

OPOS ADK

Application Development Guide

General Functions

Version 3.00 Feb. 2019

Notes

- (1) Reproduction of this documentation by any means in part or in whole is prohibited.
- (2) Contents of this documentation are subject to change without notice.
- (3) Epson will not be responsible for any consequences resulting from the use of any information in this documentation.
- (4) Comments and notification of any mistakes in this documentation are gratefully accepted.

Trademarks

Microsoft®, Windows®, Windows Server®, Visual Basic® and Visual C++® are trademarks or registered trademarks of Microsoft Corporation in the United States and/or other countries.
EPSON® and ESC/POS® are registered trademarks of Seiko Epson Corporation.
Other product and company names used herein are for identification purposes only and may be trademarks or registered trademarks of their respective companies.

Contents

SECTION 1. PREFACE	1
SECTION 2. GENERAL INFORMATION	2
2.1 Object Names	2
2.2 Device Information Reference	2
2.3 Opening and Closing Devices.....	2
2.4 Device Claim/Release.....	4
2.5 Device Enable/Disable	5
2.6 Device Self Diagnostics	5
2.7 Character Sets	6
2.8 Event Management.....	7
2.8.1 <i>DataEvent</i>	7
2.8.2 <i>DirectIOEvent</i>	7
2.8.3 <i>ErrorEvent</i>	7
2.8.4 <i>OutputCompleteEvent</i>	8
2.8.5 <i>StatusUpdateEvent</i>	8
2.9 Results of Changing Properties or Running Methods	8
2.10 Extended Errors	8
2.11 Clearing the Input and Output Buffers	9
2.12 Capability Property.....	9
2.13 Notifying Power Status.....	9
2.14 Device Statistics.....	10
SECTION 3. POS PRINTER	11
3.1 Printer Stations	11
3.2 Escape Sequences	12
3.3 MapMode Settings	13
3.4 Line Information	13
3.5 Sending Data to the Printer.....	14
3.5.1 <i>Synchronous Printing on One Station</i>	14
3.5.2 <i>Asynchronous Printing on One Station</i>	15
3.5.3 <i>Printing on Two Stations at The Same Time</i>	17
3.5.4 <i>Setting the Logo</i>	18
3.5.5 <i>Printing Bar Codes</i>	18
3.5.6 <i>Bitmap Printing</i>	19
3.5.7 <i>Rotated Printing</i>	22
3.5.8 <i>Immediate Printing</i>	23
3.5.9 <i>Collective Printing</i>	23
3.6 Form Insertion/Removal and Slip Printing	24
3.7 Paper Cutting	25
3.8 Checking the Printer State	27
3.9 Printer Errors and Status	28
3.10 Clearing the Output Buffer	29
3.11 Testing with the CheckHealth Method	29
3.12 Cartridge State	29
3.13 Color Printing	30
3.14 Mark Sensed Paper Support.....	31
3.15 Printing on Both Sides	31
3.16 Things to Consider when Using Properties	31
SECTION 4. LINE DISPLAY	32
4.1 Window Creation/Destruction	32
4.2 Window Rows/Columns	34
4.3 Showing Data on the Display.....	34
4.4 Window Clear/Refresh	35

4.5 Descriptors	36
4.6 Scrolling the Display	37
4.7 Marquee Settings	38
4.8 Testing with the CheckHealth Method	40
4.9 Setting the Glyph Character Definition	40
SECTION 5. MICR.....	42
5.1 Form Insertion/Removal.....	42
5.2 Reading Data from the MICR.....	43
5.3 Error Management	44
5.4 Testing with the CheckHealth Method	45
SECTION 6. CASH DRAWERS.....	46
6.1 Drawer Open/Close	46
6.2 Checking Drawer Status	46
6.3 Testing with the CheckHealth Method	47
6.4 Multi-drawer Configuration Support	47
6.4.1 Multi-drawers with One Status	47
6.4.2 Multi-drawers with Two Status	47
SECTION 7. CHECKSCANNER.....	48
7.1 Form Insertion/Removal.....	48
7.2 Reading Data from the CheckScanner	49
7.3 Saving/Reading/Deleting Scanned Data	50
7.4 Error Management	51
7.5 Testing with the CheckHealth Method	52
SECTION 8. ELECTRONIC JOURNAL.....	53
8.1 Writing Electronic Journal data	53
8.2 Marker setting	53
8.3 Specifications for specified range	53
8.4 Non-simultaneously printing of ElectronicJournal data.....	53

Section 1. Preface

This manual is an application development guide containing information that will assist in building POS applications using the API functions supported by OPOS. Included in this guide are numerous details on how to utilize Visual Basic to take advantage of the many functions of OPOS that run the various devices used in a POS system. There are also many device-specific programming examples included that run the various capabilities of devices supported by OPOS. Please use this manual for your benefit.

Throughout the manual, the "OPOS Application Development Kit" will be shown as "OPOS ADK".

For information on installing and setting up the OPOS ADK, please refer to the "EPSON OPOS ADK User's Manual (Installer/ SetupPOS/ TMUSB)". For detailed explanations of API functions, please refer to the "UPOS 1.14.1" created by OPOS Committee.

Section 2. General Information

This section contains an explanation of programming functions that are common to all devices.

2.1 Object Names

When explaining how to use OPOS API in Visual Basic, API's will have the designator "[Object]." before the API name. Object names are created automatically when an ActiveX control is placed on a form, but can be changed by the user. Check the object's names in the form's properties box.

Default name:

Line Display OPOSLineDisplay*n*
POSPrinter..... OPOSPOSPrinter*n*
MICR..... OPOSMicr*n*
Cash Drawer OPOSCashDrawer*n*
CheckScanner OPOSCheckScanner*n*
ElectronicJournal OPOSElectronicJournal*n*

n is a number determined when the control is placed on the form.

2.2 Device Information Reference

All devices include the following information. Use this information as the need arises.

ControlObjectDescription Returns a string distinguishing the custom control.
ControlObjectVersion Returns the custom control's version number.
ServiceObjectDescription Returns a string distinguishing the service object.*
ServiceObjectVersion Returns the service object's version number.*
DeviceDescription Returns a string distinguishing the device.*
DeviceName Returns the name of the device.*

*Values cannot be viewed unless the device is opened successfully.

2.3 Opening and Closing Devices

In order to use the devices that are connected to the POS system, it is necessary to open each device. By opening a device, it is connected to the application and is made available for use. Each device must be opened individually.

The Open method is used to open devices. The DeviceName (a variable used to distinguish the device) parameter that is passed to the Open method uses the DeviceNameKey* or LogicalDeviceName* for each specific device. If you are not sure of a device's name, it can be found by running the SetupPOS utility. For more details, refer to UPOS.

DeviceNameKey and LogicalDeviceName values can be added or changed from the SetupPOS utility. LogicalDeviceName is an alias name used to distinguish a device from others. By using a LogicalDeviceName, a program can be made more portable and easier to update.

After using the Open method, a result is returned to the program. If the Open failed, the ResultCode property is handed an OPOS_E_CLOSED value. From the UPOS Release 1.5, an

error detail will be further explained in the OpenResult property. It can be referred to when necessary. A return value of the Open method can also be referred. If the Open method is successful, the ResultCode property is set to zero and any other codes returned by methods used after that can be referred to when necessary. To learn more about the OpenResult or the ResultCode property, consult UPOS.

The following is an example of opening a printer device.

```
Dim RC As Long
RC = OPOSPOSPrinter1.Open ("Unit1")
If RC<>OPOS_SUCCESS Then
    If OPOSPOSPrinter1.OpenResult = OPOS_OR_ALREADYOPEN Then
        MsgBox"Error On Opening POS Printer – Already Open"
    ElseIf OPOSPOSPrinter1.OpenResult = OPOS_OR_REGBADNAME Then
        MsgBox"Error On Opening POS Printer – DeviceName Invalid"
    ElseIf .....
    .....
    .....
    Else
        Open = 1
    End If
```

For more details on errors that may occur when opening devices, please consult the “Device Class Programming” sections of this manual.

After finishing use of a device in a program, please be sure to use the Close method to close the device. This will insure that the device will not cause any unexpected problems and become unable to be used.

After using the Close method, a result is returned to the program. When a device is closed, an OPOS_E_CLOSED value is placed in the ResultCode property so the program can check to insure that the close method was successful.

The following is an example of closing a printer device.

```
Dim RC As Long
RC = OPOSPOSPrinter1.Close
If RC <> OPOS_SUCCESS Then
    MsgBox "Error on closing POSPrinter."
Else
    Openflag = 0
End If
```

2.4 Device Claim/Release

An application can gain exclusive use of a device through the ClaimDevice method. In the API, there are devices that can and cannot be used if they are not claimed. To confirm whether a device needs to be claimed or not, please refer to the “Device Summary” section of UPOS.

By using the ClaimDevice and ReleaseDevice methods, other applications and multi-processes can use the device. For example, a printer is claimed by process A. While the printer is claimed, process B cannot use the printer. The ClaimDevice method will return an error. If process A uses the ReleaseDevice method to release the printer, then process B can claim the printer and use it. This example assumes that the printer is the same physical machine for both processes, but different devices using the same port through hydra settings will act in the same manner. When a printer and display are connected through hydra settings, if either one is claimed by a process, neither can be used by another process.

When using the ClaimDevice method, the maximum number of milliseconds to wait for exclusive access to be satisfied is passed as a variable.

If the ResultCode property has the value OPOS_E_TIMEOUT after the ClaimDevice method has been called, another process has accessed the device. If OPOS_SUCCESS is returned, the device is available for use by the process.

For more information on the errors that may occur when claiming devices, please consult manuals of each device.

The following is an example of Claiming and Releasing a printer device.

```
OPOSPOSPrinter1.ClaimDevice 2000
If OPOSPOSPrinter1.ResultCode = OPOS_SUCCESS Then
    OPOSPOSPrinter1.DeviceEnabled = True
    OPOSPOSPrinter1.AsyncMode = False
    OPOSPOSPrinter1.PrintNormal PTR_S_RECEIPT,"Chocolate $1.00"
                                + Chr(13) + Chr(10)
    OPOSPOSPrinter1.ReleaseDevice
Elseif OPOSPOSPrinter1.ResultCode = OPOS_E_TIMEOUT Then
    MsgBox"Error On Claim – Printer Device is used by Another process."
Elseif .....
End If
```

It is also possible to tell if a device is claimed or not by checking the Claimed property.

```
If Not OPOSPOSPrinter1.Claimed Then
    OPOSPOSPrinter1.ClaimDevice 2000
End If
```


2.5 Device Enable/Disable

After opening a device, the device is placed in a disabled state. In the API, there are devices that need to be enabled before use and some that do not. To confirm whether a device needs to be enabled or not, please refer to the "Device Summary" section of UPOS.

In the case where a device requires claiming, the device needs to execute ClaimDevice method before bring it to an enable state. If the device is brought to an enable state without execution of the method, OPOS_E_NOTCLAIMED is returned

If a device is in a disabled state, the device cannot transmit and receive data.

The following is an example of switching a device (printer) between enabled and disabled states.

```
If Not OPOSPOSPrinter1.DeviceEnabled Then
    OPOSPOSPrinter1.DeviceEnabled = True      'Enable
Else
    OPOSPOSPrinter1.DeviceEnabled = False     'Disable
End If
```

2.6 Device Self Diagnostics

Device diagnostics can be performed using the CheckHealth method. There are three levels of diagnostics available that can be set with a corresponding number. For a list of diagnostic levels and explanations, please refer to the "CheckHealth Method" section of UPOS. Different devices allow different levels of diagnostic support. Please refer to each device's [Testing with the CheckHealth Method's INTERACTIVE Check] section for further information.

The results of using the CheckHealth method are placed in the CheckHealthText property. The string returned to the CheckHealthText property is decided by the SO automatically, so there are not any common values for all devices.

If a device is in the middle of sending data, an OPOS_E_BUSY error is returned. Please run the CheckHealth method after all data has finished transmitting. If a CheckHealth level that is not supported by the device is used, an OPOS_E_ILLEGAL error will be returned. When any value other than OPOS_SUCCESS (e.g. OPOS_E_BUSY, OPOS_E_ILLEGAL, etc.) is returned, the CheckHealthText property cannot be updated. The value in this property will remain the result of the prior condition.

The following is an example of using the CheckHealth method to perform a diagnostic check.

```
Dim RC As Long
RC = OPOSPOSPrinter1.CheckHealth (OPOS_CH_INTERACTIVE)
If RC = OPOS_SUCCESS Then
    MsgBox "CheckHealth Successful = " + OPOSPOSPrinter1.CheckHealthText
Else
    MsgBox "CheckHealth Failed"
End If
```

2.7 Character Sets

The types of character objects that make up a character set are listed below.

- Unique device character set
- Code page
- ASCII character set
- Windows ANSI character set

For information on what code pages are valid, please refer to your device's product manual.

Supported character set numbers can be referenced in the CharacterSetList property.

```
Dim CSList As String
CSList = OPOSPOSPrinter1.CharacterSetList
```

By running the above code, the supported character set numbers are placed in the CSList variable as a text string. (e.g.. "437,850,860,863,865,998") If it becomes necessary to determine if a character set is valid, the above property can be used. In order to actually specify a character set, use the CharacterSet property.

```
OPOSPOSPrinter1.CharacterSet = 437 'Code page: American English.
OPOSPOSPrinter1.CharacterSet = DISP_CS_ASCII 'ASCII character set.
```

The Open method passes a default character set value of 437 (American English) when initialized.

For information on the default values, please refer to the each device manual. The CharacterSetList property can be viewed immediately after the Open method has been called, but the CharacterSet property can only be viewed or set after the Open, Claim, and Enabled methods have been called.

The following are examples of using character settings in a program.

```
OPOSPOSPrinter1.CharacterSet = 858           *Code page FPC858
*Printing EURO characters
OPOSPOSPrinter1.BinaryConversion = OPOS_BC_NIBBLE
OPOSPOSPrinter1.PrintNormal PTR_S_RECEIPT,"=50=0:"

OPOSPOSPrinter1.BinaryConversion = OPOS_BC_DECIMAL
OPOSPOSPrinter1.PrintNormal PTR_S_RECEIPT,"213013010"
```

When outputting more than 80H codes, needed to modify the method for specifying the characters using the BinaryConversion property. For more information on the method for specifying the characters, please refer to the "BinaryConversion Property" in UPOS

2.8 Event Management

In programs using the OPOS API, much information is given to the application through Events. By placing a device's object on a form, events that can be used by the object are pre-set in the application's code. Event management can be performed by using events to determine what to do when an Event fires. As described below, there are five types of events available.

To temporarily pause an event, use the FreezeEvents property. If the FreezeEvents property is set to TRUE, events cannot be passed to the application. To allow the processing of events, the FreezeEvents property should be set to FALSE, which will allow all events that occurred during the frozen time to fire in order.

The five following sections contain explanations of the different types of events that may occur.

2.8.1 DataEvent

DataEvent lets the application know when data has been received from a device. Events can be enabled/disabled by changing the TRUE/FALSE value of the DataEventEnabled property. All data gathered while disabled will fire when control is returned by setting the DataEventEnabled property back to TRUE.

The Auto Disable property that has been added to the UPOS Release 1.3 and later, influences on the DataEvent. If set the AutoDisable property to TRUE, the DeviceEnable property is set to FALSE at every time when the DataEvent is fired.

The way to utilize this event depends upon the type of device that is being used. Please consult the device's programming section.

2.8.2 DirectIOEvent

DirectIOEvent are unique events that correspond to devices supported by us. For detailed explanations, please refer to the "DirectIOEvent" section of respective device's "EPSON OPOS ADK MANUAL APPLICATION DEVELOPMENT GUIDE".

2.8.3 ErrorEvent

ErrorEvents are used to inform the program if an error has occurred while executing (e.g. While the printer is printing data). This event will not fire if the program is not executing. Error management can be performed by using the parameters passed to the application by the event.

ResultCode --

Error reasons are included in the code.

ResultCodeExtended --

If the result code returned is OPOS_E_EXTENDED, the ResultCodeExtended will contain a value that will further detail the meaning of the ResultCode property. These codes are device-dependent. Please refer to the "OPOS Application Header Files" appendix of UPOS or the OPOS .BAS file contents.

Even if the ResultCode property is given a value other than OPOS_E_EXTENDED, it is possible that the ResultCodeExtended property will also be given a value. To allow a program to obtain detailed information on possible errors, the ResultCodeExtended property can be checked.

When no extended information is available, 0 will be the value.

For information on these error values, please refer to respective device's "EPSON OPOS ADK MANUAL APPLICATION DEVELOPMENT GUIDE".

Error Locus --

This value shows the place where the error occurred. The value is device-dependent. Please refer to the device's explanation in UPOS.

pErrorResponse --

This value designates the response to the ErrorEvent. The value is device-dependent. Please refer to the device's explanation in UPOS.

2.8.4 OutputCompleteEvent

An OutputCompleteEvent is fired to the application when synchronous data has finished sending. An OutputID is returned with this event. The OutputID can be compared to the data sent ID to decide whether synchronous data has been completely sent or not. The OutputID value can be checked by using the OutputID property after sending data. The OutputID values are changeable, so variables should be used to store the values when necessary.

Use of this event is device-dependent. Please consult the device's programming section.

2.8.5 StatusUpdateEvent

StatusUpdateEvent is fired to show that there has been a change in device status.

The new functions that notifying the power status has been added to the UPOS Release 1.3 and later. If the CapPowerReporting supports the level (NONE, STANDARD, or ADVANCED), the status can be checked by using the event.

Use of this event is device-dependent. Please consult the device's programming section.

2.9 Results of Changing Properties or Running Methods

The ResultCode property will be given a value after a property is changed or a method is run. When the property change or method is successful, the ResultCode property is given the value OPOS_SUCCESS. If an error occurs, the error is sent to the ResultCode and/or ResultCodeExtended properties. For information on these error values, please refer to the "ResultCode property" section in UPOS. For device specific error explanations, please refer to the device's error explanations in the "Device Class Programming" section.

Use the properties and method return values to perform error management. If an error is not properly handled, it may cause problems later on in the program.

The codes used for ResultCodeExtended are determined by the device that is being used.

However, if the device is hydra connected, there are times when codes for the first device in the connection will be returned. For example, when a line display is hydra connected to a printer and an error occurs on the display because the printer's cover is open, the display will send it's own error, but the ResultCodeExtended will be OPOS_EPTR_COVER_OPEN from the printer.

2.10 Extended Errors

As for our own extended errors, please refer to the device's error explanations in the "Device Class Programming" section. Only Commons are mentioned as follows.

[Common]

OPOS_EX_BADCO	: Invalid CO Interface
OPOS_EX_BADPORT	: Invalid Port
OPOS_EX_BADDEVICE	: Invalid DeviceName
OPOS_EX_BADPROPIDX	: Invalid index inside property
OPOS_EX_BADPROPVAL	: Invalid property value
OPOS_EX_NOTSUPPORTED	: Function not supported
OPOS_EX_NOASB	: No ASB data returned
OPOS_EX_INPUT	: No data returned

OPOS_EX_BUSY	: Device busy by (Async output)
OPOS_EX_INCAPABLE	: Incapable of the function (corresponding capability property is false)
OPOS_EX_INVALIDMODE	: Invalid device mode
OPOS_EX_REOPEN	: Device reopened
OPOS_EX_BADPEEKRange	: PeekRange invalid
OPOS_EX_BADDISPRANGE	: DispatchRange invalid
OPOS_EX_NOTCLAIMED	: Not claimed (release method used)
OPOS_EX_TIMEOUT	: Sync output timeout
OPOS_EX_PORTUSED	: Port used by another
OPOS_EX_MICRMODE	: MICR mode
OPOS_EX_PORTBUSY	: HOST Port busy
OPOS_EX_MICRMODE	: MICR in mode
OPOS_EX_BADINF	: Invalid INF file
OPOS_EX_DEVBUSY	: Device busy
OPOS_EX_SOVERSION	: Invalid SO version
OPOS_EX_BADPARAM	: Invalid parameter (general)
(OPOS_EX_BADPARAM+1)	: Invalid first parameter (general)
(OPOS_EX_BADPARAM+2)	: Invalid second parameter (general)
(OPOS_EX_BADPARAM+3)	: Invalid third parameter (general)
: : Invalid xxx parameter	
: :	

2.11 Clearing the Input and Output Buffers

By using the ClearInput and ClearOutput methods, all buffered data can be cleared from the buffer. All relevant events will also be cleared at this time.

2.12 Capability Property

Devices' capabilities vary depending upon the device class that is being used. Due to this, it is impossible for this manual to provide complete class specific programming explanations for every device's capability. Properties that start with "Cap" hold information regarding available functions of the device. It is recommended that these properties be used to manage cases where different devices supporting different functions may be attached.

2.13 Notifying Power Status

The new functions that notifying power status has been added to the UPOS Release 1.3 and later. These are CapPowerReporting, PowerState, and the PowerNotify properties. For more details, please refer to the information on the "Device Power Reporting Model" section in UPOS. When the CapPowerReporting property is OFF_OFFLINE, the device power state cannot be distinguished whether it is powered on, or off.

Under consideration of those features, please execute the recovery of the application. Each device's supports for the PowerReporting property model are written in the each manual. When the device is connected using the pass-through connection (for instance, printer and display, etc.), the state of the signal line depends on the state of the currently selected device. For instance, the state of the signal line does not change even if the printer is turned off when the display is selected, and the power supply notification is not sent. The device that basically transmitted the last command is selected.

2.14 Device Statistics

The DeviceStatistics function is added in response to the compliance of the "UPOS 1.8". Please refer to the "EPSON OPOS ADK MANUAL APPLICATION GUIDE Device Statistics" for the details of the Device Statistics.

Section 3. POS Printer

Programming examples of how to use API functions relating to a POS Printer are shown below.

3.1 Printer Stations

The printer control recognizes the following three types of stations.

JournalPTR_S_JOURNAL
ReceiptPTR_S_RECEIPT
SlipPTR_S_SLIP

The control's properties that are supported by each individual station may vary, so be sure to use the properties that correspond to the station you are using. Printer methods are passed the station names as their first parameter in order to connect to the proper station. Different printers may not have all three stations available, so please check the printer's properties carefully.

```
Journal)
If OPOSPOSPrinter1.CapJrnPresent = True Then
    'Journal functions are available
Else
    'Journal functions are not available
End If
```

```
Receipt)
If OPOSPOSPrinter1.CapRecPresent = True Then
    'Receipt functions are available
Else
    'Receipt functions are not available
End If
```

```
Slip)
If OPOSPOSPrinter1.CapSlpPresent = True Then
    'Slip functions are available
Else
    'Slip functions are not available
End If
```

3.2 Escape Sequences

POS Printers support escape sequences that can be treated as printing data. For a listing of escape sequence types, please refer to the “Data Characters and Escape Sequences” section of UPOS.

The following is an example of how to use escape sequences.

When data that is being printed is to be underlined, the escape sequence “ESC|uC” is placed before the character string to be printed.

```
Dim Pdata As String
Pdata = Chr(&H1B) + “|uC” + “Print Data” + Chr(&H0D) + Chr(&H0A)
```

If the character string in the variable Pdata is printed, the underlined string Print Data is sent to the printer. If the Chr(&H1B)+”|uC” is not put in front of the data, the string will not be underlined when printed. Also, a printer must support the escape sequence for the sequence to be performed the way the program specifies. It is possible for a program to check if a printer has a certain capability by looking at the printer’s properties.

```
If OPOSPOSPrinter1.CapRecUnderline = True Then
    ‘Underline printing is supported.
Else
    ‘Underline printing is not supported.
End If
```

Escape sequences are only valid once per method, so escape sequences cannot be used in the following manner:

```
OPOSPOSPrinter1.PrintNormal PTR_S_JOURNAL,Chr(&H1B)+”|uC”+”123”
OPOSPOSPrinter1.PrintNormal PTR_S_JOURNAL,”456” + Chr(13) + Chr(10)
```

An underline will appear below the “123” characters, but not under “456”. When using a new method to print characters, it is necessary to add the desired escape sequence again.

During rotate printing mode (Right90, Left90), the following escape sequences will not work.
”ESC|cA”, ”ESC|rA”

The following escape sequences will not work if they are not the first character of a line.

Thermal station	”ESC cA”, ”ESC rA”, ”ESC #P”, ”ESC rC”
Except Thermal station	”ESC #P”, ”ESC rC”

The number position shown with # should be replaced by a number from 1 to 4. Any number greater than 4 or less than 1 will not be recognized as an escape sequence.

On printers with escape sequence capabilities it is possible to get information about the capabilities from capability properties, and for information about setting parameter numbers on machines without capabilities, the ValidateData method can be used. It is not possible to perform these checks after printing has occurred. When it is necessary to perform an escape sequence check before printing, use the ValidateData method.

When setting all printing data using the ValidateData method, it becomes possible to tell if data is valid, if escape sequences are correct, and if escape sequence functions are supported.


```

Dim Pdata As String
Dim RC As Long
Pdata = Chr(&H1B) + "uC" + "Print Data"+Chr(13)+Chr(10)
RC=OPOSPOSPrinter1.ValidateData (PTR_S_RECEIPT, Pdata)
If RC = OPOS_SUCCESS Then
`Pdata is valid
Elseif RC = OPOS_E_ILLEGAL Then
`parameter is illegal
Elseif RC = OPOS_E_FAILURE Then
`the capability is not supported
End If

```

3.3 MapMode Settings

By changing a printer's MapMode setting, the height, width, spacing, and other attributes of printed data can be changed. The default setting is dot pitch that depends the printer being used.

Settings can be changed in the following units by using the MapMode property to set new values.

Unit	Setting Value
Print dot width	PTR_MM_DOT
1/1440 of an inch	PTR_MM_TWIPS
0.001 inch	PTR_MM_ENGLISH
0.01 millimeter	PTR_MM_METRIC

For example, in a program like is shown below, the dot pitch default is being used, so the setting for D1 is 512. Next, after using MapMode to change the units to inches and rechecking RecLineWidth, D2 becomes 512/Printer Resolution (dpi x 1000). The actual line width is not changed.

```

Dim D1 As Long
Dim D2 As Long
D1=OPOSPOSPrinter1.RecLineWidth
OPOSPOSPrinter1.MapMode = PTR_MM_ENGLISH
D2 = OPOSPOSPrinter1.RecLineWidth

```

3.4 Line Information

Using the properties below, information about printing lines can be obtained and set.

XXXLineCharsThe number of characters that can be printed on a single line can be browsed or set.

XXXLineCharsList...The width of supported characters can be browsed or set.

XXXLineHeightThe height of a single line can be obtained.
XXXLineSpacing.....The space between lines can be browsed or set.
XXXLineWidthThe width of a single line can be obtained.

XXX should be replaced by the desired station name.

The values of the above parameters XXXLineHeight, XXXLineSpacing, and XXXLineWidth are changed by the MapMode property.

The XXXLineSpacing property can be set to values from xxxLineHeigh to 127 on thermal station.

The XXXLineSpacing property can be set to values from 0 to 127 on dot station.

The number of characters that can be printed on one line can be set for each station using the XXXLineChars property, but stations cannot be set individually. When one station is set, all stations are set to corresponding values. When a value from the JrnLineCharsList property is specified as the value of the JrnLineChars property, the corresponding value in the SlpLineCharsList property will be automatically set in the SlpLineChars property, and vice versa. For example, if the JrnLineCharsList is 48, 54,... and the SlpLineCharsList property is 62,88,..., the first value of the JrnLineChars Property is set (48) JrnLineChars, then the first value of the SlpLineChars property is set to (62) automatically.

It is possible to change the speed and quality of a printing by using the XXXLetterQuality property.

RecLetterQuality = TRUE 'Print in quality mode
RecLetterQuality = FALSE 'Print in speed mode
SlpLetterQuality = TRUE 'Print in quality mode
SlpLetterQuality = FALSE 'Print in speed mode

Default is speed mode. If printing mode is changed to quality mode, printing will be slower, but a higher quality. If printing mode is changed back to speed mode, printing becomes faster, but lower quality if the station being used is a thermal printer, printing is smoothed automatically.

3.5 Sending Data to the Printer

The following are explanations of data printing functions.

3.5.1 Synchronous Printing on One Station

The PrintNormal method is used to send data to the printer. By setting the AsyncMode property to FALSE and running the PrintNormal method, data is printed as it is sent and will not release control of the printer until the data is completely sent.

The following is an example of a synchronous printing procedure.

```
OPOSPOSPrinter1.AsyncMode = False
OPOSPOSPrinter1.PrintNormal PTR_S_JOURNAL,
    "Print Data" + Chr(13) + Chr(10)
If OPOSPOSPrinter1.ResultCode = OPOS_SUCCESS Then
    'Printing was successful.
ElseIf OPOSPOSPrinter1.ResultCode = OPOS_E_BUSY Then
    'The printer is printing in asynchronous mode.
```

```

    ElseIf OPOSPOSPrinter1.ResultCode = OPOS_E_ILLEGAL Then
    'This printing function is not available, etc.

    ElseIf OPOSPOSPrinter1.ResultCode = OPOS_E_EXTENDED Then
        If OPOSPOSPrinter1.ResultCodeExtended =
            OPOS_EPTR_COVER_OPEN Then
            'The cover is open.

            ElseIf OPOSPOSPrinter1.ResultCodeExtended =
                OPOS_EPTR_JRN_EMPTY Then
                'No paper.
            End If
        Else
        'Other error
        End If
    
```

Errors can be determined from the value returned by a method, or can be read from the ResultCode property. Using values returned by a method is shown in the example below.

```

    Dim RC As Long
    OPOSPOSPrinter1.AsyncMode = False
    RC=OPOSPOSPrinter1.PrintNormal (PTR_S_JOURNAL,
                                    "Print Data" + Chr(13) + Chr(10))

    If RC = OPOS_SUCCESS Then
    'Printing was successful

        ElseIf RC = OPOS_E_BUSY Then
        'The printer is printing in asynchronous mode.

        ElseIf RC = OPOS_E_ILLEGAL Then
        'The function does not exist, etc.

        ElseIf RC = OPOS_E_EXTENDED Then
        'Extended error.
    Else
    'Other error.
    End If
    
```

3.5.2 Asynchronous Printing on One Station

Asynchronous output, especially when sending large amounts of data, is more efficient. Asynchronous output allows printing to happen in the background and thus has less effect on keyboard and mouse operations in a POS application. By using asynchronous printing, more user-friendly POS applications can be built.

The PrintNormal method is used to send data to the printer. By setting the AsyncMode property to TRUE and running the PrintNormal method, data is printed asynchronously. Printing data is stored in a buffer and control is returned to the application immediately. After the data has been sent to the buffer, an ID for the data is set to the OutputID property. When the data has finished printing, an OutputCompleteEvent is fired to the application. This event's identifier is the OutputID. The value of the OutputID property changes with different results, so it can be saved and used as needed in an application.

When asynchronous printing is started, the State property becomes OPOS_S_BUSY. When data has been sent, the State property returns to OPOS_S_IDLE.

However, after asynchronous printing, the following State monitor loop must not be done in order to know whether the output has been completed.

```
While OPOSPOSPrinter1.State = OPOS_S_BUSY
    DoEvents
Wend
```

When an error occurs and ErrorEvent cannot be applied, this kind of loop can become infinite loop. When an error occurs, ErrorEvent can be applied, and State value doesn't change unless retry (default) or clear can be done.

Errors in asynchronous printing are reported to the application by an ErrorEvent. The ErrorEvent value is placed in the ResultCode parameter and can be used from there. Extended errors such as cover open, out of paper, etc., can be read from this value. After an error occurs, the choice of whether to try again or clear the error can be made by setting the parameter of the pErrorResponse to the corresponding value.

The following are examples of using asynchronous printing in a program.

[Main Program]

```
Global PrintID1 As Long
Global PrintID2 As Long

OPOSPOSPrinter1.AsyncMode = True
OPOSPOSPrinter1.PrintNormal PTR_S_JOURNAL,
    "Print Data" + Chr(13) + Chr(10)
If OPOSPOSPrinter1.ResultCode = OPOS_SUCCESS Then
    'Data was sent successfully.
    PrintID1 = OPOSPOSPrinter1.OutputID
Elseif OPOSPOSPrinter1.ResultCode = OPOS_E_ILLEGAL Then
    'The station does not exist.
Else
    'Other error.
End If

If PrintID2 = PrintID1 Then
    'Data was printed successfully.
End If
```

[Event Program]

```
Private Sub OPOSPOSPrinter1_OutputCompleteEvent(ByVal OutputID As Long)
PrintID2 = OutputID
End Sub
```

```
Private Sub OPOSPOSPrinter1_ErrorEvent(ByVal ResultCode As Long,
    ByVal ResultCodeExtended As Long,
    ByVal ErrorLocus As Long,
```

```

pErrorResponse As Long )
    'ResultCode is obtained and the error can be dealt with.
    'Errors such as OPOS_E_TIMEOUT, OPOS_E_BUSY,
    'and OPOS_E_OFFLINE are possible results.
    If ResultCode = OPOS_E_EXTENDED Then
        If ResultCodeExtended = OPOS_EPTR_COVER_OPEN Then
            'Cover Open
        ElseIf ResultCodeExtended = OPOS_EPTR_JRN_EMPTY
            Then
            'Paper End
        End If
    End If
    pErrorResponse = OPOS_ER_RETRY 'When the program should try again
    pErrorResponse = OPOS_ER_CLEAR 'When the program should clear the data
End Sub

```

3.5.3 Printing on Two Stations at The Same Time

It is possible to print data on two stations at the same time. Using the PrintTwoNormal method allows concurrent printing. On a printer with more than one station, the stations can be specified with identifiers, and similar or different data can be printed at the same time. When the information is to be similar on both stations, the second parameter of the PrintTwoNormal method is the string to be printed, and the third parameter is passed an empty string. When printing different data on the two stations, the data to be printed needs to be put in the second and third parameters corresponding to the station names.

When using two stations at one time, please confirm that the printer supports dual printing by using the following properties:

When using the Journal and Receipt stations	CapConcurrentJrnRec property
When using the Journal and Slip stations	CapConcurrentJrnSlp property
When using the Slip and Receipt stations	CapConcurrentRecSlp property

The following is an example of how to use the API to perform concurrent printing on two stations. The JOURNAL station will be sent "Data 2" and the RECEIPT station will be sent "Data 1". If the first parameter of PrintTwoNormal method is PTR_S_JOURNAL_RECEIPT, data from the second parameter is printed on the receipt station and data from the third parameter is printed on the journal station. Error handling and asynchronous printing control are described in the "2.5.1. Synchronous Printing on One Station" and "2.5.2. Asynchronous Printing on One Station".

```

Dim D1 As String
Dim D2 As String
D1 = "Data 1"
D2 = "Data 2"
OPOSPOSPrinter1.AsyncMode = False
OPOSPOSPrinter1.PrintTwoNormal PTR_S_JOURNAL_RECEIPT,D1,D2

```

If an escape sequence that is not supported by both stations when using PrintTwoNormal

method (e.g. Paper Cut) is used, the method will not be successful. Right and Center justification escape sequence also cannot be used.

To determine synchronous or asynchronous mode, use the AsyncMode property. The length of Data1 and Data2 should be keep within one printed line. CR and LF characters are ignored.

3.5.4 Setting the Logo

Along with the data sent by the program, it is possible to print a top and bottom logo on the printer. The logo is set by using the SetLogo method. The method as printing data is used to send the logo to the printer by escape sequences (top logo is ESC|tL and bottom logo is ESC|bL). The logo escape sequences are set and transferred in character strings.

```
Dim D1 As String
Dim D2 As String
Dim D3 As String
D1 = "OPOS 2000" + Chr(13) + Chr(10)
D2 = "Thank you!!!" + Chr(13) + Chr(10)
D3 = Chr(&H1B) + "|tL" + "Data 1" + Chr(13) + Chr(10) +
    Chr(&H1B) + "|bL"

OPOSPOSPrinter1.SetLogo PTR_L_TOP,D1
OPOSPOSPrinter1.SetLogo PTR_L_BOTTOM,D2
OPOSPOSPrinter1.AsyncMode = False
OPOSPOSPrinter1.PrintNormal PTR_S_RECEIPT,D3
```

The printed results of the above program are as follows.

```
OPOS 2000
Data 1
Thank you!!!
```

3.5.5 Printing Bar Codes

Bar codes can be printed on a station if the printer supports bar code printing. To use this function, check to see if the printer is able to print bar codes, and if so send the data.

```
If OPOSPOSPrinter1.CapRecBarCode = True Then
'Bar codes can be printed.
Else
'Bar codes cannot be printed.
End If
```

It is possible to decide whether to print bar codes rotated or not before the PrintBarCode method is used. Some printers support 90-degree right, 90-degree left, and 180-degree rotated printing, but it is necessary to check the printer being used by using the RecBarCodeRotationList (SlpBarCodeRotationList) property. The supported rotation degree numbers will be seen in a string there. If the desired rotation is available, it can then be set in the RotateSpecial property.

By using the following example, 180-degree rotated bar code printing can be performed.

```
If InStr(OPOSPOSPrinter1.RecBarCodeRotationList, "180") Then
```

```
    OPOSPOSPrinter1.RotateSpecial = PTR_RP_ROTATE180
```

The following is an example of setting the parameters needed to use the PrintBarCode method.

```
Dim RC As Long
```

```
RC = OPOSPOSPrinter1.PrintBarCode (PTR_S_RECEIPT, 'The Slip or Receipt station can be specified.
```

```
"495462406082", 'A bar code number.
```

```
PTR_BCS_UPCA, 'Bar code symbol type.
```

```
100, 'Bar code height.
```

```
150, 'Bar code width.
```

```
PTR_BC_CENTER, 'Position of bar code.
```

```
PTR_BC_TEXT_BELOW'Position of bar code text.
```

```
If RC = OPOS_E_BUSY Then
```

```
    'Data is being sent.
```

```
Elseif RC = OPOS_E_ILLEGAL Then
```

```
    'There is a problem with the parameters, etc.
```

```
Elseif RC = OPOS_E_EXTENDED Then 'Extended error
```

```
    If OPOSPOSPrinter1.ResultCodeExtended = OPOS_EPTR_COVER_OPEN Then
```

```
        'Cover is open.
```

```
    Elseif OPOSPOSPrinter1.ResultCodeExtended = OPOS_EPTR_REC_EMPTY Then
```

```
        'Paper out.
```

```
    End If
```

```
End If
```

For detailed explanations of the parameters and their legal values, please refer to UPOS. When printing bar codes and specifying the bar code symbol type, other parameters are affected.

Supported symbol types may vary depending on the printer device being used, so it is recommended that further information be obtained from either the "Device Specific Programming" section or the device's product manual.

The PrintBarCode method can use both synchronous and asynchronous printing functions by setting the appropriate AsyncMode property.

It is possible to print wide bar codes using 90-degree rotated printing. Use this function when ever necessary.

Error handling and asynchronous printing control are described in the "2.5.1. Synchronous Printing on One Station" and "2.5.2. Asynchronous Printing on One Station".

3.5.6 Bitmap Printing

Bitmaps can be printed on a station that supports bitmap printing. To use this function, check to see if the printer is able to print bitmaps, and if so send the data.

```
If OPOSPOSPrinter1.CapRecBitmap = True Then
```

```
    'Bitmaps can be printed.
```

```
Else
```

```
    'Bitmaps cannot be printed.
```

```
End If
```

It is possible to change the speed and quality of a bitmap by using the XXXLetterQuality

property.

RecLetterQuality = TRUE 'print Receipt's bitmap in quality mode
RecLetterQuality = FALSE 'print Receipt's bitmap in speed mode
SlpLetterQuality = TRUE 'print Slip's bitmap in quality mode
SlpLetterQuality = FALSE 'print Slip's bitmap in speed mode

The default is speed mode. If bitmap printing mode is changed to quality mode, bitmap printing is slower, but at a higher quality. If bitmap printing mode is changed to speed mode, bitmap printing becomes faster, but lower quality. If the station is a dot printer, density will be single no matter which mode is set. If the station is a thermal printer, the density is as follows. (When the letter quality is set to 180dpi.)

Mode	Density
Speed	60 x 90 dpi, 90 x 90 dpi (When down load)
Quality	180 x 180 dpi

There are two ways of printing a bitmap. One way is by using the PrintBitmap method to directly print the bitmap.

The following is an example of setting the parameters needed to use the PrintBitmap method.

```
Dim RC As Long
RC = OPOSPOSPrinter1.PrintBitmap (PTR_S_RECEIPT, 'The Slip or Receipt station to
be used.
"Bitmap.BMP", 'The .BMP file to be printed.
PTR_BM_ASIS, 'Bitmap width.
PTR_BM_CENTER) 'Bitmap alignment.
If RC = OPOS_E_BUSY Then
'Data is being sent
Elseif RC = OPOS_E_ILLEGAL Then
'There is a problem with the parameters, etc.
Elseif RC = OPOS_E_NOEXIST Then
'The specified file does not exist.
Elseif RC = OPOS_E_EXTENDED Then
If OPOSPOSPrinter1.ResultCodeExtended = OPOS_EPTR_TOOBIG Then
'Bitmap image is too large.
Elseif OPOSPOSPrinter1.ResultCodeExtended = OPOS_EPTR_BADFORMAT Then
'The format is incorrect.
Elseif OPOSPOSPrinter1.ResultCodeExtended = OPOS_EPTR_COVER_OPEN Then
'Cover is open.
Elseif OPOSPOSPrinter1.ResultCodeExtended = OPOS_EPTR_REC_EMPTY Then
'Paper out.
End If
End If
```

For detailed explanations of the parameters and their legal values, please refer to UPOS.

The PrintBitmap method can use both synchronous and asynchronous printing functions by setting the AsyncMode property.

Error handling and asynchronous printing control are described in the "2.5.1. Synchronous Printing on One Station" and "2.5.2. Asynchronous Printing on One Station". Errors relating to bitmaps are OPOS_EPTR_TOOBIG and OPOS_EPTR_BADFORMAT.

Another way to print a bitmap is to use escape sequences. The SetBitmap method can be used to define the bitmap.

There are two ways of printing bitmaps by using escape sequences. The first is to use the printer's download bitmap image function. This is called the download bitmap printing mode. The SetBitmap method can be used to save a bitmap on the printer, and every time the correct escape sequence is used, the image will be printed. The first BitmapNumber parameter is used for this. This method has limits on the size of the bitmap that can be stored, but printing is very fast.

The second way is to send the bitmap to the printer with escape sequences each time it is to be printed. This allows any bitmap to be printed up to the size of the paper being used, but printing is slower than the first method.

The size of the bitmap that is stored is limited by the hardware being used (please refer to the bitmap printing command section of the each printer's product manual), and only one bitmap can be stored at one time (BitmapNumber parameter of the SetBitmap method is set to 1). Printing with this method, however, is much faster.

The second way is not using printer download image function. Using this way, the bitmap size can be changed to any size up to the width of the paper being used, but printing is slow. This method can be chosen by setting the BitmapNumber parameter of the SetBitmap method to 2. Either of the bitmap printing methods specified above can be set by using the DirectIO method. Please consult the class' device manual DirectIO method setting section.

The following is an example of setting the parameters needed to use the SetBitmap method.

```
Dim RC As Long
RC = OPOSPOSPrinter1.SetBitmap (1,          'Bitmap number.
    PTR_S_RECEIPT, 'The Slip or Receipt station can be specified.
    "Bitmap.BMP",  'The .BMP file to be printed.
    PTR_BM_ASIS,   'Bitmap width.
    PTR_BM_CENTER) 'Bitmap alignment.
If RC = OPOS_E_ILLEGAL Then
    'There is a problem with the parameters, etc.
Elseif RC = OPOS_E_NOEXIST Then
    'The specified file does not exist.
Elseif RC = OPOS_E_EXTENDED Then
    If OPOSPOSPrinter1.ResultCodeExtended = OPOS_EPTR_TOOBIG Then
        'Bitmap image is too large.
    Elseif OPOSPOSPrinter1.ResultCodeExtended = OPOS_EPTR_BADFORMAT Then
        'The format is incorrect.
    Elseif OPOSPOSPrinter1.ResultCodeExtended = OPOS_EPTR_COVER_OPEN Then
        'Cover is open.
    Elseif OPOSPOSPrinter1.ResultCodeExtended = OPOS_EPTR_REC_EMPTY Then
        'Paper is out
    End If
End If
End If
```

For detailed explanations of the parameters and their legal values, please refer to UPOS.

Next, the PrintNormal or PrintImmediate method is used with PrintBitmap's escape sequence that holds the bitmap's data as a parameter. The escape sequence code is "ESC|nB". *n* is the number of the bitmap specified in the SetBitmap method. It is possible to record up to two

bitmaps at one time. By using bitmap numbers, numerous bitmaps can be recorded and printed. *n* must be a 1 column integer.

```
OPOSPOSPrinter1.AsyncMode = False
OPOSPOSPrinter1.PrintNormal PTR_S_RECEIPT, Chr(&H1B) + "|1B"
```

When using the PrintNormal method, it is possible to print bitmap picture rotated to 90-degree or 180-degree.

Printing will not occur when printing to a station other than the station specified in the SetBitmap method.

3.5.7 Rotated Printing

Printed data can be turned 90-degree or 180-degree. To use this function, check to see if the printer is able to rotate printing, and if so send the data.

```
[90-degree Right Printing]
If OPOSPOSPrinter1.CapRecRight90 = True Then
    '90-degree right printing is possible.
Else
    '90-degree right printing is not possible.
End If
```

```
[90-degree Left Printing]
If OPOSPOSPrinter1.CapRecLeft90 = True Then
    '90-degree left printing is possible.
Else
    '90-degree left printing is not possible.
End If
```

```
[180-degree printing]
If OPOSPOSPrinter1.CapRecRotate180 = True Then
    '180-degree printing is possible.
Else
    '180-degree printing is not possible.
End If
```

The following is an example of setting the parameters needed to use the RotatePrint method.

By setting the rotate printing mode, rotated printing can be performed by using the PrintNormal or PrintImmediate methods. (90-degree rotated printing can be performed by using PrintNormal only.)

```
Dim RC As Long
RC = OPOSPOSPrinter1.RotatePrint (PTR_S_RECEIPT, PTR_RP_RIGHT90)
If RC = OPOS_SUCCESS Then
    OPOSPOSPrinter1.PrintNormal PTR_S_RECEIPT, "Right 90 Printing."+ Chr(13)+ Chr(10)
    + "AAAAA" + Chr(13) + Chr(10) + "BBBBB" + Chr(13) + Chr(10)
RC = OPOSPOSPrinter1.RotatePrint (PTR_S_RECEIPT, PTR_RP_NORMAL)
    If RC = OPOS_E_BUSY Then
        'Data is being sent
    ElseIf RC = OPOS_E_ILLEGAL Then
        'Specified station does not exist
```

```

Elseif RC = OPOS_E_EXTENDED Then
    If OPOSPOSPrinter1.ResultCodeExtended = OPOS_EPTR_COVER_OPEN Then
        'Cover is open.
    Elseif OPOSPOSPrinter1.ResultCodeExtended = OPOS_EPTR_REC_EMPTY Then
        'Paper out.
    End If
End If
Else
    'Error
End If

```

After 90-degree rotated printing has been specified by the RotatePrint method, all print data is buffered. By changing the rotation parameter of the RotatePrint method or executing TransactionPrint method, all buffered data is printed based on the before rotation parameter.

When using 90-degree rotated printing, there are limits on the possible number of lines and columns. These limits can be obtained by checking the XXXSidewaysMaxChars and XXXSidewaysMaxLines properties. (XXX should be replaced by Rec or Slp.) When the number specified by MaxLine is passed, data will be discarded.

3.5.8 Immediate Printing

Immediate Printing is the ability to move data to the top of the priority list. Data can even be sent and printed before other data that has been sent in asynchronous mode. The PrintImmediate method is used to accomplish this.

For example, if the LineFeed(&H0A) command is sent while asynchronous data is being sent, a line feed will occur in the middle of the data that is currently being printed. When the immediate data has finished, the normal data will continue printing.

```

Dim RC As Long
RC = OPOSPOSPrinter1.PrintImmediate (PTR_S_JOURNAL,Chr(13)+Chr(10))
If RC = OPOS_SUCCESS Then
    'Data printed successfully.
Elseif RC = OPOS_E_ILLEGAL Then
    'Specified station does not exist, etc.
Elseif RC = OPOS_E_EXTENDED Then 'Extended error.
    If OPOSPOSPrinter1.ResultCodeExtended = OPOS_EPTR_COVER_OPEN Then
        'Cover is open.
    Elseif OPOSPOSPrinter1.ResultCodeExtended = OPOS_EPTR_JRN_EMPTY Then
        'Paper out.
    End If
End If

```

3.5.9 Collective Printing

Collective printing mode can be set by using the TransactionPrint method. When this function is used, be sure to confirm that the printer supports collective printing by checking its properties.

```

If OPOSPOSPrinter1.CapTransaction = True Then
    'collective printing is possible
Else
    'collective printing is not possible
End If

```

All print data specified by the PrintNormal method for collective printing is buffered. It is not

possible to print with any other method than PrintNormal. The AsyncMode property also becomes unusable when the PrintNormal method is used for collective printing. The following is a programming example of collective printing.

```
Dim RC As Long
RC=OPOSPOSPrinter1.TransactionPrint (PTR_S_RECEIPT, PTR_TP_TRANSACTION)
If RC = OPOS_SUCESS Then
    OPOSPOSPrinter1.PrintNormal PTR_S_RECEIPT, " Print Data 1" + Chr(13) + Chr(10)
    OPOSPOSPrinter1.PrintNormal PTR_S_RECEIPT, "Print Data 2" + Chr(13) + Chr(10)
    OPOSPOSPrinter1.PrintNormal PTR_S_RECEIPT, "Print Data 3" + Chr(13) + Chr(10)
    OPOSPOSPrinter1.PrintNormal PTR_S_RECEIPT, "Print Data 4" + Chr(13) + Chr(10)
    `nothing is printed up to this point
    OPOSPOSPrinter1.AsyncMode = True      `synchronous or asynchronous mode is chosen
here
    RC=OPOSPOSPrinter1.TransactionPrint(PTR_S_RECEIPT, PTR_TP_NORMAL)
    If RC = OPOS_SUCESS Then
        ` printing was successful
    ElseIf RC = OPOS_E_BUSY Then
        ` data is still being sent
    ElseIf RC = OPOS_E_EXTENDED Then
        If OPOSPOSPrinter1.ResultCodeExtended = OPOS_EPTR_COVER_OPEN Then
            `cover is open
        ElseIf OPOSPOSPrinter1.ResultCodeExtended = OPOS_EPTR_REC_EMPTY
Then
            `out of paper
        End If
    End If
ElseIf RC = OPOS_E_ILLEGAL Then
    `the specified station does not exist or collective printing is not supported
End If
```

3.6 Form Insertion/Removal and Slip Printing

When using the Slip station, methods can be used to insert and remove forms.

The BeginInsertion method makes the station ready to receive a form. If a form is inserted within the set time interval, the method returns OPOS_SUCCESS. If a form has been inserted in the station, the EndInsertion method locks the form in place (closes the jaws) making it ready for printing. If the station is waiting to receive a form, the EndInsertion method cancels the ready state of the station.

To remove a form, the BeginRemoval method can be used to release the jaws and eject the paper. The EndRemoval method can be used to cancel the closed state of the station.

If there is no roll paper, an error will be returned when a form is inserted or removed, because there may be a paper jam in the printer. If an error occurs, please check the roll paper.

To insert a form:

```
Dim RC As Long
RC = OPOSPOSPrinter1.BeginInsertion (5000)
If RC = OPOS_SUCCESS Then
    OPOSPOSPrinter1.EndInsertion
    `Slip is ready to be printed on.
ElseIf RC = OPOS_E_TIMEOUT Then
```

```

'Slip has not been inserted in a setting time
    Else
        'Other error
End If

```

Printing can occur on the slip station after BeginInsertion and EndInsertion method have been executed.

```
OPOSPOSPrinter1.PrintNormal PTR_S_SLIP, "Print Slip Station" + Chr(13) + Chr(10)
```

To remove a form:

```

Dim RC As Long
RC = OPOSPOSPrinter1.BeginRemoval (5000)
If RC = OPOS_SUCCESS Then
    RC=OPOSPOSPrinter1.EndRemoval
Elseif RC = OPOS_E_TIMEOUT Then
    'Paper is still in the station
Else
    'Other error
End If

```

Immediately after asynchronous data has been sent, it is possible that the BeginRemoval method will return an OPOS_E_BUSY error. It is suggested that programs wait until the OPOS_E_BUSY error is cleared. This can be done by checking the OutputID or FlagWhenIdle properties after asynchronous data has been sent.

Also, while a station is printing, it is not possible to remove the paper, so a timeout will occur if the BeginRemoval method is used before printing is finished. Be sure to only use the BeginRemoval method after the printer is finished printing.

If BeginRemoval is used to remove paper from the printer, OPOS_SUCCESS will not be returned until the paper is completely removed from the station. Paper is not completely removed while the slip LED is flashing. It is usually necessary to remove paper by hand.

3.7 Paper Cutting

Paper can be cut by using the automatic cutter on the Receipt station. To use this function, check to see if the printer is able to perform automatic cuts, and if so instruct it to do so.

```

If OPOSPOSPrinter1.CapRecPapercut = True Then
    'The printer has a paper cutter.
Else
    'The printer does not have a paper cutter.
End If

```

There are two ways to perform a paper cut. One way is by using the CutPaper method to directly cut the receipt.

The percentage of paper to cut can be set with the CutPaper method. The exact cut percentage that can be passed by paper cutting is as follows.

0%	No cut.
70%	All but three points will be cut.

90% All but one point will be cut.
100% Full cut. If full cut is not possible, all but one point will be cut.

0 to 100 percentage values other than the above is handled as follows.

0 Nothing cut
1-79 All but three points will be cut. If this cut is not possible, all but one point will be cut.
80-99 All but one point will be cut.
100 Full cut. If full cut is not possible, all but one point will be cut.
100< Error

Any other percentage used when data is checked with ValidateData method will cause an OPOS_E_ILLEGAL error to be returned.

Cutting ability varies by printer. For detailed information, please refer to UPOS and Device Specific Programming manuals.

```
OPOSPOSPrinter1.CutPaper 100 'Full Cut.  
If OPOSPOSPrinter1.ResultCode = OPOS_SUCCESS Then  
    ' OK  
Else  
    ' NG  
End If
```

The CutPaper method can use both synchronous and asynchronous printing functions by setting the appropriate AsyncMode property.

Error handling and asynchronous printing control are described in the “2.5.1. Synchronous Printing on One Station” and “2.5.2. Asynchronous Printing on One Station”.

Another way to perform a paper cut is by using an escape sequence. The PrintNormal and PrintImmediate methods can pass the paper cut escape sequence to the printer as a parameter. The paper cut escape sequence is ESC|*n*P, where *n* is the percentage of paper to cut. *n* can be given a number from 0 to 100 that is passed as a character string. Specified percentage is treated the same as by the CutPaper method, but if a value of more than 100 is set or no value is specified, it will default to 100.

```
OPOSPOSPrinter1.AsyncMode = False  
OPOSPOSPrinter1.PrintNormal PTR_S_RECEIPT, Chr(&H1B) + “|100P”
```

When using a printer's paper cut function, please refer to the RecLineToPaperCut property in order to advance the paper far enough before it is cut. By using the “ESC|*#f*P” escape sequence, paper can be cut after being advanced the number of lines specified without using the RecLineToPaperCut property.

While there is still output data in the printer buffer and the CutPaper method is used, the paper will be cut after all data in the buffer has been printed, but if an escape sequence is used to cut the paper, nothing will happen.

Printers without cutter capabilities will return OPOS_E_FAILURE.

3.8 Checking the Printer State

The state of the printer can be checked through properties supported by the printer. For example, if the user wants to check if the printer cover is open, the CoverOpen property can be used.

```
If OPOSPOSPrinter1.CoverOpen = True Then
    MsgBox "Cover is open!"
End If
```

This and many other events can also be checked by firing a StatusUpdateEvent.

[Event management]

```
Private Sub OPOSPOSPrinter1_StatusUpdateEvent (ByVal Data As Long)
If Data = PTR_SUE_COVER_OPEN Then
    MsgBox "Cover is open!"
End If
End Sub
```

StatusUpdateEvent can return information on the following items.

STATUS	INFORMATION
PTR_SUE_COVER_OPEN	Cover is open.
PTR_SUE_COVER_OK	Cover is closed.
PTR_SUE_JRN_EMPTY	Journal paper is out.
PTR_SUE_JRN_NEAREMPTY	Journal paper is near the end.
PTR_SUE_JRN_PAPEROK	Journal paper is OK.
PTR_SUE_JRN_CARTRIDGE_EMPTY	Journal cartridge is out.
PTR_SUE_JRN_CARTRIDGE_NEAREMP TY	Journal cartridge is near the end.
PTR_SUE_JRN_HEAD_CLEANING	Journal cartridge starts cleaning
PTR_SUE_JRN_CARTRIDGE_OK	Journal cartridge is OK.
PTR_SUE_REC_EMPTY	Receipt paper is out.
PTR_SUE_REC_NEAREMPTY	Receipt paper is near the end.
PTR_SUE_REC_PAPEROK	Receipt paper is OK.
PTR_SUE_REC_CARTRIDGE_EMPTY	Receipt cartridge is out.
PTR_SUE_REC_CARTRIDGE_NEAREMP TY	Receipt cartridge is near the end.
PTR_SUE_REC_HEAD_CLEANING	Receipt cartridge starts cleaning.
PTR_SUE_REC_CARTRIDGE_OK	Receipt cartridge is OK.
PTR_SUE_SLP_EMPTY	Slip paper is out.
PTR_SUE_SLP_NEAREMPTY	Slip paper is near the end.
PTR_SUE_SLP_PAPEROK	Slip paper is OK.
PTR_SUE_SLP_CARTRIDGE_EMPTY	Slip cartridge is out.
PTR_SUE_SLP_CARTRIDGE_NEAREMP TY	Slip cartridge is near the end.
PTR_SUE_SLP_HEAD_CLEANING	Slip cartridge starts cleaning.
PTR_SUE_SLP_CARTRIDGE_OK	Slip cartridge is OK.
PTR_SUE_IDLE	Printer State is idle.

When the FlagWhenIdle property is set to TRUE, PTR_SUE_IDLE is sent to inform the application that the printer is idle. Other than when data is being sent, the printer is in an idle

state, so if FlagWhenIdle is TRUE, an event will be fired when printing is finished. After the event is fired, FlagWhenIdle will be set to FALSE. By using this value, the information below can be found out.

*Finding out when multiple asynchronous print jobs have finished printing.

When multiple asynchronous print jobs have been sent to the printer, it is possible to know when they have finished printing. After setting the AsyncMode property to TRUE and running the PrintNormal method, change the FlagWhenIdle property to TRUE. When all data has finished printing, the printer becomes idle and a StatusUpdateEvent is fired to the application with the value of PTR_SUE_IDLE.

3.9 Printer Errors and Status

A change in printer status when asynchronous data is being sent is made available to the program by the firing of an ErrorEvent and StatusUpdateEvent. When the printer changes status while nothing is happening, the change is told to the program by a StatusUpdateEvent only.

As an example, assume that the printer cover becomes open. Usually, when data is not being sent to the printer and the cover is opened, a StatusUpdateEvent is fired to the application. After the AsyncMode property is set to TRUE and a method is used to print data, if the cover is opened while the data is being sent, the program is notified by an ErrorEvent. ErrorEvent are fired when the error has interrupted the data that is being sent. Only StatusUpdateEvent will fire when the error does not affect the data and the data will continue being sent as normal.

Reasons for the ErrorEvent being fired and the corresponding error names are listed below.

ResultCode/ ResultCodeExtended	Reason
OPOS_E_ILLEGAL	There is an abnormality with the device. (Includes the following 1 error)
OPOS_EX_INVALIDMODE	During slip printing mode, printed to another station
OPOS_E_EXTENDED	Error determined by the device's SO (Includes the following 4 errors)
OPOS_EPTR_COVER_OPEN	Cover is open.
OPOS_EPTR_JRN_EMPTY	Journal paper is empty.
OPOS_EPTR_REC_EMPTY	Receipt paper is empty.
OPOS_EPTR_SLP_EMPTY	There is no form in the slip station.
OPOS_E_FAILURE	Hard error (Includes the following 6 errors).
OPOS_EPTR_UNRECOVERABLE	Error that cannot be recovered form.
OPOS_EPTR_CUTTER	Error with the automatic cutter.
OPOS_EPTR_MECHANICAL	Mechanical error.
OPOS_EPTR_OVERHEAT	Head overheat error.
OPOS_EX_MICRMODE	In MICR mode error.
OPOS_EX_DEVBUSY	Device busy error.
OPOS_EPTR_JRN_CARTRIDGE_REMOVED	Journal cartridge is removed.
OPOS_EPTR_JRN_CARTRIDGE_EMPTY	Journal cartridge is empty.
OPOS_EPTR_JRN_HEAD_CLEANING	Journal head starts cleaning.
OPOS_EPTR_REC_CARTRIDGE_REMOVED	Receipt cartridge is removed.

OPOS_EPTR_REC_CARTRIDGE_EMPTY	Receipt cartridge is empty.
OPOS_EPTR_REC_HEAD_CLEANNING	Receipt head starts cleaning.
OPOS_EPTR_SLP_CARTRIDGE_REMOVED	Slip cartridge is removed.
OPOS_EPTR_SLP_CARTRIDGE_EMPTY	Slip cartridge is empty.
OPOS_EPTR_SLP_HEAD_CLEANNING	Slip head starts cleaning.

After an error occurs, detailed information about the error can be obtained from the ErrorLevel, ErrorStation, and ErrorString properties.

3.10 Clearing the Output Buffer

Data that is currently being output is not affected when the ClearOutput method is used.

For example, when the following data has been buffered,

Output Data 1	OutputID=1 Being sent now
Output Data 2	OutputID=2
Output Data 3	OutputID=3
Output Data 4	OutputID=4

the data in "Output Data 1" will all be printed. All the data in Output Data 2, Output Data 3, and Output Data 4 will be cleared. After the data is cleared, an OutputCompleteEvent will not be fired to the application.

3.11 Testing with the CheckHealth Method

Printers support the CheckHealth method's Level 1, 2 and 3 setting. The tests can be used to ensure the correct connection of the device. For more details, please refer to the "EPSON OPOS ADK MANUAL APPLICATION DEVELOPMENT GUIDE POSPrinter (TM Series)".

3.12 Cartridge State

From the UPOS Release 1.5, the methods and the properties to confirm the cartridge state have been added. For those printers that have the cartridge sensor can confirm the cartridge state by events. First, confirm whether a printer has the cartridge sensor or not using the CapXxxCartridgeSensor property. Then specify the cartridge to be used, and confirm the status of the cartridge. When setting the CartridgeNotify to ENABLE, the cartridge state will be notified by the event.

[Main Management]

If OPOSPOSPrinter1.CapJrnCartridgeSensor = True Then 'Journal Cartridge exists

 If OPOSPOSPrinter1.CapJrnColor And PTR_COLOR_CUSTOM1 = True then
 'Secondary color selection

```

OPOSPOSPrinter1.DeviceEnable = TRUE
OPOSPOSPrinter1.CartridgeNotify = PTR_CN_DISABLED
OPOSPOSPrinter1.JrnCurrentCartridge = PTR_COLOR_CUSTOM1
If OPOSPOSPrinter1. JrnCartridgeState = PTR_CART_REMOVED then
    'Cartridge is removed.
Elseif OPOSPOSPrinter1. JrnCartridgeState = PTR_CART_CLEANING then
    'Cartridge is in cleaning
Elseif OPOSPOSPrinter1. JrnCartridgeState = PTR_CART_NEAREND then
    'Cartridge is near the end
Else
    'Other errors
End If
End If
End If

```

[Event Management]

```

Private Sub OPOSPOSPrinter1_StatusUpdateEvent(ByVal Data As Long)
    If Data = OPOS_EPTR_JRN_CARTRIDGE_REMOVED Then
        OPOSPOSPrinter1.JrnCurrentCartridge = PTR_COLOR_CUSTOM1
        If OPOSPOSPrinter1. JrnCartridgeState = PTR_CART_REMOVED then
            'Cartridge is removed.
        End If
    End If
End Sub

```

3.13 Color Printing

From the UPOS Release 1.5, the methods and the properties for color printing have been added. As referring to the CapXxxColor property, supported colors can be confirmed. After confirming the available colors, color printing can be done using ESC |#rC or ESC |rC.

```

Dim Color As String
If OPOSPOSPrinter1.CapJrnColor And PTR_COLOR_CUSTOM1 = True then 'Secondary
color selection
    OPOSPOSPrinter1.AsyncMode = False
    Color = CStr iPTR_COLOR_CUSTOM1 j
    OPOSPOSPrinter1.PrintNormal PTR_S_JOURNAL,Chr(&H1B) + "|" + Color + "rC" +
"Color Custom1"
End If

```

3.14 Mark Sensed Paper Support

From the UPOS Release 1.5, the method for the mark sensed receipt paper has been added. To confirm whether the available device or not, please refer to the CapRecMarkFeed property. The various kinds of mark sensed paper feedings are done by the MarkFeed method.

3.15 Printing on Both Sides

From the UPOS Release 1.5, the function for printing on both sides has been added. To confirm whether the available device or not, please refer to the CapSlpBothSidePrint property. The printing side can be changed by the ChangePrintSide method. The changed value is in the SlpPrintSide property.

```
If OPOSPOSPrinter1.CapSlpBothSidePrint = True Then    'The printer can print on both
sides
    If OPOSPOSPrinter1.SlpPrintSide = PTR_PS_SIDE2 then 'The reverse side
        OPOSPOSPrinter1.ChangePrintSide PTR_PS_SIDE1
    End If
    OPOSPOSPrinter1.PrintNormal PTR_S_SLIP C"Default Print Side"+Chr(13) + Chr(10)
    OPOSPOSPrinter1.ChangePrintSide PTR_PS_SIDE2
    OPOSPOSPrinter1.PrintNormal PTR_S_SLIP C"Reverse Side of Default Side"
+Chr(13) + Chr(10)
End If
```

3.16 Things to Consider when Using Properties

There are times when a command will be sent to the printer when a property is set. When the printer is not able to accept commands (printer is busy, etc), properties will not be able to be set, and an error will be returned.

There are some properties that cannot be set while data is being buffered for 90-degree rotated printing. An error will be returned in this case.

Properties that send commands to the printer cannot be set while the printer is printing in asynchronous mode.

Section 4. Line Display

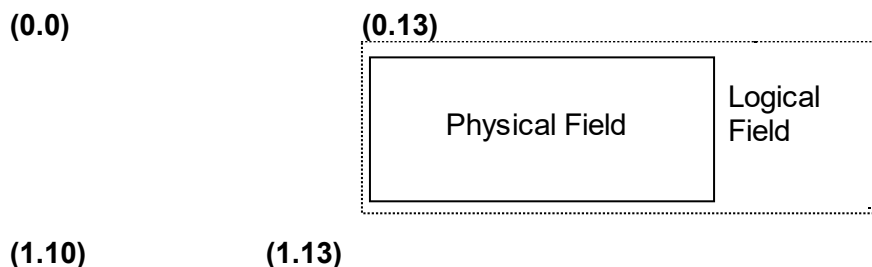
Programming examples of how to use API functions relating to a Line Display are shown below.

4.1 Window Creation/Destruction

In the line display window, there is a physical field that shows in the display area, and a logical field that extends beyond the physical field to allow updating and character movement. All windows on the display are made up of these two fields, and by default the physical field is always contained within the logical field. The logical field may be larger than the physical field in either the horizontal or the vertical direction. It is not possible to create a physical field window that is larger than the logical field window. The logical and physical fields can be set to the same size.

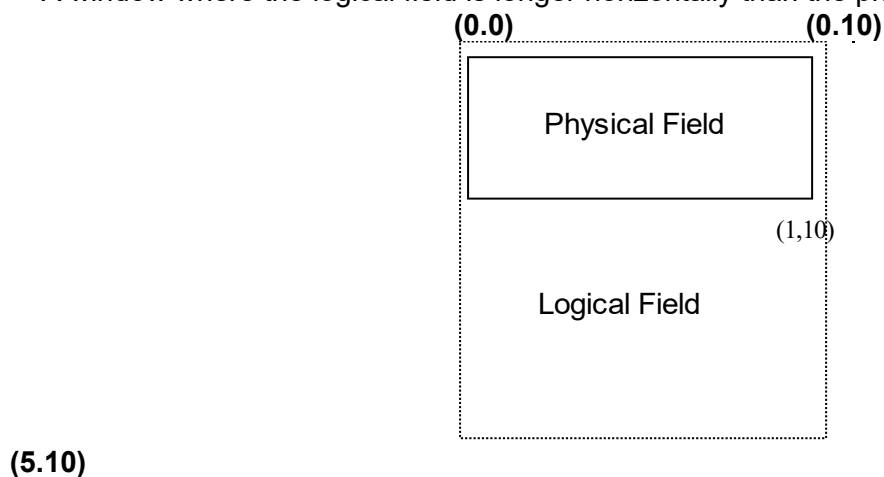
The following are examples of the three types of windows that are possible.

A window where the logical field is longer horizontally than the physical field:



WINDOW
Physical Field: (0.0) - (1.10)
Logical Field: (0.0) - (1.13)

A window where the logical field is longer horizontally than the physical field:



(5.10)

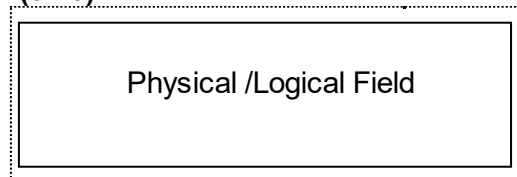
WINDOW
Physical Field: (0.0) - (1.10)

Logical Field: (0.0) - (5.10)

A window where the logical and physical fields are the same:

(0.0)

(0.10)



(1,10)

WINDOW

Physical Field: (0.0) - (1.10)

Logical Field: (0.0) - (1.10)

A window is created by using the CreateWindow method. Until the CreateWindow method is used, window 0 is used as the current window. The physical field and logical field of window 0 are initially the same size, and fill the maximum possible area that the display settings allow. The field's sizes vary depending on the type of display that is being used, and are made available to the program by the DeviceRows and DeviceColumns properties.

In addition to the default window, any other logical windows can be created.

The following is the code necessary to create the window.

```
Dim RC As Long
Dim Wno As Long
RC = OPOSLineDisplay1.CreateWindow (0,0,2,11,2,14) 'Window dimensions.
Wno = OPOSLineDisplay1.CurrentWindow 'The window that is now
displayed.
If RC <> OPOS_SUCCESS Then
    'Error
Else
    'OK
End If
```

When a window is created, the new window is automatically set as the current window. After creating a new window and obtaining the window number from the CurrentWindow property, the number can be used to return to the window after changing to a different window. By setting the CurrentWindow property, the window that is shown on the display can be easily changed.

```
Dim Wno1 As Long
Dim Wno2 As Long
OPOSLineDisplay1.CreateWindow 0,0,2,11,2,14
Wno1 = OPOSLineDisplay1.CurrentWindow
OPOSLineDisplay1.CreateWindow 0,0,1,20,1,20
Wno2 = OPOSLineDisplay1.CurrentWindow
OPOSLineDisplay1.CurrentWindow = Wno1
```

In the above program, the CurrentWindow property is changed after creating the second window, making the first window the current window.

When there is no longer a need for a window, the window can be destroyed by using the DestroyWindow method. The current window is destroyed and the CurrentWindow property is reset to the default Window 0. Window 0 cannot be destroyed.

4.2 Window Rows/Columns

The number of rows and columns in the current window can be obtained from the Rows and Columns properties. These properties are determined by the parameters set when the window was created. Window 0 rows and columns are determined by the type of display. (This is same value at the DeviceRows and DeviceColumns property.)

4.3 Showing Data on the Display

There are two ways to show data on a display. One way is by sending a character string to the display and letting the data appear from the point where the cursor is. If the CursorUpdate property is TRUE, the CursorRow and CursorColumn properties are changeable. After the data is displayed, the CursorRow and CursorColumn properties are set to the next position after the value last data object. By changing the CursorRow and CursorColumn properties, data can be displayed at the discretion of the display. If the CursorUpdate property is FALSE, the CursorRow and CursorColumn properties cannot be changed.

The character attribute can be specified by setting the second parameter of the Display Text method to DISP_DT_NORMAL, DISP_DT_BLINK, DISP_DT_REVERSE, or DISP_DT_BLINK_REVERSE.

To display data at the current cursor position:

```
OPOSLineDisplay1.DisplayText "Data 1", DISP_DT_NORMAL
If OPOSLineDisplay1.ResultCode <> OPOS_SUCCESS Then
    'Error
End If
```

To start data showing at the upper left hand corner of the display:

```
OPOSLineDisplay1.CursorRow = 0
OPOSLineDisplay1.CursorColumn = 0
OPOSLineDisplay1.DisplayText "Data 1", DISP_DT_NORMAL
If OPOSLineDisplay1.ResultCode <> OPOS_SUCCESS Then
    'Error
End If
```

The other way to display data is by using the DisplayTextAt method to determine where the data should be shown. As above, the CursorRow and CursorColumn properties are dependent on the CursorUpdate property being TRUE.

To start data showing at the 10th column of the 1st row of the display:

```
OPOSLineDisplay1.DisplayTextAt 0, 10, "Data 1", DISP_DT_NORMAL
If OPOSLineDisplay1.ResultCode <> OPOS_SUCCESS Then
    'Error
End If
```

To start data showing at the upper left hand corner of the display:

```
OPOSLineDisplay1.DisplayTextAt 0, 0, "Data 1", DISP_DT_NORMAL
If OPOSLineDisplay1.ResultCode <> OPOS_SUCCESS Then
    'Error
End If
```

The InterCharacterWait method can be used to create a time interval between displayed data. By sending a large number as a parameter, the time between characters is made large, and smaller numbers will make the interval shorter. The default value is "0".

When marquee mode has not been set and the InterCharacterWait property is set to 0, the display is in "immediate mode" and the specified data will be displayed in the display field instantly. If the InterCharacterWait property is set to a value other than 0, the display is placed in "teletype mode". Data will be displayed one character at a time separated by the specified interval. Use these modes to create the desired display effects.

If data is set that is longer than the logical field, the letters will scroll until the entire data string has been displayed. Due to this, the data at the beginning of the string that is pushed off the top of the editing window will disappear. Please confirm the size of the window before displaying data.

If more than 0x80 data is sent, please use the BinaryConversion.

The following is the example of the programming.

```
OPOSLineDisplay1.CharacterSet = 858                                * Code page: PC858
*Printing EURO characters
OPOSLineDisplay1.BinaryConversion = OPOS_BC_NIBBLE
OPOSLineDisplay1.DisplayText "=5=5=5",DISP_DT_NORMAL

OPOSLineDisplay1.BinaryConversion = OPOS_BC_DECIMAL
OPOSLineDisplay1.DisplayText "213213213",DISP_DT_NORMAL
```

If an un-displayable character is sent with a data string, that character is not displayed but the character's position will be included by the cursor. Please confirm that all characters sent are displayable.

4.4 Window Clear/Refresh

The RefreshWindow method is used to update any specified window. By passing the window number to the device as a parameter, the window to refresh can be chosen.

```
Dim Wno1 As Long
Dim Wno2 As Long
OPOSLineDisplay1.CreateWindow 0,0,2,11,2,14
Wno1 = OPOSLineDisplay1.CurrentWindow
OPOSLineDisplay1.DisplayText "Window 1", DISP_DT_NORMAL
|
OPOSLineDisplay1.CreateWindow 0,0,1,20,1,20
Wno2 = OPOSLineDisplay1.CurrentWindow
OPOSLineDisplay1.DisplayText "Window 2", DISP_DT_NORMAL
|
OPOSLineDisplay1.RefreshWindow Wno1
```

In the above program, the windows Wno1 and Wno2 are created with different parameters and data is sent to display on each. After both windows are created, the data passed to window Wno2 is showing on the display. The RefreshWindow is used to update window Wno1, bringing Wno1 to the top and allowing its data to be shown on the display. After specifying Wno1 with the RefreshWindow method, Wno1 is placed in the CurrentWindow property.

To clear data from a logical window, use the ClearText method. The entire logical field of the

window will be cleared.

To make re-displaying process easy to understand, windows with overlapping parts have been created here. In an application, these overlapping windows may develop inconsistencies, therefore they are not recommended to use.

4.5 Descriptors

Descriptors can be turned ON/OFF on displays with descriptor displaying capabilities. The descriptor ability of a display can be checked with the CapDescriptors property. Descriptors can be shown on a display by using the SetDescriptor method. The descriptor position is determined by setting a parameter. The number of descriptors that can be shown on a display varies depending on the device model being used and can be checked by using the DeviceDescriptors property. The ClearDescriptors method will clear all descriptors, or can be used to clear specific descriptors by setting the attribute parameter of the SetDescriptor method.

The following is an example of a program to turn descriptors ON/OFF.

To display all descriptors:

```
Dim I As Long
If OPOSLineDisplay1.CapDescriptors = True Then
    For I = 0 To OPOSLineDisplay1.DeviceDescriptors - 1
        OPOSLineDisplay1.SetDescriptor I, DISP_SD_ON
    Next I
End If
```

To display the third descriptor:

```
If OPOSLineDisplay1.CapDescriptors = True Then
    OPOSLineDisplay1.SetDescriptor 2, DISP_SD_ON
End If
```

To clear all descriptors:

```
If OPOSLineDisplay1.CapDescriptors = True Then
    OPOSLineDisplay1.ClearDescriptors
End If
```

To clear the third descriptor:

```
If OPOSLineDisplay1.CapDescriptors = True Then
    OPOSLineDisplay1.SetDescriptor 2, DISP_SD_OFF
End If
```


4.6 Scrolling the Display

The data on a display screen can be scrolled. The entire logical field of a window is used to show and hide data to create the scrolling effect.

Scrolling performs differently when the size of the physical field and logical field are the same and when they are different.

The following are examples of how to use the logical and physical fields to scroll data on the display.

When the physical and logical fields are the same size:

```
OPOSLineDisplay1.CreateWindow 0,0,2,10,2,10  
OPOSLineDisplay1.DisplayTextAt 0,0, "This is scroll data.", DISP_DT_NORMAL
```

The above code results in the following data showing on the display.

```
This is sc  
roll data.
```

After this is accomplished, the ScrollText method is used to scroll the data left or right.

```
OPOSLineDisplay1.ScrollText DISP_ST_LEFT, 2
```

This line moves the data left two columns, resulting in the following display.

```
is is sc  
ll data.
```

The window has been scrolled to the left two columns, so the first two characters have disappeared. Next, the program will use the ScrollText method to move the data right.

```
OPOSLineDisplay1.ScrollText DISP_ST_RIGHT, 2
```

This line results in the following data showing on the display.

```
is is sc  
ll data.
```

As can be seen, the data has been moved to the right by two columns, but data that was pushed beyond the logical field boundary has disappeared and cannot be recovered.

To move the data up or down, the same method is used.

```
OPOSLineDisplay1.ScrollText DISP_ST_UP, 1
```

This line results in the following data showing on the display.

```
ll data.
```

The second line shows nothing. Using DISP_ST_DOWN to move the data down will not bring back the first line because it has been moved beyond the logical field's boundaries.

When the physical and logical fields are different sizes:

```
OPOSLineDisplay1.CreateWindow 0,0,2,10,2,20  
OPOSLineDisplay1.DisplayTextAt 0,0, "This is scroll data. OPOS ADK XXX Co. LTD!*"
```

The above code results in the following data showing on the display.

```
This is Sc
```

OPOS ADK X

After this is accomplished, the ScrollText method is used to scroll the data left or right.

```
OPOSLineDisplay1.ScrollText DISP_ST_LEFT, 2
```

This line moves the data left two columns, resulting in the following display.

is is scro

OS ADK XXX

The window has been scrolled to the left two columns, bringing out the next two columns from the logical field, which contain “ro” and “XX “. Next, the program will use the ScrollText method to move the data right.

```
OPOSLineDisplay1.ScrollText DISP_ST_RIGHT, 2
```

This line results in the following data showing on the display.

This is sc

OPOS ADK X

As can be seen, the data has been moved to the right by two columns, and the data that was pushed beyond the physical field boundary has been recovered.

Setting a value that is beyond the boundary of the logical field,

```
OPOSLineDisplay1.ScrollText DISP_ST_LEFT, 25
```

will result in the following data showing on the display.

roll data.

N Co. LTD!*

No error will be returned after using this command.

When scrolling up or down in a window where the physical field and logical field only differs in width, there is no difference from when both fields are exactly the same size.

When scrolling up or down in a window where the fields are of different height, characters that cannot be seen in the display can be displayed again, but when scrolling left or right, the window acts the same as if both the physical field and logical fields are the same size.

4.7 Marquee Settings

A marquee is a character string that is scrolled across the display in the direction and style specified by the settings.

There are two types of marquees: In-place type and walking type. For details on these types of marquee, please refer to UPOS. The marquee type can be chosen by setting the MarqueeFormat property.

By setting the MarqueeType property to DISP_MT_INIT, the marquee string to be displayed can be specified. When DISP_MT_INIT is set, the DisplayText, DisplayTextAt, or ClearText methods can be used. At this time, display management cannot be performed. After this, by setting the MarqueeType property to a value other than DISP_MT_NONE or DISP_MT_INIT, the marquee function will start. To stop a marquee, set the MarqueeType property to DISP_MT_NONE. When a marquee is started, the specified character string will scroll until the marquee function is stopped. When the MarqueeType property is set from DISP_MT_NONE to DISP_MT_LEFT/RIGHT/UP/DOWN, the character strings that are displayed at that moment will be the target of the marquee.

There are two other properties besides the MarqueeType that affect the marquee. The first, MarqueeUnitWait, can be used to create a pause before each data character is displayed. This property is valid for all scrollable characters. The second property is MarqueeRepeatWait, which creates a pause in between the display of data strings. After a data string has been shown, the program waits for the amount of time specified in the property and then displays the same string again.

Marquee functions can only be used on displays that support marquees. To confirm a display's ability to move a marquee up and down, check the CapVMarquee property. For left and right movement, check the CapHMarquee property.

The following is the procedure needed to use a marquee:

1. Confirm that the display supports marquee settings. (CapVMarquee and CapHMarquee).
2. Decide the marquee display field.
3. Build a window by the CreateWindow method.
4. Choose either place type or walk type at MarqueeFormat property.
5. Set the display speed and interval using properties.(MarqueeRepeatWait and MarqueeUnitWait)
6. Set the MarqueeType property to DISP_MT_INIT.
7. The data string to become the marquee is shown by using the DisplayText or DisplayTextAt method.
8. Set the MarqueeType property to something other than DISP_MT_NONE and DISP_MT_INIT.

Marquees cannot be shown on window 0. To display a marquee, it is necessary to create a window for use by the marquee.

Marquees can be displayed when the logical field is larger than the physical field.

The following is an example of creating and displaying a marquee.

[Marquee ON]

```
If OPOSLineDisplay1.CapHMarquee = True Then
OPOSLineDisplay1.CreateWindow 0,0,1,20,1,33
OPOSLineDisplay1.MarqueeFormat = DISP_MF_WALK           'walking type
OPOSLineDisplay1.MarqueeUnitWait = 100                  '100 milliseconds
OPOSLineDisplay1.MarqueeRepeatWait = 300                 '300 milliseconds
OPOSLineDisplay1.MarqueeFormat = DISP_MT_INIT
OPOSLineDisplay1.DisplayText "Welcome to our shop!!! Thank you.", DISP_DT_NORMAL
OPOSLineDisplay1.MarqueeType = DISP_MT_LEFT              'Scroll left
End If
```

The data string "Welcome to our shop!!! Thank you." will be repeatedly displayed on the first row of the window and scrolled to the left.

The following is a list of actions that should result from the above code:

1. The character string 'Welcome to our shop!!!' will be displayed from the end of the physical field one character at a time after the delay specified in the MarqueeUnitWait property.
2. After the entire character string has been displayed, the display waits for the amount of

time set in the MarqueeRepeatWait property. After the time interval has passed, the screen will be cleared and the marquee will start displaying again from number 1.

To disable the marquee function:

```
OPOSLineDisplay1.MarqueeType = DISP_MT_NONE
```

It is possible for the marquee to be set for multiple windows at the same time. However, the field position of the windows can affect the way that the marquee is displayed. When displaying the marquee in multiple windows at the same time, please confirm that there are no conflicts between each window field.

4.8 Testing with the CheckHealth Method

Line displays support the CheckHealth method's Level 1, 2 and 3 setting. The tests can be used to ensure the correct connection of the device. For more details, please refer to the "EPSON OPOS ADK MANUAL APPLICATION DEVELOPMENT GUIDE LineDisplay (DM-D1xx/ DM-D2xx)".

4.9 Setting the Glyph Character Definition

The Glyph character is a special character defined by the user. Only one Glyph character can be defined in each execution of the method. The character data to define the Glyph character expresses the dot which is turned on and off in the display area as a bit unit, and specifies the obtained value.

The DefineGlyph method is used to define the Glyph character. Specify the defined character code and the defined character data. After they are defined, the Glyph character can be displayed by using the DisplayText method and the DisplayTextAt method.

The Glyph character definition function is effective only on the display where the function is provided. The presence of the function can be confirmed with the CapCustomGlyph property. In the definition of the Glyph character, three related properties are involved. One is the CustomGlyphList property. This property shows the character code that can be defined by the DefineGlyph method. The other two properties are GlyphHeight and GlyphWidth, which show the size of the Glyph character. The values of these three properties are different depending on the display. Refer to the "EPSON OPOS ADK MANUAL APPLICATION DEVELOPMENT GUIDE LineDisplay (DM-D1xx/ DM-D2xx)" for details.

Follow the procedure below to use the Glyph definition.

- 1) Check the Glyph definition function. (CapCustomGlyph).
- 2) Check the character code in which Glyph can be defined. (CustomGlyphList).

- 3) Check the size of the character in which Glyph can be defined. (GlyphHeight & GlyphWidth).
- 4) Define the Glyph character using the DefineGlyph method, with consideration of CustomGlyphList, GlyphHeight, and GlyphWidth (explained in the description part of the program).
- 5) To display the defined character, use the DisplayText method and the DisplayTextAt method.

The definition program of the Glyph character is described as follows:

[When the Glyph character in the following figure is defined with: GlyheHeight = 7, GlyphWidth = 5, character code 49 ("1")]

The format of the defined character data is as follows:

Bit position	7	6	5	4	3	2	1	0	Binary	Hexadecimal
GlyphHeight (7)					1				00001000	08
						1			00000100	04
				1			1		00010010	12
					1			1	00001001	09
						1			00000100	04
							1		00000010	02
								1	00000001	01
GlyphWidth (5)										

The displayed data is expressed in binary value, where the dot that turns on is 1 and the dot that turns off is 0. After this value is converted into a form corresponding to BinaryConversion, specify for the DefineGlyph method and define the Glyph character. The following program is an example where BinaryConversion= OPOS_BC_NIBBLE.

```
OPOSLineDisplay1.BinaryConversion = OPOS_BC_NIBBLE
OPOSLineDisplay1.DefineGlyph(49, 08041209040201)
```

Afterwards, if you want to display the defined character, add the following line.

```
OPOSLineDisplay1.DisplayText("1", DISP_MT_NONE)
```

* Notes

The Glyph character registered by the DefineGlyph method does not release the Glyph character as long as the Close method is not called. When specifying a character code, use a code that does not cause trouble to the normal display.

Section 5. MICR

Programming examples of how to use API functions relating to MICR devices are shown below.

5.1 Form Insertion/Removal

If a MICR device is going to be used with the system, there are methods to allow check insertion into and removal from the device.

The BeginInsertion only checks the state of the MICR, because the MICR's jaw is always open. When the EndInsertion method is used to insert a check into the MICR, the MICR is placed in a state of readiness and awaits the check insertion.

When the BeginRemoval method is used to remove a check from the MICR, the MICR is again placed in a state of readiness and awaits the check removal. Using the EndRemoval method will cause the MICR to close this state of removal.

The BeginInsertion and BeginRemoval methods both use a parameter to pass the time the MICR is to wait for check insertion/removal before issuing a time-out which will close the ready state. If the check is not inserted or removed within the time specified, the method returns an OPOS_E_TIMEOUT result code to the application.

If methods are used on a MICR that is an internal part of a POSPrinter and the printer is in a BUSY state (sending or receiving data, etc.), then the methods will return an OPOS_E_BUSY result code to the application. If OPOS_E_BUSY is returned, the application must wait until the printer is no longer busy or continue trying until the method is successful.

The following is an example of how to read check data by inserting/removing the check from the MICR.

```
Dim RC As Long
RC = OPOSMICR1.BeginInsertion (5000)
If RC = OPOS_SUCCESS Then
    RC = OPOSMICR1.EndInsertion
    If RC = OPOS_SUCCESS Then
        RC = OPOSMICR1.BeginRemoval (5000)
        If RC = OPOS_SUCCENSS Then
            RC = OPOSMICR1.EndRemoval
        Else
            'Error
        End If
    Else
        'Error
    End If
Else
    'Error
End If

OPOSMICR1.DataEventEnabled = TRUE
```

When the above code is run, the methods await the check data's DataEvent. When the

EndInsertion method is actually run, in order to receive data, a data event is fired to the device by check insertion.

5.2 Reading Data from the MICR

When data comes from the MICR, a DataEvent is fired to inform the application. DataEvent management requires the use of the MICR's properties to read the data from a check. Once a DataEvent is fired, the DataEventEnabled property is set to FALSE, so the application will not be able to receive any further data. After the data from the first DataEvent has been managed or when it is necessary to read data from another check, it is necessary to set the DataEventEnabled property to TRUE. If there is any data that has been spooled while the property was FALSE, a DataEvent is immediately fired and the data is sent to the program. If there is no data waiting, no DataEvent is fired.

Information received from a check's data is separated and placed into corresponding properties. Before a DataEvent is fired, the control's properties will contain either the previous check's data, or if no check has been read, the properties will contain empty strings. The following is a list of the properties used to store a check's data. These properties can be used to manage the data needed from a check.

RawData	Data read from the MICR is stored with no changes. MICR specific characters will be re-written into substitution characters. (For an explanation of these characters, please consult UPOS.)
AccountNumber	The account number written on the check.
Amount	The amount of the check.
BankNumber	The bank's number.
EPC	The check's EPC.
SerialNumber	The check's serial number.
TransmitNumber	The check's transmit Number.
CheckType	The check's check type.
CountryCode	The check's country code.

Below is an example of a program that manages check data received from a MICR device using the DataEvent.

```
Global GANData As String
Global GAmData As String
Global GCData As String
Global GBNDData As String
```

```
Private Sub OPOSMICR1_DataEvent(ByVal Status As Long)
GANData = OPOSMICR1.AccountNumber
GBNDData = OPOSMICR1.BankNumber
GAmData = OPOSMICR1.Amount
GCData = OPOSMICR1.CountryCode
OPOSMICR1.DataEventEnabled = True
End Sub
```

5.3 Error Management

If any error occurs on the device while check data is being read, the application is informed by an event. By checking the ResultCode parameter and the ResultCodeExtended parameter, the error can be checked and handled. When this event occurs, ResultCode is passed the value OPOS_E_FAILURE. If the data is not valid after an error, the ErrorLocus property is passed an OPOS_EL_INPUT value, and if the data is valid, an OPOS_EL_INPUT_DATA value is passed. OPOS_EL_INPUT is sent to ErrorLocus and OPOS_EMICR_COMPORT is sent to ResultCodeExtended when there is a port's error during data transmission, and OPOS_EL_INPUT_DATA is sent to ErrorLocus and OPOS_EMICR_DATAERROR is sent to ResultCodeExtended when there is a problem with the data format. All data is sent after an OPOS_EMICR_DATAERROR DataEvent is fired. After the data in which the error occurred has been managed, the ResultCodeExtended parameter is given the value OPOS_EMICR_DATAEND, and an ErrorEvent containing the value OPOS_EL_INPUT is sent to the ErrorLocus parameter.

This shows that some of the data passed by the DataEvent between the time that the OPOS_EMICR_DATAERROR and OPOS_EMICR_DATAEND occurred has an error in it. It is not possible to determine what data contains the error. However, even if the device says the data is valid after an error occurs, it is recommended that the check data be read in again. By using the ErrorLocus parameter, errors can be managed. Depending on what value is in the ErrorLocus parameter, the program can choose either OPOS_ER_CLEAR (clear the data) or OPOS_ER_CONTINUEINPUT (continue using the data). Placing one of these values in the pErrorResponse parameter will allow the program to handle the error. When the ErrorLocus parameter is set to OPOS_EL_INPUT, pErrorResponse is set to OPOS_ER_CLEAR as a default, and if ErrorLocus is set to OPOS_EL_INPUT_DATA, pErrorResponse is set to OPOS_ER_CONTINUEINPUT. When OPOS_EL_INPUT is the set value, OPOS_ER_CONTINUEINPUT cannot be specified.

```
Private Sub OPOSMICR1_ErrorEvent(ByVal ResultCode As Long,
ByVal ResultCodeExtended As Long,
ByVal ErrorLocus As Long
pErrorResponse As Long)
'Consult ResultCode, then manage the error.
If ErrorLocus = OPOS_EL_INPUT_DATA Then 'Data is invalid
pErrorResponse = OPOS_ER_CLEAR 'Clear invalid data
End If
End Sub
```

The reason for this is, for example, when data is entered in the data input buffer and an error event occurs, if OPOS_ER_CLEAR is used, the data that has been enter into the buffer is cleared. If OPOS_ER_CONTINUEINPUT is used, the data remains in the buffer for use by the program. The data in the buffer will follow the directions of the ErrorLocus property value.

The reasons for and types of errors that cause an ErrorEvent to be fired are listed below.

ResultCode	Reason
OPOS_E_FAILURE	The MICR data was not received properly.
OPOS_E_FAILURE	There is a problem with the device or its connection, and an error is originated from the port.

If an OPOS_E_FAILURE occurs, please confirm the MICR's settings and connections.

5.4 Testing with the CheckHealth Method

MICR devices support the CheckHealth method's Level 1 and Level 3 setting. The tests can be used to ensure the correct connection of the device. For more details, please refer to the "EPSON OPOS ADK MANUAL APPLICATION DEVELOPMENT GUIDE MICR (TM Series)".

Section 6. Cash Drawers

Programming examples of how to use API functions relating to a Cash Drawer are shown below.

6.1 Drawer Open/Close

The cash drawer is opened by using the `OpenDrawer` method. The `DrawerOpened` property can be used to check the current state of the drawer. To pause the program until the drawer is closed, the `WaitForDrawerClose` method is used. When using this method, the computer makes a sound that starts when the drawer is opened or after a specified amount of time, and lasts until the drawer is closed or for a specified time. The amount of time is passed to the method by a parameter. The frequency of the beep can also be controlled by parameters. Until the drawer is closed, the `WaitForDrawerClose` method will not return control of the application. The beep frequency and time settings depend on the Windows API beep functions. When setting the beep functions parameters in Windows 2000, Windows XP or Windows Vista, follow the specified settings. The `CashDrawer` control is not able to tell when the drawer is shut while a beep is sounding. Please use the `BeepDuration` parameter to make the beep sound as short as possible for the situation.

The following is an example of a program to open and close a cash drawer.

```
If Not OPOSCashdrawer1.DrawerOpened Then
    OPOSCashdrawer1.OpenDrawer
    OPOSCashdrawer1.WaitForDrawerClose 10000, 1000, 100, 100
End If
```

6.2 Checking Drawer Status

There are methods other than explained in section 2.4.1 that can be used to check drawer status. A `StatusUpdateEvent` is fired whenever the drawer is opened or closed.

The following are examples of programs that use events to check if the drawer has been opened or closed.

[Main Program]

```
Global DrawerFlag As Boolean
If Not OPOSCashdrawer1.DrawerOpened Then
    DrawerFlag = True
    OPOSCashdrawer1.OpenDrawer
    While (DrawerFlag = True) 'This part uses timer management to
        DoEvents
    Wend
    'check the status of DrawerFlag.
End If
```

[Event Management]

```
Private Sub OPOSCashdrawer1_StatusUpdateEvent(ByVal Data As Long)
    If Data = False Then
        DrawerFlag = False
    End If
End Sub
```

6.3 Testing with the CheckHealth Method

Cash Drawers support the CheckHealth method's Level 1, 2 and 3 setting. The tests can be used to ensure the correct connection of the device. For more details, please refer to the "EPSON OPOS ADK MANUAL APPLICATION DEVELOPMENT GUIDE CashDrawer".

6.4 Multi-drawer Configuration Support

Depending on a printer or terminal supports, the multi-drawer configuration connects more than one drawer independently via the same channel or hardware port. For more details about the multi-drawer configuration, please confirm the information on the connected printer or terminal.

6.4.1 Multi-drawers with One Status

When the CapMultiDrawerDetect property is set to False, the operation shows the multi-drawer with one status. The multiple drawers can be setting, but one status is shared. If either of the drawers is opened, the StatusUpdateEvent is fired indicating that the both drawers are opened. When either of the drawers is closed, nothing is changed. Only when the both drawers are closed, the status indicates that the drawers are closed.

6.4.2 Multi-drawers with Two Status

When the CapMultiDrawerDetect property is set to TRUE, the operation shows the multi-drawers with two statuses. Each drawer can have its status individually.

Section 7. CheckScanner

Programming examples of how to use API functions relating to CheckScanner devices are shown below. This device is Epson special.

7.1 Form Insertion/Removal

If a CheckScanner device is going to be used with the system, there are methods to allow check insertion into and removal from the device.

The BeginInsertion only check the state of the CheckScanner, because the CheckScanner's jaw is always open. When the EndInsertion method is used to insert a check into the CheckScanner, the CheckScanner is placed in a state of readiness and awaits the check's insertion. Then, the EndInsertion method actually reads a check.

After executing BeginInsertion and EndInsertion, and then reading a check, you can handle data using RetrievalImage, RetrieveMemory, and StoreImage.

When the BeginRemoval method is used to remove a check from the CheckScanner, the CheckScanner is again placed in a state of readiness and awaits the check's removal. Using the EndRemoval method will cause the CheckScanner to close this state of removal.

The BeginInsertion and BeginRemoval methods both use a parameter to pass the time the CheckScanner is to wait for check insertion/removal before issuing a time out, which will close the readied state. If the check is not inserted or removed within the time specified, the method returns an OPOS_E_TIMEOUT result code to the application.

If methods are used on a CheckScanner that is an internal part of a POS printer and the printer is in a BUSY state (sending or receiving data, etc.), then the methods will return an OPOS_E_BUSY result code to the application. If OPOS_E_BUSY is returned, the application must wait until the printer is no longer busy or continue trying until the method is successful.

The following is an example of how to read check data by inserting/removing the check from the CheckScanner.

```
Dim RC As Long
RC = OPOSCheckScanner1.BeginInsertion (5000)
If RC = OPOS_SUCCESS Then
    RC = OPOSCheckScanner1.EndInsertion
    If RC = OPOS_SUCCESS Then
        RC = OPOSCheckScanner1.RetrievalImage(ALL)
        If RC = OPOS_SUCCENSS Then
            'Success
        Else
            'Error
        End If
        RC = OPOSCheckScanner1.BeginRemoval (5000)
        RC = OPOSCheckScanner1.EndRemoval
    Else
        'Error
    End If
```

```
Else
    'Error
End If
```

```
OPOSCheckScanner1.DataEventEnabled = TRUE
```

When the above code is run, the methods await the check data's DataEvent. DataEvent is issued when data is set to the property, that is, after you execute RetrieveImage or RetrieveMemory. Since the EndInsertion method only reads a check and does not set the property, DataEvent is not issued.

7.2 Reading Data from the CheckScanner

When the scanned data is set to the property, a DataEvent is fired to inform the application. DataEvent management requires the use of the CheckScanner's properties to read the data from a check. Once a DataEvent is fired, the DataEventEnabled property is set to FALSE, so the application will not be able to receive any further data. After the data from the first DataEvent has been managed or when it is necessary to read data from another check, it is necessary to set the DataEventEnabled property to TRUE. If there is any data that has been spooled while the property was FALSE, a DataEvent is immediately fired and the data is sent to the program. If there is no data waiting, no DataEvent is fired.

Information received from a check's data is placed in the properties. The read data is stored in the ImageData property. In this case, the storage format varies according to the settings of the ImageFormat property.

Before a DataEvent is fired, the control's properties will contain either the previous check's data, or if no check has been read, the properties will contain empty strings. The following is a list of the properties used to store a check's data.

The following is an example of a program that manages check data received from a CheckScanner device using the DataEvent.

[Main Program]

```
Global DataReadFlag As Boolean
DataReadFlag = False
RC = OPOSCheckScanner1.RetrieveImage(ENTIRE_IMAGE)
While (DataReadFlag = False)    'Wait data read end
DoEvents
Wend
'ImageData Read
```

[Event management]

```
Private Sub OPOSCheckScanner1_DataEvent(ByVal Status As Long)
DataReadFlag = True
End Sub
```

7.3 Saving/Reading/Deleting Scanned Data

CheckScanner can save, read, and delete scanned data. Saved data is not deleted although you call the Close method. Be sure that data is not deleted unless you call the ClearImage method.

The following code shows how to save scanned data.

```
Dim IRet As Long
Dim IFlag As Long

'Sets FileID
OPOSCheckScanner1.FileID = "ID1"
OPOSCheckScanner1.ImageTagData = "Tag1"
'Saves the storage data
IFlag = ECHK_CROP_AREA_ENTIRE_IMAGE
IRet = OPOSCheckScanner1.StoreImage(IFlag)
If IRet = OPOS_SUCCENSS Then
    'Success
Else
    'Error
End If
```

The code saves data using values set in the FileID and ImageTagData properties. The FileIndex property is automatically allocated when data is saved successfully. You cannot save it by specifying the FileID and ImageTagData properties corresponding to the data already saved.

The following code shows how to read the saved data. (It uses ImageTagData and reads the data.)

```
Dim IRet As Long
Dim IFlag As Long

'Sets ImageTagData
OPOSCheckScanner1.ImageTagData = "Tag1"
'Reads the storage data
IFlag = ECHK_LOCATE_BY_IMAGETAGDATA
IRet = OPOSCheckScanner1.RetrieveMemory(IFlag)
If IRet = OPOS_SUCCENSS Then
    'Success
Else
    'Error
End If
```

After the RetrieveMemory method is executed and the data read, information about the read data is set to the FileID, FileIndex, ImageTagData, and ImageData properties.

The following code shows how to delete the saved data. (It uses FileIndex and reads the data.)

```
Dim IRet As Long
Dim IFlag As Long

'Sets FileIndex
OPOSCheckScanner1.FileIndex = 1
'Deletes the storage data
IFlag = ECHK_LOCATE_BY_FILEINDEX
IRet = OPOSCheckScanner1.ClearImage(IFlag)
If IRet = OPOS_SUCCENSS Then
    'Success
Else
```

‘Error

End If

If you want to delete all saved data, set ECHK_CLR_ALL to the ClearImage parameter and then call it.

7.4 Error Management

If an error occurs on the device while check data is being read, the application is informed by an event. By checking the ResultCode parameter and the ResultCodeExtended parameter, the error can be checked and handled. When this event occurs, ResultCode is passed the value OPOS_E_FAILURE. If the data is not valid after an error, the ErrorLocus property is passed an OPOS_EL_INPUT value, and if the data is valid, an OPOS_EL_INPUT_DATA value is passed. OPOS_EL_INPUT_DATA is sent to ErrorLocus and OPOS_ECHK_DATAERROR is sent to ResultCodeExtended when there is a problem with the data format. All data is sent after an OPOS_ECHK_DATAERROR DataEvent is fired. After the data in which the error occurred has been managed, the ResultCodeExtended parameter is given the value OPOS_ECHK_DATAEND, and an ErrorEvent containing the value OPOS_EL_INPUT is sent to the ErrorLocus parameter.

This shows that some of the data passed by the DataEvent between the time that the OPOS_ECHK_DATAERROR and OPOS_ECHK_DATAEND occurred has an error in it. It is not possible to determine what data contains the error. However, even if the device says the data is valid after an error occurs, it is recommended that the check data be read in again. By using the ErrorLocus parameter, errors can be managed. Depending on what value is in the ErrorLocus parameter, the program can choose either OPOS_ER_CLEAR (clear the data) or OPOS_ER_CONTINUEINPUT (continue using the data). Placing one of these values in the pErrorResponse parameter will allow the program to handle the error. When the ErrorLocus parameter is set to OPOS_EL_INPUT, pErrorResponse is set to OPOS_ER_CLEAR as a default, and if ErrorLocus is set to OPOS_EL_INPUT_DATA, pErrorResponse is set to OPOS_ER_CONTINUEINPUT. When OPOS_EL_INPUT is the set value, OPOS_ER_CONTINUEINPUT cannot be specified.

```
Private Sub OPOSCheckScanner1_ErrorEvent(ByVal ResultCode As Long,
ByVal ResultCodeExtended As Long,
ByVal ErrorLocus As Long
pErrorResponse As Long)
‘Consult ResultCode, then manage the error.
If ErrorLocus = OPOS_EL_INPUT_DATA Then ‘Data is invalid
pErrorResponse = OPOS_ER_CLEAR ‘Clear invalid data
End If
End Sub
```

For example, if OPOS_ER_CLEAR is specified upon occurrence of an error, all the stored data in an input buffer that are read from MSR are cleared. If OPOS_ER_CONTINUEINPUT is specified, the data are kept in the input buffer. However, the validity of the data with error is decided by ErrorLocus value.

The reasons for and types of errors that cause an ErrorEvent to be fired are listed below.

ResultCode	Reason
OPOS_E_FAILURE	The Check data was not received properly.
OPOS_E_FAILURE	There is a problem with the device or its connection, and an error is originated from the port.

If an OPOS_E_FAILURE occurs, please confirm the CheckScanner's settings and connections.

7.5 Testing with the CheckHealth Method

CheckScanner devices support the CheckHealth method's Level 1 and Level 3 setting. The tests can be used to ensure the correct connection of the device. For more details, please refer to the "EPSON OPOS ADK MANUAL APPLICATION DEVELOPMENT GUIDE CheckScanner".

Section 8. Electronic Journal

Programming examples of how to use API functions relating to ElectronicJournal devices are shown below. This device is Epson special.

8.1 Writing Electronic Journal data

When the property is set to “true”, ServiceObject of Electronic Journal enables output data of POSPrinter writeable. Even POSPrinter ServiceObject is not printable, the property can be set to “true”. When initialization of Electronic Journal file is not completed, writing output data of POSPrinter will be error and error event queuing.

The printing data are stored in HDD that for saving the files through the SetupPOS setting specified. When a free disk size reaches to a set value, StatusUpdateEvent will be notified. When a process for writing output data from the printer is failed StatusUpdateEvent will be notified.

8.2 Marker setting

The printing data are stored with contiguous data.

When the write data is acquired and printed, the marker are specified to indicate the data range as an index. Several markers can be added on the same place.

Strings can be specified as a marker name is as follows:

- Strings exclude ASCII code (0x00-0x1F and 0x7F)
- Maximum string size: 1024 characters

The null character cannot be used for marker.

(Example) Acquired data range: 1 day's data
Start Marker: The day's work starting time
End Marker: The day's work closing time

(Example) Acquired data range: 1 receipt's data
Start Marker: Before the appropriate receipt printing start
End Marker: After the appropriate receipt printing end

When the Printer's print method executed non-simultaneously is printed completed, at the time the ElectronicJournal data is stored. Therefore, add the marker after the Printer's OutputCompleteEvent is notified.

8.3 Specifications for specified range

The process becomes successful when a specified range is empty. It is the same as when process of the empty printing data.

QueryContent method:	The empty extract file is created.
PrintContentFile method:	Ends without printing any data.

8.4 Non-simultaneously printing of ElectronicJournal data

When AsyncMode property is set to “true”, the ElectronicJournal data is executed with non-

simultaneous print. When non-simultaneous print of Electronic Journal data is completed, the method notifies OutputCompleteEvent. When one of the processes is failed, ErrorEvent is notified.

The suspend mode is available at non-simultaneously printing.

When suspend mode is set, all the non-simultaneous output processes will be suspended. And restarts printing by the ResumePrintContent method.

When suspend mode is set, all the non-simultaneous output processes will be suspended.

Suspend mode is for posing printing process when event error is issued during non-simultaneous outputs, and executable process is limited during the time.

<When the method is executed while State property is OPOS_S_ERROR>

A command is sent to clear data in the device buffer in case there is any because printing process is suspended due to error.

After printing is restarted, transaction being sent when the SuspendPrintContent method is executed will also be restarted.

<When the method is executed while State property is OPOS_S_BUSY>

When Suspended property is set to "true", SUCCESS is returned.

After transaction being sent when the method is executed is completed, non-simultaneous print process will be stopped.

When print error occurs after the SuspendPrintContent method is executed, a result will be the same as when the State property is executed while OPOS_S_ERROR occurs.

When printing is suspended correctly, output data will be printed after a transaction being sent when the SuspendPrintContent method is executed.