN_SDT_OCRB        = 122;  // OCR "B"

// Two dimensional symbologies
const LONG SCAN_SDT_PDF417      = 201;
const LONG SCAN_SDT_MAXICODE    = 202;

// Special cases
const LONG SCAN_SDT_OTHER       = 501;  // Start of Scanner-Specific bar
                                //   code symbologies
const LONG SCAN_SDT_UNKNOWN     =   0;  // Cannot determine the barcode
                                //   symbology.


#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)
```

OposTone.h : Tone Indicator Header File

```
/////////////////////////////////////////////////////////////////////
//
// OposTone.h
//
//   Tone Indicator header file for OPOS Applications.
//
// Modification history
// ------------------------------------------------------------------
// 97-06-04 OPOS Release 1.2                              CRM
//
/////////////////////////////////////////////////////////////////////

#if !defined(OPOSTONE_H)
#define      OPOSTONE_H


#include "Opos.h"


// No definitions required for this version.


#endif                  // !defined(OPOSTONE_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)
```

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 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++, however, requires more consideration.

BSTR is usually quite straightforward to use:

- <u>BSTR Method Parameters</u>
    - ♦ **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.
- <u>BSTR Return Type</u> (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 no 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.

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 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++, again, requires more consideration.

Looking at the cases as with non-binary data, BSTR handling is as follows:

- <u>BSTR Method Parameters</u>
    - ♦ **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.

- <u>BSTR Return Type</u> (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);
```

End of Application Programmer's Guide