

# Java for Retail POS

## Programming Guide

Version 1.5

September 24, 2000

International Standard

For Implementation of POS Peripherals on a Java Based System

**Java for Retail POS Committee Members:**

**Epson, Fujitsu-ICL, Home Depot, IBM, JCPenney,  
MGV, NCR, PCMS Datafit, Research Computer Services,  
Sears Roebuck and Co., Sun Microsystems, Telxon, Wincor Nixdorf**

## **Java for Retail POS**

© Java for Retail POS Committee

Programmer's Guide

Information in this document is subject to change without notice.

JavaPOS is a trademark of Sun Microsystems, Inc.

# Table of Contents

## INTRODUCTION AND ARCHITECTURE

JAVA FOR RETAIL POS .....	1
<b>WHAT IS JAVA FOR RETAIL POS? .....</b>	<b>1</b>
BENEFITS .....	2
DEPENDENCIES .....	2
RELATIONSHIP TO OPOS .....	2
WHO SHOULD READ THIS DOCUMENT .....	3
CHAPTER OVERVIEW .....	4
<b>ARCHITECTURAL OVERVIEW .....</b>	<b>5</b>
ARCHITECTURAL COMPONENTS .....	6
<b>DEVICE BEHAVIOR MODELS .....</b>	<b>9</b>
INTRODUCTION TO PROPERTIES, METHODS, AND EVENTS .....	9
DEVICE INITIALIZATION AND FINALIZATION .....	10
<i>Initialization</i> .....	10
<i>Finalization</i> .....	10
<i>Summary</i> .....	11
DEVICE SHARING MODEL .....	12
<i>Exclusive-Use Devices</i> .....	13
<i>Sharable Devices</i> .....	13
DATA TYPES .....	14
EXCEPTIONS .....	15
<i>ErrorCode</i> .....	16
<i>ErrorCodeExtended</i> .....	17
EVENTS.....	18
<i>Registering for Events</i> .....	20
<i>Event Delivery</i> .....	20
DEVICE INPUT MODEL .....	22
DEVICE OUTPUT MODELS .....	25
<i>Synchronous Output</i> .....	25
<i>Asynchronous Output</i> .....	25
DEVICE POWER REPORTING MODEL.....	27
<i>Model</i> .....	27
<i>Properties</i> .....	28
<i>Power Reporting Requirements for DeviceEnabled</i> .....	29
DEVICE STATES.....	30
THREADS .....	31
VERSION HANDLING .....	32

<b>CLASSES AND INTERFACES .....</b>	<b>33</b>
SYNOPSIS .....	33
<i>Application</i> .....	33
<i>Device Control</i> .....	34
<i>Device Service</i> .....	34
<i>Helper Classes</i> .....	35
SAMPLE CLASS AND INTERFACE HIERARCHIES .....	36
<i>Application</i> .....	36
<i>Device Controls</i> .....	36
<i>Device Service</i> .....	37
SAMPLE APPLICATION CODE .....	39
PACKAGE STRUCTURE .....	40
<i>jpos</i> .....	40
<i>jpos.events</i> .....	41
<i>jpos.services</i> .....	41
<b>DEVICE CONTROLS .....</b>	<b>43</b>
DEVICE CONTROL RESPONSIBILITIES .....	43
DEVICE SERVICE MANAGEMENT .....	44
<i>jpos.config/loader (JCL)</i> <i>and JavaPOS Entry Registry (JER)</i> .....	44
<i>jpos.config/loader Characteristics</i> .....	44
PROPERTY AND METHOD FORWARDING .....	46
EVENT HANDLING .....	47
<i>Event Listeners and Event Delivery</i> .....	47
<i>Event Callbacks</i> .....	48
VERSION HANDLING .....	49
<b>DEVICE SERVICES .....</b>	<b>51</b>
DEVICE SERVICE RESPONSIBILITIES .....	51
PROPERTY AND METHOD PROCESSING .....	51
EVENT GENERATION .....	52
PHYSICAL DEVICE ACCESS .....	52
<b>CHAPTER 1</b>	
<b>COMMON PROPERTIES, METHODS, AND EVENTS .....</b>	<b>53</b>
SUMMARY .....	53
GENERAL INFORMATION .....	55
PROPERTIES .....	56
METHODS .....	68
EVENTS .....	74
<b>CHAPTER 2</b>	
<b>BUMP BAR .....</b>	<b>79</b>
SUMMARY .....	79
GENERAL INFORMATION .....	83
PROPERTIES .....	88
METHODS .....	94
EVENTS .....	99

---

<b>CHAPTER 3</b>	
<b>CASH CHANGER .....</b>	<b>105</b>
SUMMARY .....	105
GENERAL INFORMATION .....	108
PROPERTIES .....	113
METHODS .....	125
EVENTS .....	132
<b>CHAPTER 4</b>	
<b>CASH DRAWER .....</b>	<b>135</b>
SUMMARY .....	135
GENERAL INFORMATION .....	137
PROPERTIES .....	138
METHODS .....	140
EVENTS .....	141
<b>CHAPTER 5</b>	
<b>CAT-CREDIT AUTHORIZATION TERMINAL .....</b>	<b>143</b>
SUMMARY .....	143
GENERAL INFORMATION .....	146
PROPERTIES .....	153
METHODS .....	174
EVENTS .....	182
<b>CHAPTER 6</b>	
<b>COIN DISPENSER .....</b>	<b>187</b>
SUMMARY .....	187
GENERAL INFORMATION .....	190
PROPERTIES .....	191
METHODS .....	193
EVENTS .....	194
<b>CHAPTER 7</b>	
<b>FISCAL PRINTER .....</b>	<b>197</b>
SUMMARY .....	197
GENERAL INFORMATION .....	205
PROPERTIES .....	217
METHODS .....	242
EVENTS .....	296
<b>CHAPTER 8</b>	
<b>HARD TOTALS .....</b>	<b>303</b>
SUMMARY .....	303
GENERAL INFORMATION .....	307
PROPERTIES .....	311
METHODS .....	314
EVENTS .....	325

<b>CHAPTER 9</b>	
<b>KEYLOCK .....</b>	<b>327</b>
SUMMARY .....	327
GENERAL INFORMATION .....	329
PROPERTIES .....	330
METHODS .....	331
EVENTS.....	332
<b>CHAPTER 10</b>	
<b>LINE DISPLAY .....</b>	<b>335</b>
SUMMARY .....	335
GENERAL INFORMATION .....	338
PROPERTIES .....	341
METHODS .....	358
EVENTS.....	369
<b>CHAPTER 11</b>	
<b>MICR – MAGNETIC INK CHARACTER RECOGNITION</b>	
<b>READER.....</b>	<b>371</b>
SUMMARY .....	371
GENERAL INFORMATION .....	374
<i>MICR Character Substitution</i> .....	377
PROPERTIES .....	378
METHODS .....	383
EVENTS.....	387
<b>CHAPTER 12</b>	
<b>MSR – MAGNETIC STRIPE READER.....</b>	<b>391</b>
SUMMARY .....	391
GENERAL INFORMATION .....	394
PROPERTIES .....	396
EVENTS.....	409
<b>CHAPTER 13</b>	
<b>PINPAD .....</b>	<b>415</b>
SUMMARY .....	415
GENERAL INFORMATION .....	419
PROPERTIES .....	422
METHODS .....	436
EVENTS .....	440
<b>CHAPTER 14</b>	
<b>POINTCARD READER WRITER.....</b>	<b>443</b>
SUMMARY .....	443
GENERAL INFORMATION .....	447
PROPERTIES .....	455
METHODS .....	474
EVENTS.....	482

---

<b>CHAPTER 15</b>	
<b>POS KEYBOARD.....</b>	<b>487</b>
SUMMARY .....	487
GENERAL INFORMATION .....	489
PROPERTIES .....	490
EVENTS.....	492
<b>CHAPTER 16</b>	
<b>POS POWER.....</b>	<b>495</b>
SUMMARY .....	495
GENERAL INFORMATION .....	498
PROPERTIES .....	500
METHODS .....	504
EVENTS.....	505
<b>CHAPTER 17</b>	
<b>POS PRINTER.....</b>	<b>507</b>
SUMMARY .....	507
GENERAL INFORMATION .....	513
PROPERTIES .....	522
METHODS .....	572
EVENTS.....	609
<b>CHAPTER 18</b>	
<b>REMOTE ORDER DISPLAY.....</b>	<b>615</b>
SUMMARY .....	615
GENERAL INFORMATION .....	619
PROPERTIES .....	625
METHODS .....	636
EVENTS.....	657
<b>CHAPTER 19</b>	
<b>SCALE .....</b>	<b>661</b>
SUMMARY .....	661
GENERAL INFORMATION .....	664
PROPERTIES .....	666
METHODS .....	671
EVENTS.....	674
<b>CHAPTER 20</b>	
<b>SCANNER (BAR CODE READER).....</b>	<b>677</b>
SUMMARY .....	677
GENERAL INFORMATION .....	679
PROPERTIES .....	680
EVENTS.....	685

<b>CHAPTER 21</b>	
<b>SIGNATURE CAPTURE .....</b>	<b>689</b>
SUMMARY .....	689
GENERAL INFORMATION .....	691
PROPERTIES .....	693
METHODS .....	697
EVENTS.....	699
<b>CHAPTER 22</b>	
<b>TONE INDICATOR.....</b>	<b>703</b>
SUMMARY .....	703
GENERAL INFORMATION .....	706
PROPERTIES .....	708
METHODS .....	712
EVENTS.....	714
<b>APPENDIX A</b>	
<b>CHANGE HISTORY .....</b>	<b>A-1</b>
RELEASE 1.3.....	A-1
RELEASE 1.4.....	A-3
RELEASE 1.5.....	A-4
<b>APPENDIX B</b>	
<b>OPOS AND JAVAPOS .....</b>	<b>B-1</b>
API MAPPING RULES .....	B-1
<i>Data Types</i> .....	<i>B-2</i>
<i>Property &amp; Method Names</i> .....	<i>B-2</i>
<i>Events</i> .....	<i>B-3</i>
<i>Constants</i> .....	<i>B-3</i>
API DEVIATIONS.....	B-4
FUTURE VERSIONS .....	B-5



# Java for Retail POS

## What Is Java for Retail POS?

Java for Retail POS (or JavaPOS™) is a standard that defines:

- An architecture for Java-based POS (Point-Of-Service or Point-Of-Sale) device access.
- A set of POS device interfaces (APIs) sufficient to support a range of POS solutions.

The Java for Retail POS standards committee was formed by a collection of retail vendors and end users, with a primary goal of providing device interfaces for the retail applications written in Java.

The JavaPOS committee will produce the following:

- JavaPOS Programmer's Guide (this document).
- Java source files, including:
  - Definition files. Various interface and class files described in the standard.
  - jpos.config/loader (JCL), configuration and service loader example.
  - Example files. These will include a set of sample Device Control classes, to illustrate the interface presented to an application.

The JavaPOS committee will **not** provide the following:

- Complete software components. Hardware providers or third-party providers develop and distribute these components.
- Certification mechanism.

## Benefits

The benefits of JavaPOS include:

- The opportunity for reduced POS terminal costs, through the use of thinner clients.
- Platform-independent applications, where the application is separated from both hardware and operating system specifics.
- Reduced administration costs, because an application and supporting software may be maintained on a server and loaded on demand by Java.

## Dependencies

Deployment of JavaPOS depends upon the following software components:

- Java Communications Port API (COM/API) or optionally some other Java communications API that supports hardware device connectivity.
- `jpos.config/loader (JCL)`
- For more information concerning the availability and any other up-to-date information about these components, see <http://www.javapos.com/>.

## Relationship to OPOS

The OLE for Retail POS (OPOS) standards committee developed device interfaces for Win32-based terminals using ActiveX technologies. The OPOS standard was used as the starting point for JavaPOS, due to:

- **Similar purposes.** Both standards involve developing device interfaces for a segment of the software community.
- **Reuse of device models.** The majority of the OPOS documentation specifies the properties, methods, events, and constants used to model device behavior. These behaviors are in large part independent of programming language.
- **Reduced learning curve.** Many application and hardware vendors are already familiar with using and implementing the OPOS APIs.
- **Early deployment.** By sharing device models, JavaPOS “wrappers” or “bridges” may be built to migrate existing OPOS device software to JavaPOS.

Therefore, most of the OPOS APIs were mapped into the Java language. The general translation rules are given in the Appendix “OPOS and JavaPOS” on page B-1.

---

## Who Should Read This Document

The JavaPOS Programmer's Guide is targeted to both the application developer who will use JavaPOS Devices and the system developer who will write JavaPOS Devices.

This guide assumes that the application developer is familiar with the following:

- General characteristics of POS peripheral devices.
- Java terminology and architecture.
- A Java development environment, such as Javasoft's JDK, Sun's Java Workshop, IBM's VisualAge for Java, or others.

A system developer must understand the above, plus the following:

- The POS peripheral device to be supported.
- The host operating system, if the JavaPOS Device will require a specific operating system.
- A thorough knowledge of the JavaPOS models and the APIs of the device.

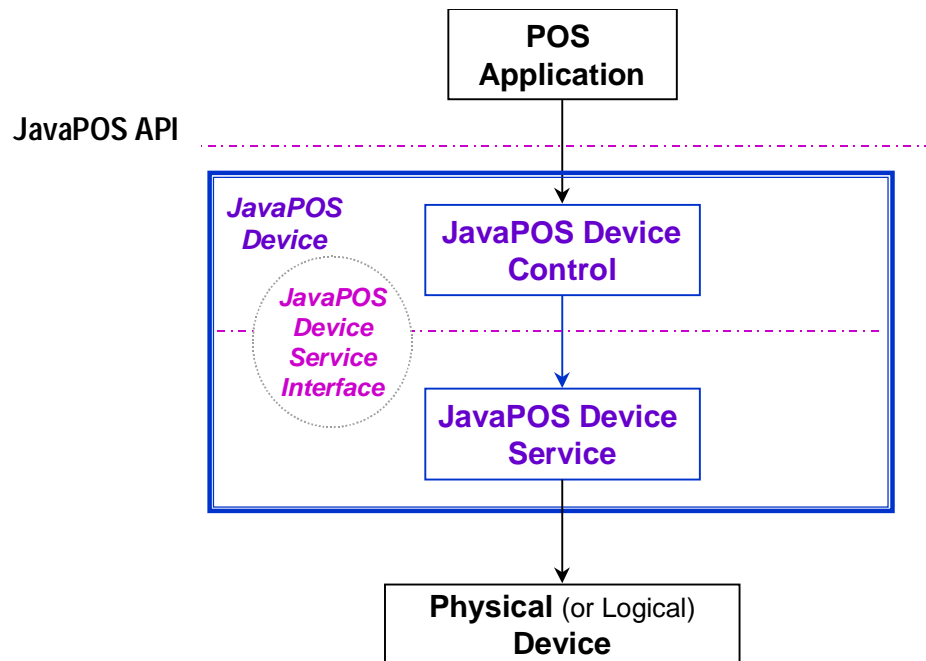
## Chapter Overview

This chapter contains the following major sections:

Section Name	Developer Audience
What Is “Java for Retail POS?”	App and System
Architectural Overview (page 5)	App and System
Device Behavior Models (page 9)	App and System
Classes and Interfaces (page 33)	App and System
Device Controls (page 43)	System
Device Services (page 53)	System

## Architectural Overview

JavaPOS defines a multi-layered architecture in which a POS Application interacts with the Physical or Logical Device through the JavaPOS Device.



## Architectural Components

The **POS Application** (or **Application**) is either a Java Application or applet that uses one or more JavaPOS Devices. An application accesses the JavaPOS Device through the **JavaPOS Device Interface**, which is specified by Java interfaces.

**JavaPOS Devices** are divided into categories called **Device Categories**, such as Cash Drawer and POS Printer.

Each JavaPOS Device is a combination of these components:

- **JavaPOS Device Control** (or **Device Control**) for a device category. The Device Control class provides the interface between the Application and the device category. It contains no graphical component and is therefore invisible at runtime, and conforms to the JavaBeans API.

The Device Control has been designed so that all implementations of a device category's control will be compatible. Therefore, the Device Control can be developed independently of a Device Service for the same device category (they can even be developed by different companies).

- **JavaPOS Device Service** (or **Device Service**), which is a Java class that is called by the Device Control through the **JavaPOS Device Service Interface** (or **Service Interface**). The Device Service is used by the Device Control to implement JavaPOS-prescribed functionality for a Physical Device. It can also call special event methods provided by the Device Control to deliver events to the Application.

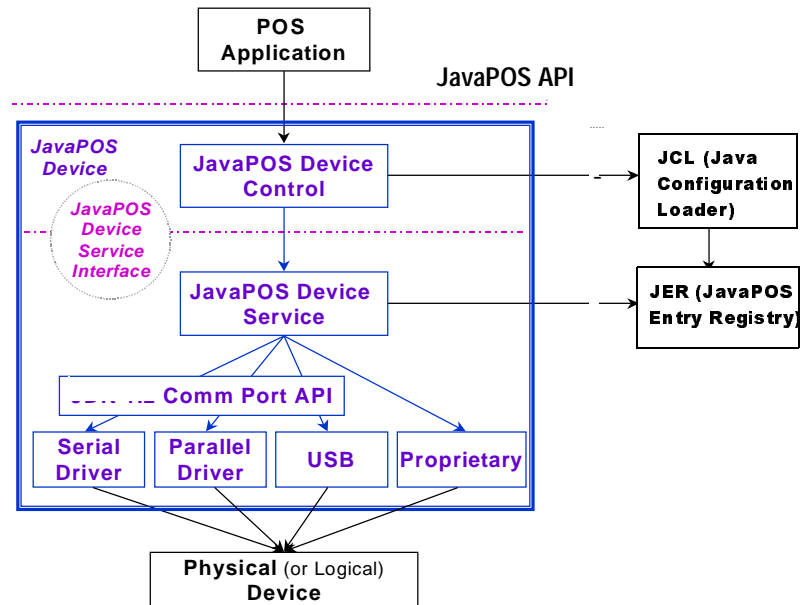
A set of Device Service classes can be implemented to support Physical Devices with multiple Device Categories.

The Application manipulates the **Physical Device** (the hardware unit or peripheral) by calling the JavaPOS Device APIs. Some Physical Devices support more than one device category. For example, some POS Printers include a Cash Drawer kickout, and some Bar Code Scanners include an integrated Scale. However with JavaPOS, an application treats each of these device categories as if it were an independent Physical Device. The JavaPOS Device writer is responsible for presenting the peripheral in this way.

**Note:** Occasionally, a Device may be implemented in software with no user-exposed hardware, in which case it is called a **Logical Device**.

## Additional Layers and APIs

The JavaPOS architecture contains additional layers and APIs in order to integrate well with the Java development environment.



*Note: Comm Port API refers to the Java Communications Port API (COM/API) or optionally some other Java communications API that supports hardware device connectivity.*

## JavaPOS Development Environment

JavaPOS will use these packages:

- **JavaPOS Configuration / Loader (JCL) *Added in Release 1.5***  
The `jpos.config/loader` (JCL) is a simple binding (configuration and loading) API which enables a JavaPOS control to bind to the correct JavaPOS service in a manner independent of the actual configuration mechanism. For POS applications, it represents a somewhat minimum (however, extensible) functional equivalent of the “NT Registry”, **JposEntryRegistry**. All JavaPOS Device Controls should use this API.
- **Communications Port API (for example, JavaComm v2.0 API)**, so that Applications can make standard access to devices that may use serial (RS-232), parallel, USB, and other future communication methods.



## Device Behavior Models

### Introduction to Properties, Methods, and Events

An application accesses a JavaPOS Device via the JavaPOS APIs.

The three elements of JavaPOS APIs are:

- **Properties.** Properties are device characteristics or settings. A type is associated with each property, such as **boolean** or **String**. An application may retrieve a property's value, and it may set a writable property's value. JavaPOS properties conform to the JavaBean property design pattern.

To read a property value, use the method:

*Type* **getSampleProperty()** throws **JposException**;

where *Type* is the data type of the property and *SampleProperty* is the property name.

To write a property value (assuming that the property is writable), use the method:

**void setSampleProperty(*Type value*)** throws **JposException**;

where *Type* is the data type of the property and *SampleProperty* is the property name.

- **Methods.** An application calls a method to perform or initiate some activity at a device. Some methods require parameters of specified types for sending and/or returning additional information.

A JavaPOS method has the form:

**void sampleMethod(*parameters*)** throws **JposException**;

where *sampleMethod* is the method name and *parameters* is a list of zero or more parameters.

Since JavaPOS uses Method names that are consistent with OPOS (See Appendix page B-1) some Methods may appear to be Property getters/setters (for example, **setDate** page 288 in Fiscal Printer). BeanInfo classes are used to properly describe the Properties and Methods to provide clarification so that various vendors builder tools will properly function.

- **Events.** A JavaPOS Device may call back into the application via events. The application must specifically register for each event type that it needs to receive. JavaPOS events conform to the JavaBean event design pattern.

See "Events" on page 18 for further details.

## Device Initialization and Finalization

### Initialization

The first actions that an application must take to use a JavaPOS Device are:

- Obtain a reference to a JavaPOS Device Control, either by creating a new instance or by accessing an existing one.
- Call Control methods to register for the events that the application needs to receive. (See “Events” on page 18.)

To initiate activity with the Physical Device, an application calls the Control’s **open** method:

**void open(String *logicalDeviceName*) throws JposException;**

The *logicalDeviceName* parameter specifies a logical device to associate with the JavaPOS Device. The **open** method performs the following steps:

1. Creates and initializes an instance of the proper Device Service class for the specified name.
2. Initializes many of the properties, including the descriptions and version numbers of the JavaPOS Device.

More than one instance of a Device Control may have a Physical Device open at the same time. Therefore, after the Device is opened, an application might need to call the **claim** method to gain exclusive access to it. Claiming the Device ensures that other Device instances do not interfere with the use of the Device. An application can **release** the Device to share it with another Device Control instance— for example, at the end of a transaction.

Before using the Device, an application must set the **DeviceEnabled** property to true. This value brings the Physical Device to an operational state, while false disables it. For example, if a Scanner JavaPOS Device is disabled, the Physical Device will be put into its non-operational state (when possible). Whether physically operational or not, any input is discarded until the JavaPOS Device is enabled.

### Finalization

After an application finishes using the Physical Device, it should call the **close** method. If the **DeviceEnabled** property is true, **close** disables the Device. If the **Claimed** property is true, **close** releases the claim.

Before exiting, an application should close all open JavaPOS Devices to free device resources in a timely manner, rather than relying on the Java garbage collection mechanism to free resources at some indeterminate time in the future.

## Summary

In general, an application follows this general sequence to open, use, and close a Device:

- Obtain a Device Control reference.
- Register for events (add listeners).
- Call the **open** method to instantiate a Device Service and link it to the Device Control.
- Call the **claim** method to gain exclusive access to the Physical Device. Required for exclusive-use Devices; optional for some sharable Devices. (See “Device Sharing Model” on page 12 for more information).
- Set the **DeviceEnabled** property to true to make the Physical Device operational. (For sharable Devices, the Device may be enabled without first **claiming** it.)
- Use the device.
- Set the **DeviceEnabled** property to false to disable the Physical Device.
- Call the **release** method to release exclusive access to the Physical Device.
- Call the **close** method to unlink the Device Service from the Device Control.
- Unregister from events (remove listeners).

## Device Sharing Model

JavaPOS Devices fall into two sharing categories:

- Devices that are to be used exclusively by one JavaPOS Device Control instance.
- Devices that may be partially or fully shared by multiple Device Control instances.

Any Physical Device may be open by more than one Device Control instance at a time. However, activities that an application can perform with a Device Control may be restricted to the Device Control instance that has claimed access to the Physical Device.

**Note:** Currently, device exclusivity and sharing can only be guaranteed within an application's Java Virtual Machine. This is because the Java language and environment does not directly support inter-virtual machine communication or synchronization mechanisms. At some time in the future, this restriction may be lifted. Until then, the sharing model will typically be of little benefit because a single application will seldom find value in opening a Physical Device through multiple Device Control instances.

## Exclusive-Use Devices

The most common device type is called an **exclusive-use device**. An example is the POS printer. Due to physical or operational characteristics, an exclusive-use device can only be used by one Device Control at a time. An application must call the Device's **claim** method to gain exclusive access to the Physical Device before most methods, properties, or events are legal. Until the Device is claimed and enabled, calling methods or accessing properties may cause a **JposException** with an error code of `JPOS_E_NOTCLAIMED`, `JPOS_E_CLAIMED`, or `JPOS_E_DISABLED`. No events are delivered until the Device is claimed.

An application may in effect share an exclusive-use device by calling the Device Control's **claim** method before a sequence of operations, and then calling the **release** method when the device is no longer needed. While the Physical Device is released, another Device Control instance can claim it.

When an application calls the **claim** method again (assuming it did not perform the sequence of **close** method followed by **open** method on the device), some settable device characteristics are restored to their condition at the **release**. Examples of restored characteristics are the line display's brightness, the MSR's tracks to read, and the printer's characters per line. However, state characteristics are not restored, such as the printer's sensor properties. Instead, these are updated to their current values.

## Sharable Devices

Some devices are "sharable devices." An example is the keylock. A sharable device allows multiple Device Control instances to call its methods and access its properties. Also, it may deliver its events to all Device Controls that have registered listeners. A sharable device may still limit access to some methods or properties to the Device Control that has claimed it, or it may deliver some events only to the Device Control that has claimed it.

## Data Types

JavaPOS uses the following data types:

Type	Usage
boolean	Boolean true or false.
boolean[1]	Modifiable boolean.
byte[]	Array of bytes. May be modified, but size of array cannot be changed.
int	32-bit integer.
int[1]	Modifiable 32-bit integer.
long	64-bit integer. Sometimes used for currency values, where 4 decimal places are implied. For example, if the integer is “1234567”, then the currency value is “123.4567”.
long[1]	Modifiable 64-bit integer.
String	Text character string.
String[1]	Modifiable text character string.
Point[]	Array of points. Used by Signature Capture.
Object	An object. This will usually be subclassed to provide a Device Service-specific parameter.

The convention of *type*[1] (an array of size 1) is used to pass a modifiable basic type. This is required since Java’s primitive types, such as **int** and **boolean**, are passed by value, and its primitive wrapper types, such as **Integer** and **Boolean**, do not support modification.

For strings and arrays, do not use a null value to report no information. Instead use an empty string (“”) or an empty array (zero length).

In some chapters, an integer may contain a “bit-wise mask”. That is, the integer data may be interpreted one or more bits at a time. The individual bits are numbered beginning with Bit 0 as the least significant bit.

## Exceptions

Every JavaPOS method and property accessor may throw a **JposException** upon failure, except for the properties **DeviceControlVersion**, **DeviceControlDescription**, and **State**. No other types of exceptions will be thrown.

**JposException** is in the package **jpos**, and extends **java.lang.Exception**. The constructor variations are:

```
public JposException(int errorCode);
public JposException(int errorCode, int errorCodeExtended);
public JposException(int errorCode, String description);
public JposException(int errorCode, int errorCodeExtended,
    String Description);
public JposException(int errorCode, String description,
    Exception origException);
public JposException(int errorCode, int errorCodeExtended,
    String description, Exception origException)
```

The parameters are:

Parameter	Description
<i>errorCode</i>	The JavaPOS error code. Access is through the <b>getErrorCode</b> method.
<i>errorCodeExtended</i>	May contain an extended error code. If not provided by the selected constructor, then is set to zero. Access is through the <b>getErrorCodeExtended</b> method.
<i>description</i>	A text description of the error. If not provided by the selected constructor, then one is formed from the <i>errorCode</i> and <i>errorCodeExtended</i> parameters. Access is through the superclass' methods <b>getMessage</b> or <b>toString</b> .
<i>origException</i>	Original exception. If the JavaPOS Device caught a non-JavaPOS exception, then an appropriate <i>errorCode</i> is selected and the original exception is referenced by this parameter. Otherwise, it is set to null. Access is through the <b>getOrigException</b> method.

## ErrorCode

This section lists the general meanings of the error code property of an **ErrorEvent** or a **JposException**. In general, the property and method descriptions in later chapters list error codes only when specific details or information are added to these general meanings.

The error code is set to one of the following values:

<b>Value</b>	<b>Meaning</b>
JPOS_E_CLOSED	An attempt was made to access a closed JavaPOS Device.
JPOS_E_CLAIMED	An attempt was made to access a Physical Device that is claimed by another Device Control instance. The other Control must release the Physical Device before this access may be made. For exclusive-use devices, the application will also need to claim the Physical Device before the access is legal.
JPOS_E_NOTCLAIMED	An attempt was made to access an exclusive-use device that must be claimed before the method or property set action can be used. If the Physical Device is already claimed by another Device Control instance, then the status JPOS_E_CLAIMED is returned instead.
JPOS_E_NOSERVICE	The Control cannot communicate with the Service, normally because of a setup or configuration error.
JPOS_E_DISABLED	Cannot perform this operation while the Device is disabled.
JPOS_E_ILLEGAL	An attempt was made to perform an illegal or unsupported operation with the Device, or an invalid parameter value was used.
JPOS_E_NOHARDWARE	The Physical Device is not connected to the system or is not powered on.
JPOS_E_OFFLINE	The Physical Device is off-line.
JPOS_E_NOEXIST	The file name (or other specified value) does not exist.
JPOS_E_EXISTS	The file name (or other specified value) already exists.
JPOS_E_FAILURE	The Device cannot perform the requested procedure, even though the Physical Device is connected to the system, powered on, and on-line.



---

JPOS_E_TIMEOUT	The Service timed out waiting for a response from the Physical Device, or the Control timed out waiting for a response from the Service.
JPOS_E_BUSY	The current Device Service state does not allow this request. For example, if asynchronous output is in progress, certain methods may not be allowed.
JPOS_E_EXTENDED	A device category-specific error condition occurred. The error condition code is available by calling <b>getErrorCodeExtended</b> .

### **ErrorCodeExtended**

The extended error code is set as follows:

- When *errorCode* is JPOS\_E\_EXTENDED, *errorCodeExtended* is set to a device category-specific value, and must match one of the values given in this document under the appropriate device category chapter.
- When *errorCode* is any other value, *errorCodeExtended* **may** be set by the Service to any Device Service-specific value. These values are only meaningful if an application adds Service-specific code to handle them.

## Events

Java for Retail POS uses events to inform the application of various activities or changes with the JavaPOS Device. The five event types follow.

Event Class	Description	Supported When A Device Category Supports...
DataEvent	Input data has been placed into device class-category properties.	Event-driven input
ErrorEvent	An error has occurred during event-driven input or asynchronous output.	Event-driven input -or- Asynchronous output
OutputCompleteEvent	An asynchronous output has successfully completed.	Asynchronous output
StatusUpdateEvent	A change in the Physical Device's status has occurred. <b>Release 1.3 and later:</b> All devices may be able to report device power state. See "Device Power Reporting Model" on page 27.	Status change notification
DirectIOEvent	This event may be defined by a Device Service provider for purposes not covered by the specification.	Always, for Service-specific use

Each of these events contains the following properties:

Property	Type	Description
<i>Source</i>	<i>Object</i>	Reference to the Device Control delivering the event. If the application defines a class that listens for events from more than one Device, then it uses this property to determine the Device instance that delivered the event.
<i>SequenceNumber</i>	<i>long</i>	JavaPOS event sequence number. This number is a sequence number that is global across all JavaPOS Devices. Each JavaPOS event increments the global sequence number, then places its value in this property.
<i>When</i>	<i>long</i>	An event timestamp; value is set to <b>System.currentTimeMillis()</b> .

Chapter 1, "Events" on page 74, provides details about each of these events, including additional properties.

The Device Service must enqueue these events on an internally created and managed queue. All JavaPOS events are delivered in a first-in, first-out manner. (The only exception is that a special input error event is delivered early if some data events are also enqueued. See “Device Input Model” on page 22.) Events are delivered by an internally created and managed Device Service thread. The Device Service causes event delivery by calling an event firing callback method in the Device Control, which then calls each registered listener's event method in the order in which they were added.

The following conditions cause event delivery to be delayed until the condition is corrected:

- The application has set the property **FreezeEvents** to true.
- The event type is a **DataEvent** or an input **ErrorEvent**, but the property **DataEventEnabled** is false. (See “Device Input Model” on page 22.)

Rules for event queue management are:

- The JavaPOS Device may only enqueue new events while the Device is enabled.
- The Device delivers enqueued events until the application calls the **release** method (for exclusive-use devices) or the **close** method (for any device), at which time any remaining events are deleted.
- For input devices, the **clearInput** method clears data and input error events.
- For output devices, the **clearOutput** method clears output error events.
- The application returns from the JPOS\_EL\_INPUT\_DATA **ErrorEvent** with *ErrorResponse* set to JPOS\_ER\_CLEAR.

## Registering for Events

JavaPOS events use the event delegation model first outlined in JDK 1.1. With this model, an application registers for events by calling a method supplied by the event source, which is the Device Control. The method is supplied a reference to an application class that implements a listener interface extended from `java.util.EventListener`.

The following table specifies the event interfaces and methods for each event class:

<b>Event Class</b>	<b>Listener Interface and Methods</b> Implemented in an application class	<b>Source Methods</b> Implemented in the Device Control
Data Event	<b>DataListener</b> dataOccurred (DataEvent e)	addDataListener (DataListener l) removeDataListener (DataListener l)
Error Event	<b>ErrorListener</b> errorOccurred (ErrorEvent e)	addErrorListener (ErrorListener l) removeErrorListener (ErrorListener l)
Status Update Event	<b>StatusUpdateListener</b> statusUpdateOccurred (StatusUpdateEvent e)	addStatusUpdateListener (StatusUpdateListener l) removeStatusUpdateListener (StatusUpdateListener l)
Output Complete Event	<b>OutputCompleteListener</b> outputCompleteOccurred (OutputCompleteEvent e)	addOutputCompleteListener (OutputCompleteListener l) removeOutputCompleteListener (OutputCompleteListener l)
DirectIO Event	<b>DirectIOListener</b> directIOOccurred (DirectIOEvent e)	addDirectIOListener (DirectIOListener l) removeDirectIOListener (DirectIOListener l)

Although more than one listener may be registered for an event type, the typical case is for only one listener, or at least only one primary listener. This listener takes actions such as processing data events and direct I/O events, and responding to error events.

## Event Delivery

A Device delivers an event by calling the listener method of each registered listener. The listener processes the event, then returns to the Device Control.

An application must not assume that events are delivered in the context of any particular thread. The JavaPOS Device delivers events on a privately created and managed thread. It is an application's responsibility to synchronize event processing with its threads as needed.

While an application is processing an event within its listener method, no additional events will be delivered by the Device.

While within a listener method, an application may access properties and call methods of the Device. However, an application must not call the **release** or **close** methods from an event method, because the **release** method may shut down event handling (possibly including a thread on which the event was delivered) and **close** must shut down event handling before returning.

## Device Input Model

The standard JavaPOS input model for exclusive-use devices is event-driven input. Event-driven input allows input data to be received after **DeviceEnabled** is set to true. Received data is enqueued as a **DataEvent**, which is delivered to an application as detailed in the “Events” (page 18). If the **AutoDisable** property is true when data is received, then the JavaPOS Device will automatically disable itself, setting **DeviceEnabled** to false. This will inhibit the Device from enqueueing further input and, when possible, physically disable the device.

When the application is ready to receive input from the JavaPOS Device, it sets the **DataEventEnabled** property to true. Then, when input is received (usually as a result of a hardware interrupt), the Device delivers a **DataEvent**. (If input has already been enqueued, the **DataEvent** will be delivered immediately after **DataEventEnabled** is set to true.) The **DataEvent** may include input status information through its Status property. The Device places the input data plus other information as needed into device category-specific properties just before the event is delivered.

Just before delivering this event, the JavaPOS Device disables further data events by setting the **DataEventEnabled** property to false. This causes subsequent input data to be enqueued by the Device while an application processes the current input and associated properties. When an application has finished the current input and is ready for more data, it enables data events by setting **DataEventEnabled** to true.

## Error Handling

If the JavaPOS Device encounters an error while gathering or processing event-driven input, then the Device:

- Changes its state to JPOS\_S\_ERROR.
- Enqueues an **ErrorEvent** with locus JPOS\_EL\_INPUT to alert an application of the error condition. This event is added to the end of the queue
- If one or more **DataEvents** are already enqueued for delivery, an additional **ErrorEvent** with locus JPOS\_EL\_INPUT\_DATA is enqueued before the **DataEvents**, as a pre-alert.

This event (or events) is not delivered until the **DataEventEnabled** property is true, so that orderly application sequencing occurs.

ErrorLocus	Description
JPOS_EL_INPUT_DATA	<p>Only delivered if the error occurred when one or more <b>DataEvents</b> are already enqueued.</p> <p>This event gives the application the ability to immediately clear the input, or to optionally alert the user to the error before processing the buffered input. This error event is enqueued before the oldest <b>DataEvent</b>, so that an application is alerted of the error condition quickly.</p> <p>This locus was created especially for the Scanner: When this error event is received from a Scanner JavaPOS Device, the operator can be immediately alerted to the error so that no further items are scanned until the error is resolved. Then, the application can process any backlog of previously scanned items before error recovery is performed.</p>
JPOS_EL_INPUT	<p>Delivered when an error has occurred and there is no data available.</p> <p>If some input data was buffered when the error occurred, then an <b>ErrorEvent</b> with the locus JPOS_EL_INPUT_DATA was delivered first, and then this error event is delivered after all <b>DataEvents</b> have been delivered.</p> <p><b>Note:</b> This JPOS_EL_INPUT event is not delivered if: an JPOS_EL_INPUT_DATA event was delivered and the application event handler responded with a JPOS_ER_CLEAR.</p>

The application's event listener method can set the **ErrorResponse** property to one of the following:

<b>ErrorResponse</b>	<b>Description</b>
JPOS_ER_CLEAR	Clear the buffered <b>DataEvents</b> and <b>ErrorEvents</b> and exit the error state, changing <b>State</b> to JPOS_S_IDLE. This is the default response for locus JPOS_EL_INPUT.
JPOS_ER_CONTINUE_INPUT	This response acknowledges the error and directs the Device to continue processing. The Device remains in the error state, and will deliver additional data events as directed by the <b>DataEventEnabled</b> property. When all input has been delivered and the <b>DataEventEnabled</b> property is again set to true, another <b>ErrorEvent</b> is delivered with locus JPOS_EL_INPUT. This is the default response when the locus is JPOS_EL_INPUT_DATA, and is legal only with this locus.
JPOS_ER_RETRY	This response directs the Device to retry the input. The error state is exited, and <b>State</b> is changed to JPOS_S_IDLE. This response may only be selected when the device chapter specifically allows it and when the locus is JPOS_EL_INPUT. An example is the scale.

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

- The application returns from the JPOS\_EL\_INPUT **ErrorEvent**.
- The application returns from the JPOS\_EL\_INPUT\_DATA **ErrorEvent**.
- The application calls the **clearInput** method.

### Miscellaneous

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

The **DataCount** property contains the number of **DataEvents** enqueued by the JavaPOS Device.

Calling the **clearInput** method deletes all input enqueued by a JavaPOS Device. **clearInput** may be called after **open** for sharable devices and after **claim** for exclusive-use devices.

The general event-driven input model does not specifically rule out the definition of device categories containing methods or properties that return input data directly. Some device categories define such methods and properties in order to operate in a more intuitive or flexible manner. An example is the Keylock Device. This type of input is sometimes called "synchronous input."



## Device Output Models

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

### Synchronous Output

The application calls a category-specific method to perform output. The JavaPOS Device does not return until the output is completed.

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

### Asynchronous Output

The application calls a category-specific method to start the output. The JavaPOS Device validates the method parameters and throws an exception immediately if necessary. If the validation is successful, the JavaPOS Device does the following:

1. Buffers the request.
2. Sets the **OutputID** property to an identifier for this request.
3. Returns as soon as possible.

When the JavaPOS Device successfully completes a request, an **OutputCompleteEvent** is enqueued for delivery to the application. A property of this event contains the output ID of the completed request. If the request is terminated before completion, due to reasons such as the application calling the **clearOutput** method or responding to an **ErrorEvent** with a `JPOS_ER_CLEAR` response, then no **OutputCompleteEvent** is delivered.

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

**Note:** Asynchronous output is always performed on a first-in first-out basis.

## Error Handling

If an error occurs while performing an asynchronous request, the error state `JPOS_S_ERROR` is entered and an **ErrorEvent** is enqueued with the **ErrorLocus** property set to `JPOS_EL_OUTPUT`. The application is guaranteed that the request in error is the one following the request whose output ID was most recently reported by an **OutputCompleteEvent**. An application's event listener method can set the **ErrorResponse** property to one of the following:

ErrorResponse	Description
<code>JPOS_ER_CLEAR</code>	Clear the outstanding output and exit the error state (to <code>JPOS_S_IDLE</code> ).
<code>JPOS_ER_RETRY</code>	Exit the error state (to <code>JPOS_S_BUSY</code> ) and retry the outstanding output. If the condition that caused the error was not corrected, then the Device may immediately reenter the error state and enqueue another <b>ErrorEvent</b> . This is the default response.

## Miscellaneous

Calling the **clearOutput** method deletes all output buffered by the JavaPOS Device. This method also stops any output that may be in progress (when possible).

**Note:** Currently, only the POS printer uses the complete Asynchronous Output model described here. Other device categories use portions of the model.

---

## Device Power Reporting Model

### *Added in JavaPOS Release 1.3.*

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

### Model

JavaPOS segments device power into three states:

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

In addition, one combination state is defined:

- **OFF\_OFFLINE.** The device is either off or offline, and the Device Service cannot distinguish these states.

Power reporting only occurs while the device is open, claimed (if the device is exclusive-use), and enabled.

---

#### **Note - Enabled/Disabled vs. Power States**

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

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

---

## Properties

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

- **CapPowerReporting** property. Identifies the reporting capabilities of the device. This property may be one of:
  - JPOS\_PR\_NONE. The Device Service cannot determine the state of the device. Therefore, no power reporting is possible.
  - JPOS\_PR\_STANDARD. The Device Service can determine and report two of the power states - OFF\_OFFLINE (that is, off or offline) and ONLINE.
  - JPOS\_PR\_ADVANCED. The Device Service can determine and report all three power states - ONLINE, OFFLINE, and OFF.
- **PowerState** property. Maintained by the Device Service at the current power condition, if it can be determined. This property may be one of:
  - JPOS\_PS\_UNKNOWN
  - JPOS\_PS\_ONLINE
  - JPOS\_PS\_OFF
  - JPOS\_PS\_OFFLINE
  - JPOS\_PS\_OFF\_OFFLINE
- **PowerNotify** property. The application may set this property to enable power reporting via **StatusUpdateEvents** and the **PowerState** property. This property may only be set before the device is enabled (that is, before **DeviceEnabled** is set to true). This restriction allows simpler implementation of power notification with no adverse effects on the application. The application is either prepared to receive notifications or doesn't want them, and has no need to switch between these cases. This property may be one of:
  - JPOS\_PN\_DISABLED
  - JPOS\_PN\_ENABLED

## Power Reporting Requirements for DeviceEnabled

The following semantics are added to **DeviceEnabled** when

**CapPowerReporting** is not JPOS\_PR\_NONE, and  
**PowerNotify** is JPOS\_PN\_ENABLED:

- When the Control changes from **DeviceEnabled** false to true, then begin monitoring the power state:
  - If the Physical Device is ONLINE, then:
    - PowerState** is set to JPOS\_PS\_ONLINE.
    - A **StatusUpdateEvent** is enqueued with its *Status* property set to JPOS\_SUE\_POWER\_ONLINE.
  - If the Physical Device's power state is OFF, OFFLINE, or OFF\_OFFLINE, then the Device Service may choose to fail the enable by throwing a **JposException** with error code JPOS\_E\_NOHARDWARE or JPOS\_E\_OFFLINE.

However, if there are no other conditions that cause the enable to fail, and the Device Service chooses to return success for the enable, then:

    - PowerState** is set to JPOS\_PS\_OFF, JPOS\_PS\_OFFLINE, or JPOS\_PS\_OFF\_OFFLINE.
    - A **StatusUpdateEvent** is enqueued with its *Status* property set to JPOS\_SUE\_POWER\_OFF, JPOS\_SUE\_POWER\_OFFLINE, or JPOS\_SUE\_POWER\_OFF\_OFFLINE.
- When the Device changes from **DeviceEnabled** true to false, JavaPOS assumes that the Device is no longer monitoring the power state and sets the value of **PowerState** to JPOS\_PS\_UNKNOWN.

## Device States

JavaPOS defines a property **State** with the following values:

```
JPOS_S_CLOSED  
JPOS_S_IDLE  
JPOS_S_BUSY  
JPOS_S_ERROR
```

The **State** property is set as follows:

- **State** is initially JPOS\_S\_CLOSED.
- **State** is changed to JPOS\_S\_IDLE when the **open** method is successfully called.
- **State** is set to JPOS\_S\_BUSY when the Device Service is processing output. The **State** is restored to JPOS\_S\_IDLE when the output has completed.
- The **State** is changed to JPOS\_S\_ERROR when an asynchronous output encounters an error condition, or when an error is encountered during the gathering or processing of event-driven input.

After the Device Service changes the **State** property to JPOS\_S\_ERROR, it enqueues an **ErrorEvent**. The properties of this event are the error code and extended error code, the locus of the error, and a modifiable response to the error. See Input Model on Error Handling on page 23 and Output Model on Error Handling on page 26 for further details.

## Threads

The Java language directly supports threads, and an application may create additional threads to perform different jobs. The use of threads can add complexity, however, often requiring synchronization to arbitrate sharing of resources. For applications that share a control instance among multiple threads, actions of one thread may have undesirable effects on the other thread(s). For example, cancelled I/O (e.g., `clearOutput`) can result in any pending synchronous requests of other threads being completed with a JPOS exception with an error code of `JPOS_E_FAILURE`. These situations can be avoided by insuring a control instance is managed by a single thread.

An application must be aware of multiple threads in the following cases:

- **Properties and Methods.** Calling some JavaPOS methods or setting some properties can cause other property values to be changed. When an application needs to access these properties, it must either access the properties and methods from only one thread, or ensure that its threads synchronize these sequences as required.
- **Events.** An application must not assume that events are delivered in the context of any particular thread. The JavaPOS Device typically will deliver events on a privately created and managed thread. It is an application's responsibility to synchronize event processing with its threads if necessary.

## Version Handling

As JavaPOS evolves, additional releases will introduce enhanced versions of some Devices. JavaPOS imposes the following requirements on Device Control and Service versions:

- **Device Control requirements.** A Device Control for a device category must operate with any Device Service for that category, as long as its major version number matches the Service's major version number. If they match, but the Control's minor version number is greater than the Service's minor version number, the Control may support some new methods or properties that are not supported by the Service's release. If an application calls one of these methods or accesses one of these properties, a **JposException** with error code **JPOS\_E\_NOSERVICE** will be thrown.
- **Device Service requirements.** A Device Service for a device category must operate with any Device Control for that category, as long as its major version number matches the Control's major version number. If they match, but the Service's minor version number is greater than the Control's minor version number, then the Service may support some methods or properties that cannot be accessed from the Control.

When an application wishes to take advantage of the enhancements of a version, it must first determine that the Device Control and Device Service are at the proper major version and at or greater than the proper minor version. The versions are reported by the properties **DeviceControlVersion** and **DeviceServiceVersion**.



## Classes and Interfaces

### Synopsis

This section lists the JavaPOS classes and interfaces used by applications, Device Controls and Device Services. Further details about their usage appear later in this document.

In the tables that follow, the following substitutions should be made for *italic* type:

Substitution Name	Description
<i>Event</i>	Replace with one of the five event types: <b>Data, Error, OutputComplete, StatusUpdate, DirectIO</b>
<i>event</i>	Replace with one of the five event types: <b>data, error, outputComplete, statusUpdate, directIO</b>
<i>Devcat</i>	Replace with one of the device categories: <b>BumpBar, CashChanger, CashDrawer, CAT, CoinDispenser, FiscalPrinter, HardTotals, Keylock, LineDisplay, MICR, MSR, PINPad, PointCardRW, POSKeyboard, POSPower, POSPrinter, RemoteOrderDisplay, Scale, Scanner, SignatureCapture, ToneIndicator</b>
<i>Rr</i>	Replace with the JavaPOS release number. For example, Release 1.2 is shown as 12. When an interface or class uses a release number, interfaces for later releases at the same major version number extend the previous release's interface or class.
<i>Pp</i>	Replace with the JavaPOS release number prior to <i>Rr</i> . For example, if <i>Rr</i> is 13, then <i>Pp</i> is 12.

The classes and interfaces defined or used by JavaPOS are summarized in the following tables, organized by the software entity that implements them.

### Application

Class or Interface	Name	Description	Extends / Implements
Interface	<b>jpos.EventListener</b> (Ex: <b>DataListener</b> )	Application defines and registers a class that implements this interface. Events are delivered by calling the <i>eventOccurred</i> (ex: <b>dataOccurred</b> ) method of this interface with an <i>EventEvent</i> (ex: <b>DataEvent</b> ) instance.	Extends: <b>java.util.EventListener</b>

## Device Control

Class or Interface	Name	Description	Extends / Implements
Class	<b>jpos.Devcat</b> (ex: <b>Scanner</b> , <b>POSPrinter</b> )	Device Control Class. One fixed name per device category.	Implements: <b>jpos.DevcatControlRr</b> (ex: <b>ScannerControl12</b> , <b>POSPrinterControl13</b> ) Implements (as an Inner Class): <b>jpos.services.</b> <b>EventCallbacks</b>
Interface	<b>jpos.DevcatControlRr</b> (ex: <b>ScannerControl12</b> , <b>POSPrinterControl13</b> )	Contains the methods and properties specific to Device Controls for this device category and release.	Extends either: <b>jpos.BaseControl</b> (for first release) or <b>jpos.DevcatControlPp</b> (for later releases) (ex: <b>POSPrinterControl13</b> )
Interface	<b>jpos.BaseControl</b>	Contains the methods and properties common to all Device Controls.	--
Interface	<b>jpos.services.</b> <b>EventCallbacks</b>	Includes one callback method per event type. The Device Service calls these methods to cause events to be delivered to the application.	--

## Device Service

Class or Interface	Name	Description	Extends / Implements
Class	Vendor-defined name	Device Service Class.	Implements: <b>jpos.services.</b> <b>DevcatServiceRr</b> (ex: <b>ScannerService12</b> , <b>POSPrinterService13</b> )
Interface	<b>jpos.services.</b> <b>DevcatServiceRr</b> (ex: <b>ScannerService12</b> , <b>POSPrinterService13</b> )	Contains the methods and properties specific to Device Services for this device category and release.	Extends either: <b>jpos.services.</b> <b>BaseService</b> (for first release) or <b>jpos.services.</b> <b>DevcatServicePp</b> (for later releases) (ex: <b>POSPrinterService13</b> )
Interface	<b>jpos.services.</b> <b>BaseService</b>	Contains the methods and properties common to all Device Services.	--

## Helper Classes

Class or Interface	Name	Description	Extends / Implements
Interface	<b>jpos.JposConst</b>	Interface containing the JavaPOS constants that are common to several device categories.	--
Interface	<b>jpos.DevcatConst</b> (ex: <b>ScannerConst</b> , <b>POSPrinterConst</b> )	Interface containing the JavaPOS constants specific to a device category.	--
Class	<b>jpos.JposEvent</b>	Abstract class from which all JavaPOS event classes are extended.	Extends: <b>java.util.EventObject</b>
Class	<b>jpos.EventEvent</b> (ex: <b>DataEvent</b> )	The Device Service creates <i>Event</i> event instances of this class and delivers them through the Device Control's event callbacks to the application.	Extends: <b>jpos.JposEvent</b>
Class	<b>jpos.JposException</b>	Exception class. The Device Control and Device Service create and throw exceptions on method and property access failures.	Extends: <b>java.lang.Exception</b>

## Sample Class and Interface Hierarchies

The following example class hierarchies are given for the scanner Release 1.2 (the initial Release) and for the printer (Release 1.3). Assume that neither Device Service generates any DirectIO events in which the application is interested.

### Application

“MyApplication” class hierarchy:

- **DataListener.** Implement to receive Scanner data events.
- **ErrorListener.** Implement to receive Scanner and POSPrinter error events.
- **OutputCompleteListener.** Implement to receive POSPrinter output complete events.
- **StatusUpdateListener.** Implement to receive POSPrinter status update events.

(Frequently, an application will define additional classes that implement one or more of the listener interfaces.)

The “MyApplication” Application class also uses the following:

- **Scanner** and **POSPrinter.** Instances of the Device Controls.
- **JposConst, ScannerConst,** and **POSPrinterConst.** Use constants, either by fully qualified package names or by adding to the “implements” clause of an application class.
- **DataEvent.** Instance of this class received by the **DataListener's** method **dataOccurred.**
- **ErrorEvent.** Instance of this class received by the **ErrorListener's** method **errorOccurred.**
- **OutputCompleteEvent.** Instance of this class received by the **OutputCompleteListener's** method **outputCompleteOccurred.**
- **StatusUpdateEvent.** Instance of this class received by the **StatusUpdateListener's** method **statusUpdateOccurred.**
- **JposException.** Instance of this class is caught when a Scanner or POSPrinter method or property access fails.

### Device Controls

#### Scanner

**Scanner** class hierarchy:

- **ScannerControl12.** Implement scanner’s methods and properties.
- **EventCallbacks.** Derive an inner class to pass to Service so that it may generate events.

The **Scanner** Control class also uses the following:

- **JposConst** and **ScannerConst**. Use constants, either by fully qualified package names or by adding to the “implements” clause of the Device Control.
- **JposException**. Instance of this class is thrown when a method or property access fails.

## POSPrinter

**POSPrinter** class hierarchy:

- **POSPrinterControl13**. Implement printer’s methods and properties and extends **POSPrinterControl12**.
- **EventCallbacks**. Derive an inner class to pass to Service so that it may generate events.

The **POSPrinter** Control class also uses the following:

- **JposConst** and **POSPrinterConst**. Use constants, either by fully qualified package names or by adding to the “implements” clause of the Device Control.
- **JposException**. Instance of this class is thrown when a method or property access fails.

## Device Service

### “MyScannerService”

“MyScannerService” class hierarchy:

- **ScannerService12**. Implement scanner’s methods and properties.

The “MyScannerService” Service class also uses the following:

- **JposConst** and **ScannerConst**. Use constants, either by fully qualified package names or by adding to the “implements” clause of the Device Service.
- **DataEvent**. Instance of this class created as data is received. It is delivered to an application when the event delivery preconditions are met by calling the **fireDataEvent** method of the Control's derived **EventCallbacks** class.
- **ErrorEvent**. Instance of this class created when an error is detected while reading scanner data. It is delivered to an application when the event delivery preconditions are met by calling the **fireErrorEvent** method of the Control's derived **EventCallbacks** class.
- **JposException**. Instance of this class is thrown when a method or property access fails.

## “MyPrinterService”

“MyPrinterService” class hierarchy:

- **POSPrinterService13.** Implement printer’s methods and properties and extends **POSPrinterService12.**

The “MyPrinterService” Service class also uses the following:

- **JposConst** and **POSPrinterConst.** Use constants, either by fully qualified package names or by adding to the “implements” clause of the Device Service.
- **ErrorEvent.** Instance of this class created when an error is detected while printing asynchronous data. It is delivered to an application when the event delivery preconditions are met by calling the **fireErrorEvent** method of the Control's derived **EventCallbacks** class.
- **OutputCompleteEvent.** Instance of this class created when an asynchronous output request completes. It is delivered to an application when the event delivery preconditions are met by calling the **fireOutputCompleteEvent** method of the Control's derived **EventCallbacks** class.
- **StatusUpdateEvent.** Instance of this class created when a printer status change is detected. It is delivered to an application when the event delivery preconditions are met by calling the **fireStatusUpdateEvent** method of the Control's derived **EventCallbacks** class.
- **JposException.** Instance of this class is thrown when a method or property access fails.

## Sample Application Code

The following code snippet shows how to use a scanner.

```
//import ...;
import jpos.*;
import jpos.events.*;

public class MyApplication implements DataListener
{
    // Data listener's method to process incoming scanner data.
    public void dataOccurred(DataEvent e)
    {
        jpos.Scanner dc = (jpos.Scanner) e.getSource();
        String Msg = "Scanner DataEvent (Status=" + e.getStatus() +
            ") received.";
        System.out.println (Msg);
        try {
            dc.setDataEventEnabled(true);
        } catch (JposException e){}
    }

    // Method to initialize the scanner.
    public void initScanner(String openName) throws jpos.JposException
    {
        // Create scanner instance and register for data events.
        jpos.Scanner myScanner1 = new jpos.Scanner();
        myScanner1.addDataListener(this);
        // Initialize the scanner. Exception thrown if a method fails.
        myScanner1.open(openName);
        myScanner1.claim(1000);
        myScanner1.setDeviceEnabled(true);
        myScanner1.setDataEventEnabled(true);
        //...Success! Continue doing work...
    }

    //...Other methods, including main...
}
```

## Package Structure

The JavaPOS packages and files for Release 1.4 are as follows:

*Note: The only difference between Release 1.3 and Release 1.4 of JavaPOS is the inclusion of the CAT device. No other technical changes were made. Therefore the JavaPOS packages and files for devices covered under Release 1.3 may be used for Release 1.4. Additional device classifications of Point Card Reader Writer and POS Power were added in Release 1.5.*

### jpos

BaseControl.java  
JposConst.java  
JposException.java

CashChanger.java  
CashChangerBeanInfo.java  
CashChangerConst.java  
CashChangerControl13.java

CashDrawer.java  
CashDrawerBeanInfo.java  
CashDrawerConst.java  
CashDrawerControl13.java

CoinDispenser.java  
CoinDispenserBeanInfo.java  
CoinDispenserConst.java  
CoinDispenserControl13.java

HardTotals.java  
HardTotalsBeanInfo.java  
HardTotalsConst.java  
HardTotalsControl13.java

Keylock.java  
KeylockBeanInfo.java  
KeylockConst.java  
KeylockControl13.java

LineDisplay.java  
LineDisplayBeanInfo.java  
LineDisplayConst.java  
LineDisplayControl13.java

MICR.java  
MICRBeanInfo.java  
MICRConst.java  
MICRControl13.java

#### *New Peripheral Device Services Added in Release 1.3*

BumpBar.java  
BumpBarBeanInfo.java  
BumpBarConst.java  
BumpBarControl13.java

FiscalPrinter.java  
FiscalPrinterBeanInfo.java  
FiscalPrinterConst.java  
FiscalPrinterControl13.java

MSR.java  
MSRBeanInfo.java  
MSRConst.java  
MSRControl13.java

POSKeyboard.java  
POSKeyboardBeanInfo.java  
POSKeyboardConst.java  
POSKeyboardControl13.java

POSPrinter.java  
POSPrinterBeanInfo.java  
POSPrinterConst.java  
POSPrinterControl13.java

Scale.java  
ScaleBeanInfo.java  
ScaleConst.java  
ScaleControl13.java

Scanner.java  
ScannerBeanInfo.java  
ScannerConst.java  
ScannerControl13.java

SignatureCapture.java  
SignatureCaptureBeanInfo.java  
SignatureCaptureConst.java  
SignatureCaptureControl13.java

ToneIndicator.java  
ToneIndicatorBeanInfo.java  
ToneIndicatorConst.java  
ToneIndicatorControl13.java

PINpad.java  
PINpadBeanInfo.java  
PINpadConst.java  
PINpadControl13.java

RemoteOrderDisplay.java  
RemoteOrderDisplayBeanInfo.java  
RemoteOrderDisplayConst.java  
RemoteOrderDisplayControl13.java

#### *New Peripheral Device Service Added in Release 1.4*

CAT.java  
CATBeanInfo.java  
CATConst.java  
CATControl14.java



*New Peripheral Device Services Added in Release 1.5*

PointCardRW.java  
PointCardRWBeanInfo.java  
PointCardRWConst.java  
PointCardRWControl15.java

POSPower.java  
POSPowerBeanInfo.java  
POSPowerConst.java  
POSPowerControl15.java

**jpos.events**

JposEvent.java

DataEvent.java  
DataListener.java  
DirectIOEvent.java  
DirectIOListener.java  
ErrorEvent.java  
ErrorListener.java  
OutputCompleteEvent.java  
OutputCompleteListener.java  
StatusUpdateEvent.java  
StatusUpdateListener.java

**jpos.services**

BaseService.java  
EventCallbacks.java

CashChangerService13.java  
CashDrawerService13.java  
CoinDispenserService13.java  
HardTotalsService13.java  
KeylockService13.java  
LineDisplayService13.java  
MICRService13.java  
MSRService13.java  
POSKeyboardService13.java  
POSPrinterService13.java  
ScaleService13.java  
ScannerService13.java  
SignatureCaptureService13.java  
ToneIndicatorService13.java

*New Peripheral Device Services Added in Release 1.3*

BumpBarService13.java  
FiscalPrinterService13.java  
PINpadService13.java  
RemoteOrderDisplayService13.java

*New Peripheral Device Services Added in Release 1.4*

CATService14.java

*New Peripheral Devie Services Added in Release 1.5*

PointCardRW15.java  
POSPower15.java



## Device Controls

*Note: This section is intended primarily for programmers who are creating JavaPOS Device Controls and Services.*

### Device Control Responsibilities

- Supporting the JavaPOS Device Interface for its category. This includes a set of properties, methods, and events.
- Managing the connection and interface to a Device Service.
- Forwarding most property accesses and method calls to the Device Service, and throwing exceptions when a property access or method call fails.
- Supporting add and remove event listener methods.
- Generating events to registered listeners upon command from the Device Service.
- Downgrading for older Device Service versions.

A Device Control is **not** responsible for:

- Managing multi-thread access to the Device Control and Service. An application must either access a Control from only one thread, or ensure that its threads synchronize sequences of requests as required to ensure that affected state and properties are maintained until the sequences have completed.
- Data buffering, including input and output data plus events. The Device Service manages all buffering and enqueueing.
- The device behavior/semantics and nuances that are specific to the functional control of the device.
- The loading functions that are to be contained in the `jpos.config/loader (JCL)`.

## Device Service Management

The Device Control manages the connection to the Device Service. The Control calls upon the `jpos.config/loader` (JCL) to accomplish the connection and disconnection.

### **jpos.config/loader (JCL) and JavaPOS Entry Registry (JER)**

The `jpos.config/loader` (JCL) along with the JavaPOS Entry Registry (JER) is used as the binding (configuration and loading) API that allows a JavaPOS control to bind to the correct JavaPOS service in a manner independent of the actual configuration mechanism. For POS applications, it represents a somewhat minimum (but extensible) functional equivalent of the “NT Registry” called the **JposEntryRegistry**.

All JavaPOS Device Controls that use this API and additional helpful reference material can be obtained on the JavaPOS website ([www.javapos.com](http://www.javapos.com)). In addition other standards information may be obtained from the [www.NRF-ARTS.org](http://www.NRF-ARTS.org) website.

A reference open source implementation of the JCL is available on this website and maintained under the control of the JavaPOS technical committee. Included on the website is a functioning JCL with complete JavaDoc documentation, examples, sample code, a browser-based configuration editor and additional explanatory material.

A brief description of the JCL process is given below. However, for additional detailed information on the JCL one should consult the referenced web sites for the most up to date information.

### **jpos.config/loader (JCL) Characteristics**

The `jpos.config/loader` is the name for the minimal set of classes (1) and interfaces (6) which are necessary to abstract into the JavaPOS specification. They provide for an independent way of configuring, loading and creating JavaPOS device services while maintaining the following important goals.

- Minimize the impact on existing controls
- Allow services to easily support multiple `jpos.config/loader` implementations
- Abstract as much as possible using Java interfaces to separate the JCL specification from its implementation
- Keep to a minimum the number of necessary classes and interfaces

The `jpos.config/loader` class/interfaces are added in two packages named `jpos.config` and `jpos.loader`. A `jpos` implementation is dependent upon the `jpos` and `jpos.loader` packages included in the `jpos.loader` class/interfaces, the `jpos.JposConst` interfaces and the `jpos.JposException` classes.

The `jpos.config/loader` specification contains 1 class and 6 interfaces. The single class is the `jpos.loader.ServiceLoader` which bootstraps the implementation of the `jpos.config/loader` to be used in the JVM by creating the manager object (an instance of the `jpos.loader.JposServiceManager` interface). It also defaults to the simple `jpos.config/loader` implementation if no bootstrap is defined. The following table gives the name and a brief description of the class and interfaces that are involved.

Class or Interface	Name	Description
class	<code>jpos.loader.ServiceLoader</code>	This is the only class in the <code>jpos.config</code> and <code>jpos.loader</code> packages. It maintains a <b>JposServiceManager</b> instance (manager) which it uses to create a <b>JposServiceConnection</b> . The manager is created by looking for a Java property " <b>jpos.loader.serviceManagerClass</b> ". If this property is defined, then the class that it defines will be loaded and an instance of this class created as the manager (NOTE: this also assumes that the class implements <code>JposServiceManager</code> interface and has a 0-argument constructor). If the property is not defined then the "simple" JCL reference implementation manager is created ( <b>jpos.loader.simple.SimpleServiceManager</b> ).
interface	<code>jpos.loader.JposServiceManager</code>	This interface defines a manager used to create <b>JposServiceConnection</b> and allows access to the <b>JposEntryRegistry</b> .
interface	<code>jpos.loader.JposServiceConnection</code>	Defines a mediator between the service and the user of the service. The JavaPOS controls use this interface to connect to the service and then get the <b>JposServiceInstance</b> associated with the connection. Once disconnected the <b>JposServiceInstance</b> is no longer valid and a re-connect is necessary.
interface	<code>jpos.config.JposEntry</code>	Defines an interface for configuring a service. Properties can be added, queried, modified and removed. The <b>JposServiceInstanceFactory</b> uses the information in the object implementing this interface to create the current <b>JposServiceInstance</b> and configure it.
interface	<code>jpos.loader.JposEntryRegistry</code>	This interface defines a way to statically and dynamically add known <b>JposEntry</b> objects to the system.
interface	<code>jpos.loader.JposServiceInstance</code>	Only interface required to be implemented by all JavaPOS services. It defines one method that is used to indicate to the service that the connection has been disconnected.
interface	<code>jpos.loader.JposServiceInstanceFactory</code>	Factory interface to create <b>JposServiceInstance</b> objects (i.e. the JavaPOS services). It is passed a <b>JposEntry</b> which it uses to create the correct service.

The configuration information is described as a set of properties in the **JposEntry**. These are entered as *<key, value>* pairs. The key is a String and the value is a Java Object of type: String, Integer, Long, Float, Boolean, Character or Byte (which are the String and primitive wrapper classes provided in the java.lang package). The following are two properties which must be defined by all the entries in the **JposEntry** in order for it to be considered valid.

Property Name	Property Type	Description
<b>logicalName</b>	String	This is the unique name that identifies this entry. The control uses this name to bind itself to the service.
<b>serviceInstanceFactoryClass</b>	String	Defines the factory class which should be used to create the service. This class must implement the <b>jpos.loader.JposServiceInstanceFactory</b> interface and it must have a default constructor.

All other properties are optionally provided or needed for the correct creation and initialization of the JavaPOS service. Note the service providers will most likely want to define their own set of properties and require them to be in the **JposEntry** in order to allow their **JposServiceFactory** to be used and their device service to be configured and loaded.

Future releases of the reference jpos.config/loader (JCL) might be modified to define a standard set of properties (in addition to the two mandated above) that all JavaPOS services would need to define.

## Property and Method Forwarding

The Device Control must use the Device Service to implement all properties and methods defined by the JavaPOS Device Interface for a device category, with the following exceptions:

- **open** method.
- **close** method.
- **DeviceControlDescription** property. The Control returns its description.
- **DeviceControlVersion** property. The Control returns its version.
- **State** property. The Control forwards the request to the Service as shown in the following paragraphs. Any exception is changed to a return value of JPOS\_S\_CLOSED; an exception is never thrown to an application.

For all other properties and methods, the Device Control forwards the request to the identically named method or property of the Device Service. A template for set property and method request forwarding follows:

```
public void name(Parameters) throws JposException
{
    try
        service.name(Parameters);
    catch(JposException je)
        throw je;
    catch(Exception e)
        throw new JposException(JPOS_E_CLOSED,
            "Control not opened", e);
}
```

Similarly, a template for get property request forwarding is:

```
public Type name() throws JposException
{
    try
        return service.name();
    catch(JposException je)
        throw je;
    catch(Exception e)
        throw new JposException(JPOS_E_CLOSED,
            "Control not opened", e);
}
```

The general forwarding sequence is to call the Service to process the request, and return to the application if no exception occurs. If an exception occurs and the exception is **JposException**, rethrow it to the application.

Otherwise wrap the exception in a **JposException** and throw it. This should only occur if an **open** has not successfully linked the Service to the Control, that is, if the **service** field contains a null reference. (Any exceptions that occur while in the Service should be caught by it, and the Service should rethrow it as a **JposException**.) This allows the Control to set the message text to “Control not opened” with reasonable certainty.

## Event Handling

### Event Listeners and Event Delivery

An application must be able to register with the Device Control to receive events of each type supported by the Device, as well as unregister for these events. To conform to the JavaBean naming pattern for events, the registration methods have the form:

```
void addXxxListener(XxxListener l);  
void removeXxxListener(XxxListener l);
```

where *Xxx* is replaced by one of the event types: **Data**, **Error**, **OutputComplete**, **StatusUpdate**, or **DirectIO**.

An example add listener method is:

```
protected Vector dataListeners;  
public void addDataListener(DataListener l)  
{  
    synchronized(dataListeners)  
        dataListeners.addElement(l);  
}
```

When the Device Service requests that an event be delivered, the Control calls the event method of each listener that has registered for that event. (Typically, only one listener will register for each event type. However, diagnostic or other software may choose to listen, also.) The event methods have the form:

```
void xxxOccurred(XxxEvent e)
```

where *xxx* is replaced by: **data**, **error**, **outputComplete**, **statusUpdate**, or **directIO**.



## Event Callbacks

The Device Service requests that an event be delivered by calling a method in a callback instance. This instance is created by the Control and passed to the Service in the **open** method.

The callback instance is typically created as an inner class of the Control. An example callback inner class is:

```
protected class ScannerCallbacks implements EventCallbacks
{
    public BaseControl getEventSource()
    {
        return (BaseControl)Scanner.this;
    }

    public void fireDataEvent(DataEvent e)
    {
        synchronized(Scanner.this.dataListeners)
            // deliver the event to all registered listeners
            for(int x = 0; x < dataListeners.size(); x++)
                ((DataListener)dataListeners.elementAt(x)).
                    dataOccurred(e);
    }

    public void fireDirectIOEvent(DirectIOEvent e)
    {
        //...Removed code similar to fireDataEvent...
    }

    public void fireErrorEvent(ErrorEvent e)
    {
        //...Removed code similar to fireDataEvent...
    }

    public void fireOutputCompleteEvent(OutputCompleteEvent e)
    {
    }

    public void fireStatusUpdateEvent(StatusUpdateEvent e)
    {
    }
}
```

## Version Handling

The Device Control responsibilities given in the preceding sections “Device Service Management” and “Property and Method Forwarding” are somewhat simplified: They do not take into account version handling.

Both the Device Control and the Device Service have version numbers. Each version number is broken into three parts: Major, minor, and build. The major and minor portions indicate compliance with a release of the JavaPOS specifications. For example, release 1.4 compatibility is represented by a major version of one and a minor version of four. The build portion is set by the JavaPOS Device writer.

The JavaPOS version requirement is that a Device Control for a device category must operate and return reasonable results with any Device Service for that class, as long as its major version number matches the Service’s major version number.

In order to support this requirement, the following steps must be taken by the Control:

- **open** method. The Control must validate and determine the version of the Service, and save this version for later use (the “validated version”). The steps are as follows:

1. After connecting to the Device Service and obtaining its reference, determine the level of JavaPOS Service interface supported by the Service (the “interface version”). This test ensures that the Service complies with the property and method requirements of the interface.

For example, assume that the Scanner Control is at version 1.3. First attempt to cast the Service reference to the original release version, **ScannerService12**. If this succeeds, the “interface version” is at least 1.2; otherwise fail the open. Next, attempt to cast to **ScannerService13**. If this succeeds, the “interface version” is 1.3.

2. After calling the Service’s **open** method, get its **DeviceServiceVersion** property. If the major version does not match the Control’s major version, then fail the open.
3. At this point we know that some level of Service interface is supported, and that the major Control and Service versions match. Now determine the “validated version”:

```

if ( service_version <= interface_version )
{
    // The Service version may match the interface
    // version, or it may be less. The latter case may
    // be true for a Service that wraps or bridges to
    // OPOS software, because the Service may be able to
    // support a higher interface version, but
    // downgrades its reported Service version to that of
    // the OPOS software.
    // Remember the Services real version.
    validated_version = service_version;
}
else if ( service_version > interface_version )

```

```
{
    // The Service is newer than the Control.
    // Look at two subcases.
    if ( control_version == interface_version )
    {
        // The Service is newer than the Control, and it
        // supports all the Controls methods and
        // properties (and perhaps more that the Control
        // will not call).
        // Remember the maximum version that the Control
        // supports.
        validated_version = interface_version;
    }
    else if ( service_version > interface_version )
    {
        //... Fail the open!
        // The Service is reporting a version for which it
        // does not support all the required methods and
        // properties.
    }
}
```

- Properties and other methods. If an application accesses a property or calls a method supported by the Control's version but not by the "validated version" of the Service, the Control must throw a **JposException** with error code **JPOS\_E\_NOSERVICE**.



## Device Services

*Note: This section is intended primarily for programmers creating JavaPOS Device Controls and Services.*

### Device Service Responsibilities

A Device Service for a device category is responsible for:

- Supporting the JavaPOS Device Service Interface for its category. This includes a set of properties and methods, plus event generation and delivery.
- Implementing property accesses and method calls, and throwing exceptions when a property access or method call fails.
- Enqueuing events and delivering them (through calls to Device Control event callback methods) when the preconditions for delivering the event are satisfied.
- Managing access to the Physical Device.

The Device Service requires the `jpos.config/loader (JCL) JposEntry` object which contains all the configuration information.

### Property and Method Processing

The Device Service performs the actual work for the property access and method processing. If the Service is successful in carrying out the request, it returns to the application. Otherwise, it must throw a **JposException**.

At the beginning of property and method processing, the Service will typically need to validate that an application has properly initialized the device before it is processed. If the device must first be claimed, the Service throws an exception with the error code `JPOS_E_CLAIMED` (if the device is already claimed by another JPOS Device) or `JPOS_E_NOTCLAIMED` (if the device is available to be claimed). If the device must first be enabled, then the Service throws an exception with the error code `JPOS_E_DISABLED`.

Some special cases are:

- **open** method. The Service must perform additional housekeeping and initialization during this method. Initialization will often include accessing the Java System Database (Release 1.4 and prior) or `JposEntryRegistry` (Release 1.5 and beyond) to obtain parameters specific to the Service and the Physical Device.
- **close** method. The Service releases all resources that were acquired during or after **open**.

## Event Generation

The Device Service has the responsibility of enqueueing events and delivering them in the proper sequence. The Service must enqueue and deliver them one at a time, in a first-in, first-out manner. (The only exception is when a `JPOS_EL_INPUT_DATA` event must be delivered early on an input error because some data events are also enqueued.) Events are delivered by an internally created and managed Service thread. They are delivered by calling an event firing callback method in the Device Control, which then calls each registered listener's event method. (See “Event Handling” on page 48.)

The following conditions cause event delivery to be delayed until the condition is corrected:

- The application has set the property **FreezeEvents** to true.
- The event type is a **DataEvent** or an input **ErrorEvent**, but the property **DataEventEnabled** is false. (See “Device Input Model” on page 22.)

Rules on the management of the queue of events are:

- The JavaPOS Device may only enqueue new events while the Device is enabled.
- The Device may deliver enqueued events until the application calls the **release** method (for exclusive-use devices) or the **close** method (for any device), at which time any remaining events are deleted.
- For input devices, the **clearInput** method clears data and input error events.
- For output devices, the **clearOutput** method clears output error events.

## Physical Device Access

The Device Service is responsible for managing the Physical Device. Often, this occurs by using a communications Port API (supplied or custom). At other times, the Service may need to use other device drivers or techniques to control the device.

# Common Properties, Methods, and Events

The following Properties, Methods, and Events are used for all device categories unless noted otherwise in the *Usage Notes* table entry. For an overview of the general rules and usage guidelines, see “Device Behavior Models” on page 9.

## Summary

### Properties

<i>Name</i>	<i>Usage Notes</i>	<i>Ver</i>	<i>Type</i>	<i>Access</i>
<b>AutoDisable</b>	<i>1</i>		boolean	R/W
<b>CapPowerReporting</b>		1.3	int	R
<b>CheckHealthText</b>			String	R
<b>Claimed</b>			boolean	R
<b>DataCount</b>	<i>1</i>		int	R
<b>DataEventEnabled</b>	<i>1</i>		boolean	R/W
<b>DeviceEnabled</b>			boolean	R/W
<b>FreezeEvents</b>			boolean	R/W
<b>OutputID</b>	<i>2</i>		int	R
<b>PowerNotify</b>		1.3	int	R/W
<b>PowerState</b>		1.3	int	R
<b>State</b>			int	R
<b>DeviceControlDescription</b>			String	R
<b>DeviceControlVersion</b>			int	R
<b>DeviceServiceDescription</b>			String	R
<b>DeviceServiceVersion</b>			int	R
<b>PhysicalDeviceDescription</b>			String	R
<b>PhysicalDeviceName</b>			String	R

**Methods**

<i>Name</i>	<i>Usage Notes</i>
<b>open</b>	
<b>close</b>	
<b>claim</b>	
<b>release</b>	
<b>checkHealth</b>	
<b>clearInput</b>	
<b>clearOutput</b>	
<b>directIO</b>	

**Events**

<i>Name</i>	<i>Usage Notes</i>
<b>DataEvent</b>	1
<b>DirectIOEvent</b>	
<b>ErrorEvent</b>	
<b>OutputCompleteEvent</b>	2
<b>StatusUpdateEvent</b>	

Usage Notes:

1. Used only with Devices that have Event Driven Input.
2. Used only with Asynchronous Output Devices.



## General Information

This section lists properties, methods, and events that are common to many of the peripheral devices covered in this standard.

The summary section of each device category marks those common properties, methods, and events that do not apply to that category as “Not Supported.” Items identified in this fashion are not present in the device control’s class.

This section relies heavily on the user being familiar with Java programming techniques covered in JDK version 1.1 and later. In addition, a good understanding of the features of the JavaPOS architecture model is required. Please see “Device Behavior Models” on page 9 for additional information.

## Properties

### AutoDisable Property R/W

<b>Type</b>	<b>boolean</b>
<b>Remarks</b>	<p>If true, the Device Service will set <b>DeviceEnabled</b> to false after it receives and enqueues data as a <b>DataEvent</b>. Before any additional input can be received, the application must set <b>DeviceEnabled</b> to true.</p> <p>If false, the Device Service does not automatically disable the device when data is received.</p> <p>This property provides the application with an additional option for controlling the receipt of input data. If an application wants to receive and process only one input, or only one input at a time, then this property should be set to true. This property applies only to event-driven input devices.</p> <p>This property is initialized to false by the <b>open</b> method.</p>
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
<b>See Also</b>	“Device Input Model” on page 22

### CapPowerReporting Property R *Added in Release 1.3*

<b>Type</b>	<b>int</b>								
<b>Remarks</b>	<p>Identifies the reporting capabilities of the Device. It has one of the following values:</p> <table> <thead> <tr> <th><b>Value</b></th> <th><b>Meaning</b></th> </tr> </thead> <tbody> <tr> <td>JPOS_PR_NONE</td> <td>The Device Service cannot determine the state of the device. Therefore, no power reporting is possible.</td> </tr> <tr> <td>JPOS_PR_STANDARD</td> <td>The Device Service can determine and report two of the power states - OFF_OFFLINE (that is, off or offline) and ONLINE.</td> </tr> <tr> <td>JPOS_PR_ADVANCED</td> <td>The Device Service can determine and report all three power states - OFF, OFFLINE, and ONLINE.</td> </tr> </tbody> </table> <p>This property is initialized by the <b>open</b> method.</p>	<b>Value</b>	<b>Meaning</b>	JPOS_PR_NONE	The Device Service cannot determine the state of the device. Therefore, no power reporting is possible.	JPOS_PR_STANDARD	The Device Service can determine and report two of the power states - OFF_OFFLINE (that is, off or offline) and ONLINE.	JPOS_PR_ADVANCED	The Device Service can determine and report all three power states - OFF, OFFLINE, and ONLINE.
<b>Value</b>	<b>Meaning</b>								
JPOS_PR_NONE	The Device Service cannot determine the state of the device. Therefore, no power reporting is possible.								
JPOS_PR_STANDARD	The Device Service can determine and report two of the power states - OFF_OFFLINE (that is, off or offline) and ONLINE.								
JPOS_PR_ADVANCED	The Device Service can determine and report all three power states - OFF, OFFLINE, and ONLINE.								
<b>Errors</b>	None.								
<b>See Also</b>	“Device Power Reporting Model” on page 27; <b>PowerState</b> Property, <b>PowerNotify</b> Property								

## CheckHealthText Property R

<b>Type</b>	<b>String</b>
<b>Remarks</b>	<p>Holds the results of the most recent call to the <b>checkHealth</b> method. The following examples illustrate some possible diagnoses:</p> <ul style="list-style-type: none"><li>• “Internal HCheck: Successful”</li><li>• “External HCheck: Not Responding”</li><li>• “Interactive HCheck: Complete”</li></ul> <p>This property is empty (“”) before the first call to the <b>checkHealth</b> method.</p>
<b>Errors</b>	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15</p>
<b>See Also</b>	<b>checkHealth</b> Method

## Claimed Property R

<b>Type</b>	<b>boolean</b>
<b>Remarks</b>	<p>If true, the device is claimed for exclusive access. If false, the device is released for sharing with other applications.</p> <p>Many devices must be claimed before the Control will allow access to many of its methods and properties, and before it will deliver events to the application.</p> <p>This property is initialized to false by the <b>open</b> method.</p>
<b>Errors</b>	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>
<b>See Also</b>	<p>“Device Initialization and Finalization” on page 10, “Device Sharing Model” on page 12, <b>claim</b> Method, <b>release</b> Method</p>

## DataCount Property R

<b>Type</b>	<b>int</b>
<b>Remarks</b>	<p>Holds the number of enqueued <b>DataEvents</b>.</p> <p>The application may read this property to determine whether additional input is enqueued from a device, but has not yet been delivered because of other application processing, freezing of events, or other causes.</p> <p>This property is initialized to zero by the <b>open</b> method.</p>
<b>Errors</b>	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>
<b>See Also</b>	<p>“Device Input Model” on page 22, <b>DataEvent</b></p>

## DataEventEnabled Property R/W

Type	<b>boolean</b>
Remarks	<p>If true, a <b>DataEvent</b> will be delivered as soon as input data is enqueued. If changed to true and some input data is already queued, then a <b>DataEvent</b> is delivered immediately. (Note that other conditions may delay “immediate” delivery: if <b>FreezeEvents</b> is true or another event is already being processed at the application, the <b>DataEvent</b> will remain queued at the Device Service until the condition is corrected.)</p> <p>If false, input data is enqueued for later delivery to the application. Also, if an input error occurs, the <b>ErrorEvent</b> is not delivered while this property is false.</p> <p>This property is initialized to false by the <b>open</b> method.</p>
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	“Events” on page 18, <b>DataEvent</b>

## DeviceControlDescription Property R

Type	<b>String</b>
Remarks	<p>Holds an identifier for the Device Control and the company that produced it.</p> <p>A sample returned string is:</p> <pre>“POS Printer JavaPOS Control, (C) 1998 Epson”</pre> <p>This property is always readable.</p>
Errors	None.
See Also	<b>DeviceControlVersion</b> Property

**DeviceControlVersion Property R****Type**      **int****Remarks**      Holds the Device Control version number.

Three version levels are specified, as follows:

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

A sample version number is:

1002038

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

This property is always readable.

**Errors**      None.**See Also**      “Version Handling” on page 32, **DeviceControlDescription** Property

## DeviceEnabled Property R/W

<b>Type</b>	<b>boolean</b>
<b>Remarks</b>	<p>If true, the device is in an operational state. If changed to true, then the device is brought to an operational state.</p> <p>If false, the device has been disabled. If changed to false, then the device is physically disabled when possible, any subsequent input will be discarded, and output operations are disallowed.</p> <p>Changing this property usually does not physically affect output devices. For consistency, however, the application must set this property to true before using output devices.</p> <p><b>Release 1.3 and later:</b> The Device's power state may be reported while <b>DeviceEnabled</b> is true; See "Device Power Reporting Model" on page 27 for details.</p> <p>This property is initialized to false by the <b>open</b> method. Note that an exclusive use device must be claimed before the device may be enabled.</p>
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see "Exceptions" on page 15.
<b>See Also</b>	"Device Initialization and Finalization" on page 10

## DeviceServiceDescription Property R

<b>Type</b>	<b>String</b>
<b>Remarks</b>	<p>Holds an identifier for the Device Service and the company that produced it.</p> <p>A sample returned string is:</p> <p style="padding-left: 40px;">"TM-U950 Printer JPOS Service Driver, (C) 1998 Epson"</p> <p>This property is initialized by the <b>open</b> method.</p>
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see "Exceptions" on page 15

**DeviceServiceVersion Property R****Type**      **int****Remarks**      Holds the Device Service version number.

Three version levels are specified, as follows:

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

A sample version number is:

1002038

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

This property is initialized by the **open** method.**Errors**      A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.**See Also**      “Version Handling” on page 32, **DeviceServiceDescription** Property



## FreezeEvents Property R/W

<b>Type</b>	<b>boolean</b>
<b>Remarks</b>	<p>If true, events will not be delivered. Events will be enqueued until this property is set to false.</p> <p>If false, the application allows events to be delivered. If some events have been held while events were frozen and all other conditions are correct for delivering the events, then changing this property to false will allow these events to be delivered. An application may choose to freeze events for a specific sequence of code where interruption by an event is not desirable.</p> <p>This property is initialized to false by the <b>open</b> method.</p>
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

## OutputID Property R

<b>Type</b>	<b>int</b>
<b>Remarks</b>	<p>Holds the identifier of the most recently started asynchronous output.</p> <p>When a method successfully initiates an asynchronous output, the Device assigns an identifier to the request. When the output completes, an <b>OutputCompleteEvent</b> will be enqueued with this output ID as a parameter.</p> <p>The output ID numbers are assigned by the Device and are guaranteed to be unique among the set of outstanding asynchronous outputs. No other facts about the ID should be assumed.</p>
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
<b>See Also</b>	“Device Output Models” on page 25, <b>OutputCompleteEvent</b>

**PowerNotify Property R/W Added in Release 1.3**

**Type**            **int**

**Remarks**       Contains the type of power notification selection made by the Application. It has one of the following values:

<b>Value</b>	<b>Meaning</b>
JPOS_PN_DISABLED	The Device Service will not provide any power notifications to the application. No power notification <b>StatusUpdateEvents</b> will be fired, and <b>PowerState</b> may not be set.
JPOS_PN_ENABLED	The Device Service will fire power notification <b>StatusUpdateEvents</b> and update <b>PowerState</b> , beginning when <b>DeviceEnabled</b> is set to true. The level of functionality depends upon <b>CapPowerReporting</b> .

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

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

**Errors**            A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
JPOS_E_ILLEGAL	One of the following occurred:  The device is already enabled.  <b>PowerNotify</b> = JPOS_PN_ENABLED but <b>CapPowerReporting</b> = JPOS_PR_NONE.

**See Also**        “Device Power Reporting Model” on page 27; **CapPowerReporting** Property, **PowerState** Property

**PowerState Property R** *Added in Release 1.3*

**Type**        **int**

**Remarks**    Identifies the current power condition of the device, if it can be determined. It has one of the following values:

<b>Value</b>	<b>Meaning</b>
JPOS_PS_UNKNOWN	Cannot determine the device's power state for one of the following reasons:  <b>CapPowerReporting</b> = JPOS_PR_NONE; the device does not support power reporting.  <b>PowerNotify</b> = JPOS_PN_DISABLED; power notifications are disabled.  <b>DeviceEnabled</b> = false; Power state monitoring does not occur until the device is enabled.
JPOS_PS_ONLINE	The device is powered on and ready for use. Can be returned if <b>CapPowerReporting</b> = JPOS_PR_STANDARD or JPOS_PR_ADVANCED.
JPOS_PS_OFF	The device is powered off or detached from the POS terminal. Can only be returned if <b>CapPowerReporting</b> = JPOS_PR_ADVANCED.
JPOS_PS_OFFLINE	The device is powered on but is either not ready or not able to respond to requests. Can only be returned if <b>CapPowerReporting</b> = JPOS_PR_ADVANCED.
JPOS_PS_OFF_OFFLINE	The device is either off or offline. Can only be returned if <b>CapPowerReporting</b> = JPOS_PR_STANDARD.

This property is initialized to JPOS\_PS\_UNKNOWN by the **open** method. When **PowerNotify** is set to enabled and **DeviceEnabled** is true, then this property is updated as the Device Service detects power condition changes.

**Errors**        None.

**See Also**     “Device Power Reporting Model” on page 27; **CapPowerReporting** Property, **PowerNotify** Property

## PhysicalDeviceDescription Property R

Type	<b>String</b>
Remarks	<p>Holds an identifier for the physical device.</p> <p>A sample returned string is:</p> <pre>"NCR 7192-0184 Printer, Japanese Version"</pre> <p>This property is initialized by the <b>open</b> method.</p>
Errors	<p>A JposException may be thrown when this property is accessed. For further information, see "Exceptions" on page 15.</p>
See Also	<b>PhysicalDeviceName</b> Property

## PhysicalDeviceName Property R

Type	<b>String</b>
Remarks	<p>Holds a short name identifying the physical device. This is a short version of <b>PhysicalDeviceDescription</b> and should be limited to 30 characters.</p> <p>This property will typically be used to identify the device in an application message box, where the full description is too verbose. A sample returned string is:</p> <pre>"IBM Model II Printer, Japanese"</pre> <p>This property is initialized by the <b>open</b> method.</p>
Errors	<p>A JposException may be thrown when this property is accessed. For further information, see "Exceptions" on page 15.</p>
See Also	<b>PhysicalDeviceDescription</b> Property

**State Property R**

**Type**        **int**

**Remarks**    Holds the current state of the Device. It has one of the following values:

<b>Value</b>	<b>Meaning</b>
JPOS_S_CLOSED	The Device is closed.
JPOS_S_IDLE	The Device is in a good state and is not busy.
JPOS_S_BUSY	The Device is in a good state and is busy performing output.
JPOS_S_ERROR	An error has been reported, and the application must recover the Device to a good state before normal I/O can resume.

This property is always readable.

**Errors**        None.

**See Also**      “Device States” on page 30

## Methods

### checkHealth Method

**Syntax**      **void checkHealth (int *level*) throws JposException;**

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

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

**Remarks**      Tests the state of a device.

A text description of the results of this method is placed in the **CheckHealthText** property. The health of many devices can only be determined by a visual inspection of these test results.

This method is always synchronous.

**Errors**      A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_ILLEGAL	The specified health check level is not supported by the Device Service.

**See Also**      **CheckHealthText** Property

## claim Method

**Syntax**      **void claim (int *timeout*) throws JposException;**

The *timeout* parameter gives the maximum number of milliseconds to wait for exclusive access to be satisfied. If zero, then immediately either returns (if successful) or throws an appropriate exception. If JPOS\_FOREVER (-1), the method waits as long as needed until exclusive access is satisfied.

**Remarks**      Requests exclusive access to the device. Many devices require an application to claim them before they can be used.

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

**Errors**          A JposException may be thrown when this method is invoked. For further information, “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_ILLEGAL	This device cannot be claimed for exclusive access, or an invalid <i>timeout</i> parameter was specified.
JPOS_E_TIMEOUT	Another application has exclusive access to the device, and did not relinquish control before <i>timeout</i> milliseconds expired.

**See Also**      “Device Sharing Model” on page 12, **release** Method

## clearInput Method

**Syntax**      **void clearInput () throws JposException;**

**Remarks**      Clears all device input that has been buffered.

Any data events or input error events that are enqueued – usually waiting for **DataEventEnabled** to be set to true and **FreezeEvents** to be set to false – are also cleared.

**Errors**          A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

**See Also**      “Device Input Model” on page 22

## clearOutput Method

<b>Syntax</b>	<b>void clearOutput () throws JposException;</b>
<b>Remarks</b>	<p>Clears all device output that has been buffered. Also, when possible, halts outputs that are in progress.</p> <p>Any output error events that are enqueued – usually waiting for <b>FreezeEvents</b> to be set to false – are also cleared.</p>
<b>Errors</b>	A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.
<b>See Also</b>	“Device Output Models” on page 25

## close Method

<b>Syntax</b>	<b>void close () throws JposException;</b>
<b>Remarks</b>	<p>Releases the device and its resources.</p> <p>If the <b>DeviceEnabled</b> property is true, then the device is disabled.</p> <p>If the <b>Claimed</b> property is true, then exclusive access to the device is released.</p>
<b>Errors</b>	A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.
<b>See Also</b>	“Device Initialization and Finalization” on page 10, <b>open</b> Method



**directIO Method**

**Syntax**      **void directIO (int *command*, int[] *data*, Object *object*) throws JposException;**

<b>Parameter</b>	<b>Description</b>
<i>command</i>	Command number whose specific values are assigned by the Device Service.
<i>data</i>	An array of one modifiable integer whose specific values or usage vary by <i>command</i> and Device Service.
<i>object</i>	Additional data whose usage varies by <i>command</i> and Device Service.

**Remarks**      Communicates directly with the Device Service.

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

Use of this method will make an application non-portable. The application may, however, maintain portability by performing **directIO** calls within conditional code. This code may be based upon the value of the **DeviceServiceDescription**, **PhysicalDeviceDescription**, or **PhysicalDeviceName** property.

**Errors**      A JposException may be thrown when this method is invoked. For further information, see "Exceptions" on page 15.

**See Also**      **DirectIOEvent**

**open Method**

**Syntax**      **void open(String *logicalDeviceName*) throws JposException;**

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

**Remarks**      Opens a device for subsequent I/O.

The device name specifies which of one or more devices supported by this Device Control should be used.

In Controls from version 1.4 and prior, The *logicalDeviceName* must exist in the Java System Database (JSD) for this device category so that its relationship to the physical device can be determined. Entries in the JSD are created by a setup or configuration utility.

In Controls from version 1.5 and beyond, The *logicalDeviceName* must exist in the **JposEntryRegistry** for this device category so that its relationship to the physical device can be determined. JposEntry objects in the registry are created by a populator or some configuration utility like the JCL GUI editor.

When this method is successful, it initializes the properties **Claimed**, **DeviceEnabled**, **DataEventEnabled** and **FreezeEvents**, as well as descriptions and version numbers of the JavaPOS software layers. Additional category-specific properties may also be initialized.

**Errors**      A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
JPOS_E_ILLEGAL	The Control is already open.
JPOS_E_NOEXIST	The specified <i>logicalDeviceName</i> was not found.
JPOS_E_NOSERVICE	Could not establish a connection to the corresponding Device Service.

**See Also**      “Device Initialization and Finalization” on page 10, “Version Handling” on page 32, **close** Method

## release Method

**Syntax**      **void release () throws JposException;**

**Remarks**      Releases exclusive access to the device.

If the **DeviceEnabled** property is true, and the device is an exclusive-use device, then the device is also disabled (this method does not change the device enabled state of sharable devices).

**Errors**      A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
JPOS_E_ILLEGAL	The application does not have exclusive access to the device.

**See Also**      “Device Sharing Model” on page 12, **claim** Method

## Events

### DataEvent

**Interface** `jpos.events.DataListener`

**Method** `dataOccurred (DataEvent e)`

**Description** Notifies the application that input data is available from the device.

**Properties** This event contains the following property:

Property	Type	Description
<i>Status</i>	<i>int</i>	The input status with its value dependent upon the device category; it may describe the type or qualities of the input data.

**Remarks** When this event is delivered to the application, the **DataEventEnabled** property is changed to false, so that no further data events will be delivered until the application sets **DataEventEnabled** back to true. The actual *byte array* input data is placed in one or more device-specific properties.

If **DataEventEnabled** is false at the time that data is received, then the data is enqueued in an internal buffer, the device-specific input data properties are not updated, and the event is not delivered. When **DataEventEnabled** is subsequently changed back to true, the event will be delivered immediately if input data is enqueued and **FreezeEvents** is false.

**See Also** “Events” on page 18, “Device Input Model” on page 22, **DataEventEnabled** Property, **FreezeEvents** Property

## DirectIOEvent

**Interface** `jpos.events.DirectIOListener`

**Method** `directIOOccurred (DirectIOEvent e);`

**Description** Provides Device Service information directly to the application. This event provides a means for a vendor-specific Device Service to provide events to the application that are not otherwise supported by the Device Control.

**Properties** This event contains the following properties:

Property	Type	Description
<i>EventNumber</i>	<i>int</i>	Event number whose specific values are assigned by the Device Service.
<i>Data</i>	<i>int</i>	Additional numeric data. Specific values vary by the <i>EventNumber</i> and the Device Service. This property is settable.
<i>Object</i>	<i>Object</i>	Additional data whose usage varies by the <i>EventNumber</i> and the Device Service. This property is settable.

**Remarks** This event is to be used only for those types of vendor specific functions that are not otherwise described as part of the JavaPOS standard. Use of this event may restrict the application program from being used with other vendor's devices which may not have any knowledge of the Device Service's need for this event.

**See Also** "Events" on page 18, **directIO** Method

**ErrorEvent**

<b>Interface</b>	<b>jpos.events.ErrorListener</b>
<b>Method</b>	<b>errorOccurred (ErrorEvent e);</b>
<b>Description</b>	Notifies the application that an error has been detected and a suitable response is necessary to process the error condition.
<b>Properties</b>	This event contains the following properties:

Property	Type	Description
<i>ErrorCode</i>	<i>int</i>	Error Code causing the error event. See the list of <i>ErrorCodes</i> on page 16.
<i>ErrorCodeExtended</i>	<i>int</i>	Extended Error Code causing the error event. These values are device category specific.
<i>ErrorLocus</i>	<i>int</i>	Location of the error. See values below.
<i>ErrorResponse</i>	<i>int</i>	Error response, whose default value may be overridden by the application (i.e., this property is settable). See values below.

The *ErrorLocus* parameter has one of the following values:

Value	Meaning
JPOS_EL_OUTPUT	Error occurred while processing asynchronous output.
JPOS_EL_INPUT	Error occurred while gathering or processing event-driven input. No input data is available.
JPOS_EL_INPUT_DATA	Error occurred while gathering or processing event-driven input, and some previously buffered data is available.

The application's error event listener can set the *ErrorResponse* property to one of the following values:

Value	Meaning
JPOS_ER_RETRY	Retry the asynchronous output. The error state is exited. May be valid only when locus is JPOS_EL_INPUT. Default when locus is JPOS_EL_OUTPUT.
JPOS_ER_CLEAR	Clear the asynchronous output or buffered input data. The error state is exited. Default when locus is JPOS_EL_INPUT.

**JPOS\_ER\_CONTINUEINPUT**

Acknowledges the error and directs the Device to continue input processing. The Device remains in the error state and will deliver additional **DataEvents** as directed by the **DataEventEnabled** property. When all input has been delivered and **DataEventEnabled** is again set to true, then another **ErrorEvent** is delivered with locus JPOS\_EL\_INPUT.

Use only when locus is JPOS\_EL\_INPUT\_DATA.  
Default when locus is JPOS\_EL\_INPUT\_DATA.

**Remarks** This event is enqueued when an error is detected and the Device's **State** transitions into the error state. This event is not delivered until **DataEventEnabled** is true, so that proper application sequencing occurs.

**See Also** "Device Input Model" on page 22, "Device Input Model" on page 22, "Device States" on page 30

**OutputCompleteEvent**

**Interface** **jpos.events.OutputCompleteListener**

**Method** **outputCompleteOccurred (OutputCompleteEvent e);**

**Description** Notifies the application that the queued output request associated with the *OutputID* property has completed successfully.

**Properties** This event contains the following property:

<b>Property</b>	<b>Type</b>	<b>Description</b>
<i>OutputID</i>	<i>int</i>	The ID number of the asynchronous output request that is complete.

**Remarks** This event is enqueued after the request's data has been both sent and the Device Service has confirmation that it was processed by the device successfully.

**See Also** "Device Output Models" on page 25

## StatusUpdateEvent

**Interface**     `jpos.events.StatusUpdateListener`

**Method**        `statusUpdateOccurred (StatusUpdateEvent e);`

**Description**    Notifies the application when a device has detected an operation status change.

**Properties**      This event contains the following property:

Property	Type	Description
<i>Status</i>	<i>int</i>	Device category-specific status, describing the type of status change.

*Note that Release 1.3* added Power State Reporting with additional *Status* values of:

Value	Meaning
JPOS_SUE_POWER_ONLINE	The device is powered on and ready for use. Can be returned if <b>CapPowerReporting</b> = JPOS_PR_STANDARD or JPOS_PR_ADVANCED.
JPOS_SUE_POWER_OFF	The device is off or detached from the terminal. Can only be returned if <b>CapPowerReporting</b> = JPOS_PR_ADVANCED.
JPOS_SUE_POWER_OFFLINE	The device is powered on but is either not ready or not able to respond to requests. Can only be returned if <b>CapPowerReporting</b> = JPOS_PR_ADVANCED.
POS_SUE_POWER_OFF_OFFLINE	The device is either off or offline. Can only be returned if <b>CapPowerReporting</b> = JPOS_PR_STANDARD.  The common property <b>PowerState</b> is also maintained at the current power state of the device.

**Remarks**        This event is enqueued when a Device needs to alert the application of a device status change. Examples are a change in the cash drawer position (open vs. closed) or a change in a POS printer sensor (form present vs. absent).

When a device is enabled, this event may be delivered to inform the application of the device state. This behavior, however, is not required.

**See Also**        “Events” on page 18, “Device Power Reporting Model” on page 27, **CapPowerReporting** Property, **PowerNotify** Property.



# Bump Bar

## Summary

### Properties

<i>Common</i>	<i>Ver</i>	<i>Type</i>	<i>Access</i>	<i>May Use After</i>
<b>AutoDisable</b>	1.3	boolean	R/W	<i>Not Supported</i>
<b>CapPowerReporting</b>	1.3	int	R	open
<b>CheckHealthText</b>	1.3	String	R	open
<b>Claimed</b>	1.3	boolean	R	open
<b>DataCount</b>	1.3	int	R	open
<b>DataEventEnabled</b>	1.3	boolean	R/W	open
<b>DeviceEnabled</b>	1.3	boolean	R/W	open & claim
<b>FreezeEvents</b>	1.3	boolean	R/W	open
<b>OutputID</b>	1.3	int	R	open
<b>PowerNotify</b>	1.3	int	R/W	open
<b>PowerState</b>	1.3	int	R	open
<b>State</b>	1.3	int	R	--
<b>DeviceControlDescription</b>	1.3	String	R	--
<b>DeviceControlVersion</b>	1.3	int	R	--
<b>DeviceServiceDescription</b>	1.3	String	R	open
<b>DeviceServiceVersion</b>	1.3	int	R	open
<b>PhysicalDeviceDescription</b>	1.3	String	R	open
<b>PhysicalDeviceName</b>	1.3	String	R	open

**Properties (Continued)**

<i>Specific</i>	<i>Ver</i>	<i>Type</i>	<i>Access</i>	<i>May Use After</i>
<b>AsyncMode</b>	1.3	boolean	R/W	open, claim, & enable
<b>Timeout</b>	1.3	int	R/W	open
<b>UnitsOnline</b>	1.3	int	R	open, claim, & enable
<b>CurrentUnitID</b>	1.3	int	R/W	open, claim, & enable
<b>CapTone</b>	1.3	boolean	R	open, claim, & enable
<b>AutoToneDuration</b>	1.3	int	R/W	open, claim, & enable
<b>AutoToneFrequency</b>	1.3	int	R/W	open, claim, & enable
<b>BumpBarDataCount</b>	1.3	int	R	open, claim, & enable
<b>Keys</b>	1.3	int	R	open, claim, & enable
<b>ErrorUnits</b>	1.3	int	R	open
<b>ErrorString</b>	1.3	String	R	open
<b>EventUnitID</b>	1.3	int	R	open & claim
<b>EventUnits</b>	1.3	int	R	open & claim
<b>EventString</b>	1.3	String	R	open & claim

**Methods***Common*

	<i>Ver</i>	<i>May Use After</i>
<b>open</b>	1.3	--
<b>close</b>	1.3	open
<b>claim</b>	1.3	open
<b>release</b>	1.3	open & claim
<b>checkHealth</b>	1.3	open, claim, & enable
<b>clearInput</b>	1.3	open & claim
<b>clearOutput</b>	1.3	open & claim
<b>directIO</b>	1.3	open

*Specific*

<b>bumpBarSound</b>	1.3	open, claim, & enable
<b>setKeyTranslation</b>	1.3	open, claim, & enable

**Events**

<i>Name</i>	<i>Ver</i>	<i>May Occur After</i>
<b>DataEvent</b>	1.3	open, claim, & enable
<b>DirectIOEvent</b>	1.3	open & claim
<b>ErrorEvent</b>	1.3	open, claim, & enable
<b>OutputCompleteEvent</b>	1.3	open, claim, & enable
<b>StatusUpdateEvent</b>	1.3	open, claim, & enable

## General Information

The Bump Bar Control's class name is "jpos.BumpBar".  
The device constants are contained in the class "jpos.BumpBarConst".  
See "Package Structure" on page 40.

*This device was added in JavaPOS Release 1.3.*

### Capabilities

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

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

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

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

## Model

The general model of a bump bar is an input device but may also be an output device when, in some implementations, the bump bar device can produce an output tone to indicate that it has been depressed:

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

The subsystem can support up to 32 bump bar units.

One application on one workstation or POS Terminal will typically manage and control the entire subsystem of bump bars. If applications on the same or other workstations and POS Terminals will need to access the subsystem, then this application must act as a subsystem server and expose interfaces to other applications.

- All specific methods are broadcast methods. This means that the method can apply to one unit, a selection of units or all online units. The *units* parameter is an **int**, with each bit identifying an individual bump bar unit. (One or more of the constants `BB_UID_1` through `BB_UID_32` are bitwise ORed to form the bitmask.) The Device Service will attempt to satisfy the method for all unit(s) indicated in the *units* parameter. If an error is received from one or more units, the **ErrorUnits** property is updated with the appropriate units in error. The **ErrorString** property is updated with a description of the error or errors received. The method will then throw the corresponding `JposException`. In the case where two or more units encounter different errors, the Device Service should determine the most severe `JposException` to throw.
- The common methods **checkHealth**, **clearInput**, and **clearOutput** are not broadcast methods and use the unit ID indicated in the **CurrentUnitID** property. (One of the constants `BB_UID_1` through `BB_UID_32` are selected.) See the description of these common methods to understand how the current unit ID property is used.
- When the current unit ID property is set by the application, all the corresponding properties are updated to reflect the settings for that unit.

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

The **CurrentUnitID** uniquely represents a single bump bar unit. The definitions range from `BB_UID_1` to `BB_UID_32`. These definitions are also used to create the bitwise parameter, *units*, used in the broadcast methods.

## Input – Bump Bar

The Bump Bar follows the general “Device Input Model” for event-driven input with some differences:

- When input is received, a **DataEvent** is enqueued.
- This device does not support the **AutoDisable** property, so the device will not automatically disable itself when a **DataEvent** is enqueued.
- An enqueued **DataEvent** can be delivered to the application when the **DataEventEnabled** property is true and other event delivery requirements are met. Just before delivering this event, data is copied into corresponding properties, and further data events are disabled by setting the **DataEventEnabled** property to false. This causes subsequent input data to be enqueued while the application processes the current input and associated properties. When the application has finished the current input and is ready for more data, it reenables events by setting **DataEventEnabled** to true.
- An **ErrorEvent** or events are enqueued if an error is encountered while gathering or processing input, and are delivered to the application when the **DataEventEnabled** property is true and other event delivery requirements are met.
- The **BumpBarDataCount** property may be read to obtain the number of bump bar **DataEvents** for a specific unit ID enqueued. The **DataCount** property can be read to obtain the total number of data events enqueued.
- Queued input may be deleted by calling the **clearInput** method. See **clearInput** method description for more details.

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

## Output – Tone

The bump bar follows the general “Device Output Model,” with some enhancements:

- The **bumpBarSound** method is performed either synchronously or asynchronously, depending on the value of the **AsyncMode** property. When **AsyncMode** is false, then this method operates synchronously and the Device returns to the application after completion. When operating synchronously, a **JposException** is thrown if the method could not complete successfully.
- When **AsyncMode** is true, then this method operates as follows:
  - The Device buffers the request, sets the **OutputID** property to an identifier for this request, and returns as soon as possible. When the device completes the request successfully, the **EventUnits** property is updated and an **OutputCompleteEvent** is enqueued. A property of this event contains the output ID of the completed request.

Asynchronous methods will not throw a **JposException** due to a bump bar problem, such as communications failure. These errors will only be reported by an **ErrorEvent**. A **JposException** is thrown only if the bump bar is not claimed and enabled, a parameter is invalid, or the request cannot be enqueued. The first two error cases are due to an application error, while the last is a serious system resource exception.

- If an error occurs while performing an asynchronous request, an **ErrorEvent** is enqueued. The **EventUnits** property is set to the unit or units in error. The **EventString** property is also set.  
*Note: **ErrorEvent** updates **EventUnits** and **EventString**. If an error is reported by a broadcast method, then **ErrorUnits** and **ErrorString** are set instead.*

The event handler may call synchronous bump bar methods (but not asynchronous methods), then can either retry the outstanding output or clear it.

- Asynchronous output is performed on a first-in first-out basis.
- All output buffered may be deleted by setting the **CurrentUnitID** property and calling the **clearOutput** method. **OutputCompleteEvents** will not be enqueued for cleared output. This method also stops any output that may be in progress (when possible).



## Device Sharing

The bump bar is an exclusive-use device, as follows:

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

## Properties

### AsyncMode Property R/W

Type	<b>boolean</b>
Remarks	If true, then the bumpBarSound method will be performed asynchronously. If false, tones are generated synchronously.  This property is initialized to false by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>bumpBarSound</b> Method, “Device Output Models” on page 25

### AutoToneDuration Property R/W

Type	<b>int</b>
Remarks	Holds the duration (in milliseconds) of the automatic tone for the bump bar unit specified by the <b>CurrentUnitID</b> property.  This property is initialized to the default value for each online bump bar unit when the device is first enabled following the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15
See Also	<b>CurrentUnitID</b> Property

### AutoToneFrequency Property R/W

Type	<b>int</b>
Remarks	Holds the frequency (in Hertz) of the automatic tone for the bump bar unit specified by the <b>CurrentUnitID</b> property.  This property is initialized to the default value for each online bump bar unit when the device is first enabled following the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>CurrentUnitID</b> Property

## BumpBarDataCount Property R

Type	int
Remarks	<p>Holds the number of <b>DataEvents</b> enqueued for the bump bar unit specified by the <b>CurrentUnitID</b> property.</p> <p>The application may read this property to determine whether additional input is enqueued from a bump bar unit, but has not yet been delivered because of other application processing, freezing of events, or other causes.</p> <p>This property is initialized to zero by the <b>open</b> method.</p>
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>CurrentUnitID</b> Property, <b>DataEvent</b>

## CapTone Property R

Type	boolean
Remarks	<p>If true, the bump bar unit specified by the <b>CurrentUnitID</b> property supports an enunciator.</p> <p>This property is initialized when the device is first enabled following the <b>open</b> method.</p>
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>CurrentUnitID</b> Property

## CurrentUnitID Property R/W

Type	int
Remarks	<p>Holds the current bump bar unit ID. Up to 32 units are allowed for one bump bar device. The unit ID definitions range from BB_UID_1 to BB_UID_32.</p> <p>Setting this property will update other properties to the current values that apply to the specified unit. The following properties and methods apply only to the selected bump bar unit ID:</p> <ul style="list-style-type: none"><li>• Properties: <b>AutoToneDuration</b>, <b>AutoToneFrequency</b>, <b>BumpBarDataCount</b>, <b>CapTone</b>, and <b>Keys</b>.</li><li>• Methods: <b>checkHealth</b>, <b>clearInput</b>, <b>clearOutput</b>.</li></ul> <p>This property is initialized to BB_UID_1 when the device is first enabled following the <b>open</b> method.</p>
Errors	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>

## DataCount Property (Common) R

Type	int
Remarks	<p>Holds the total number of <b>DataEvents</b> enqueued. All units online are included in this value. The number of enqueued events for a specific unit ID is stored in the <b>BumpBarDataCount</b> property.</p> <p>The application may read this property to determine whether additional input is enqueued, but has not yet been delivered because of other application processing, freezing of events, or other causes.</p> <p>This property is initialized to zero by the <b>open</b> method.</p>
Errors	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>
See Also	<p><b>BumpBarDataCount</b> Property, <b>DataEvent</b> Event, “Device Input Model” on page 22.</p>

## ErrorString Property R

Type	<b>String</b>
Remarks	<p>Holds a description of the error which occurred on the unit(s) specified by the <b>ErrorUnits</b> property, when an error occurs for any method that acts on a bitwise set of bump bar units.</p> <p>If an error occurs during processing of an asynchronous request, the <b>ErrorEvent</b> updates the property <b>EventString</b> instead.</p> <p>This property is initialized to an empty string by the <b>open</b> method.</p>
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>ErrorUnits</b> Property

## ErrorUnits Property R

Type	<b>int</b>
Remarks	<p>Holds a bitwise mask of the unit(s) that encountered an error, when an error occurs for any method that acts on a bitwise set of bump bar units.</p> <p>If an error occurs during processing of an asynchronous request, the <b>ErrorEvent</b> updates the property <b>EventUnits</b> instead.</p> <p>This property is initialized to zero by the <b>open</b> method.</p>
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>ErrorString</b> Property

## EventString Property R

Type	<b>String</b>
Remarks	<p>Holds a description of the error which occurred to the unit(s) specified by the <b>EventUnits</b> property, when an <b>ErrorEvent</b> is delivered.</p> <p>This property is initialized to an empty string by the <b>open</b> method.</p>
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>EventUnits</b> Property, <b>ErrorEvent</b>

### EventUnitID Property R

Type	int
Remarks	Holds the bump bar unit ID causing a <b>DataEvent</b> . This property is set just before a <b>DataEvent</b> is delivered. The unit ID definitions range from BB_UID_1 to BB_UID_32.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>DataEvent</b>

### EventUnits Property R

Type	int
Remarks	Holds a bitwise mask of the unit(s) when an <b>OutputCompleteEvent</b> , <b>ErrorEvent</b> , or <b>StatusUpdateEvent</b> is delivered.  This property is initialized to zero by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>OutputCompleteEvent</b> , <b>ErrorEvent</b> , <b>StatusUpdateEvent</b>

### Keys Property R

Type	int
Remarks	Holds the number of keys on the bump bar unit specified by the <b>CurrentUnitID</b> property.  This property is initialized when the device is first enabled following the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>CurrentUnitID</b> Property

### Timeout Property R/W

Type	int
Remarks	<p>Holds the timeout value in milliseconds used by the bump bar device to complete all output methods supported. If the device cannot successfully complete an output method within the timeout value, then the method throws a <code>JposException</code> if <code>AsyncMode</code> is false, or enqueues an <code>ErrorEvent</code> if <code>AsyncMode</code> is true.</p> <p>This property is initialized to a Device Service dependent timeout following the <code>open</code> method.</p>
Errors	A <code>JposException</code> may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<code>AsyncMode</code> Property, <code>ErrorString</code> Property, <code>bumpBarSound</code> Method

### UnitsOnline Property R

Type	int
Remarks	<p>Bitwise mask indicating the bump bar units online, where zero or more of the unit constants <code>BB_UID_1</code> (bit 0 on) through <code>BB_UID_32</code> (bit 31 on) are bitwise ORed. 32 units are supported.</p> <p>This property is initialized when the device is first enabled following the <code>open</code> method. This property is updated as changes are detected, such as before a <code>StatusUpdateEvent</code> is enqueued and during the <code>checkHealth</code> method.</p>
Errors	A <code>JposException</code> may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<code>checkHealth</code> Method, <code>StatusUpdateEvent</code>





**Errors** A `JposException` may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s `ErrorCode` property are:

Value	Meaning
JPOS_E_ILLEGAL	<p>One of the following errors occurred:</p> <p><i>numberOfCycles</i> is neither a positive, non-zero value nor JPOS_FOREVER.</p> <p><i>numberOfCycles</i> is JPOS_FOREVER when <b>AsyncMode</b> is false.</p> <p>A negative <i>interSoundWait</i> was specified.</p> <p><i>units</i> is zero or a non-existent unit was specified.</p> <p>A unit in <i>units</i> does not support the <b>CapTone</b> capability.</p> <p>The <b>ErrorUnits</b> and <b>ErrorString</b> properties may be updated before the exception is thrown.</p>
JPOS_E_FAILURE	<p>An error occurred while communicating with one of the bump bar units specified by the <i>units</i> parameter. The <b>ErrorUnits</b> and <b>ErrorString</b> properties are updated before the exception is thrown. (Can only occur if <b>AsyncMode</b> is false.)</p>

**See Also** **AsyncMode** Property, **ErrorUnits** Property, **ErrorString** Property, **CapTone** Property, **clearOutput** Method

**checkHealth Method (Common)**

**Type**      **void checkHealth (int *level*) throws JposException;**

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

Value	Meaning
JPOS_CH_INTERNAL	Perform a health check that does not physically change the device. The device is tested by internal tests to the extent possible.
JPOS_CH_EXTERNAL	Perform a more thorough test that may change the device.
JPOS_CH_INTERACTIVE	Perform an interactive test of the device. The Device Service will typically display a modal dialog box to present test options and results.

**Remarks**      When JPOS\_CH\_INTERNAL or JPOS\_CH\_EXTERNAL level is requested, the method will check the health of the bump bar unit specified by the **CurrentUnitID** property. When the current unit ID property is set to a unit that is not currently online, the device will attempt to check the health of the bump bar unit and report a communication error if necessary. The JPOS\_CH\_INTERACTIVE health check operation is up to the Device Service designer.

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

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

This method is always synchronous.

**Errors**      A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_FAILURE	An error occurred while communicating with the bump bar unit specified by the <b>CurrentUnitID</b> property.

**See Also**      **CurrentUnitID** Property, **UnitsOnline** Property

### clearInput Method (Common)

<b>Syntax</b>	<b>void clearInput () throws JposException;</b>
<b>Remarks</b>	<p>Clears the device input that has been buffered for the unit specified by the <b>CurrentUnitID</b> property.</p> <p>Any data events that are enqueued – usually waiting for <b>DataEventEnabled</b> to be set to true and <b>FreezeEvents</b> to be set to false – are also cleared.</p>
<b>Errors</b>	A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.
<b>See Also</b>	<b>CurrentUnitID</b> Property, “Device Input Model” on page 22.

### clearOutput Method (Common)

<b>Syntax</b>	<b>void clearOutput () throws JposException;</b>
<b>Remarks</b>	<p>Clears the tone outputs that have been buffered for the unit specified by the <b>CurrentUnitID</b> property.</p> <p>Any output complete and output error events that are enqueued – usually waiting for <b>DataEventEnabled</b> to be set to true and <b>FreezeEvents</b> to be set to false – are also cleared.</p>
<b>Errors</b>	A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.
<b>See Also</b>	<b>CurrentUnitID</b> Property, “Device Output Models” on page 25.



## Events

### DataEvent

**Interface** `jpos.events.DataListener`

**Method** `dataOccurred (DataEvent e);`

**Description** Notifies the application when input from the bump bar is available.

**Properties** This event contains the following property:

Property	Type	Description
----------	------	-------------

<i>Status</i>	<i>int</i>	See below.
---------------	------------	------------

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

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

**Remarks** Enqueued to present input data from a bump bar unit to the application. The low word contains the Event Type. The high word contains additional data depending on the Event Type. When the Event Type is BB\_DE\_KEY, the low byte of the high word contains the LogicalKeyCode for the key pressed on the bump bar unit. The LogicalKeyCode value is device independent. It has been translated by the Device Service from its original hardware specific value. Valid ranges are 0-255.

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

**See Also** “Device Input Model” on page 22, **EventUnitID** Property, **DataEventEnabled** Property, **FreezeEvents** Property

## DirectIOEvent

**Interface** `jpos.events.DirectIOListener`

**Method** `directIOOccurred (DirectIOEvent e);`

**Description** Provides Device Service information directly to the application. This event provides a means for a vendor-specific Bump Bar Device Service to provide events to the application that are not otherwise supported by the Device Control.

**Properties** This event contains the following property:

Property	Type	Description
<i>EventNumber</i>	<i>int</i>	Event number whose specific values are assigned by the Device Service.
<i>Data</i>	<i>int</i>	Additional numeric data. Specific values vary by the <i>EventNumber</i> and the Device Service. This property is settable.
<i>Object</i>	<i>Object</i>	Additional data whose usage varies by the <i>EventNumber</i> and Device Service. This property is settable.

**Remarks** This event is to be used only for those types of vendor specific functions that are not otherwise described as part of the JavaPOS standard. Use of this event may restrict the application program from being used with other vendor's Bump Bar devices which may not have any knowledge of the Device Service's need for this event.

**See Also** "Events" on page 18, `directIO` Method

## ErrorEvent

**Interface** `jpos.events.ErrorListener`

**Method** `errorOccurred (ErrorEvent e);`

**Description** Notifies the application that a Bump Bar error has been detected and a suitable response by the application is necessary to process the error condition.

**Properties** This event contains the following properties:

Property	Type	Description
<i>ErrorCode</i>	<i>int</i>	Result code causing the error event. See a list of <i>ErrorCodes</i> on page 16.
<i>ErrorCodeExtended</i>	<i>int</i>	Extended Error code causing the error event. If <i>ErrorCode</i> is <code>JPOS_E_EXTENDED</code> , then see values below. Otherwise, it may contain a Service-specific value.
<i>ErrorLocus</i>	<i>int</i>	Location of the error. See values below.

*ErrorResponse* *int* Error response, whose default value may be overridden by the application. (i.e., this property is settable). See values below.

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

Value	Meaning
JPOS_EL_OUTPUT	Error occurred while processing asynchronous output.
JPOS_EL_INPUT	Error occurred while gathering or processing event-driven input. No input data is available.
JPOS_EL_INPUT_DATA	Error occurred while gathering or processing event-driven input, and some previously buffered data is available.

The contents of the *ErrorResponse* property are preset to a default value, based on the *ErrorLocus*. The application's error event listener may change *ErrorResponse* to one of the following values:

Value	Meaning
JPOS_ER_RETRY	Use only when locus is JPOS_EL_OUTPUT. Retry the asynchronous output. The error state is exited. Default when locus is JPOS_EL_OUTPUT.
JPOS_ER_CLEAR	Clear the buffered input data. The error state is exited. Default when locus is JPOS_EL_INPUT.
JPOS_ER_CONTINUEINPUT	Use only when locus is JPOS_EL_INPUT_DATA. Acknowledges the error and directs the Device to continue processing. The Device remains in the error state, and will deliver additional <b>DataEvents</b> as directed by the <b>DataEventEnabled</b> property. When all input has been delivered and the <b>DataEventEnabled</b> property is again set to true, then another <b>ErrorEvent</b> is delivered with locus JPOS_EL_INPUT. Default when locus is JPOS_EL_INPUT_DATA.

**Remarks** Enqueued when an error is detected while gathering data from or processing asynchronous output for the bump bar.

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

The **EventUnits** and **EventString** properties are updated before the event is delivered.

**See Also** "Device Output Models" on page 25, "Device States" on page 30, **DataEventEnabled** Property, **EventUnits** Property, **EventString** Property

## OutputCompleteEvent

**Interface**    **jpos.events.OutputCompleteListener**

**Method**      **outputCompleteOccurred (OutputCompleteEvent e);**

**Description**    Notifies the application that the queued output request associated with the *OutputID* property has completed successfully.

**Properties**     This event contains the following property:

<b>Property</b>	<b>Type</b>	<b>Description</b>
<i>OutputID</i>	<i>int</i>	The ID number of the asynchronous output request that is complete. The EventUnits property is updated before delivering.

**Remarks**     Enqueued when a previously started asynchronous output request completes successfully.

**See Also**      **EventUnits** Property, “Device Output Models” on page 25.



## StatusUpdateEvent

**Interface** `jpos.events.StatusUpdateListener`

**Method** `statusUpdateOccurred (StatusUpdateEvent e);`

**Description** Notifies the application that the bump bar has had an operation status change.

**Properties** This event contains the following property:

Property	Type	Description
<i>Status</i>	<i>int</i>	Reports a change in the power state of a bump bar unit.

*Note that Release 1.3* added Power State Reporting with additional *Power reporting StatusUpdateEvent values*. See “StatusUpdateEvent” description on page 78.

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

Deviation from the standard **StatusUpdateEvent** (See “StatusUpdateEvent” on page 78.)

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

**See Also** **EventUnits** Property



# Cash Changer

## Summary

### Properties

<i>Common</i>	<i>Ver</i>	<i>Type</i>	<i>Access</i>	<i>May Use After</i>
<b>AutoDisable</b>		boolean	R/W	<i>Not Supported</i>
<b>CapPowerReporting</b>	1.3	int	R	open
<b>CheckHealthText</b>		String	R	open
<b>Claimed</b>		boolean	R	open
<b>DataCount</b>	1.5	int	R	<i>open</i>
<b>DataEventEnabled</b>	1.5	boolean	R/W	<i>open</i>
<b>DeviceEnabled</b>		boolean	R/W	open & claim
<b>FreezeEvents</b>		boolean	R/W	open
<b>OutputID</b>		int	R	<i>Not Supported</i>
<b>PowerNotify</b>	1.3	int	R/W	open
<b>PowerState</b>	1.3	int	R	open
<b>State</b>		int	R	--
<b>DeviceControlDescription</b>		String	R	--
<b>DeviceControlVersion</b>		int	R	--
<b>DeviceServiceDescription</b>		String	R	open
<b>DeviceServiceVersion</b>		int	R	open
<b>PhysicalDeviceDescription</b>		String	R	open
<b>PhysicalDeviceName</b>		String	R	open

<i>Specific</i>	<i>Ver</i>	<i>Type</i>	<i>Access</i>	<i>May Use After</i>
<b>CapDeposit</b>	1.5	boolean	R	open
<b>CapDepositDataEvent</b>	1.5	boolean	R	open
<b>CapDiscrepancy</b>		boolean	R	open
<b>CapEmptySensor</b>		boolean	R	open
<b>CapFullSensor</b>		boolean	R	open
<b>CapNearEmptySensor</b>		boolean	R	open
<b>CapNearFullSensor</b>		boolean	R	open
<b>CapPauseDeposit</b>	1.5	boolean	R	open
<b>CapRepayDeposit</b>	1.5	boolean	R	open
<b>AsyncMode</b>		boolean	R/W	open
<b>AsyncResultCode</b>		int	R	open, claim, & enable
<b>AsyncResultCodeExtended</b>		int	R	open, claim, & enable
<b>CurrencyCashList</b>		String	R	open
<b>CurrencyCode</b>		String	R/W	open
<b>CurrencyCodeList</b>		String	R	open
<b>CurrentExit</b>		int	R/W	open
<b>DepositAmount</b>	1.5	int	R	open
<b>DepositCashList</b>	1.5	String	R	open
<b>DepositCodeList</b>	1.5	String	R	open
<b>DepositCounts</b>	1.5	String	R	open
<b>DepositStatus</b>	1.5	int	R	open
<b>DeviceExits</b>		int	R	open
<b>ExitCashList</b>		String	R	open
<b>DeviceStatus</b>		int	R	open, claim, & enable
<b>FullStatus</b>		int	R	open, claim, & enable

**Methods**

<i>Common</i>	<i>Ver</i>	<i>May Use After</i>
<b>open</b>		--
<b>close</b>		open
<b>claim</b>		open
<b>release</b>		open & claim
<b>checkHealth</b>		open, claim, & enable
<b>clearInput</b>		<i>Not Supported</i>
<b>clearOutput</b>		<i>Not Supported</i>
<b>directIO</b>		open & claim
 <i>Specific</i>		
<b>beginDeposit</b>	1.5	open, claim, & enable
<b>dispenseCash</b>		open, claim, & enable
<b>dispenseChange</b>		open, claim, & enable
<b>endDeposit</b>	1.5	open, claim, & enable
<b>fixDeposit</b>	1.5	open, claim, & enable
<b>pauseDeposit</b>	1.5	open, claim, & enable
<b>readCashCounts</b>		open, claim, & enable

**Events**

<i>Name</i>	<i>Ver</i>	<i>May Occur After</i>
<b>DataEvent</b>	1.5	open, claim, & enable
<b>DirectIOEvent</b>	1.3	open & claim
<b>ErrorEvent</b>		<i>Not Supported</i>
<b>OutputCompleteEvent</b>		<i>Not Supported</i>
<b>StatusUpdateEvent</b>		open, claim, & enable

## General Information

The Cash Changer Control's class name is "jpos.CashChanger".  
The device constants are contained in the class "jpos.CashChangerConst".  
See "Package Structure" on page 40.

### Capabilities

The Cash Changer has the following capabilities:

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

The Cash Changer may also have the following additional capabilities:

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

### ***Release 1.5 and later – Support for the cash acceptance is added as an option.***

- The money (bills and coins) which is deposited into the device between the start and end of cash acceptance is reported to the application. The contents of the report are cash units and cash counts.

## Model

The general model of a Cash Changer is:

- Supports several cash types such as coins, bills, and combinations of coins and bills. The supported cash type for a particular currency is noted by the list of cash units in the **CurrencyCashList** property.
- Consists of any combination of features to aid in the cash processing functions such as a cash entry holding bin, a number of slots or bins which can hold the cash, and cash exits.
- Prior to Release 1.5 this specification provides programmatic control *only for the dispensing of cash*. The accepting of cash by the device (for example, to replenish cash) cannot be controlled by the APIs provided in this model. The application can call **readCashCounts** to retrieve the current unit count for each cash unit, but cannot control when or how cash is added to the device.
- May have multiple exits. The number of exits is specified in the **DeviceExits** property. The application chooses a dispensing exit by setting the **CurrentExit** property. The cash units which may be dispensed to the current exit are indicated by the **ExitCashList** property. When **CurrentExit** is 1, the exit is considered the “primary exit” which is typically used during normal processing for dispensing cash to a customer following a retail transaction. When **CurrentExit** is greater than 1, the exit is considered an “auxiliary exit.” An “auxiliary exit” typically is used for special purposes such as dispensing quantities or types of cash not targeted for the “primary exit.”
- Dispenses cash into the exit specified by **CurrentExit** when either **dispenseChange** or **dispenseCash** is called. With **dispenseChange**, the application specifies a total amount to be dispensed, and it is the responsibility of the Cash Changer device or the Control to dispense the proper amount of cash from the various slots or bins. With **dispenseCash**, the application specifies a count of each cash unit to be dispensed.
- Dispenses cash either synchronously or asynchronously, depending on the value of the **AsyncMode** property.

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

When **AsyncMode** is true and no exception is thrown by either **dispenseChange** or **dispenseCash**, then the method is performed asynchronously and its completion is indicated by a **StatusUpdateEvent** with its *Data* property set to **CHAN\_STATUS\_ASYNC**. The request’s completion status is set in the **AsyncResultCode** and **AsyncResultCodeExtended** properties.

The values of **AsyncResultCode** and **AsyncResultCodeExtended** are the same as those for the *ErrorCode* and *ErrorCodeExtended* properties of a *Jpo-sException* when an error occurs during synchronous dispensing.

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

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

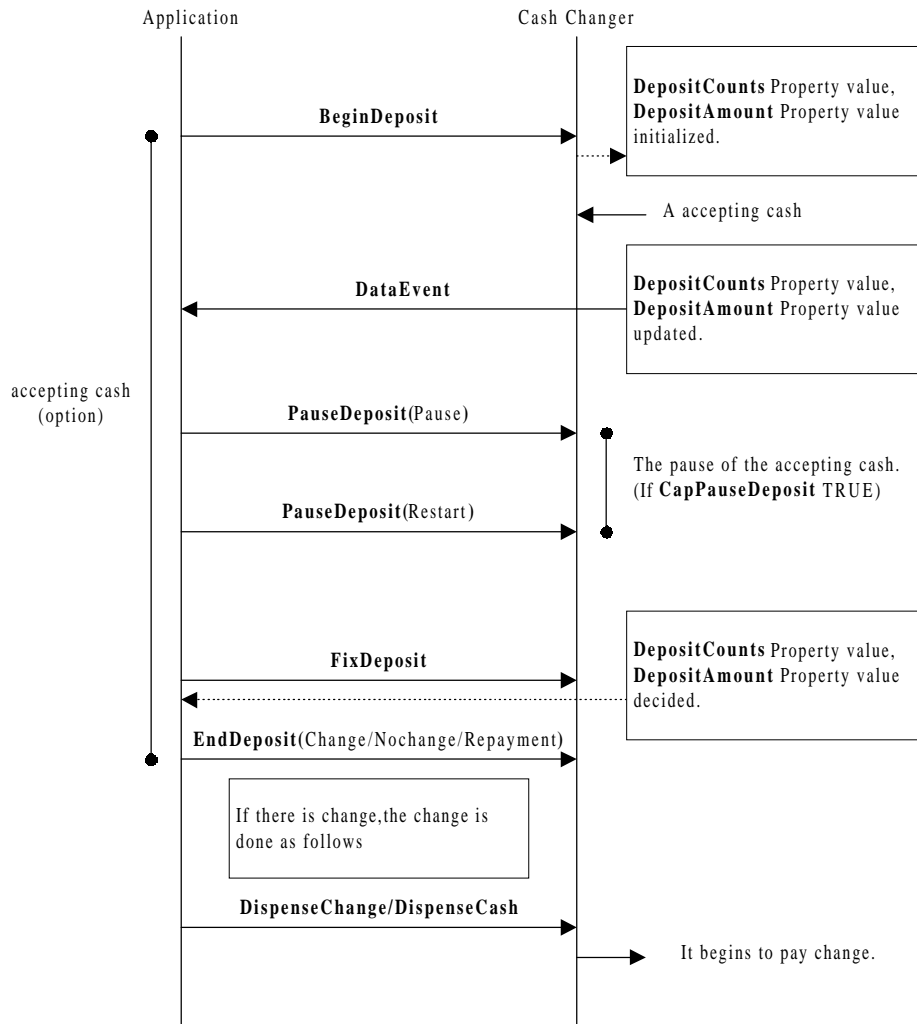
- May support more than one currency. The **CurrencyCode** property may be set to the currency, selecting from a currency in the list **CurrencyCodeList**. **CurrencyCashList**, **ExitCashList**, **dispenseCash**, **dispenseChange** and **readCashCounts** all act upon the current currency only.
- Sets the cash slot (or cash bin) conditions in the **DeviceStatus** property to show empty and near empty status, and in the **FullStatus** property to show full and near full status. If there are one or more empty cash slots, then **DeviceStatus** is `CHAN_STATUS_EMPTY`, and if there are one or more full cash slots, then **FullStatus** is `CHAN_STATUS_FULL`.
- **After Release 1.5--Support for cash acceptance is added as an option.**
  - The cash acceptance model is as follows:
  - Note that the **AsyncMode** property has no affect on methods that have been added for cash acceptance, since these are treated as input methods.
  - The dispensing of change function of this device is not dependent upon the availability of a “cash acceptance” function option. Dispensing of change and collection of money are two independent functions.
  - Receipt of cash (cash acceptance function) is an option that may be provided by the Cash Changer device. Cash acceptance into the “cash acceptance mechanism” is started by invoking the **beginDeposit** method. The previous values of the properties **DepositCounts** and **DepositAmount** are initialized to zero.
  - The total amount of cash placed into the device continues to be accumulated until either the **fixDeposit** method or the **pauseDeposit** method is executed. When the **fixDeposit** method is executed, the total amount of accumulated cash is stored in the **DepositCounts** and **DepositAmount** properties. If the **CapDepositDataEvent** capability was previously set to true, then a **DataEvent** is generated to inform the application that cash has been collected. If the **pauseDeposit** method is executed with a parameter value of `CHAN_DEPOSIT_PAUSE`, then the counting of the deposited cash is suspended and the current amount of accumulated cash is also updated to the **DepositCounts** and **DepositAmount** properties. When **pauseDeposit** method is executed with a parameter value of `CHAN_DEPOSIT_RESTART`, counting of deposited cash is resumed and added to the accumulated totals. When the **fixDeposit** method is executed, the current amount of accumulated cash is updated in the **DepositCounts** and **DepositAmount** properties, and the process remains static until an **endDeposit** method is executed. At this point the “cash acceptance” mechanism is notified to stop accepting cash. If **endDeposit** method receives a `CHAN_DEPOSIT_CHANGE` parameter, then



the mechanism will dispense cash change back to the user. If **endDeposit** is invoked with a `CHAN_DEPOSIT_NOCHANGE` parameter, then the mechanism will not dispense cash change back to the user. Finally, if **endDeposit** is invoked with a `CHAN_DEPOSIT_REPAY` parameter, then all collected cash is returned back to the user by the mechanism.

- Two types of Cash Changer mechanisms are covered by this standard. In one case where **CapRepayDeposit** is true, the bins that are used for collecting the cash are the same bins that are used for dispensing the cash as change. In the other case where **CapRepayDeposit** is false, the bins that are used for collecting the cash are different from the bins that are used for dispensing the change. In the first case, if a transaction is aborted for any reason, the same cash the user input to the mechanism will be returned to the user. In the second case, it is up to the application to dispense an equivalent amount of cash (not the same physical cash collected) back to the user for an aborted transaction.
- The Cash Changer mechanisms can only be used in one mode at a time. While the mechanism is collecting deposited cash, it can not dispense change at the same time. Therefore, while **beginDeposit** method is being executed, no payment of change can occur. Only after an **endDeposit** method call can the proper amount of change be determined (either by the application or by a “smart” Cash Changer) and dispensed to the user. Each Cash Changer manufacturer must determine the amount of time it takes to process the received cash and place in storage bins before it completes the **endDeposit** method.

- When the **clearInput** method is executed, the queued **DataEvent** associated with the receipt of cash is cleared. The **DepositCounts** and **DepositAmount** properties remain set and are not cleared.
- The processing flow of cash acceptance is shown in the following diagram:



### Device Sharing

The Cash Changer is an exclusive-use device, as follows:

- The application must claim the device before enabling it.
- The application must claim and enable the device before accessing some of the properties, dispensing or collecting, or receiving status update events.
- See the “Summary” table for precise usage prerequisites.

## Properties

### AsyncMode Property R/W

Type	<b>boolean</b>
Remarks	If true, the <b>dispenseCash</b> and <b>dispenseChange</b> methods will be performed asynchronously. If false, these methods will be performed synchronously.  This property is initialized to false by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>dispenseCash</b> Method, <b>dispenseChange</b> Method, <b>AsyncResultCode</b> Property; <b>AsyncResultCodeExtended</b> Property

### AsyncResultCode Property R

Type	<b>int</b>
Remarks	Holds the completion status of the last asynchronous dispense request (i.e., when <b>dispenseCash</b> or <b>dispenseChange</b> was called with <b>AsyncMode</b> true). This property is set before a <b>StatusUpdateEvent</b> event is delivered with a <i>Data</i> value of CHAN_STATUS_ASYNC.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>dispenseCash</b> Method, <b>dispenseChange</b> Method, <b>AsyncMode</b> Property

### AsyncResultCodeExtended Property R

Type	<b>int</b>
Remarks	Holds the extended completion status of the last asynchronous dispense request (i.e., when <b>dispenseCash</b> or <b>dispenseChange</b> was called with <b>AsyncMode</b> true). This property is set before a <b>StatusUpdateEvent</b> event is delivered with a <i>Data</i> value of CHAN_STATUS_ASYNC.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>dispenseCash</b> Method, <b>dispenseChange</b> Method, <b>AsyncMode</b> Property

**CapDeposit Property R*****Added in Release 1.5***

<b>Type</b>	<b>boolean</b>
<b>Remarks</b>	If true, the Cash Changer supports cash acceptance. This property is initialized by the <b>open</b> method.
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
<b>See Also</b>	<b>beginDeposit</b> Method, <b>endDeposit</b> Method, <b>fixDeposit</b> Method, <b>pauseDeposit</b> Method.

**CapDepositDataEvent Property R*****Added in Release 1.5***

<b>Type</b>	<b>boolean</b>
<b>Remarks</b>	If true, the Cash Changer can report a cash acceptance event. This property is initialized by the <b>open</b> method.
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
<b>See Also</b>	<b>beginDeposit</b> Method, <b>endDeposit</b> Method, <b>fixDeposit</b> Method, <b>pauseDeposit</b> Method.

**CapDiscrepancy Property R**

<b>Type</b>	<b>boolean</b>
<b>Remarks</b>	If true, the <b>readCashCounts</b> method can report a valid <i>discrepancy</i> value. This property is initialized by the <b>open</b> method.
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
<b>See Also</b>	<b>readCashCounts</b> Method

### CapEmptySensor Property R

Type	<b>boolean</b>
Remarks	If true, the Cash Changer can report the condition that some cash slots are empty.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>DeviceStatus</b> Property, <b>StatusUpdateEvent</b>

### CapFullSensor Property R

Type	<b>boolean</b>
Remarks	If true, the Cash Changer can report the condition that some cash slots are full.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>FullStatus</b> Property, <b>StatusUpdateEvent</b>

### CapNearEmptySensor Property R

Type	<b>boolean</b>
Remarks	If true, the Cash Changer can report the condition that some cash slots are nearly empty.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>DeviceStatus</b> Property, <b>StatusUpdateEvent</b>

**CapNearFullSensor Property R**

<b>Type</b>	<b>boolean</b>
<b>Remarks</b>	If true, the Cash Changer can report the condition that some cash slots are nearly full.  This property is initialized by the <b>open</b> method.
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
<b>See Also</b>	<b>FullStatus</b> Property, <b>StatusUpdateEvent</b>

**CapPauseDeposit Property R**      *Added in Release 1.5*

<b>Type</b>	<b>boolean</b>
<b>Remarks</b>	If true, the Cash Changer has the capability to suspend cash acceptance processing temporarily. This property is initialized by the <b>open</b> method.
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
<b>See Also</b>	<b>pauseDeposit</b> Method.

**CapRepayDeposit Property R**      *Added in Release 1.5*

<b>Type</b>	<b>boolean</b>
<b>Remarks</b>	If true, the Cash Changer has the capability to return money that was deposited. This property is initialized by the <b>open</b> method.
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
<b>See Also</b>	<b>endDeposit</b> Method.

## CurrencyCashList Property R

Type	String
Remarks	<p>Holds the cash units supported in the Cash Changer for the currency represented by the <b>CurrencyCode</b> property. It consists of ASCII numeric comma delimited values which denote the units of coins, then the ASCII semicolon character (“;”) followed by ASCII numeric comma delimited values for the bills that can be used with the Cash Changer. If a semicolon (“;”) is absent, then all units represent coins.</p> <p>Below are sample <b>CurrencyCashList</b> values in Japan.</p> <ul style="list-style-type: none"><li>• “1,5,10,50,100,500” — 1, 5, 10, 50, 100, 500 yen coin.</li><li>• “1,5,10,50,100,500;1000,5000,10000” — 1, 5, 10, 50, 100, 500 yen coin and 1000, 5000, 10000 yen bill.</li><li>• “;1000,5000,10000” — 1000, 5000, 10000 yen bill.</li></ul> <p>This property is initialized by the <b>open</b> method, and is updated when <b>CurrencyCode</b> is set.</p>
Errors	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>
See Also	<p><b>CurrencyCode</b> Property</p>

**CurrencyCode Property R/W**

<b>Type</b>	<b>String</b>				
<b>Remarks</b>	<p>Holds the active currency code to be used by Cash Changer operations. This value is one of the set of currencies specified by the <b>CurrencyCodeList</b> property.</p> <p>This property is initialized to an appropriate value by the <b>open</b> method.</p>				
<b>Errors</b>	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p> <p>Some possible values of the exception’s <i>ErrorCode</i> property are:</p> <table> <thead> <tr> <th><b>Value</b></th> <th><b>Meaning</b></th> </tr> </thead> <tbody> <tr> <td>JPOS_E_ILLEGAL</td> <td>A value was specified that is not within <b>CurrencyCodeList</b>.</td> </tr> </tbody> </table>	<b>Value</b>	<b>Meaning</b>	JPOS_E_ILLEGAL	A value was specified that is not within <b>CurrencyCodeList</b> .
<b>Value</b>	<b>Meaning</b>				
JPOS_E_ILLEGAL	A value was specified that is not within <b>CurrencyCodeList</b> .				
<b>See Also</b>	<b>CurrencyCodeList</b> Property				

**CurrencyCodeList Property R**

<b>Type</b>	<b>String</b>
<b>Remarks</b>	<p>Holds the currency code indicators. It is a list of ASCII three-character ISO 4217 currency codes separated by commas.</p> <p>For example, if the string is “JPY,USD”, then the Cash Changer supports both Japanese and U.S. monetary units.</p> <p>This property is initialized by the <b>open</b> method.</p>
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
<b>See Also</b>	<b>CurrencyCode</b> Property



**CurrentExit Property R/W**

<b>Type</b>	<b>int</b>				
<b>Remarks</b>	<p>Holds the current cash dispensing exit. The value 1 represents the primary exit (or <i>normal</i> exit), while values greater than 1 are considered auxiliary exits. Legal values range from 1 to <b>DeviceExits</b>.</p> <p>Below are examples of typical property value sets in Japan. <b>CurrencyCode</b> is “JPY” and <b>CurrencyCodeList</b> is “JPY.”</p> <ul style="list-style-type: none"> <li>• Cash Changer supports coins; only one exit supported:  <b>CurrencyCashList</b> = “1,5,10,50,100,500”  <b>DeviceExits</b> = 1  <b>CurrentExit</b> = 1: <b>ExitCashList</b> = “1,5,10,50,100,500”</li> <li>• Cash Changer supports both coins and bills; an auxiliary exit is used for larger quantities of bills:  <b>CurrencyCashList</b> = “1,5,10,50,100,500;1000,5000,10000”  <b>DeviceExits</b> = 2  When <b>CurrentExit</b> = 1: <b>ExitCashList</b> = “1,5,10,50,100,500;1000,5000”  When <b>CurrentExit</b> = 2: <b>ExitCashList</b> = “;1000,5000,10000”</li> <li>• Cash Changer supports bills; an auxiliary exit is used for larger quantities of bills:  <b>CurrencyCashList</b> = “;1000,5000,10000”  <b>DeviceExits</b> = 2  When <b>CurrentExit</b> = 1: <b>ExitCashList</b> = “;1000,5000”  When <b>CurrentExit</b> = 2: <b>ExitCashList</b> = “;1000,5000,10000”</li> </ul> <p>This property is initialized to 1 by the <b>open</b> method.</p>				
<b>Errors</b>	<p>A <i>JposException</i> may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p> <p>Some possible values of the exception’s <i>ErrorCode</i> property are:</p> <table> <thead> <tr> <th><b>Value</b></th> <th><b>Meaning</b></th> </tr> </thead> <tbody> <tr> <td>JPOS_E_ILLEGAL</td> <td>An invalid <b>CurrentExit</b> value was specified.</td> </tr> </tbody> </table>	<b>Value</b>	<b>Meaning</b>	JPOS_E_ILLEGAL	An invalid <b>CurrentExit</b> value was specified.
<b>Value</b>	<b>Meaning</b>				
JPOS_E_ILLEGAL	An invalid <b>CurrentExit</b> value was specified.				
<b>See Also</b>	<b>CurrencyCashList</b> Property, <b>DeviceExits</b> Property, <b>ExitCashList</b> Property				

**DepositAmount Property R**      **Added in Release 1.5**

Type	<b>int</b>
Remarks	<p>The total amount of deposited cash.</p> <p>For example, if the currency is Japanese yen and <b>DepositAmount</b> is set to <b>18057</b>, after the call to the <b>beginDeposit</b> method, there would be 18,057 yen in the Cash Changer.</p> <p>This property is initialized by the <b>open</b> method.</p>
Errors	A <code>JposException</code> may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	<b>CurrencyCode</b> Property.

**DepositCashList Property R**      **Added in Release 1.5**

Type	<b>String</b>
Remarks	<p>Holds the cash units supported in the Cash Changer for the currency represented by the <b>CurrencyCode</b> property. It is set to null when the cash acceptance process is not supported.</p> <p>It consists of ASCII numeric comma delimited values which denote the units of coins, then the ASCII semicolon character (“;”) followed by ASCII numeric comma delimited values for the bills that can be used with the Cash Changer. If the semicolon (“;”) is absent, then all units represent coins.</p> <p>Below are sample <b>DepositCashList</b> values in Japan.</p> <ul style="list-style-type: none"><li>• “1,5,10,50,100,500” --- 1, 5, 10, 50, 100, 500 yen coin.</li><li>• “1,5,10,50,100,500;1000,5000,10000” --- 1, 5, 10, 50, 100, 500 yen coin and 1000, 5000, 10000 yen bill.</li><li>• “;1000,5000,10000” --- 1000, 5000, 10000 yen bill.</li></ul> <p>This property is initialized by the <b>open</b> method, and is updated when <b>CurrencyCode</b> is set.</p>
Errors	A <code>JposException</code> may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	<b>CurrencyCode</b> Property.

**DepositCodeList Property R**      **Added in Release 1.5**

Type	<b>String</b>
Remarks	<p>Holds the currency code indicators for cash accepted. It is set to null when the cash acceptance process is not supported.</p> <p>It is a list of ASCII three-character ISO 4217 currency codes separated by commas. For example, if the string is “JPY,USD”, then the Cash Changer supports both Japanese and U.S. monetary units.</p> <p>This property is initialized by the <b>open</b> method.</p>
Errors	A JposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	<b>CurrencyCode</b> Property.

**DepositCounts Property R**      **Added in Release 1.5**

Type	<b>String</b>
Remarks	<p>Holds the total of the cash accepted by the cash units. The format of the string is the same as <i>cashCounts</i> in the <b>dispenseCash</b> method. Cash units inside the string are the same as the <b>DepositCashList</b> property, and are in the same order. It is set to null when the cash acceptance function is not supported.</p> <p>For example if the currency is Japanese yen and string of the <b>DepositCounts</b> property is set to:</p> <p>1:80,5:77,10:0,50:54,100:0,500:87</p> <p>After the call to the <b>beginDeposit</b> method, there would be 80 one yen coins, 77 five yen coins, 54 fifty yen coins, and 87 five hundred yen coins in the Cash Changer.</p> <p>This property is initialized by the <b>open</b> method</p>
Errors	A JposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	<b>CurrencyCode</b> Property.

**DepositStatus Property R*****Added in Release 1.5*****Type**      **int****Remarks**      Holds the current status of the cash acceptance operation. It may be one of the following values:**Value****Meaning**

CHAN\_STATUS\_DEPOSIT\_START

Cash acceptance started.

CHAN\_STATUS\_DEPOSIT\_END

Cash acceptance stopped.

CHAN\_STATUS\_DEPOSIT\_NONE

Cash acceptance not supported.

CHAN\_STATUS\_DEPOSIT\_COUNT

Counting or repaying the deposited money.

CHAN\_STATUS\_DEPOSIT\_JAM

A mechanical fault has occurred.

This property is initialized and kept current while the device is enabled. This property is set to CHAN\_STATUS\_DEPOSIT\_END after initialization, or to CHAN\_STATUS\_DEPOSIT\_NONE if the device does not support cash acceptance.

**Errors**      A JposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

**DeviceExits Property R**

<b>Type</b>	<b>int</b>
<b>Remarks</b>	<p>Holds the number of exits for dispensing cash.</p> <p>This property is initialized by the <b>open</b> method.</p>
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
<b>See Also</b>	<b>CurrentExit</b> Property

**DeviceStatus Property R**

<b>Type</b>	<b>int</b>										
<b>Remarks</b>	<p>Holds the current status of the Cash Changer. It has of the following values:</p> <table> <thead> <tr> <th><b>Value</b></th> <th><b>Meaning</b></th> </tr> </thead> <tbody> <tr> <td>CHAN_STATUS_OK</td> <td>The current condition of the Cash Changer is satisfactory.</td> </tr> <tr> <td>CHAN_STATUS_EMPTY</td> <td>Some cash slots are empty.</td> </tr> <tr> <td>CHAN_STATUS_NEAREMPTY</td> <td>Some cash slots are nearly empty.</td> </tr> <tr> <td>CHAN_STATUS_JAM</td> <td>A mechanical fault has occurred.</td> </tr> </tbody> </table> <p>This property is initialized and kept current while the device is enabled. If more than one condition is present, then the order of precedence starting at the highest is jam (or mechanical fault), empty, and near empty.</p>	<b>Value</b>	<b>Meaning</b>	CHAN_STATUS_OK	The current condition of the Cash Changer is satisfactory.	CHAN_STATUS_EMPTY	Some cash slots are empty.	CHAN_STATUS_NEAREMPTY	Some cash slots are nearly empty.	CHAN_STATUS_JAM	A mechanical fault has occurred.
<b>Value</b>	<b>Meaning</b>										
CHAN_STATUS_OK	The current condition of the Cash Changer is satisfactory.										
CHAN_STATUS_EMPTY	Some cash slots are empty.										
CHAN_STATUS_NEAREMPTY	Some cash slots are nearly empty.										
CHAN_STATUS_JAM	A mechanical fault has occurred.										
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.										

**ExitCashList Property R**

<b>Type</b>	<b>String</b>
<b>Remarks</b>	<p>Holds the cash units which may be dispensed to the exit which is denoted by <b>CurrentExit</b> property. The supported cash units are either the same as <b>CurrencyCashList</b>, or a subset of it. The string format is identical to that of <b>CurrencyCashList</b>.</p> <p>This property is initialized by the <b>open</b> method, and is updated when <b>CurrencyCode</b> or <b>CurrentExit</b> is set.</p>
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
<b>See Also</b>	<b>CurrencyCode</b> Property, <b>CurrencyCashList</b> Property, <b>CurrentExit</b> Property

**FullStatus Property R**

<b>Type</b>	<b>int</b>								
<b>Remarks</b>	<p>Holds the current full status of the cash slots. It may be one of the following:</p> <table> <thead> <tr> <th><b>Value</b></th> <th><b>Meaning</b></th> </tr> </thead> <tbody> <tr> <td>CHAN_STATUS_OK</td> <td>All cash slots are neither nearly full nor full.</td> </tr> <tr> <td>CHAN_STATUS_FULL</td> <td>Some cash slots are full.</td> </tr> <tr> <td>CHAN_STATUS_NEARFULL</td> <td>Some cash slots are nearly full.</td> </tr> </tbody> </table> <p>This property is initialized and kept current while the device is enabled.</p>	<b>Value</b>	<b>Meaning</b>	CHAN_STATUS_OK	All cash slots are neither nearly full nor full.	CHAN_STATUS_FULL	Some cash slots are full.	CHAN_STATUS_NEARFULL	Some cash slots are nearly full.
<b>Value</b>	<b>Meaning</b>								
CHAN_STATUS_OK	All cash slots are neither nearly full nor full.								
CHAN_STATUS_FULL	Some cash slots are full.								
CHAN_STATUS_NEARFULL	Some cash slots are nearly full.								
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.								

## Methods

### beginDeposit Method

*Added in Release 1.5*

**Syntax**      **void beginDeposit ( ) throws JposException;**

**Remarks**      Cash acceptance is started.

The following property values are initialized by the call to this method:

- The value of each cash unit of the **DepositCounts** property is set to zero.
- The **DepositAmount** property is set to zero.

After calling this method, if **CapDepositDataEvent** is true, cash acceptance is reported by **DataEvents** until **fixDeposit** is called while the deposit process is not paused.

**Errors**      A JposException may be thrown when this method is invoked. For further information, see “Errors” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
JPOS_E_ILLEGAL	Either the Cash Changer does not support cash acceptance, or the call sequence is not correct.

**See Also**      **DepositCounts** Property, **DepositAmount** Property, **CapDepositDataEvent** Property, **endDeposit** Method, **fixDeposit** Method, **pauseDeposit** Method.

**dispenseCash Method**

<b>Syntax</b>	<p><b>void dispenseCash (String <i>cashCounts</i>) throws JposException;</b></p> <p>The <i>cashCounts</i> parameter contains the dispensing cash units and counts, represented by the format of “cash unit:cash counts,;:, cash unit:cash counts.” Units before “;” represent coins, and units after “;” represent bills. If “;” is absent, then all units represent coins.</p>						
<b>Remarks</b>	<p>Dispenses the cash from the Cash Changer into the exit specified by <b>CurrentExit</b>. The cash dispensed is specified by pairs of cash units and counts.</p> <p>This method is performed synchronously if <b>AsyncMode</b> is false, and asynchronously if <b>AsyncMode</b> is true.</p> <p>Some <i>cashCounts</i> examples, using Japanese Yen as the currency, are below.</p> <ul style="list-style-type: none"> <li>• “10:5,50:1,100:3,500:1” Dispense 5 ten yen coins, 1 fifty yen coin, 3 one hundred yen coins, 1 five hundred yen coin.</li> <li>• “10:5,100:3;1000:10” Dispense 5 ten yen coins, 3 one hundred yen coins, and 10 one thousand yen bills.</li> <li>• “;1000:10,10000:5” Dispense 10 one thousand yen bills and 5 ten thousand yen bills.</li> </ul>						
<b>Errors</b>	<p>A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.</p> <p>Some possible values of the exception’s <i>ErrorCode</i> property are:</p> <table border="1"> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>JPOS_E_ILLEGAL</td> <td>A <i>cashCounts</i> parameter value was illegal for the current exit.</td> </tr> <tr> <td>JPOS_E_EXTENDED</td> <td><i>ErrorCodeExtended</i> = JPOS_ECHAN_OVERDISPENSE: The specified cash cannot be dispensed because of a cash shortage.</td> </tr> </tbody> </table>	Value	Meaning	JPOS_E_ILLEGAL	A <i>cashCounts</i> parameter value was illegal for the current exit.	JPOS_E_EXTENDED	<i>ErrorCodeExtended</i> = JPOS_ECHAN_OVERDISPENSE: The specified cash cannot be dispensed because of a cash shortage.
Value	Meaning						
JPOS_E_ILLEGAL	A <i>cashCounts</i> parameter value was illegal for the current exit.						
JPOS_E_EXTENDED	<i>ErrorCodeExtended</i> = JPOS_ECHAN_OVERDISPENSE: The specified cash cannot be dispensed because of a cash shortage.						
<b>See Also</b>	<b>AsyncMode</b> Property, <b>CurrentExit</b> Property						



## dispenseChange Method

**Syntax**      **void dispenseChange (int *amount*) throws JposException;**

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

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

This method is performed synchronously if **AsyncMode** is false, and asynchronously if **AsyncMode** is true.

**Errors**          A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
JPOS_E_ILLEGAL	A negative or zero <i>amount</i> was specified, or it is impossible to dispense the <i>amount</i> based on the values specified in <b>ExitCashList</b> for the current exit.
JPOS_E_EXTENDED	<i>ErrorCodeExtended</i> = JPOS_ECHAN_OVERDISPENSE: The specified change cannot be dispensed because of a cash shortage.

**See Also**      **AsyncMode** Property, **CurrentExit** Property

**endDeposit Method****Added in Release 1.5****Syntax**      **void endDeposit ( int *success* ) throws JposException;**

The *success* parameter holds the value of how to deal with the cash that was deposited. Contains one of the following values:

Parameter	Description
CHAN_DEPOSIT_CHANGE	The deposit is accepted and the deposited amount is greater than the amount required.
CHAN_DEPOSIT_NOCHANGE	The deposit is accepted and the deposited amount is equal to or less than the amount required.
CHAN_DEPOSIT_REPAY	The deposit is to be repaid through the cash deposit exit or the cash payment exit.

**Remarks**      Cash acceptance is completed.

Before calling this method, the application must calculate the difference between the amount of the deposit and the amount required.

If the deposited amount is greater than the amount required then *success* is set to CHAN\_DEPOSIT\_CHANGE. If the deposited amount is equal to or less than the amount required then *success* is set to CHAN\_DEPOSIT\_NOCHANGE.

If *success* is set to CHAN\_DEPOSIT\_REPAY then the deposit is repaid through either the cash deposit exit or the cash payment exit without storing the actual deposited cash.

When the deposit is repaid, it is repaid in the exact cash unit quantities that were deposited. Depending on the actual device, the cash repaid may be the exact same bills and coins that were deposited, or it may not.

The application must call the **fixDeposit** method before calling this method.

**Errors**      A JposException may be thrown when this method is invoked. For further information, see “Errors” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_ILLEGAL	One of the following errors occurred: <ul style="list-style-type: none"> <li>• Cash acceptance is not supported.</li> <li>• The call sequence is invalid. <b>beginDeposit</b> and <b>fixDeposit</b> must be called in sequence before calling this method.</li> </ul>

**See Also**      **DepositCounts** Property, **DepositAmount** Property, **CapDepositDataEvent** Property, **beginDeposit** Method, **fixDeposit** Method, **pauseDeposit** Method.

**fixDeposit Method*****Added in Release 1.5***

- Syntax**      **void fixDeposit ( ) throws JposException;**
- Remarks**    When this method is called, all property values are updated to reflect the current values in the Cash Changer.
- Errors**        A JposException may be thrown when this method is invoked. For further information, see “Errors” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
JPOS_E_ILLEGAL	One of the following errors occurred: <ul style="list-style-type: none"> <li>• Cash acceptance is not supported.</li> <li>• The call sequence is invalid. <b>beginDeposit</b> must be called before calling this method.</li> </ul>

- See Also**      **DepositCounts** Property, **DepositAmount** Property, **beginDeposit** Method, **endDeposit** Method, **pauseDeposit** Method.

**pauseDeposit Method****Added in Release 1.5****Syntax**      **void pauseDeposit ( int *control* ) throws JposException;**The *control* parameter contains one of the following values:

Parameter	Description
CHAN_DEPOSIT_PAUSE	Cash acceptance is paused.
CHAN_DEPOSIT_RESTART	Cash acceptance is resumed.

**Remarks**      Called to suspend or resume the process of depositing cash.

If *control* is CHAN\_DEPOSIT\_PAUSE, the cash acceptance operation is paused. The deposit process will remain paused until this method is called with *control* set to CHAN\_DEPOSIT\_RESTART. It is valid to call **fixDeposit** then **endDeposit** while the deposit process is paused.

When the deposit process is paused, the **depositCounts** and **depositAmount** properties are updated to reflect the current state of the Cash Changer. The property values are not changed again until the deposit process is resumed.

If *control* is CHAN\_DEPOSIT\_RESTART, the deposit process is resumed.

**Errors**      A JposException may be thrown when this method is invoked. For further information, see “Errors” on page 15.Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_ILLEGAL	One of the following errors occurred: <ul style="list-style-type: none"> <li>• Cash acceptance is not supported.</li> <li>• The call sequence is invalid. <b>beginDeposit</b> must be called before calling this method.</li> <li>• The deposit process is already paused and <i>control</i> is set to CHAN_DEPOSIT_PAUSE, or the deposit process is not paused and <i>control</i> is set to CHAN_DEPOSIT_RESTART.</li> </ul>

**See Also**      **DepositCounts** Property, **DepositAmount** Property, **CapDepositDataEvent** Property, **CapPauseDeposit** Property, **beginDeposit** Method, **endDeposit** Method, **fixDeposit** Method.

## readCashCounts Method

**Syntax** `void readCashCounts (String cashCounts, boolean[] discrepancy)  
throws JposException;`

<b>Parameter</b>	<b>Description</b>
<i>cashCounts</i>	The cash count data is placed into the string.
<i>discrepancy</i>	If set to true, then there is some cash which was not able to be included in the counts reported in <i>cashCounts</i> .

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

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

If **CapDiscrepancy** property is false, then *discrepancy* is always false.

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

**Errors** A `JposException` may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

**See Also** **dispenseCash** Method, **CapDiscrepancy** Property, **CurrencyCashList** Property

## Events

### DataEvent

*Added in Release 1.5*

**Interface** `jpos.events.DataEventListener`

**Method** `dataOccurred (DataEvent e);`

**Description** Notifies the application when cash counts change while cash acceptance is in progress.

**Properties** This event contains the following property:

Property	Type	Description
<i>Status</i>	<i>int</i>	This parameter is always set to "0".

**See Also** "Events" on page 18.

### DirectIOEvent

**Interface** `jpos.events.DirectIOListener`

**Method** `directIOOccurred (DirectIOEvent e);`

**Description** Provides Device Service information directly to the application. This event provides a means for a vendor-specific Cash Changer Device Service to provide events to the application that are not otherwise supported by the Device Control.

**Properties** This event contains the following properties:

Property	Type	Description
<i>EventNumber</i>	<i>int</i>	Event number whose specific values are assigned by the Device Service.
<i>Data</i>	<i>int</i>	Additional numeric data. Specific values vary by the <i>EventNumber</i> and the Device Service. This property is settable.
<i>Object</i>	<i>Object</i>	Additional data whose usage varies by the <i>EventNumber</i> and Device Service. This property is settable.

**Remarks** This event is to be used only for those types of vendor specific functions that are not otherwise described as part of the JavaPOS standard. Use of this event may restrict the application program from being used with other vendor's Cash Changer devices which may not have any knowledge of the Device Service's need for this event.

**See Also** "Events" on page 18, **directIO** Method

## StatusUpdateEvent

**Interface** `jpos.events.StatusUpdateListener`

**Method** `statusUpdateOccurred(StatusUpdateEvent e)`

**Description** Notifies the application when the Cash Changer detects a status change.

**Properties** This event contains the following property:

Property	Type	Description
<i>Status</i>	<i>int</i>	The status reported from the Cash Changer.

The *Status* property has one of the following values:

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

**Note that Release 1.3** added Power State Reporting with additional *Power reporting StatusUpdateEvent values*. See “StatusUpdateEvent” description on page 78.

**See Also** “Events” on page 18.





# Cash Drawer

## Summary

### Properties

<i>Common</i>	<i>Ver</i>	<i>Type</i>	<i>Access</i>	<i>May Use After</i>
<b>AutoDisable</b>		boolean	R/W	<i>Not Supported</i>
<b>CapPowerReporting</b>	1.3	int	R	open
<b>CheckHealthText</b>		String	R	open
<b>Claimed</b>		boolean	R	open
<b>DataCount</b>		int	R	<i>Not Supported</i>
<b>DataEventEnabled</b>		boolean	R/W	<i>Not Supported</i>
<b>DeviceEnabled</b>		boolean	R/W	open
<b>FreezeEvents</b>		boolean	R/W	open
<b>OutputID</b>		int	R	<i>Not Supported</i>
<b>PowerNotify</b>	1.3	int	R/W	open
<b>PowerState</b>	1.3	int	R	open
<b>State</b>		int	R	--
<b>DeviceControlDescription</b>		String	R	--
<b>DeviceControlVersion</b>		int	R	--
<b>DeviceServiceDescription</b>		String	R	open
<b>DeviceServiceVersion</b>		int	R	open
<b>PhysicalDeviceDescription</b>		String	R	open
<b>PhysicalDeviceName</b>		String	R	open
<i>Specific</i>		<i>Type</i>	<i>Access</i>	<i>May Use After</i>
<b>CapStatus</b>		boolean	R	open
<b>CapStatusMultiDrawerDetect</b>	1.5	boolean	R	open
<b>DrawerOpened</b>		boolean	R	open & enable

**Methods**

<i>Common</i>	<i>Ver</i>	<i>May Use After</i>
<b>open</b>		--
<b>close</b>		open
<b>claim</b>		open
<b>release</b>		open & claim
<b>checkHealth</b>		open & enable; <i>Note</i>
<b>clearInput</b>		<i>Not Supported</i>
<b>clearOutput</b>		<i>Not Supported</i>
<b>directIO</b>		open
 <i>Specific</i>		
<b>openDrawer</b>		open & enable; <i>Note</i>
<b>waitForDrawerClose</b>		open & enable; <i>Note</i>

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

**Events**

<i>Name</i>	<i>Ver</i>	<i>May Occur After</i>
<b>DataEvent</b>		<i>Not Supported</i>
<b>DirectIOEvent</b>	1.3	open & claim
<b>ErrorEvent</b>		<i>Not Supported</i>
<b>OutputCompleteEvent</b>		<i>Not Supported</i>
<b>StatusUpdateEvent</b>		open & enable

## General Information

The Cash Drawer Control's class name is "jpos.CashDrawer".

The device constants are contained in the class "jpos.CashDrawerConst".

See "Package Structure" on page 40.

### Capabilities

The Cash Drawer Control has the following capability:

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

The cash drawer may have the following additional capability:

- Drawer status reporting of such a nature that the service can determine whether a particular drawer is open or closed in environments where the drawer is the only drawer accessible via a hardware port.
- Drawer unique status reporting of such a nature that the service can determine whether a particular drawer is open or closed in environments where more than one drawer is accessible via the same hardware port.

### Device Sharing

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

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

## Properties

### CapStatus Property R

Type	<b>boolean</b>
Remarks	If true, the drawer can report status. If false, the drawer is not able to determine whether cash drawer is open or closed.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapStatusMultiDrawerDetect Property R

**Added in Release 1.5**

Type	<b>boolean</b>
Remarks	If true, the status unique to each drawer in a multiple cash drawer configuration <sup>1</sup> can be reported.  If false, the following possibilities exist:  <b>DrawerOpened:</b> value of false indicates that there are no drawers open.  <b>DrawerOpened:</b> value of true indicates that at least one drawer is open and it <i>might</i> be the particular drawer in question. This case can occur in multiple cash drawer configurations where only one status is reported indicating either a) all drawers are closed, or b) one or more drawers are open.  <i>Note:</i> A multiple cash drawer configuration is defined as one where a terminal or printer supports opening more than one cash drawer independently via the same channel or hardware port. A typical example is a configuration where a “Y” cable, connected to a single hardware printer port, has separate drawer open signal lines but the drawer open status from each of the drawers is “wired-or” together. It is not possible to determine which drawer is open.  This property is only meaningful if <b>CapStatus</b> is true.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	CapStatus Property, DrawerOpened Property.

---

<sup>1</sup>. Multiple cash drawer configuration -- A hardware configuration where a printer or terminal controls more than one cash drawer independently via the same channel or hardware port. A typical example is a configuration with a “Y” cable connected to a single hardware port that controls two cash drawers.

## DrawerOpened Property R

<b>Type</b>	<b>boolean</b>
<b>Remarks</b>	<p>If true, the drawer is open. If false, the drawer is closed.</p> <p>If the capability <b>CapStatus</b> is false, then the device does not support status reporting, and this property is always false.</p> <p><b>Note:</b> If the capability <b>CapStatusMultiDrawerDetect</b> is false, then a <b>DrawerOpened</b> value of true indicates at least one drawer is open, and it <i>might</i> be the particular drawer in question in a multiple cash drawer configuration. See <b>CapStatusMultiDrawerDetect</b> for further clarification.</p> <p>This property is initialized and kept current while the device is enabled.</p>
<b>Errors</b>	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>

## Methods

### openDrawer Method

<b>Syntax</b>	<b>void openDrawer () throws JposException;</b>
<b>Remarks</b>	Opens the drawer.
<b>Errors</b>	A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

### waitForDrawerClose Method

<b>Syntax</b>	<b>void waitForDrawerClose (int beepTimeout, int beepFrequency, int beepDuration, int beepDelay) throws JposException;</b>										
	<table border="1"> <thead> <tr> <th>Parameter</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td><i>beepTimeout</i></td> <td>Number of milliseconds to wait before starting an alert beeper.</td> </tr> <tr> <td><i>beepFrequency</i></td> <td>Audio frequency of the alert beeper in hertz.</td> </tr> <tr> <td><i>beepDuration</i></td> <td>Number of milliseconds that the beep tone will be sounded.</td> </tr> <tr> <td><i>beepDelay</i></td> <td>Number of milliseconds between the sounding of beeper tones.</td> </tr> </tbody> </table>	Parameter	Description	<i>beepTimeout</i>	Number of milliseconds to wait before starting an alert beeper.	<i>beepFrequency</i>	Audio frequency of the alert beeper in hertz.	<i>beepDuration</i>	Number of milliseconds that the beep tone will be sounded.	<i>beepDelay</i>	Number of milliseconds between the sounding of beeper tones.
Parameter	Description										
<i>beepTimeout</i>	Number of milliseconds to wait before starting an alert beeper.										
<i>beepFrequency</i>	Audio frequency of the alert beeper in hertz.										
<i>beepDuration</i>	Number of milliseconds that the beep tone will be sounded.										
<i>beepDelay</i>	Number of milliseconds between the sounding of beeper tones.										
<b>Remarks</b>	<p>Waits until the cash drawer is closed. If the drawer is still open after <i>beepTimeout</i> milliseconds, then the system alert beeper is started.</p> <p>Not all POS implementations may support the typical PC speaker system alert beeper. However, by setting these parameters the application will insure that the system alert beeper will be utilized if it is present.</p> <p>Unless a JposException is thrown, this method will not return to the application while the drawer is open. In addition, in a multiple cash drawer configuration where the <b>CapStatusMultiDrawerDetect</b> property is false, this method will not return to the application while any of the drawers are open. When all drawers are closed, the beeper is turned off.</p> <p>If <b>CapStatus</b> is false, then the device does not support status reporting, and this method will return immediately.</p>										
<b>Errors</b>	A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.										
<b>See Also</b>	<b>CapStatus</b> Property, <b>CapStatusMultiDrawerDetect</b> Property.										

## Events

### DirectIOEvent

**Interface** `jpos.events.DirectIOListener`

**Method** `directIOOccurred (DirectIOEvent e);`

**Description** Provides Device Service information directly to the application. This event provides a means for a vendor-specific Cash Drawer Device Service to provide events to the application that are not otherwise supported by the Device Control.

**Properties** This event contains the following properties:

Property	Type	Description
<i>EventNumber</i>	<i>int</i>	Event number whose specific values are assigned by the Device Service.
<i>Data</i>	<i>int</i>	Additional numeric data. Specific values vary by the <i>EventNumber</i> and the Device Service. This property is settable.
<i>Object</i>	<i>Object</i>	Additional data whose usage varies by the <i>EventNumber</i> and Device Service. This property is settable.

**Remarks** This event is to be used only for those types of vendor specific functions that are not otherwise described as part of the JavaPOS standard. Use of this event may restrict the application program from being used with other vendor's Cash Drawer devices which may not have any knowledge of the Device Service's need for this event.

**See Also** "Events" on page 18, **directIO** Method

## StatusUpdateEvent

**Interface**     `jpos.events.StatusUpdateListener`

**Method**        `statusUpdateOccurred (StatusUpdateEvent e);`

**Description**    Notifies the application when the status of the Cash Drawer changes.

**Properties**     This event contains the following property:

Property	Type	Description
<i>Status</i>	<i>int</i>	The status reported from the Cash Drawer.

The *Status* property has one of the following values:

Value	Meaning
CASH_SUE_DRAWERCLOSED (=0)	The drawer is closed.
CASH_SUE_DRAWEROPEN (=1)	The drawer is open.

*Note that Release 1.3* added Power State Reporting with additional *Power reporting StatusUpdateEvent values*. See “StatusUpdateEvent” description on page 78.

**Remarks**     If `CapStatus` is false, then the device does not support status reporting, and this event will never be delivered.

If `CapStatusMultiDrawerDetect` is false, then a CASH\_SUE\_DRAWEROPEN value indicates that at least one cash drawer is open and it *might* be the particular drawer in question for multiple cash drawer configurations.

**See Also**     “Events” on page 18



# CAT-Credit Authorization Terminal

## Summary

### Properties

<i>Common</i>	<i>Ver</i>	<i>Type</i>	<i>Access</i>	<i>May Use After</i>
<b>AutoDisable</b>	1.4	boolean	R/W	<i>Not Supported</i>
<b>CapPowerReporting</b>	1.4	int	R	open
<b>CheckHealthText</b>	1.4	String	R	open
<b>Claimed</b>	1.4	boolean	R	open
<b>DataCount</b>	1.4	int	R	<i>Not Supported</i>
<b>DataEventEnabled</b>	1.4	boolean	R/W	<i>Not Supported</i>
<b>DeviceEnabled</b>	1.4	boolean	R/W	open & claim
<b>FreezeEvents</b>	1.4	boolean	R/W	open
<b>OutputID</b>	1.4	int	R	open
<b>PowerNotify</b>	1.4	int	R/W	open
<b>PowerState</b>	1.4	int	R	open
<b>State</b>	1.4	int	R	--
<b>DeviceControlDescription</b>	1.4	String	R	--
<b>DeviceControlVersion</b>	1.4	int	R	--
<b>DeviceServiceDescription</b>	1.4	String	R	open
<b>DeviceServiceVersion</b>	1.4	int	R	open
<b>PhysicalDeviceDescription</b>	1.4	String	R	open
<b>PhysicalDeviceName</b>	1.4	String	R	open

<i>Specific</i>	<i>Ver</i>	<i>Type</i>	<i>Access</i>	<i>May Use After</i>
<b>AccountNumber</b>	1.4	String	R	open
<b>AdditionalSecurityInformation</b>	1.4	String	R/W	open
<b>ApprovalCode</b>	1.4	String	R	open
<b>AsyncMode</b>	1.4	boolean	R/W	open
<b>CapAdditionalSecurityInformation</b>	1.4	boolean	R	open
<b>CapAuthorizeCompletion</b>	1.4	boolean	R	open
<b>CapAuthorizePreSales</b>	1.4	boolean	R	open
<b>CapAuthorizeRefund</b>	1.4	boolean	R	open
<b>CapAuthorizeVoid</b>	1.4	boolean	R	open
<b>CapAuthorizeVoidPreSales</b>	1.4	boolean	R	open
<b>CapCenterResultCode</b>	1.4	boolean	R	open
<b>CapCheckCard</b>	1.4	boolean	R	open
<b>CapDailyLog</b>	1.4	int	R	open
<b>CapInstallments</b>	1.4	boolean	R	open
<b>CapPaymentDetail</b>	1.4	boolean	R	open
<b>CapTaxOthers</b>	1.4	boolean	R	open
<b>CapTransactionNumber</b>	1.4	boolean	R	open
<b>CapTrainingMode</b>	1.4	boolean	R	open
<b>CardCompanyID</b>	1.4	String	R	open
<b>CenterResultCode</b>	1.4	String	R	open
<b>DailyLog</b>	1.4	String	R	open
<b>PaymentCondition</b>	1.4	int	R	open
<b>PaymentMedia</b>	1.5	int	R/W	open
<b>PaymentDetail</b>	1.4	String	R	open
<b>SequenceNumber</b>	1.4	int	R	open
<b>SlipNumber</b>	1.4	String	R	open
<b>TrainingMode</b>	1.4	boolean	R/W	open
<b>TransactionNumber</b>	1.4	String	R	open
<b>TransactionType</b>	1.4	int	R	open

**Methods***Common*

	<i>Ver</i>	<i>May Use After</i>
<b>open</b>	1.4	--
<b>close</b>	1.4	open
<b>claim</b>	1.4	open & claim
<b>release</b>	1.4	open & claim
<b>checkHealth</b>	1.4	open, claim, & enable
<b>clearInput</b>	1.4	<i>Not Supported</i>
<b>clearOutput</b>	1.4	open & claim
<b>directIO</b>	1.4	open & claim

*Specific*

<b>accessDailyLog</b>	1.4	open, claim, & enable
<b>authorizeCompletion</b>	1.4	open, claim, & enable
<b>authorizePreSales</b>	1.4	open, claim, & enable
<b>authorizeRefund</b>	1.4	open, claim, & enable
<b>authorizeSales</b>	1.4	open, claim, & enable
<b>authorizeVoid</b>	1.4	open, claim, & enable
<b>authorizeVoidPreSales</b>	1.4	open, claim, & enable
<b>checkCard</b>	1.4	open, claim, & enable

**Events**

<i>Name</i>	<i>Ver</i>	<i>May Use After</i>
<b>DataEvent</b>	1.4	<i>Not Supported</i>
<b>DirectIOEvent</b>	1.4	open & claim
<b>ErrorEvent</b>	1.4	open, claim, & enable
<b>OutputCompleteEvent</b>	1.4	open, claim, & enable
<b>StatusUpdateEvent</b>	1.4	open, claim, & enable

## General Information

The CAT Control's class name is "jpos.CAT".

The device constants are contained in the class "jpos.CATConst".

See "Package Structure" on page 40.

***This device was added in JavaPOS Release 1.4.***

The CAT device described in this chapter is currently in use in Japan only.

### Description of terms

- **Authorization method**  
Methods defined by this device class that have the *authorize* prefix in their name. These methods require communication with an approval agency.
- **Authorization operation**  
The period from the invocation of an authorization method until the authorization is completed. This period differs depending upon whether operating in synchronous or asynchronous mode.
- **Credit Authorization Terminal (CAT) Device**  
A CAT device typically consists of a display, keyboard, magnetic stripe card reader, receipt printing device, and a communications device. CAT devices are predominantly used in Japan where they are required by law. Essentially a CAT device can be considered a device that shields the encryption, message formatting, and communication functions of an electronic funds transfer (EFT) operation from an application.
- **Purchase**  
The transaction that allows credit card or debit card payment at the POS. It is independent of payment methods (for example, lump-sum payment, payment in installments, revolving payment, etc.).
- **Cancel Purchase**  
The transaction to request voiding a purchase *on the date of purchase*.
- **Refund Purchase**  
The transaction to request voiding a purchase *after the date of purchase*. This differs from cancel purchase in that a cancel purchase operation can often be handled by updating the daily log at the CAT device, while the refund purchase operation typically requires interaction with the approval agency.
- **Authorization Completion**  
The state of a purchase when the response from the approval agency is "suspended." The purchase is later completed after a voice approval is received from the card company.
- **Pre-Authorization**  
The transaction to reserve an estimated amount in advance of the actual purchase with customer's credit card presentation and card entry at CAT.
- **Cancel Pre-Authorization**  
The transaction to request canceling pre-authorization.

- **Card Check**  
The transaction to perform a negative card file validation of the card presented by the customer. Typically negative card files contain card numbers that are known to fail approval. Therefore the Card Check operation removes the need for communication to the approval agency in some instances.
- **Daily log**  
The daily log of card transactions that have been approved by the card companies.
- **Payment condition**  
Condition of payment such as lump-sum payment, payment by bonus, payment in installments, revolving payment, and the combination of those payments. Debit payment is also available. See the **PaymentCondition**, **PaymentMedia**, and **PaymentDetail** properties for details.
- **Approval agency**  
The agency to decide whether or not to approve the purchase based on the card information, the amount of purchase, and payment type. The approval agency is generally the card company.

## Capabilities

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

- This standard defines the application interface with the CAT Control and does not depend on the CAT device's hardware implementation. Therefore, the hardware implementation of a CAT device may be as follows:
  - **Separate type (POS interlock)**  
The dedicated CAT device is externally connected to the POS (for instance, via an RS-232 connection).
  - **Built-in type**  
The hardware structure is the same as the separate type but is installed within the POS housing.
- The CAT device receives each authorization request containing a purchase amount and tax from the CAT Control.
- The CAT device generally requests the user to swipe a magnetic card when it receives an authorization request from the CAT Control.
- Once a magnetic card is swiped at the CAT device, the device sends the purchase amount and tax to the approval agency using the communications device.
- The CAT device returns the result from the approval agency to the CAT Control. The returned data will be stored in the authorization properties by the CAT Control for access by applications.

## Model

The general models for the CAT Control are shown below:

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

Function	Method name	Associated Capability Property
Purchase	<b>authorizeSales</b>	None Available
Cancel Purchase	<b>authorizeVoid</b>	<b>CapAuthorizeVoid</b>
Refund Purchase	<b>authorizeRefund</b>	<b>CapAuthorizeRefund</b>
Authorization Completion	<b>authorizeCompletion</b>	<b>CapAuthorizeCompletion</b>
Pre-Authorization	<b>authorizePreSales</b>	<b>CapAuthorizePreSales</b>
Cancel Pre-Authorization	<b>authorizeVoidPreSales</b>	<b>CapAuthorizeVoidPreSales</b>

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

Function	Method name	Corresponding Capability
Card Check	<b>checkCard</b>	<b>CapCheckCard</b>
Daily log	<b>accessDailyLog</b>	<b>CapDailyLog</b>

- The CAT Control stores the authorization results in the following properties when an authorization operation successfully completes:

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

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

Description	Property Name	Corresponding Capability
Daily log	<b>DailyLog</b>	<b>CapDailyLog</b>

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

- The **AsyncMode** property determines if methods are run synchronously or asynchronously.
  - When **AsyncMode** is false, methods will be executed synchronously and their corresponding properties will contain data when the method returns.
  - When **AsyncMode** is true, methods will return immediately to the application. When the operation associated with the method completes successfully, each corresponding property will be updated prior to delivering an **OutputCompleteEvent**. If the operation associated with the method does not complete successfully, an **ErrorEvent** is enqueued. When **AsyncMode** is true, methods cannot be invoked immediately after invoking a prior method; only one outstanding asynchronous method is allowed at a time. However, **clearOutput** is an exception because its purpose is to cancel an outstanding asynchronous method.
- The methods supported and their corresponding properties vary depending on the CAT Device Service. Applications should verify that particular capabilities are supported before utilizing the dependent methods and properties.
- Whether in synchronous or asynchronous mode, the result code from the approval agency will be stored in **CenterResultCode**.
- Training mode occurs continually when **TrainingMode** is true. To discontinue training mode, set **TrainingMode** to false.
- An outstanding asynchronous method can be canceled via the **clearOutput** method.
- The daily log can be collected by the **accessDailyLog** method. Collection will be run either synchronously or asynchronously according to the value of **AsyncMode**.



- Following is the general usage sequence of the CAT control.
  - Synchronous Mode:
    - **open**
    - **claim**
    - **setDeviceEnabled** (true)
    - Definition of the argument *SequenceNumber*
    - Set **PaymentMedia** Added in Version 1.5
    - **authorizeSales()**
    - Check *JposException* of the **authorizeSales** method
    - Verify that the **SequenceNumber** property matches the value of the **authorizeSales()** *sequenceNumber* argument
    - Access the properties set by **authorizeSales()**
    - **setDeviceEnabled** (false)
    - **release**
    - **close**
  - Asynchronous Mode:
    - **open**
    - **claim**
    - **setDeviceEnabled** (true)
    - **setAsyncMode** (true)
    - Definition of the argument *SequenceNumber*
    - Set **PaymentMedia** Added in Version 1.5
    - **authorizeSales()**
    - Check *JposException* of the **authorizeSales** method
    - Wait for **OutputCompleteEvent**
    - Check the argument *ErrorCode*
    - Verify that the **SequenceNumber** property matches the value of the **authorizeSales()** *SequenceNumber* argument
    - Access the properties set by **authorizeSales()**
    - **setDeviceEnabled** (false)
    - **release and close**

## Device sharing

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

- After opening the device, properties are readable.
- The application must claim the device before enabling it.
- The application must claim and enable the device before calling methods that manipulate the device.
- See the “Summary” table for precise usage prerequisites.

## Properties

### AccountNumber Property R

Type	String
Remarks	This property is initialized to an empty string by the <b>open</b> method and is updated when an authorization operation successfully completes.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### AdditionalSecurityInformation Property R/W

Type	String
Remarks	An application can send data to the CAT device by setting this property before issuing an authorization method. Also, data obtained from the CAT device and not stored in any other property as the result of an authorization operation (for example, the account code for a loyalty program) can be provided to an application by storing it in this property. Since the data stored here is device specific, this should not be used for any development that requires portability.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>CapAdditionalSecurityInformation</b> Property

### ApprovalCode Property R

Type	String
Remarks	This property is initialized to an empty string by the <b>open</b> method and is updated when an authorization operation successfully completes.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### AsyncMode Property R/W

Type	<b>boolean</b>
Remarks	If true, the authorization methods will run asynchronously. If false, the authorization methods will run synchronously.  This property is initialized to false by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	Authorization Methods

### CapAdditionalSecurityInformation Property R

Type	<b>boolean</b>
Remarks	If true, the <b>AdditionalSecurityInformation</b> property may be utilized.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>AdditionalSecurityInformation</b> Property

### CapAuthorizeCompletion Property R

Type	<b>boolean</b>
Remarks	If true, the <b>authorizeCompletion</b> method has been implemented.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>authorizeCompletion</b> Method

### CapAuthorizePreSales Property R

Type	<b>boolean</b>
Remarks	If true, the <b>authorizePreSales</b> method has been implemented.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>authorizePreSales</b> Method

### CapAuthorizeRefund Property R

Type	<b>boolean</b>
Remarks	If true, the <b>authorizeRefund</b> method has been implemented.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>authorizeRefund</b> Method

### CapAuthorizeVoid Property R

Type	<b>boolean</b>
Remarks	If true, the <b>authorizeVoid</b> method has been implemented.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>authorizeVoid</b> Method

## CapAuthorizeVoidPreSales Property R

Type	<b>boolean</b>
Remarks	If true, the <b>authorizeVoidPreSales</b> method has been implemented.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>authorizeVoidPreSales</b> Method

## CapCenterResultCode Property R

Type	<b>boolean</b>
Remarks	If true, the <b>CenterResultCode</b> property has been implemented.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>CenterResultCode</b> Property

## CapCheckCard Property R

Type	<b>boolean</b>
Remarks	If true, the <b>checkCard</b> method has been implemented.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>checkCard</b> Method

## CapDailyLog Property R

Type	int										
Remarks	Holds the daily log ability of the device.										
	<table border="1"> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>CAT_DL_NONE</td> <td>The CAT device does not have the daily log functions.</td> </tr> <tr> <td>CAT_DL_REPORTING</td> <td>The CAT device only has an intermediate total function which reads the daily log but does not erase the log.</td> </tr> <tr> <td>CAT_DL_SETTLEMENT</td> <td>The CAT device only has the “final total” and “erase daily log” functions.</td> </tr> <tr> <td>CAT_DL_REPORTING_SETTLEMENT</td> <td>The CAT device has both the intermediate total function and the final total and erase daily log function.</td> </tr> </tbody> </table>	Value	Meaning	CAT_DL_NONE	The CAT device does not have the daily log functions.	CAT_DL_REPORTING	The CAT device only has an intermediate total function which reads the daily log but does not erase the log.	CAT_DL_SETTLEMENT	The CAT device only has the “final total” and “erase daily log” functions.	CAT_DL_REPORTING_SETTLEMENT	The CAT device has both the intermediate total function and the final total and erase daily log function.
Value	Meaning										
CAT_DL_NONE	The CAT device does not have the daily log functions.										
CAT_DL_REPORTING	The CAT device only has an intermediate total function which reads the daily log but does not erase the log.										
CAT_DL_SETTLEMENT	The CAT device only has the “final total” and “erase daily log” functions.										
CAT_DL_REPORTING_SETTLEMENT	The CAT device has both the intermediate total function and the final total and erase daily log function.										
	This property is initialized by the <b>open</b> method.										
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.										
See Also	<b>DailyLog</b> Property, <b>accessDailyLog</b> Method										

## CapInstallments Property R

Type	boolean
Remarks	If true, the item “Installments” which is stored in the <b>DailyLog</b> property as the result of <b>accessDailyLog</b> will be provided.
	This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>DailyLog</b> Property

## CapPaymentDetail Property R

Type	<b>boolean</b>
Remarks	If true, the <b>PaymentDetail</b> property has been implemented.  This property is initialized by <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>PaymentDetail</b> Property

## CapTaxOthers Property R

Type	<b>boolean</b>
Remarks	If true, the item “TaxOthers” which is stored in the <b>DailyLog</b> property as the result of <b>accessDailyLog</b> will be provided.  Note that this property is not related to the “TaxOthers” argument used with the authorization methods.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>DailyLog</b> Property

## CapTransactionNumber Property R

Type	<b>boolean</b>
Remarks	If true, the <b>TransactionNumber</b> property has been implemented.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>TransactionNumber</b> Property



### CapTrainingMode Property R

Type	<b>boolean</b>
Remarks	If true, the <b>TrainingMode</b> property has been implemented.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>TrainingMode</b> Property

### CardCompanyID Property R

Type	<b>String</b>
Remarks	This property is initialized to an empty string by the <b>open</b> method and is updated when an authorization operation successfully completes.  The length of the ID string varies depending upon the CAT device.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CenterResultCode Property R

Type	<b>String</b>
Remarks	Holds the code from the approval agency. Check the approval agency for the actual codes to be stored.  This property is initialized to an empty string by the <b>open</b> method and is updated when an authorization operation successfully completes.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

**DailyLog Property R****Type**      **String**

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

The details of one transaction are shown as follows:

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

Notes from the previous table:

1) Format

Item	Format
Transaction date	YYYYMMDDHHMMSS
Purchase date	MMDD

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

2) Additional data:

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

3) If the corresponding Cap property is false:

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

4) Amount:

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

5) Purchase date:

- The date manually entered for the purchase transaction after approval.
- The authorization center only requires the month and date of the purchase date be entered.
- This value will not be set (*None* means nothing entered here); it will be set to a date only if the actual purchase date is after the pre-authorization date.

**Example** An example of daily log content is shown below.

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

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

```
102,10,19980116134530,123456,61,12345,0123456,,1234123412341234,12345,,2,12345678[CR][LF]
```

**Errors** A `JposException` may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

**See Also** `CapDailyLog` Property, `accessDailyLog` Method

## PaymentCondition Property R

<b>Type</b>	<b>int</b>																																
<b>Remarks</b>	<p>Holds the payment condition of the most recent successful authorization operation.</p> <p>This property will be set to one of the following values. See <b>PaymentDetail</b> for the detailed payment string that correlates to the following <b>PaymentCondition</b> values.</p> <table> <thead> <tr> <th><b>Value</b></th> <th><b>Meaning</b></th> </tr> </thead> <tbody> <tr> <td>CAT_PAYMENT_LUMP</td> <td>Lump-sum</td> </tr> <tr> <td>CAT_PAYMENT_BONUS_1</td> <td>Bonus 1</td> </tr> <tr> <td>CAT_PAYMENT_BONUS_2</td> <td>Bonus 2</td> </tr> <tr> <td>CAT_PAYMENT_BONUS_3</td> <td>Bonus 3</td> </tr> <tr> <td>CAT_PAYMENT_BONUS_4</td> <td>Bonus 4</td> </tr> <tr> <td>CAT_PAYMENT_BONUS_5</td> <td>Bonus 5</td> </tr> <tr> <td>CAT_PAYMENT_INSTALLMENT_1</td> <td>Installment 1</td> </tr> <tr> <td>CAT_PAYMENT_INSTALLMENT_2</td> <td>Installment 2</td> </tr> <tr> <td>CAT_PAYMENT_INSTALLMENT_3</td> <td>Installment 3</td> </tr> <tr> <td>CAT_PAYMENT_BONUS_COMBINATION_1</td> <td>Bonus combination payments 1</td> </tr> <tr> <td>CAT_PAYMENT_BONUS_COMBINATION_2</td> <td>Bonus combination payments 2</td> </tr> <tr> <td>CAT_PAYMENT_BONUS_COMBINATION_3</td> <td>Bonus combination payments 3</td> </tr> <tr> <td>CAT_PAYMENT_BONUS_COMBINATION_4</td> <td>Bonus combination payments 4</td> </tr> <tr> <td>CAT_PAYMENT_REVOLVING</td> <td>Revolving</td> </tr> <tr> <td>CAT_PAYMENT_DEBIT</td> <td>Debit card</td> </tr> </tbody> </table>	<b>Value</b>	<b>Meaning</b>	CAT_PAYMENT_LUMP	Lump-sum	CAT_PAYMENT_BONUS_1	Bonus 1	CAT_PAYMENT_BONUS_2	Bonus 2	CAT_PAYMENT_BONUS_3	Bonus 3	CAT_PAYMENT_BONUS_4	Bonus 4	CAT_PAYMENT_BONUS_5	Bonus 5	CAT_PAYMENT_INSTALLMENT_1	Installment 1	CAT_PAYMENT_INSTALLMENT_2	Installment 2	CAT_PAYMENT_INSTALLMENT_3	Installment 3	CAT_PAYMENT_BONUS_COMBINATION_1	Bonus combination payments 1	CAT_PAYMENT_BONUS_COMBINATION_2	Bonus combination payments 2	CAT_PAYMENT_BONUS_COMBINATION_3	Bonus combination payments 3	CAT_PAYMENT_BONUS_COMBINATION_4	Bonus combination payments 4	CAT_PAYMENT_REVOLVING	Revolving	CAT_PAYMENT_DEBIT	Debit card
<b>Value</b>	<b>Meaning</b>																																
CAT_PAYMENT_LUMP	Lump-sum																																
CAT_PAYMENT_BONUS_1	Bonus 1																																
CAT_PAYMENT_BONUS_2	Bonus 2																																
CAT_PAYMENT_BONUS_3	Bonus 3																																
CAT_PAYMENT_BONUS_4	Bonus 4																																
CAT_PAYMENT_BONUS_5	Bonus 5																																
CAT_PAYMENT_INSTALLMENT_1	Installment 1																																
CAT_PAYMENT_INSTALLMENT_2	Installment 2																																
CAT_PAYMENT_INSTALLMENT_3	Installment 3																																
CAT_PAYMENT_BONUS_COMBINATION_1	Bonus combination payments 1																																
CAT_PAYMENT_BONUS_COMBINATION_2	Bonus combination payments 2																																
CAT_PAYMENT_BONUS_COMBINATION_3	Bonus combination payments 3																																
CAT_PAYMENT_BONUS_COMBINATION_4	Bonus combination payments 4																																
CAT_PAYMENT_REVOLVING	Revolving																																
CAT_PAYMENT_DEBIT	Debit card																																
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.																																
<b>See Also</b>	<b>PaymentDetail</b> Property																																

**PaymentDetail Property R****Type**      **String**

**Remarks**      Holds payment condition details as the result of an authorization operation. Payment details vary depending on the value of **PaymentCondition**. The data will be stored as comma separated ASCII code. An empty string means that no data is stored.

<b>PaymentCondition</b>	<b>PaymentDetail</b>
CAT_PAYMENT_LUMP	empty string
CAT_PAYMENT_BONUS_1	empty string
CAT_PAYMENT_BONUS_2	Number of bonus payments
CAT_PAYMENT_BONUS_3	1 <sup>st</sup> bonus month
CAT_PAYMENT_BONUS_4*	Number of bonus payments, 1 <sup>st</sup> bonus month, 2 <sup>nd</sup> bonus month, 3 <sup>rd</sup> bonus month, 4 <sup>th</sup> bonus month, 5 <sup>th</sup> bonus month, 6 <sup>th</sup> bonus month
CAT_PAYMENT_BONUS_5*	Number of bonus payments, 1 <sup>st</sup> bonus month, 1 <sup>st</sup> bonus amount, 2 <sup>nd</sup> bonus month, 2 <sup>nd</sup> bonus amount, 3 <sup>rd</sup> bonus month, 3 <sup>rd</sup> bonus amount, 4 <sup>th</sup> bonus month, 4 <sup>th</sup> bonus amount, 5 <sup>th</sup> bonus month, 5 <sup>th</sup> bonus amount, 6 <sup>th</sup> bonus month, 6 <sup>th</sup> bonus amount
CAT_PAYMENT_INSTALLMENT_1	1 <sup>st</sup> billing month, Number of payments
CAT_PAYMENT_INSTALLMENT_2*	1 <sup>st</sup> billing month, Number of payments, 1 <sup>st</sup> amount, 2 <sup>nd</sup> amount, 3 <sup>rd</sup> amount, 4 <sup>th</sup> amount, 5 <sup>th</sup> amount, 6 <sup>th</sup> amount
CAT_PAYMENT_INSTALLMENT_3	1 <sup>st</sup> billing month, Number of payments, 1 <sup>st</sup> amount
CAT_PAYMENT_BONUS_COMBINATION_1	1 <sup>st</sup> billing month, Number of payments
CAT_PAYMENT_BONUS_COMBINATION_2	1 <sup>st</sup> billing month, Number of payments, bonus amount
CAT_PAYMENT_BONUS_COMBINATION_3*	1 <sup>st</sup> billing month, Number of payments, number of bonus payments, 1 <sup>st</sup> bonus month, 2 <sup>nd</sup> bonus month, 3 <sup>rd</sup> bonus month, 4 <sup>th</sup> bonus month, 5 <sup>th</sup> bonus month, 6 <sup>th</sup> bonus month
CAT_PAYMENT_BONUS_COMBINATION_4*	1 <sup>st</sup> billing month, Number of payments, number of bonus payments, 1 <sup>st</sup> bonus month, 1 <sup>st</sup> bonus amount, 2 <sup>nd</sup> bonus month, 2 <sup>nd</sup> bonus amount, 3 <sup>rd</sup> bonus month, 3 <sup>rd</sup> bonus amount, 4 <sup>th</sup> bonus month, 4 <sup>th</sup> bonus amount, 5 <sup>th</sup> bonus month, 5 <sup>th</sup> bonus amount, 6 <sup>th</sup> bonus month, 6 <sup>th</sup> bonus amount
CAT_PAYMENT_REVOLVING	empty string
CAT_PAYMENT_DEBIT	empty string

\*Maximum 6 entries

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

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

Entry item	PaymentCondition Value	CAT Name	CAT (Old CAT)	G-CAT	JET-S	SG-CAT	Master-T
		Credit Card	Not specified	Not specified	JCB	VISA	MASTER
		JavaPOS Term	<b>Card Company Terms</b>				
<b>Number of bonus payments</b> Bonus month (1) Bonus month (2) Bonus month (3) Bonus month (4) Bonus month (5) Bonus month (6)	<b>24</b>	Bonus 4	Bonus 4	Bonus 3	Bonus 3	Bonus 4 (Up to two entries for bonus month)	Bonus 4





Entry item	PaymentCondition Value	CAT Name	CAT (Old CAT)	G-CAT	JET-S	SG-CAT	Master-T	
		Credit Card	Not specified	Not specified	JCB	VISA	MASTER	
		JavaPOS Term	<b>Card Company Terms</b>					
<b>Payment start month</b> <b>Number of payments</b> Installment amount (1) Installment amount (2) Installment amount (3) Installment amount (4) Installment amount (5) Installment amount (6)	62	Installment 2	Installment 2	Does not exist.	Does not exist.	Does not exist.	Does not exist.	
<b>Payment start month</b> <b>Number of payments</b> Initial amount	63	Installment 3	Installment 3	Installment 2	Installment 2	Does not exist.	Installment 2	
<b>Payment start month</b> <b>Number of payments</b>	31	Bonus Combination 1	Bonus Combination 1	Bonus Combination 1	Bonus Combination 1	Bonus Combination 1	Bonus Combination 1	
<b>Payment start month</b> <b>Number of payments</b> Bonus amount	32	Bonus Combination 2	Bonus Combination 2	Does not exist.	Does not exist.	Bonus Combination 2	Bonus Combination 2	

Entry item	PaymentCondition Value	CAT Name	CAT (Old CAT)	G-CAT	JET-S	SG-CAT	Master-T
		Credit Card	Not specified	Not specified	JCB	VISA	MASTER
		JavaPOS Term	<b>Card Company Terms</b>				
<b>Payment start month</b> <b>Number of payments</b> <b>Number of bonus payments</b> Bonus month (1) Bonus month (2) Bonus month (3) Bonus month (4) Bonus month (5) Bonus month (6)	33	Bonus Combination 3	Bonus Combination 3	Does not exist.	Does not exist.	Bonus Combination 3 (Up to two entries for bonus month)	Bonus Combination 3

	<b>Payment start month</b> <b>Number of payments</b> <b>Number of bonus payments</b> Bonus month (1) Bonus amount (1) Bonus month (2) Bonus amount (2) Bonus month (3) Bonus amount (3) Bonus month (4) Bonus amount (4) Bonus month (5) Bonus amount (5) Bonus month (6) Bonus amount (6)	34	Bonus Combination 4	Bonus Combination 4	Bonus Combination 2	Bonus Combination 2	Bonus Combination 4  (Up to two entries for bonus month and amount)	Bonus Combination 4
<b>Revolving</b>	(None)	80	Revolving	Revolving	Revolving	Revolving	Revolving	Revolving
<b>Debit</b>	(None)	110	(Support depends upon actual device)	(Support depends upon actual device)	(Support depends upon actual device)	(Support depends upon actual device)	(Support depends upon actual device)	(Support depends upon actual device)

**Errors** A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

**See Also** **CapPaymentDetail** and **PaymentCondition** Properties

**PaymentMedia Property R      *Added in Release 1.5***

**Type**      **int**

**Remarks**      Holds the payment media type that the approval method should approve.

The application sets this property to one of the following values before issuing an approval method call. “None specified” means that payment media will be determined by the CAT device, not by the POS application.

<b>Value</b>	<b>Meaning</b>
CAT_MEDIA_UNSPECIFIED	None specified.
CAT_MEDIA_CREDIT	Credit card.
CAT_MEDIA_DEBIT	Debit card.

This property is initialized to CAT\_MEDIA\_UNSPECIFIED by the **open** method.

**Errors**      A JposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

**SequenceNumber Property R**

**Type**      **int**

**Remarks**      Holds a “sequence number” as the result of each method call. This number needs to be checked by an application to see if it matches with the value in the property **SequenceNumber** of the originating method.

The “*sequence number*” received back from the CAT device is expected to be a numeric value. If other than numeric values are returned from the CAT device, the value stored in this property will be set to zero (0).

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

**Errors**      A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

**SlipNumber Property R**

**Type**      **String**

**Remarks**      Holds a “slip number” as the result of each authorization operation.

This property is initialized to an empty string by the **open** method and is updated when an authorization operation successfully completes.

**Errors**      A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

## TrainingMode Property R/W

<b>Type</b>	<b>boolean</b>
<b>Remarks</b>	<p>If true, each operation will be run in training mode; otherwise each operation will be run in normal mode.</p> <p><b>TrainingMode</b> needs to be explicitly set to false by an application to exit from training mode, because it will not automatically be set to false after the completion of an operation.</p> <p>This property will be initialized to false by the <b>open</b> method.</p>
<b>Errors</b>	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p> <p>Some possible values of the exception’s <i>ErrorCode</i> property are:</p>

<b>Value</b>	<b>Meaning</b>
JPOS_E_ILLEGAL	<b>CapTrainingMode</b> is false.

## TransactionNumber Property R

<b>Type</b>	<b>String</b>
<b>Remarks</b>	<p>Holds a “transaction number” as the result of each authorization operation.</p> <p>This property is initialized to the empty string by the <b>open</b> method and is updated when an authorization operation successfully completes.</p>
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

## TransactionType Property R

<b>Type</b>	<b>int</b>																
<b>Remarks</b>	<p>Holds a “transaction type” as the result of each authorization operation.</p> <p>This property is initialized to zero by the <b>open</b> method and is updated when an authorization operation successfully completes.</p> <p>This property has one of the following values:</p> <table border="1"> <thead> <tr> <th><b>Value</b></th> <th><b>Meaning</b></th> </tr> </thead> <tbody> <tr> <td>CAT_TRANSACTION_SALES</td> <td>Sales</td> </tr> <tr> <td>CAT_TRANSACTION_VOID</td> <td>Cancellation</td> </tr> <tr> <td>CAT_TRANSACTION_REFUND</td> <td>Refund purchase</td> </tr> <tr> <td>CAT_TRANSACTION_COMPLETION</td> <td>Purchase after approval</td> </tr> <tr> <td>CAT_TRANSACTION_PRESALES</td> <td>Pre-authorization</td> </tr> <tr> <td>CAT_TRANSACTION_CHECKCARD</td> <td>Card Check</td> </tr> <tr> <td>CAT_TRANSACTION_VOIDPRESALES</td> <td>Cancel pre-authorization approval</td> </tr> </tbody> </table>	<b>Value</b>	<b>Meaning</b>	CAT_TRANSACTION_SALES	Sales	CAT_TRANSACTION_VOID	Cancellation	CAT_TRANSACTION_REFUND	Refund purchase	CAT_TRANSACTION_COMPLETION	Purchase after approval	CAT_TRANSACTION_PRESALES	Pre-authorization	CAT_TRANSACTION_CHECKCARD	Card Check	CAT_TRANSACTION_VOIDPRESALES	Cancel pre-authorization approval
<b>Value</b>	<b>Meaning</b>																
CAT_TRANSACTION_SALES	Sales																
CAT_TRANSACTION_VOID	Cancellation																
CAT_TRANSACTION_REFUND	Refund purchase																
CAT_TRANSACTION_COMPLETION	Purchase after approval																
CAT_TRANSACTION_PRESALES	Pre-authorization																
CAT_TRANSACTION_CHECKCARD	Card Check																
CAT_TRANSACTION_VOIDPRESALES	Cancel pre-authorization approval																
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.																

## Methods

### accessDailyLog Method

**Syntax**      **void accessDailyLog (int *sequenceNumber*, int *type*, int *timeout*) throws JposException;**

Parameter	Description
<i>sequenceNumber</i>	The sequence number to get daily log.
<i>type</i>	Specify whether the daily log is intermediate total or final total and erase.
<i>timeout</i>	The maximum waiting time (in milliseconds) until the response is received from the CAT device. JPOS_FOREVER(-1), 0, and positive values can be specified.

**Remarks**      Gets daily log from CAT. Daily log will be retrieved and stored in **DailyLog** as specified by *sequenceNumber*.

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

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

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

**Errors**      A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_ILLEGAL	Invalid or unsupported <i>type</i> or <i>timeout</i> parameter was specified, or <b>CapDailyLog</b> is false.
JPOS_E_TIMEOUT	No response was received from CAT during the specified <i>timeout</i> time in milliseconds.

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



## authorizeCompletion Method

**Syntax**     **void authorizeCompletion (int *sequenceNumber*, long *amount*, long *taxOthers*, int *timeout*) throws JposException;**

Parameter	Description
<i>sequenceNumber</i>	Sequence number for approval
<i>amount</i>	Purchase amount for approval
<i>taxOthers</i>	Tax and other amounts for approval
<i>timeout</i>	The maximum waiting time (in milliseconds) until the response is received from the CAT device. JPOS_FOREVER(-1), 0 and positive values can be specified.

**Remarks**     This method is intended to be used after a purchase is approved.

The *sequenceNumber* tracks the sale *amount* and *taxOthers* parameters once the transaction is approved.

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

**Errors**     A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_ILLEGAL	Invalid <i>timeout</i> parameter was specified, or <b>CapAuthorizeCompletion</b> is false.
JPOS_E_TIMEOUT	No response was received from CAT during the specified <i>timeout</i> time in milliseconds.

**See Also**     **CapAuthorizeCompletion** Property

**authorizePreSales Method**

**Syntax**      **void authorizePreSales (int *sequenceNumber*, long *amount*, long *taxOthers*, int *timeout*) throws JposException;**

Parameter	Description
<i>sequenceNumber</i>	Sequence number for approval
<i>amount</i>	Purchase amount for approval
<i>taxOthers</i>	Tax and other amounts for approval
<i>timeout</i>	The maximum waiting time (in milliseconds) until the response is received from the CAT device. JPOS_FOREVER(-1), 0 and positive values can be specified.

**Remarks**      This method is intended to be used with a pre-authorization sale.  
Pre-authorization for *amount* and *taxOthers* is made as the approval specified by *sequenceNumber*.

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

**Errors**      A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_ILLEGAL	Invalid <i>timeout</i> parameter was specified, or <b>CapAuthorizePreSales</b> is FALSE.
JPOS_E_TIMEOUT	No response was received from CAT during the specified <i>timeout</i> time in milliseconds.

**See Also**      **CapAuthorizePreSales** Property

## authorizeRefund Method

**Syntax**     **void authorizeRefund (int *sequenceNumber*, long *amount*, long *taxOthers*, int *timeout*) throws JposException;**

Parameter	Description
<i>sequenceNumber</i>	Sequence number for approval
<i>amount</i>	Purchase amount for approval
<i>taxOthers</i>	Tax and other amounts for approval
<i>timeout</i>	The maximum waiting time (in milliseconds) until the response is received from the CAT device. JPOS_FOREVER(-1), 0 and positive values can be specified.

**Remarks**     This method is intended to be used when a refund approval is required.

Refund purchase approval for *amount* and *taxOthers* is intended as the approval specified by *sequenceNumber*.

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

**Errors**     A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_ILLEGAL	Invalid <i>timeout</i> parameter was specified, or <b>CapAuthorizeRefund</b> is false.
JPOS_E_TIMEOUT	No response was received from CAT during the specified <i>timeout</i> time in milliseconds.

**See Also**     **CapAuthorizeRefund** Property

**authorizeSales Method**

**Syntax**     **void authorizeSales (int *sequenceNumber*, long *amount*, long *taxOthers*, int *timeout*) throws JposException;**

Parameter	Description
<i>sequenceNumber</i>	Sequence number for approval
<i>amount</i>	Purchase amount for approval
<i>taxOthers</i>	Tax and other amounts for approval
<i>timeout</i>	The maximum waiting time (in milliseconds) until the response is received from the CAT device. JPOS_FOREVER(-1), 0 and positive values can be specified.

**Remarks**     This method is intended to be used with a normal purchase transaction.

Normal purchase approval for *amount* and *taxOthers* is intended as the approval specified by *sequenceNumber*.

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

**Errors**     A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_ILLEGAL	Invalid <i>timeout</i> parameter was specified.
JPOS_E_TIMEOUT	No response was received from CAT during the specified <i>timeout</i> time in milliseconds.

## authorizeVoid Method

**Syntax**     **void authorizeVoid (int *sequenceNumber*, long *amount*, long *taxOthers*, int *timeout*) throws JposException;**

Parameter	Description
<i>sequenceNumber</i>	Sequence number for approval
<i>amount</i>	Purchase amount for approval
<i>taxOthers</i>	Tax and other amounts for approval
<i>timeout</i>	The maximum waiting time (in milliseconds) until the response is received from the CAT device. JPOS_FOREVER(-1), 0 and positive values can be specified.

**Remarks**     This method is intended to be used when a purchase needs to be cancelled. Cancellation approval for *amount* and *taxOthers* is intended as the approval specified by *sequenceNumber*.

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

**Errors**     A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_ILLEGAL	Invalid <i>timeout</i> parameter was specified, or <b>CapAuthorizeVoid</b> is false.
JPOS_E_TIMEOUT	No response was received from CAT during the specified <i>timeout</i> time in milliseconds.

**See Also**     **CapAuthorizeVoid** Property

**authorizeVoidPreSales Method**

**Syntax**      **void authorizeVoidPreSales (int *sequenceNumber*, long *amount*, long *taxOthers*, int *timeout*) throws JposException;**

Parameter	Description
<i>sequenceNumber</i>	Sequence number for approval
<i>amount</i>	Purchase amount for approval
<i>taxOthers</i>	Tax and other amounts for approval
<i>timeout</i>	The maximum waiting time (in milliseconds) until the response is received from the CAT device. JPOS_FOREVER(-1), 0 and positive values can be specified.

**Remarks**      This method is invoked when it is necessary to void a pre-authorization approval. Pre-authorization cancellation approval for *amount* and *taxOthers* is intended as the approval specified by *sequenceNumber*.

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

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

**Errors**      A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_ILLEGAL	Invalid <i>timeout</i> parameter was specified, or <b>CapAuthorizeVoidPreSales</b> is false.
JPOS_E_TIMEOUT	No response was received from CAT during the specified <i>timeout</i> time in milliseconds.

**See Also**      **CapAuthorizeVoidPreSales** Property

## checkCard Method

**Syntax**      **void checkCard (int *sequenceNumber*, int *timeout*) throws JposException;**

Parameter	Description
<i>sequenceNumber</i>	Sequence number for approval
<i>timeout</i>	The maximum waiting time (in milliseconds) until the response is received from the CAT device. JPOS_FOREVER(-1), 0 and positive values can be specified.

**Remarks**      This method is intended to be used when a card verification is required.  
 Card check will be made as specified by *sequenceNumber*.  
 When *timeout* is JPOS\_FOREVER(-1), timeout never occurs and the device waits until it receives response from the CAT.

**Errors**      A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_ILLEGAL	Invalid <i>timeout</i> parameter was specified, or <b>CapCheckCard</b> is false.
JPOS_E_TIMEOUT	No response was received from CAT during the specified <i>timeout</i> time in milliseconds.

**See Also**      **CapCheckCard** Property

## Events

### DirectIOEvent

**Interface** `jpos.events.DirectIOListener`

**Method** `directIOOccurred (DirectIOEvent e);`

**Description** Provides Device Service information directly to the application. This event provides a means for a vendor-specific CAT Device Service to provide events to the application that are not otherwise supported by the Device Control.

**Properties** This event contains the following properties:

Property	Type	Description
<i>EventNumber</i>	<i>int</i>	Event number whose specific values are assigned by the Device Service.
<i>Data</i>	<i>int</i>	Additional numeric data. Specific values vary by the <i>EventNumber</i> and the Device Service. This property is settable.
<i>Object</i>	<i>Object</i>	Additional data whose specific values vary by the <i>EventNumber</i> and Device Service. This property is settable.

**Remarks** This to be used only for those types of vendor specific functions that are not otherwise described as part of the JavaPOS standard. Use of this event may restrict the application program from being used with other vendor's CAT devices which may not have any knowledge of the Device Service's need for this event.

**See Also** "Events" on page 18, **directIO** Method



## ErrorEvent

<b>Interface</b>	<b>jpos.events.ErrorListener</b>
<b>Method</b>	<b>errorOccurred (ErrorEvent e);</b>
<b>Description</b>	Notifies the application that a CAT error has been detected and a suitable response by the application is necessary to process the error condition.
<b>Properties</b>	This event contains the following properties:

Property	Type	Description
<i>ErrorCode</i>	<i>int</i>	Error Code causing the error event. See list of <i>ErrorCodes</i> on page 16.
<i>ErrorCodeExtended</i>	<i>int</i>	Extended Error Code causing the error event. If <i>ErrorCode</i> is JPOS_E_EXTENDED, then see values below. Otherwise, it may contain a Service-specific value.
<i>ErrorLocus</i>	<i>int</i>	Location of the error, and is set to JPOS_EL_OUTPUT indicating the error occurred while processing asynchronous output.
<i>ErrorResponse</i>	<i>int</i>	Error response, whose default value may be overridden by the application (i.e., this property is settable). See values below.

If *ErrorCode* is JPOS\_E\_EXTENDED, then *ErrorCodeExtended* has one of the following values:

Value	Meaning
JPOS_ECAT_CENTERERROR	An error was returned from the approval agency. The detail error code is defined in <b>CenterResultCode</b> .
JPOS_ECAT_COMMANDERROR	The command sent to CAT is wrong. This error is never returned so long as CAT Control is working correctly.
JPOS_ECAT_RESET	CAT was stopped during processing by CAT reset key (stop key) and so on.
JPOS_ECAT_COMMUNICATIONERROR	Communication error has occurred between the approval agency and CAT.
JPOS_ECAT_DAILYLOGOVERFLOW	Daily log was too big to be stored. Keeping daily log has been stopped and the value of <b>DailyLog</b> property is uncertain.

The application's error event listener may change *ErrorResponse* to one of the following values:

Value	Meaning
JPOS_ER_RETRY	Retries the asynchronous processing. The error state is exited. The default.
JPOS_ER_CLEAR	Clear the asynchronous processing. The error state is exited.

**Remarks** Enqueued when an error is detected while processing an asynchronous authorize group method or the **accessDailyLog** method. The Control's State transitions into the error state. This event is not delivered until **DataEventEnabled** is true, so that proper application sequencing occurs.

**See Also** "Device Output Models" on page 25, "Device States" on page 30

## OutputCompleteEvent

**Interface** `jpos.events.OutputCompleteListener`

**Method** `outputCompleteOccurred (OutputCompleteEvent e);`

**Description** Notifies the application that the queued output request associated with the *OutputID* property has completed successfully.

**Properties** This event contains the following property:

Property	Type	Description
<i>OutputID</i>	<i>int</i>	The ID number of the asynchronous output request that is complete.

**Remarks** Enqueued when the request's data has been both sent and the Device Service has confirmation that it was processed by the device successfully.

**See Also** "Device Output Models" on page 25

## StatusUpdateEvent

**Interface**     **jpos.events.StatusUpdateListener**

**Method**        **statusUpdateOccurred (StatusUpdateEvent e);**

**Description**    Notifies the application that there is a change in the power status of the CAT device.

**Properties**      This event contains the following property:

<b>Property</b>	<b>Type</b>	<b>Description</b>
<i>Status</i>	<i>int</i>	Reports a change in the power status of the CAT device.

**Remarks**        Enqueued when the CAT device detects a power state change.

**See Also**        “Events” on page 18, “Device Power Reporting Model” on page 27, **CapPowerReporting** Property, **PowerNotify** Property.



# Coin Dispenser

## Summary

### Properties

<i>Common</i>	<i>Ver</i>	<i>Type</i>	<i>Access</i>	<i>May Use After</i>
<b>AutoDisable</b>		boolean	R/W	<i>Not Supported</i>
<b>CapPowerReporting</b>	1.3	int	R	open
<b>CheckHealthText</b>		String	R	open
<b>Claimed</b>		boolean	R	open
<b>DataCount</b>		int	R	<i>Not Supported</i>
<b>DataEventEnabled</b>		boolean	R/W	<i>Not Supported</i>
<b>DeviceEnabled</b>		boolean	R/W	open & claim
<b>FreezeEvents</b>		boolean	R/W	open
<b>OutputID</b>		int	R	<i>Not Supported</i>
<b>PowerNotify</b>	1.3	int	R/W	open
<b>PowerState</b>	1.3	int	R	open
<b>State</b>		int	R	--
<b>DeviceControlDescription</b>		String	R	--
<b>DeviceControlVersion</b>		int	R	--
<b>DeviceServiceDescription</b>		String	R	open
<b>DeviceServiceVersion</b>		int	R	open
<b>PhysicalDeviceDescription</b>		String	R	open
<b>PhysicalDeviceName</b>		String	R	open

<i>Specific</i>	<i>Ver</i>	<i>Type</i>	<i>Access</i>	<i>May Use After</i>
<b>CapEmptySensor</b>		boolean	R	open
<b>CapJamSensor</b>		boolean	R	open
<b>CapNearEmptySensor</b>		boolean	R	open
<b>DispenserStatus</b>		int	R	open, claim, & enable

**Methods**

<i>Common</i>	<i>Ver</i>	<i>May Use After</i>
<b>open</b>		--
<b>close</b>		open
<b>claim</b>		open
<b>release</b>		open & claim
<b>checkHealth</b>		open, claim, & enable
<b>clearInput</b>		<i>Not Supported</i>
<b>clearOutput</b>		<i>Not Supported</i>
<b>directIO</b>		open
 <i>Specific</i>		
<b>dispenseChange</b>		open, claim, & enable

**Events**

<i>Name</i>	<i>Ver</i>	<i>May Occur After</i>
<b>DataEvent</b>		<i>Not Supported</i>
<b>DirectIOEvent</b>	1.3	open & claim
<b>ErrorEvent</b>		<i>Not Supported</i>
<b>OutputCompleteEvent</b>		<i>Not Supported</i>
<b>StatusUpdateEvent</b>		open, claim, & enable

## General Information

The Coin Dispenser Control's class name is "jpos.CoinDispenser".  
The device constants are contained in the class "jpos.CoinDispenserConst".  
See "Package Structure" on page 40.

### Capabilities

The coin dispenser has the following capability:

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

The coin dispenser may have the following additional capability:

- Status reporting, which indicates empty coin slot conditions, near empty coin slot conditions, and coin slot jamming conditions.

### Model

The general model of a coin dispenser is:

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

### Device Sharing

The coin dispenser is an exclusive-use device, as follows:

- The application must claim the device before enabling it.
- The application must claim and enable the device before accessing some of the properties, dispensing change, or receiving status update events.
- See the "Summary" table for precise usage prerequisites.



## Properties

### CapEmptySensor Property R

Type	<b>boolean</b>
Remarks	If true, the coin dispenser can report an out-of-coinage condition.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapJamSensor Property R

Type	<b>boolean</b>
Remarks	If true, the coin dispenser can report a mechanical jam or failure condition.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapNearEmptySensor Property R

Type	<b>boolean</b>
Remarks	If true, the coin dispenser can report when it is almost out of coinage.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

## DispenserStatus Property R

**Type**      **int**

**Remarks**      Holds the current status of the dispenser. It has one of the following values:

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

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

**Errors**      A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

## Methods

### dispenseChange Method

**Syntax**      **void dispenseChange (int *amount*) throws JposException;**

The *amount* parameter contains the amount of change to be dispensed.

**Remarks**      Dispenses change. The value represented by the *amount* parameter is a count of the currency units to dispense (such as cents or yen).

**Errors**          A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
JPOS_E_ILLEGAL	An <i>amount</i> parameter value of zero was specified, or the <i>amount</i> parameter contained a negative value or a value greater than the device can dispense.

## Events

### DirectIOEvent

**Interface** `jpos.events.DirectIOListener`

**Method** `directIOOccurred (DirectIOEvent e);`

**Description** Provides Device Service information directly to the application. This event provides a means for a vendor-specific Coin Dispenser Device Service to provide events to the application that are not otherwise supported by the Device Control.

**Properties** This event contains the following properties:

Property	Type	Description
<i>EventNumber</i>	<i>int</i>	Event number whose specific values are assigned by the Device Service.
<i>Data</i>	<i>int</i>	Additional numeric data. Specific values vary by the <i>EventNumber</i> and the Device Service. This property is settable.
<i>Object</i>	<i>Object</i>	Additional data whose usage varies by the <i>EventNumber</i> and Device Service. This property is settable.

**Remarks** This event is to be used only for those types of vendor specific functions that are not otherwise described as part of the JavaPOS standard. Use of this event may restrict the application program from being used with other vendor's Coin Dispenser devices which may not have any knowledge of the Device Service's need for this event.

**See Also** "Events" on page 18, **directIO** Method

## StatusUpdateEvent

**Interface** `jpos.events.StatusUpdateListener`

**Method** `statusUpdateOccurred (StatusEvent e);`

**Description** Notifies the application of a sensor status change.

**Properties** This event contains the following property:

Property	Type	Description
<i>Status</i>	<i>int</i>	The status reported from the Coin Dispenser.

The *Status* property has one of the following values:

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

*Note that Release 1.3* added Power State Reporting with additional *Power reporting StatusUpdateEvent values*. See “StatusUpdateEvent” description on page 78.

**Remarks** This event applies for status changes of the sensor types supported, as indicated by the capability properties. It also applies if Power State Reporting is enabled.



# Fiscal Printer

## Summary

### Properties

#### Common

	<i>Ver</i>	<i>Type</i>	<i>Access</i>	<i>May Use After</i>
<b>AutoDisable</b>	1.3	boolean	R/W	<i>Not Supported</i>
<b>CapPowerReporting</b>	1.3	int	R	open
<b>CheckHealthText</b>	1.3	String	R	open
<b>Claimed</b>	1.3	boolean	R	open
<b>DataCount</b>	1.3	int	R	<i>Not Supported</i>
<b>DataEventEnabled</b>	1.3	boolean	R/W	<i>Not Supported</i>
<b>DeviceEnabled</b>	1.3	boolean	R/W	open & claim
<b>FreezeEvents</b>	1.3	boolean	R/W	open
<b>OutputID</b>	1.3	int	R	open
<b>PowerState</b>	1.3	int	R	open
<b>PowerNotify</b>	1.3	int	R/W	open
<b>State</b>	1.3	int	R	--
<b>DeviceControlDescription</b>	1.3	String	R	--
<b>DeviceControlVersion</b>	1.3	int	R	--
<b>DeviceServiceDescription</b>	1.3	String	R	open
<b>DeviceServiceVersion</b>	1.3	int	R	open
<b>PhysicalDeviceDescription</b>	1.3	String	R	open
<b>PhysicalDeviceName</b>	1.3	String	R	open

<i>Specific</i>	<i>Ver</i>	<i>Type</i>	<i>Access</i>	<i>May Use After</i>
<b>CapAdditionalLines</b>	1.3	boolean	R	open
<b>CapAmountAdjustment</b>	1.3	boolean	R	open
<b>CapAmountNotPaid</b>	1.3	boolean	R	open
<b>CapCheckTotal</b>	1.3	boolean	R	open
<b>CapCoverSensor (2)</b>	1.3	boolean	R	open
<b>CapDoubleWidth</b>	1.3	boolean	R	open
<b>CapDuplicateReceipt</b>	1.3	boolean	R	open
<b>CapFixedOutput</b>	1.3	boolean	R	open
<b>CapHasVatTable</b>	1.3	boolean	R	open
<b>CapIndependentHeader</b>	1.3	boolean	R	open
<b>CapItemList</b>	1.3	boolean	R	open
<b>CapJrnEmptySensor (2)</b>	1.3	boolean	R	open
<b>CapJrnNearEndSensor (2)</b>	1.3	boolean	R	open
<b>CapJrnPresent (2)</b>	1.3	boolean	R	open
<b>CapNonFiscalMode</b>	1.3	boolean	R	open
<b>CapOrderAdjustmentFirst</b>	1.3	boolean	R	open
<b>CapPercentAdjustment</b>	1.3	boolean	R	open
<b>CapPositiveAdjustment</b>	1.3	boolean	R	open
<b>CapPowerLossReport</b>	1.3	boolean	R	open
<b>CapPredefinedPayment Lines</b>	1.3	boolean	R	open
<b>CapReceiptNotPaid</b>	1.3	boolean	R	open
<b>CapRecEmptySensor (2)</b>	1.3	boolean	R	open
<b>CapRecNearEndSensor (2)</b>	1.3	boolean	R	open
<b>CapRecPresent (2)</b>	1.3	boolean	R	open
<b>CapRemainingFiscal Memory</b>	1.3	boolean	R	open
<b>CapReservedWord</b>	1.3	boolean	R	open
<b>CapSetHeader</b>	1.3	boolean	R	open
<b>CapSetPOSID</b>	1.3	boolean	R	open
<b>CapSetStoreFiscalID</b>	1.3	boolean	R	open
<b>CapSetTrailer</b>	1.3	boolean	R	open
<b>CapSetVatTable</b>	1.3	boolean	R	open



*Specific (continued)*

	<i>Ver</i>	<i>Type</i>	<i>Access</i>	<i>May Use After</i>
<b>CapSlpEmptySensor</b> (2)	1.3	boolean	R	open
<b>CapSlpFiscalDocument</b>	1.3	boolean	R	open
<b>CapSlpFullSlip</b> (2)	1.3	boolean	R	open
<b>CapSlpNearEndSensor</b> (2)	1.3	boolean	R	open
<b>CapSlpPresent</b> (2)	1.3	boolean	R	open
<b>CapSlpValidation</b>	1.3	boolean	R	open
<b>CapSubAmountAdjustment</b>	1.3	boolean	R	open
<b>CapSubPercentAdjustment</b>	1.3	boolean	R	open
<b>CapSubtotal</b>	1.3	boolean	R	open
<b>CapTrainingMode</b>	1.3	boolean	R	open
<b>CapValidateJournal</b>	1.3	boolean	R	open
<b>CapXReport</b>	1.3	boolean	R	open
<b>AmountDecimalPlaces</b>	1.3	int	R	open, claim, & enable
<b>AsyncMode</b>	1.3	boolean	R/W	open
<b>CheckTotal</b>	1.3	boolean	R/W	open
<b>CountryCode</b>	1.3	int	R	open
<b>CoverOpen</b> (2)	1.3	boolean	R	open, claim, & enable
<b>DayOpened</b>	1.3	boolean	R	open, claim, & enable
<b>DescriptionLength</b>	1.3	int	R	open
<b>DuplicateReceipt</b>	1.3	boolean	R/W	open
<b>ErrorLevel</b>	1.3	int	R	open
<b>ErrorOutID</b>	1.3	int	R	open
<b>ErrorState</b>	1.3	int	R	open
<b>ErrorStation</b>	1.3	int	R	open
<b>ErrorString</b>	1.3	String	R	open
<b>FlagWhenIdle</b>	1.3	boolean	R/W	open
<b>JrnEmpty</b> (2)	1.3	boolean	R	open, claim, & enable
<b>JrnNearEnd</b> (2)	1.3	boolean	R	open, claim, & enable
<b>MessageLength</b>	1.3	int	R	open
<b>NumHeaderLines</b>	1.3	int	R	open
<b>NumTrailerLines</b>	1.3	int	R	open
<b>NumVatRates</b>	1.3	int	R	open

*Specific (continued)*

	<i>Ver</i>	<i>Type</i>	<i>Access</i>	<i>May Use After</i>
<b>PredefinedPaymentLines</b>	1.3	String	R	open
<b>PrinterState</b>	1.3	int	R	open
<b>QuantityDecimalPlaces</b>	1.3	int	R	open
<b>QuantityLength</b>	1.3	int	R	open
<b>RecEmpty</b> (2)	1.3	boolean	R	open, claim, & enable
<b>RecNearEnd</b> (2)	1.3	boolean	R	open, claim, & enable
<b>RemainingFiscalMemory</b>	1.3	int	R	open, claim, & enable
<b>ReservedWord</b> (1)	1.3	String	R	open
<b>SlpEmpty</b> (2)	1.3	boolean	R	open, claim, & enable
<b>SlpNearEnd</b> (2)	1.3	boolean	R	open, claim, & enable
<b>SlipSelection</b>	1.3	int	R/W	open, claim, & enable
<b>TrainingModeActive</b>	1.3	boolean	R	open, claim, & enable

**Methods*****Common***

	<b><i>Ver</i></b>	<b><i>May Use After</i></b>
<b>open</b>	1.3	--
<b>close</b>	1.3	open
<b>claim</b>	1.3	open
<b>release</b>	1.3	open & claim
<b>checkHealth</b>	1.3	open, claim, & enable
<b>clearInput</b>	1.3	<i>Not Supported</i>
<b>clearOutput</b>	1.3	open & claim
<b>directIO</b>	1.3	open

***Specific - Presetting Fiscal***

<b>setDate</b>	1.3	open, claim, & enable
<b>setHeaderLine</b>	1.3	open, claim, & enable
<b>setPOSID (1)</b>	1.3	open, claim, & enable
<b>setStoreFiscalID</b>	1.3	open, claim, & enable
<b>setTrailerLine</b>	1.3	open, claim, & enable
<b>setVatTable</b>	1.3	open, claim, & enable
<b>setVatValue</b>	1.3	open, claim, & enable

***Specific - Fiscal Receipt***

<b>beginFiscalReceipt</b>	1.3	open, claim, & enable
<b>endFiscalReceipt</b>	1.3	open, claim, & enable
<b>printDuplicateReceipt</b>	1.3	open, claim, & enable
<b>printRecItem</b>	1.3	open, claim, & enable
<b>printRecItemAdjustment</b>	1.3	open, claim, & enable
<b>printRecMessage</b>	1.3	open, claim, & enable
<b>printRecNotPaid</b>	1.3	open, claim, & enable
<b>printRecRefund</b>	1.3	open, claim, & enable
<b>printRecSubtotal</b>	1.3	open, claim, & enable
<b>printRecSubtotalAdjustment</b>	1.3	open, claim, & enable
<b>printRecTotal</b>	1.3	open, claim, & enable
<b>printRecVoid</b>	1.3	open, claim, & enable
<b>printRecVoidItem</b>	1.3	open, claim, & enable

*Specific (Continued)*

<i>Specific - Fiscal Document</i>	<i>Ver</i>	<i>May Use After</i>
<b>beginFiscalDocument</b>	1.3	open, claim, & enable
<b>endFiscalDocument</b>	1.3	open, claim, & enable
<b>printFiscalDocumentLine</b>	1.3	open, claim, & enable

*Specific - Item Lists*

<b>beginItemList</b> (1)	1.3	open, claim, & enable
<b>endItemList</b> (1)	1.3	open, claim, & enable
<b>verifyItem</b> (1)	1.3	open, claim, & enable

*Specific - Fiscal Reports*

<b>printPeriodicTotalsReport</b>	1.3	open, claim, & enable
<b>printPowerLossReport</b>	1.3	open, claim, & enable
<b>printReport</b>	1.3	open, claim, & enable
<b>printXReport</b>	1.3	open, claim, & enable
<b>printZReport</b>	1.3	open, claim, & enable

*Specific - Slip Insertion*

<b>beginInsertion</b> (2)	1.3	open, claim, & enable
<b>beginRemoval</b> (2)	1.3	open, claim, & enable
<b>endInsertion</b> (2)	1.3	open, claim, & enable
<b>endRemoval</b> (2)	1.3	open, claim, & enable

*Specific - Non-Fiscal*

<b>beginFixedOutput</b> (1)	1.3	open, claim, & enable
<b>beginNonFiscal</b>	1.3	open, claim, & enable
<b>beginTraining</b>	1.3	open, claim, & enable
<b>endFixedOutput</b> (1)	1.3	open, claim, & enable
<b>endNonFiscal</b>	1.3	open, claim, & enable
<b>endTraining</b>	1.3	open, claim, & enable
<b>printFixedOutput</b> (1)	1.3	open, claim, & enable
<b>printNormal</b>	1.3	open, claim, & enable

*Specific (Continued)*

<i>Specific - Data Requests</i>	<i>Ver</i>	<i>May Use After</i>
<b>getData</b>	1.3	open, claim, & enable
<b>getDate</b>	1.3	open, claim, & enable
<b>getTotalizer</b>	1.3	open, claim, & enable
<b>getVatEntry</b> (1)	1.3	open, claim, & enable

*Specific - Error Corrections*

<b>clearError</b>	1.3	open, claim, & enable
<b>resetPrinter</b>	1.3	open, claim, & enable

**Events**

<i>Name</i>	<i>Ver</i>	<i>May Occur After</i>
<b>DataEvent</b>	1.3	<i>Not Supported</i>
<b>DirectIOEvent</b>	1.3	open & claim
<b>ErrorEvent</b>	1.3	open, claim, & enable
<b>OutputCompleteEvent</b>	1.3	open, claim, & enable
<b>StatusUpdateEvent</b>	1.3	open, claim, & enable

**Notes:**

- 1. All methods and properties marked with (1) are specific to at least one particular country and are not required by the fiscal legislation of all countries.*
- 2. Properties and methods marked with (2) are adapted from the POS Printer device.*

## General Information

The Fiscal Printer Control's class name is "jpos.FiscalPrinter".  
The device constants are contained in the class "jpos.FiscalPrinterConst".  
See "Package Structure" on page 40.

*This device was added in JavaPOS Release 1.3.*

The Fiscal Printer Control does not attempt to encapsulate a generic graphics printer. Rather, for performance and ease of use considerations, the interfaces are defined to directly control the normal printer functions.

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

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

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

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

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

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

## General Requirements

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

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

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



## Printer Modes

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

There are three possible printer modes:

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

## Model

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

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

```
printFiscalDocumentLine  
printFixedOutput  
printNormal  
printRecItem  
printRecItemAdjustment  
printRecMessage  
printRecNotPaid  
printRecRefund  
printRecSubtotal  
printRecSubtotalAdjustment  
printRecTotal  
printRecVoid  
printRecVoidItem
```

When **AsyncMode** is false, then these methods print synchronously.

When **AsyncMode** is true, then these methods operate as follows:

- The Device buffers the request, sets the **OutputID** property to an identifier for this request, and returns as soon as possible. When the device completes the request successfully, the **OutputCompleteEvent** is enqueued. A parameter of this event contains the **OutputID** of the completed request.

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

- If an error occurs while performing an asynchronous request, an **ErrorEvent** is enqueued. The **ErrorStation** property is set to the station or stations that were printing when the error occurred. The **ErrorLevel**, **ErrorString** and **ErrorState** and **ErrorOutID** properties are also set.

The event handler may call synchronous print methods (but not asynchronous methods), then can either retry the outstanding output or clear it.

- Asynchronous output is performed on a first-in first-out basis.
- All output buffered may be deleted by calling the **clearOutput** method. **OutputCompleteEvents** will not be delivered for cleared output. This method also stops any output that may be in progress (when possible).
- The property **FlagWhenIdle** may be set to cause a **StatusUpdateEvent** to be enqueued when all outstanding outputs have finished, whether successfully or because they were cleared.

## Error Model

The printer error reporting model is as follows:

- Most of the fiscal printer error conditions are reported by setting the exception's (or `ErrorEvent`'s) *ErrorCode* to `JPOS_E_EXTENDED` and then setting *ErrorCodeExtended* to one of the following:

### **JPOS\_EFPTR\_COVER\_OPEN**

The printer cover is open.

### **JPOS\_EFPTR\_JRN\_EMPTY**

The journal station has run out of paper.

### **JPOS\_EFPTR\_REC\_EMPTY**

The receipt station has run out of paper.

### **JPOS\_EFPTR\_SLP\_EMPTY**

The slip station has run out of paper.

### **JPOS\_EFPTR\_MISSING\_DEVICES**

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

**JPOS\_EFPTR\_WRONG\_STATE**

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

**JPOS\_EFPTR\_TECHNICAL\_ASSISTANCE**

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

**JPOS\_EFPTR\_CLOCK\_ERROR**

The printer's internal clock has failed.

**JPOS\_EFPTR\_FISCAL\_MEMORY\_FULL**

The printer's fiscal memory has been exhausted.

**JPOS\_EFPTR\_FISCAL\_MEMORY\_DISCONNECTED**

The printer's fiscal memory has been disconnected.

**JPOS\_EFPTR\_FISCAL\_TOTALS\_ERROR**

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

**JPOS\_EFPTR\_BAD\_ITEM\_QUANTITY**

The quantity parameter is invalid.

**JPOS\_EFPTR\_BAD\_ITEM\_AMOUNT**

The amount parameter is invalid.

**JPOS\_EFPTR\_BAD\_ITEM\_DESCRIPTION**

The description parameter is either too long, contains illegal characters or contains a reserved word.

**JPOS\_EFPTR\_RECEIPT\_TOTAL\_OVERFLOW**

The receipt total has overflowed.

**JPOS\_EFPTR\_BAD\_VAT**

The vat parameter is invalid.

**JPOS\_EFPTR\_BAD\_PRICE**

The price parameter is invalid.

**JPOS\_EFPTR\_BAD\_DATE**

The date parameter is invalid.

**JPOS\_EFPTR\_NEGATIVE\_TOTAL**

The printer's computed total or subtotal is less than zero.

**JPOS\_EFPTR\_WORD\_NOT\_ALLOWED**

The description contains the reserved word.

- Other printer errors are reported by setting the exception's (or ErrorEvent's) *ErrorCode* to `JPOS_E_FAILURE` or another error status. These failures are typically due to a printer fault or jam, or to a more serious error.

## Device Sharing

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

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

## Printer States

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

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

The fiscal printer has the following states:

- **Monitor:**  
This is a neutral state. From this state, it is possible to move to most of the other printer states. After a successful call to the **claim** method and successful setting of the **DeviceEnabled** property to true the printer should be in this state unless there is a printer error.
- **Fiscal Receipt:**  
The printer is processing a fiscal receipt. All **printRec...** methods are available for use while in this state. This state is entered from the **Monitor** state using the **beginFiscalReceipt** method.
- **Fiscal Receipt Total:**  
The printer has already accepted at least one payment method, but the receipt’s total amount has not yet been tendered. This state is entered from the **Fiscal Receipt** state by use of the **printRecTotal** method. The printer remains in this state while the total remains unpaid. This state can be left by using the **printRecTotal**, **printRecNotPaid** or **printRecVoid** methods.
- **Fiscal Receipt Ending:**  
The printer has completed the receipt up to the **Total** line. In this state, it may be possible to print general messages using the **printRecMessage** method if it is supported by the printer. This state is entered from the **Fiscal Receipt** state via the **printRecVoid** method or from the **Fiscal Receipt Total** state using either the **printRecTotal**, **printRecNotPaid** or **printRecVoid** methods. This state is exited using the **endFiscalReceipt** method at which time the printer returns to the **Monitor** state.
- **Fiscal Document:**  
The printer is processing a fiscal document. The printer will accept the **printFiscalDocumentLine** method while in this state. This state is entered from the **Monitor** state using the **beginFiscalDocument** method. This state is exited using the **endFiscalDocument** method at which time the printer returns to the **Monitor** state.

- **Monitor** and **TrainingModeActive** are true:  
The printer is being used for training purposes. All fiscal receipt and document commands are available. This state is entered from the **Monitor** state using the **beginTraining** method. This state is exited using the **endTraining** method at which time the printer returns to the **Monitor** state.
- **Fiscal Receipt** and **TrainingModeActive** are true:  
The printer is being used for training purposes and a receipt is currently opened. To each line of the receipt, special text will be added in order to differentiate it from a fiscal receipt.
- **Fiscal Total** and **TrainingModeActive** are true:  
The printer is in training mode and receipt total is being handled.
- **Fiscal ReceiptEnding** and **TrainingModeActive** are true:  
The printer is being used for training is in the receipt ending phase.
- **NonFiscal**:  
The printer is printing non-fiscal output on either the receipt (echoed on the journal) or the slip. In this state the printer will accept the **printNormal** method. The printer prints a message that indicates that this is non-fiscal output with all application text. This state is entered from the **Monitor** state using the **beginNonFiscal** method. This state is exited using the **endNonFiscal** method at which time the printer returns to the **Monitor** state.
- **Fixed**:  
The printer is being used to print fixed, non-fiscal output to one of the printer's stations. In this state the printer will accept the **printFixedOutput** method. This state is entered from the **Monitor** state using the **beginFixedOutput** method. This state is exited using the **endFixedOutput** method at which time the printer returns to the **Monitor** state.
- **ItemList**:  
The printer is currently printing a line item report. In this state the printer will accept the **verifyItem** method. This state is entered from the **Monitor** state using the **beginItemList** method. This state is exited using the **endItemList** method at which time the printer returns to the **Monitor** state.
- **Report**:  
The printer is currently printing one of the supported types of reports. This state is entered from the **Monitor** state using one of the **printReport**, **printPeriodicTotalsReport**, **printPowerLossReport**, **printXReport** or **printZReport** methods. When the report print completes, the printer automatically returns to **Monitor** state.
- **FiscalSystemBlocked**:  
The printer is no longer operational due to one of the following reasons:
  - The printer has been disconnected or has lost power.
  - The printer's fiscal memory has been exhausted.
  - The printer's internal data has become inconsistent.In this state the printer will only accept methods to print reports and retrieve data. The printer cannot exit this state without the assistance of an authorized technician.

When the application sets the property **DeviceEnabled** to true it also monitors its

current state. In a standard situation, the **PrinterState** property is set to **FPTR\_PS\_MONITOR** after a successfully setting **DeviceEnabled** to true. This indicates that there was no interrupted operation remaining in the printer. If the printer is not in the **FPTR\_PS\_MONITOR** state, the state reflects the printer's interrupted operation and the **PowerState** property is set to **JPOS\_PS\_OFF**. In this situation, it is necessary to force the printer to a normal state by calling the **resetPrinter** method. This means that a power failure occurred or the last application that accessed the device left it in a not clear state. Notice that even in this case the method returns successfully after setting **DeviceEnabled** to true. It is required that the application checks the **PowerState** property and checks for a received **StatusUpdateEvent** with the value **JPOS\_SUE\_POWER\_OFF** in the *Status* property after successfully setting the **DeviceEnabled** property.

## Document Printing

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

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

## Ordering of Fiscal Receipt Print Requests

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

Before selling the first line item, it is possible to exit from the fiscal receipt state

by calling the **endFiscalReceipt** method. If header lines have already been printed, this method will cause also receipt voiding.

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

**printRecItem**  
**printRecItemAdjustment**  
**printRecNotPaid**  
**printRecRefund**  
**printRecSubtotal**  
**printRecSubtotalAdjustment**  
**printRecTotal**  
**printRecVoid**  
**printRecVoidItem**

The **printRecItem**, **printRecItemAdjustment**, **printRecRefund**, **printRecSubtotal**, **printRecSubtotalAdjustment** and **printRecVoidItem** will leave the printer in the Fiscal Receipt state. The **printRecNotPaid** (only available if **CapReceiptNotPaid** is true) and **printRecTotal** methods will change the printer's state to either Fiscal Receipt Total or Fiscal Receipt Ending, depending upon whether the entire receipt total has been met. The **printRecVoid** method will change the printer's state to Fiscal Receipt Ending.

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

**printRecNotPaid**  
**printRecTotal**  
**printRecVoid**

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

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

**printRecMessage**  
**endFiscalReceipt**

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

At no time can the printer's total for the receipt be negative. If this occurs the fiscal printer will generate an **ErrorEvent**.

## Receipt Layouts

The following is an example of a typical receipt layout:

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



Example of a fiscal receipt:

	definition of the line	method
name of the store address ZIP code and place fiscal identification of the store	<b>header line</b>	<b>beginFiscalReceipt</b> data stored with <b>setHeaderLine</b> and <b>setStoreFiscalID</b>
	<b>tax number line</b>	
Milk 1.000 A	<b>transaction line</b>	<b>printRecItem</b>
Beer 4.000 C	<b>transaction line</b>	<b>printRecItem</b>
Discount Beer 500- C	<b>transaction line</b>	<b>printRecItemAdjustment</b>
Bread 3.500 A	<b>transaction line</b>	<b>printRecItem</b>
Storno Bread 3.500- A	<b>transaction line</b>	<b>printRecItemVoid</b>
Apples 2.000 A	<b>transaction line</b>	<b>printRecItem</b>
VAT category A 3.000	<b>VAT summary</b>	<b>printRecTotal</b> (..., 5000,"Check")
VAT 22.00% 660		
VAT category C 3.500		
VAT 12.00% 420		
sum of VAT 1.080		
<b>TOTALE</b> 6.500	<b>total line</b>	
Check 5000	<b>payment line</b>	<b>printRecTotal</b> (..., 5000,"Cash")
Cash 5000	<b>payment line</b>	
Return - 3500	<b>Change line</b>	
Advertising messages a.s.o. possibly ongoing advertising message	<b>message line</b> <b>message line</b>	<b>printRecMessage</b> <b>printRecMessage</b>
Good Bye	<b>trailer line</b>	<b>endFiscalReceipt</b> data stored with <b>setTrailerLine</b> and at initialisation time of the fiscal printer
24/05/96 14-25 Nr. 2	<b>logo line</b>	
<b>MF</b> B5 012345678	<b>logo line</b>	

### VAT Tables

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

If the printer supports VAT tables, **CapHasVatTable** is true. If the printer allows the VAT table entries to be set or modified **CapSetVatTable** is true. The maximum number of different vat rate entries in the VAT table is given by the **NumVatRates** property. VAT tables are set through a two step process. First the application uses the **setVatValue** method to set each table entry to be sent to the printer.

Next, the **setVatTable** method is called to send the entire VAT table to the printer at one time.

## Receipt Duplication

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

## Currency amounts, percentage amounts, VAT rates, and quantity amounts

- Currency amounts (and also prices) are passed as values with the data type long. This is a 64 bit signed integer value that implicitly assumes four digits as the fractional part. For example, an actual value of 12345 represents 1.2345. So, the range supported is  
from  
-922,337,203,685,477.5808  
to  
+922,337,203,685,477.5807

The fractional part used in the calculation unit of a Fiscal Printer may differ from the long data type. The number of digits in the fractional part is stored in the **AmountDecimalPlaces** property and determined by the Fiscal Printer. The application has to take care that calculations in the application use the same fractional part for amounts.

- If **CapHasVatTable** is true, VAT rates are passed using the indexes that were sent to the **setVatValue** method.
- If **CapHasVatTable** is false, VAT rates are passed as amounts with the data type int. The number of digits in the fractional part is implicitly assumed to be four.
- Percentage amounts are used in methods which allow also surcharge and/or discount amounts. If the amounts are specified to be a percentage value the value is also passed in a parameter of type long.
- The percentage value has (as given by the long data type) four digits in the fractional part. It is the percentage (0.0001% to 99.9999%) multiplied by 10000.
- Quantity amounts are passed as values with the data type int. The number of digits in the fractional part is stored in the **QuantityDecimalPlaces** property and determined by the Fiscal Printer.

## Properties

### AmountDecimalPlaces Property R

Type	<b>int</b>
Remarks	Holds the number of decimal digits that the fiscal device uses for calculations.  This property is initialized when the device is enabled.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### AsyncMode Property R/W

Type	<b>boolean</b>
Remarks	If true, then some print methods like <b>printRecItemAdjustment</b> , <b>printRecItem</b> , <b>printNormal</b> , etc. will be performed asynchronously.  This property is initialized to false by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	“Model” on page 207 for the output model description.

### CapAdditionalLines Property R

Type	<b>boolean</b>
Remarks	If true, then the printer supports the printing of application defined lines on a fiscal receipt between the total line and the end of the fiscal receipt.  If true, then after all totals lines are printed it is possible to print application-defined strings, such as the ones used for fidelity cards. In this case, after the total lines are printed, the <b>PrinterState</b> property is set to <b>ReceiptEnding</b> and <b>printRecMessage</b> can be called.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

## CapAmountAdjustment Property R

Type	<b>boolean</b>
Remarks	If true, then the printer handles fixed amount discounts or fixed amount surcharges on items.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

## CapAmountNotPaid Property R

Type	<b>boolean</b>
Remarks	If true, then the printer allows the recording of not paid amounts.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

## CapCheckTotal Property R

Type	<b>boolean</b>
Remarks	If true, then automatic comparison of the printer’s total and the application’s total can be enabled and disabled. If false, then the automatic comparison cannot be enabled and is always considered disabled.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

## CapCoverSensor Property R

Type	<b>boolean</b>
Remarks	If true, then the printer has a “cover open” sensor.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapDoubleWidth Property R

Type	<b>boolean</b>
Remarks	If true, then the printer can print double width characters.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapDuplicateReceipt Property R

Type	<b>boolean</b>
Remarks	If true, then the printer allows printing more than one copy of the same fiscal receipt.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapFixedOutput Property R

Type	<b>boolean</b>
Remarks	If true, then the printer supports fixed format text printing through the <b>beginFixedOutput</b> , <b>printFixedOutput</b> and <b>endFixedOutput</b> methods.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapHasVatTable Property R

Type	<b>boolean</b>
Remarks	If true, then the printer has a tax table.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapIndependentHeader Property R

Type	<b>boolean</b>
Remarks	If true, then the printer supports printing the fiscal receipt header lines before the first fiscal receipt command is processed.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapItemList Property R

Type	<b>boolean</b>
Remarks	If true, then the printer can print a report of items of a specified VAT class.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapJrnEmptySensor Property R

Type	<b>boolean</b>
Remarks	If true, then the journal has an out-of-paper sensor.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapJrnNearEndSensor Property R

Type	<b>boolean</b>
Remarks	If true, then the journal has a low paper sensor.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapJrnPresent Property R

Type	<b>boolean</b>
Remarks	<p>If true, then the journal print station is present.</p> <p>Unlike POS printers, on fiscal printers the application is not able to directly access the journal. The fiscal printer itself prints on the journal if present.</p> <p>This property is initialized by the <b>open</b> method.</p>
Errors	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>

### CapNonFiscalMode Property R

Type	<b>boolean</b>
Remarks	<p>If true, then the printer allows printing in non-fiscal mode.</p> <p>This property is initialized by the <b>open</b> method.</p>
Errors	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>

### CapOrderAdjustmentFirst Property R

Type	<b>boolean</b>
Remarks	<p>If false, the application has to call <b>printRecItem</b> first and then call <b>printRecItemAdjustment</b> to give a discount or a surcharge for a single article.</p> <p>If true, the application has to call <b>printRecItemAdjustment</b> first and then call <b>printRecItem</b>.</p> <p>This property is initialized by the <b>open</b> method.</p>
Errors	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>

### CapPercentAdjustment Property R

Type	<b>boolean</b>
Remarks	<p>If true, then the printer handles percentage discounts or percentage surcharges on items.</p> <p>This property is initialized by the <b>open</b> method.</p>
Errors	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>

### CapPositiveAdjustment Property R

Type	<b>boolean</b>
Remarks	If true, then it is possible to apply surcharges via the <b>printRecItemAdjustment</b> method.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapPowerLossReport Property R

Type	<b>boolean</b>
Remarks	If true, then the printer can print a power loss report using the <b>printPowerLossReport</b> method.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapPredefinedPaymentLines Property R

Type	<b>boolean</b>
Remarks	If true, the printer can store and print predefined payment descriptions.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapReceiptNotPaid Property R

Type	<b>boolean</b>
Remarks	If true, then the printer supports using the <b>printRecNotPaid</b> method to specify a part of the receipt total that is not paid.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.



### CapRecEmptySensor Property R

Type	<b>boolean</b>
Remarks	If true, then the receipt has an out-of-paper sensor.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapRecNearEndSensor Property R

Type	<b>boolean</b>
Remarks	If true, then the receipt has a low paper sensor.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapRecPresent Property R

Type	<b>boolean</b>
Remarks	If true, then the receipt print station is present.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapRemainingFiscalMemory Property R

Type	<b>boolean</b>
Remarks	If true, then the printer supports using the <b>RemainingFiscalMemory</b> property to show the amount of Fiscal Memory remaining. If false, the printer does not support reporting the Fiscal Memory status of the printer.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapReservedWord Property R

Type	<b>boolean</b>
Remarks	<p>If true, then the printer prints a reserved word (for example, “TOTALE”) before printing the total amount.</p> <p>If true, the reserved word is stored in the <b>ReservedWord</b> property. This reserved word may not be printed using any fiscal print method.</p> <p>This property is initialized by the <b>open</b> method.</p>
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapSetHeader Property R

Type	<b>boolean</b>
Remarks	<p>If true, then it is possible to use the <b>setHeaderLine</b> method to initialize the contents of a particular line of the receipt header.</p> <p>This property is initialized by the <b>open</b> method.</p>
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapSetPOSID Property R

Type	<b>boolean</b>
Remarks	<p>If true, then it is possible to use the <b>setPOSID</b> method to initialize the values of POSID and CashierID. These values are printed on each fiscal receipt.</p> <p>This property is initialized by the <b>open</b> method.</p>
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapSetStoreFiscalID Property R

Type	<b>boolean</b>
Remarks	<p>If true, then it is possible to use the <b>setStoreFiscalID</b> method to set up the Fiscal ID number which will be printed on each fiscal receipt.</p> <p>This property is initialized by the <b>open</b> method.</p>
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapSetTrailer Property R

Type	<b>boolean</b>
Remarks	If true, then it is possible to use the <b>setTrailerLine</b> method to initialize the contents of a particular line of the receipt trailer.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapSetVatTable Property R

Type	<b>boolean</b>
Remarks	If true, then it is possible to use the <b>setVatValue</b> and <b>setVatTable</b> methods to modify the contents of the printer’s VAT table. Some printers may not allow existing VAT table entries to be modified. Only new entries may be set on these printers.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapSlpEmptySensor Property R

Type	<b>boolean</b>
Remarks	If true, then the slip has a “slip in” sensor.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapSlpFiscalDocument Property R

Type	<b>boolean</b>
Remarks	If true, then the printer allows fiscal printing to the slip station.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapSlpFullSlip Property R

Type	<b>boolean</b>
Remarks	<p>If true, then the printer supports printing full length forms on the slip station.</p> <p>It is possible to choose between full slip and validation documents by setting the <b>SlipSelection</b> property.</p> <p>This property is initialized by the <b>open</b> method.</p>
Errors	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>

### CapSlpNearEndSensor Property R

Type	<b>boolean</b>
Remarks	<p>If true, then the slip has a “slip near end” sensor.</p> <p>This property is initialized by the <b>open</b> method.</p>
Errors	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>

### CapSlpPresent Property R

Type	<b>boolean</b>
Remarks	<p>If true, then the printer has a slip station.</p> <p>This property is initialized by the <b>open</b> method.</p>
Errors	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>

### CapSlpValidation Property R

Type	<b>boolean</b>
Remarks	<p>If true, then the printer supports printing validation information on the slip station.</p> <p>It is possible to choose between full slip and validation documents by setting the <b>SlipSelection</b> property. In some countries, when printing non fiscal validations using the slip station a limited number of lines could be printed.</p> <p>This property is initialized by the <b>open</b> method.</p>
Errors	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>

### CapSubAmountAdjustment Property R

Type	<b>boolean</b>
Remarks	If true, then the printer handles fixed amount discounts on the subtotal.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapSubPercentAdjustment Property R

Type	<b>boolean</b>
Remarks	If true, then the printer handles percentage discounts on the subtotal.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapSubtotal Property R

Type	<b>boolean</b>
Remarks	If true, then it is possible to use the <b>printRecSubtotal</b> method to print the current subtotal.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapTrainingMode Property R

Type	<b>boolean</b>
Remarks	If true, then the printer supports a training mode.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

## CapValidateJournal Property R

<b>Type</b>	<b>boolean</b>
<b>Remarks</b>	If true, then it is possible to use the <b>printNormal</b> method to print a validation string on the journal station.  This property is initialized by the <b>open</b> method.
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

## CapXReport Property R

<b>Type</b>	<b>boolean</b>
<b>Remarks</b>	If true, then it is possible to use the <b>printXReport</b> method to print an X report.  This property is initialized by the <b>open</b> method.
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

## CheckTotal Property R/W

<b>Type</b>	<b>boolean</b>
<b>Remarks</b>	If true, automatic comparison between the fiscal printer’s total and the application’s total is enabled. If false, automatic comparison is disabled. This property is only valid if <b>CapCheckTotal</b> is true.  This property is initialized to true by the <b>open</b> method.
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
JPOS_E_ILLEGAL	Setting this property is not valid for this Service (see <b>CapCheckTotal</b> ).

## CountryCode Property R

<b>Type</b>	<b>int</b>																
<b>Remarks</b>	<p>Holds a value identifying which countries are supported by this Device Service. It can contain any of the following values logically ORed together:</p> <table border="1"> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>FPTR_CC_BRAZIL</td> <td>The printer supports Brazil's fiscal rules.</td> </tr> <tr> <td>FPTR_CC_GREECE</td> <td>The printer supports Greece's fiscal rules.</td> </tr> <tr> <td>FPTR_CC_HUNGARY</td> <td>The printer supports Hungary's fiscal rules.</td> </tr> <tr> <td>FPTR_CC_ITALY</td> <td>The printer supports Italy's fiscal rules.</td> </tr> <tr> <td>FPTR_CC_POLAND</td> <td>The printer supports Poland's fiscal rules.</td> </tr> <tr> <td>FPTR_CC_RUSSIA</td> <td>The printer supports Russia's fiscal rules.</td> </tr> <tr> <td>FPTR_CC_TURKEY</td> <td>The printer supports Turkey's fiscal rules.</td> </tr> </tbody> </table> <p>This property is initialized by the <b>open</b> method.</p>	Value	Meaning	FPTR_CC_BRAZIL	The printer supports Brazil's fiscal rules.	FPTR_CC_GREECE	The printer supports Greece's fiscal rules.	FPTR_CC_HUNGARY	The printer supports Hungary's fiscal rules.	FPTR_CC_ITALY	The printer supports Italy's fiscal rules.	FPTR_CC_POLAND	The printer supports Poland's fiscal rules.	FPTR_CC_RUSSIA	The printer supports Russia's fiscal rules.	FPTR_CC_TURKEY	The printer supports Turkey's fiscal rules.
Value	Meaning																
FPTR_CC_BRAZIL	The printer supports Brazil's fiscal rules.																
FPTR_CC_GREECE	The printer supports Greece's fiscal rules.																
FPTR_CC_HUNGARY	The printer supports Hungary's fiscal rules.																
FPTR_CC_ITALY	The printer supports Italy's fiscal rules.																
FPTR_CC_POLAND	The printer supports Poland's fiscal rules.																
FPTR_CC_RUSSIA	The printer supports Russia's fiscal rules.																
FPTR_CC_TURKEY	The printer supports Turkey's fiscal rules.																
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see "Exceptions" on page 15.																

## CoverOpen Property R

<b>Type</b>	<b>boolean</b>
<b>Remarks</b>	<p>If true, then the printer's cover is open.</p> <p>If <b>CapCoverSensor</b> is false, then the printer does not have a cover open sensor and this property is always false.</p> <p>This property is initialized and kept current while the device is enabled.</p>
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see "Exceptions" on page 15.

## DayOpened Property R

<b>Type</b>	<b>boolean</b>
<b>Remarks</b>	<p>If true, then the fiscal day has been started on the printer.</p> <p>The Fiscal Day of the printer can be either opened or not opened. The <b>DayOpened</b> property reflects whether or not the printer considers its Fiscal Day to be opened or not.</p> <p>Some methods may only be called while the Fiscal Day is not yet opened (<b>DayOpened</b> is false). Methods that can be called after the Fiscal Day is opened change from country to country. Usually all the configuration methods are to be</p>

called only before the Fiscal Day is opened.

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

**setDate**  
**setHeaderLine**  
**setPOSID**  
**setStoreFiscalID**  
**setTrailerLine**  
**setVatTable**  
**setVatValue**

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

**Errors** A `JposException` may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### DescriptionLength Property R

**Type** **int**

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

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

**Errors** A `JposException` may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### DuplicateReceipt Property R/W

**Type** **boolean**

**Remarks** If true, all the printing commands inside a fiscal receipt will be buffered and they can be printed again via the **printDuplicateReceipt** method.

This property is only valid if **CapDuplicateReceipt** is true.

This property is initialized to false by the **open** method.

When this property is set an exception value of `JPOS_E_ILLEGAL` is not valid for this service. See **CapDuplicateReceipt**.

**Errors** A `JposException` may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.



## ErrorLevel Property R

<b>Type</b>	<b>int</b>										
<b>Remarks</b>	<p>Holds the severity of the error condition.</p> <p>This property has one of the following values:</p> <table border="1"> <thead> <tr> <th><b>Value</b></th> <th><b>Meaning</b></th> </tr> </thead> <tbody> <tr> <td>FPTR_EL_NONE</td> <td>No error condition is present.</td> </tr> <tr> <td>FPTR_EL_RECOVERABLE</td> <td>A recoverable error has occurred. (Example: Out of paper.)</td> </tr> <tr> <td>FPTR_EL_FATAL</td> <td>A non-recoverable error has occurred. (Example: Internal printer failure.)</td> </tr> <tr> <td>FPTR_EL_BLOCKED</td> <td>A severe hardware failure which can be resolved only by authorized technicians. (Example: Fiscal memory failure.). This error can not be recovered.</td> </tr> </tbody> </table> <p>This property is set just before delivering an <b>ErrorEvent</b>. When the error is cleared, then the property is changed to FPTR_EL_NONE.</p>	<b>Value</b>	<b>Meaning</b>	FPTR_EL_NONE	No error condition is present.	FPTR_EL_RECOVERABLE	A recoverable error has occurred. (Example: Out of paper.)	FPTR_EL_FATAL	A non-recoverable error has occurred. (Example: Internal printer failure.)	FPTR_EL_BLOCKED	A severe hardware failure which can be resolved only by authorized technicians. (Example: Fiscal memory failure.). This error can not be recovered.
<b>Value</b>	<b>Meaning</b>										
FPTR_EL_NONE	No error condition is present.										
FPTR_EL_RECOVERABLE	A recoverable error has occurred. (Example: Out of paper.)										
FPTR_EL_FATAL	A non-recoverable error has occurred. (Example: Internal printer failure.)										
FPTR_EL_BLOCKED	A severe hardware failure which can be resolved only by authorized technicians. (Example: Fiscal memory failure.). This error can not be recovered.										
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.										

## ErrorOutID Property R

<b>Type</b>	<b>int</b>
<b>Remarks</b>	<p>Holds the identifier of the output in the queue which caused an <b>ErrorEvent</b>, when using asynchronous printing.</p> <p>This property is set just before an <b>ErrorEvent</b> is delivered.</p>
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

## ErrorState Property R

<b>Type</b>	<b>int</b>
<b>Remarks</b>	<p>Holds the current state of the printer when an <b>ErrorEvent</b> is delivered for an asynchronous output.</p> <p>This property is set just before an <b>ErrorEvent</b> is delivered.</p>
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
<b>See Also</b>	<b>PrinterState</b> Property.

## ErrorStation Property R

Type	<b>int</b>
Remarks	<p>Holds the station or stations that were printing when an error was detected.</p> <p>This property will be set to one of the following values: <code>FPTR_S_JOURNAL</code>, <code>FPTR_S_RECEIPT</code>, <code>FPTR_S_SLIP</code>, <code>FPTR_S_JOURNAL_RECEIPT</code>, <code>FPTR_S_JOURNAL_SLIP</code>, <code>FPTR_S_RECEIPT_SLIP</code>.</p> <p>This property is only valid if the <code>ErrorLevel</code> is not equal to <code>PTR_EL_NONE</code>. It is set just before delivering an <b>ErrorEvent</b>.</p>
Errors	<p>A <code>JposException</code> may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>

## ErrorString Property R

Type	<b>String</b>
Remarks	<p>Holds a vendor-supplied description of the current error.</p> <p>This property is set just before delivering an <b>ErrorEvent</b>. If no description is available, the property is set to an empty string. When the error is cleared, then the property is changed to an empty string.</p>
Errors	<p>A <code>JposException</code> may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>

## FlagWhenIdle Property R/W

Type	<b>boolean</b>
Remarks	<p>If true, a <b>StatusUpdateEvent</b> will be enqueued when the device is in the idle state.</p> <p>This property is automatically reset to false when the status event is delivered.</p> <p>The main use of idle status event that is controlled by this property is to give the application control when all outstanding asynchronous outputs have been processed. The event will be enqueued if the outputs were completed successfully or if they were cleared by the <b>clearOutput</b> method or by an <b>ErrorEvent</b> handler.</p> <p>If the <b>State</b> is already set to <code>JPOS_S_IDLE</code> when this property is set to true, then a <b>StatusUpdateEvent</b> is enqueued immediately. The application can therefore depend upon the event, with no race condition between the starting of its last asynchronous output and the setting of this flag.</p> <p>This property is initialized to false by the <b>open</b> method.</p>
Errors	<p>A <code>JposException</code> may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>

## JrnEmpty Property R

Type	<b>boolean</b>
Remarks	If true, the journal is out of paper. If false, journal paper is present.  If <b>CapJrnEmptySensor</b> is false, then the value of this property is always false.  This property is initialized and kept current while the device is enabled.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>JrnNearEnd</b> Property

## JrnNearEnd Property R

Type	<b>boolean</b>
Remarks	If true, the journal paper is low. If false, journal paper is not low.  If <b>CapJrnNearEndSensor</b> is false, then the value of this property is always false.  This property is initialized and kept current while the device is enabled.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>JrnEmpty</b> Property

## MessageLength Property R

Type	<b>int</b>
Remarks	Holds the maximum number of characters that may be passed as a message line in the method <b>printRecMessage</b> . The value may change in different modes of the fiscal printer. For example in the mode “Fiscal Receipt” the number of characters may be bigger than in the mode “Fiscal Receipt Total.”  This property is initialized by the <b>open</b> method.
Errors	A <code>JposException</code> may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

## NumHeaderLines Property R

Type	<b>int</b>
Remarks	Holds the maximum number of header lines that can be printed for each fiscal receipt. Header lines usually contain information like store address, store name, store Fiscal ID. Each header line is set using the <b>setHeaderLine</b> method and remains set even after the printer is switched off. Header lines are automatically printed when a fiscal receipt is initiated using the <b>beginFiscalReceipt</b> method or when the first line item inside a receipt is sold.  This property is initialized by the <b>open</b> method.
Errors	A <code>JposException</code> may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

## NumTrailerLines Property R

Type	<b>int</b>
Remarks	Holds the maximum number of trailer lines that can be printed for each fiscal receipt. Trailer lines are usually promotional messages. Each trailer line is set using the <b>setTrailerLine</b> method and remains set even after the printer is switched off. Trailer lines are automatically printed either after the last <b>printRecTotal</b> or when a fiscal receipt is closed using the <b>endFiscalReceipt</b> method.  This property is initialized by the <b>open</b> method.
Errors	A <code>JposException</code> may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

## NumVatRates Property R

Type	<b>int</b>
Remarks	<p>Holds the maximum number of vat rates that can be entered into the printer's Vat table.</p> <p>This property is initialized by the <b>open</b> method.</p>
Errors	A JposException may be thrown when this property is accessed. For further information, see "Exceptions" on page 15.

## PredefinedPaymentLines Property R

Type	<b>String</b>
Remarks	<p>Holds the list of all possible words to be used as indexes of the predefined payment lines (for example, "a, b, c, d, z"). Those indexes are used in the <b>printRecTotal</b> method for the <i>description</i> parameter.</p> <p>If <b>CapPredefinedPaymentLines</b> is true, only predefined payment lines are allowed.</p> <p>This property is initialized by the <b>open</b> method.</p>
Errors	A JposException may be thrown when this property is accessed. For further information, see "Exceptions" on page 15.

## PrinterState Property R

Type	<b>int</b>
Remarks	<p>Holds the printer's current operational state. This property controls which methods are currently legal.</p>

Values are:

Value	Meaning
FPTR_PS_MONITOR	<p>If <b>TrainingModeActive</b> is false: The printer is currently not in a specific operational mode. In this state the printer will accept any of the <b>begin...</b> methods as well as the <b>set...</b> methods.</p> <p>If <b>TrainingModeActive</b> is true: The printer is currently being used for training purposes. In this state the printer will accept any of the <b>printRec...</b> methods or the <b>endTraining</b> method.</p>
FPTR_PS_FISCAL_RECEIPT	<p>If <b>TrainingModeActive</b> is false: The printer is currently processing a fiscal receipt. In this state the printer will accept any of the <b>printRec...</b> methods.</p>

If **TrainingModeActive** is true:

The printer is currently being used for training purposes and a fiscal receipt is currently opened.

#### FPTR\_PS\_FISCAL\_RECEIPT\_TOTAL

If **TrainingModeActive** is false:

The printer has already accepted at least one payment, but the total has not been completely paid. In this state the printer will accept either the **printRecTotal** or **printRecNotPaid** methods.

If **TrainingModeActive** is true:

The printer is currently being used for training purposes and the printer has already accepted at least one payment, but the total has not been completely paid.

#### FPTR\_PS\_FISCAL\_RECEIPT\_ENDING

If **TrainingModeActive** is false:

The printer has completed the receipt up to the total line. In this state the printer will accept either the **printRecMessage** or **endFiscalReceipt** methods.

If **TrainingModeActive** is true:

The printer is currently being used for training purposes and a fiscal receipt is going to be closed.

#### FPTR\_PS\_FISCAL\_DOCUMENT

The printer is currently processing a fiscal slip. In this state the printer will accept either the **printFiscalDocumentLine** or **endFiscalDocument** methods.

#### FPTR\_PS\_FIXED\_OUTPUT

The printer is currently processing fixed text output to one or more stations. In this state the printer will accept either the **printFixedOutput** or **endFixedOutput** methods.

**FPTR\_PS\_ITEM\_LIST** The printer is currently processing an item list report. In this state the printer will accept either the **verifyItem** or **endItemList** methods.

**FPTR\_PS\_NONFISCAL** The printer is currently processing non-fiscal output to one or more stations. In this state the printer will accept either the **printNormal** or **endNonFiscal** methods.

**FPTR\_PS\_LOCKED** The printer has encountered a non-recoverable hardware problem. An authorized printer technician must be contacted to exit this state.

**FPTR\_PS\_REPORT** The printer is currently processing a fiscal report. In this state the printer will not accept any methods until the report has completed.

There are a few methods that are accepted in any state except FPTR\_PS\_LOCKED. These are **beginInsertion**, **endInsertion**, **beginRemoval**, **endRemoval**, **getDate**, **getData**, **getTotalizer**, **getVatEntry**, **resetPrinter** and **clearOutput**.

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

**Errors** A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

**See Also** “Printer States” on page 210.

### QuantityDecimalPlaces Property R

**Type** **int**

**Remarks** Holds the number of decimal digits in the fractional part that should be assumed to be in any quantity parameter.

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

**Errors** A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### QuantityLength Property R

**Type** **int**

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

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

**Errors** A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### RecEmpty Property R

**Type** **boolean**

**Remarks** If true, the receipt is out of paper. If false, receipt paper is present.

If **CapRecEmptySensor** is false, then this property is always false.

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

**Errors** A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

**See Also** **RecNearEnd** Property

## RecNearEnd Property R

Type	<b>boolean</b>
Remarks	If true, the receipt paper is low. If false, receipt paper is not low.  If <b>CapRecNearEndSensor</b> is false, then this property is always false.  This property is initialized and kept current while the device is enabled.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>RecEmpty</b> Property

## RemainingFiscalMemory Property R

Type	<b>int</b>
Remarks	Holds the remaining counter of Fiscal Memory.  This property is initialized and kept current while the device is enabled and may be updated by <b>printZReport</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>CapRemainingFiscalMemory</b> Property



## ReservedWord Property R

Type	<b>String</b>
Remarks	<p>Holds the string that is automatically printed with the total when the <b>printRecTotal</b> method is called. This word may not occur in any string that is passed into any fiscal output methods.</p> <p>This property is only valid if <b>CapReservedWord</b> is true.</p> <p>This property is initialized by the <b>open</b> method.</p>
Errors	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>

## SlpEmpty Property R

Type	<b>boolean</b>
Remarks	<p>If true, a slip form is not present. If false, a slip form is present.</p> <p>If <b>CapSlpEmptySensor</b> is false, then this property is always false.</p> <p>This property is initialized and kept current while the device is enabled.</p> <p><b>Note:</b></p> <p><i>The “slip empty” sensor should be used primarily to determine whether a form has been inserted before printing. It can also be monitored to determine whether a form is still in place. This sensor is usually placed one or more print lines above the slip print head.</i></p> <p><i>However, the “slip near end” sensor (when present) should be used to determine when nearing the end of the slip. This sensor is usually placed one or more print lines below the slip print head.</i></p>
Errors	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>
See Also	<b>SlpNearEnd</b> Property

## SlpNearEnd Property R

<b>Type</b>	<b>boolean</b>
<b>Remarks</b>	<p>If true, the slip form is near its end. If false, the slip form is not near its end. The “near end” sensor is also sometimes called the “trailing edge” sensor, referring to the bottom edge of the slip.</p> <p>If <b>CapSlpNearEndSensor</b> is false, then this property is always false.</p> <p>This property is initialized and kept current while the device is enabled.</p> <p><b>Note:</b></p> <p><i>However, the “slip near end” sensor (when present) should be used to determine when nearing the end of the slip. This sensor is usually placed one or more print lines below the slip print head.</i></p>
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
<b>See Also</b>	<b>SlpEmpty</b> Property

## SlipSelection Property R/W

<b>Type</b>	<b>int</b>						
<b>Remarks</b>	<p>Selects the kind of document to be printed on the slip station.</p> <p>This property has one of the following values:</p> <table> <thead> <tr> <th><b>Value</b></th> <th><b>Meaning</b></th> </tr> </thead> <tbody> <tr> <td>FPTR_SS_FULL_LENGTH</td> <td>Print full length documents.</td> </tr> <tr> <td>FPTR_SS_VALIDATION</td> <td>Print validation documents.</td> </tr> </tbody> </table> <p>This property is initialized to FPTR_SS_FULL_LENGTH by the <b>claim</b> method.</p>	<b>Value</b>	<b>Meaning</b>	FPTR_SS_FULL_LENGTH	Print full length documents.	FPTR_SS_VALIDATION	Print validation documents.
<b>Value</b>	<b>Meaning</b>						
FPTR_SS_FULL_LENGTH	Print full length documents.						
FPTR_SS_VALIDATION	Print validation documents.						
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.						

## TrainingModeActive Property R

**Type**        **boolean**

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

This property has one of the following values:

<b>Value</b>	<b>Meaning</b>
true	The printer is currently in training mode. That means no data are written into the EPROM of the fiscal printer.
false	The printer is currently in normal mode. All printed receipts will also update the fiscal memory.

**Errors**        A JposException may be thrown when this property is accessed. For further information, see "Exceptions" on page 15.

## Methods

### beginFiscalDocument Method

**Syntax**      **void beginFiscalDocument (int *documentAmount*) throws JposException;**

Parameter	Description
-----------	-------------

<i>documentAmount</i>	Amount of document to be stored by the printer.
-----------------------	---

**Remarks**      Initiates fiscal printing to the slip station.

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

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

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

If this method is successful, the **PrinterState** property will be changed to FPTR\_PS\_FISCAL\_DOCUMENT.

**Errors**      A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
-------	---------

JPOS_E_ILLEGAL	The slip station does not exist (see the <b>CapSlpPresent</b> property) or the printer does not support fiscal output to the slip station (see the <b>CapSlpFiscalDocument</b> property).
----------------	---

JPOS\_E\_EXTENDED: *ErrorCodeExtended* =  
JPOS\_EFPTR\_WRONG\_STATE:  
The printer’s current state does not allow this state transition.

*ErrorCodeExtended* = JPOS\_EFPTR\_SLP\_EMPTY:  
There is no paper in the slip station.

*ErrorCodeExtended* =  
JPOS\_EFPTR\_BAD\_ITEM\_AMOUNT:  
The *documentAmount* parameter is invalid.

**See Also**      **endFiscalDocument** Method, **printFiscalDocumentLine** Method  
**AmountDecimalPlaces** Property

## beginFiscalReceipt Method

**Syntax**      **void beginFiscalReceipt (boolean *printHeader*) throws JposException;**

Parameter	Description
-----------	-------------

<i>printHeader</i>	Indicates if the header lines are to be printed at this time.
--------------------	---

**Remarks**      Initiates fiscal printing to the receipt station.

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

If this method is successful, the **PrinterState** property will be changed to FPTR\_PS\_FISCAL\_RECEIPT.

**Errors**      A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
-------	---------

JPOS_E_EXTENDED:	<i>ErrorCodeExtended</i> = JPOS_EFPTR_WRONG_STATE: The printer’s current state does not allow this state transition.
------------------	---

**See Also**      **endFiscalReceipt** Method, **printRec...** Methods, **CapIndependentHeader** Property

**beginFixedOutput Method**

**Syntax**      **void beginFixedOutput (int station, int documentType) throws JposException;**

Parameter	Description
<i>station</i>	The printer station to be used. May be either FPTR_S_RECEIPT or FPTR_S_SLIP.
<i>documentType</i>	Identifier of a document stored in the printer.

**Remarks**      Initiates non-fiscal fixed text printing on a printer station. This method is only supported if **CapFixedOutput** is true.

If the *station* parameter is FPTR\_S\_SLIP, the slip paper must be inserted into the slip station using **begin/endInsertion** before calling this method.

Each fixed output will be printed using the **printFixedOutput** method. If this method is successful, the **PrinterState** property will be changed to FPTR\_PS\_FIXED\_OUTPUT. The **endFixedOutput** method ends fixed output modality and resets **PrinterState**.

**Errors**      A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_ILLEGAL	One of the following errors occurred:  Station does not exist (see the <b>CapSlpPresent</b> property).  Printer does not support fixed output (see the <b>CapFixedOutput</b> property).  <i>Station</i> parameter is invalid.  <i>DocumentType</i> is invalid.
JPOS_E_EXTENDED:	<i>ErrorCodeExtended</i> = JPOS_EFPTR_WRONG_STATE: The printer’s current state does not allow this state transition.  JPOS_EFPTR_SLP_EMPTY: There is no paper in the slip station.

**See Also**      **endFixedOutput** Method, **printFixedOutput** Method

## beginInsertion Method

**Syntax**      **void beginInsertion (int *timeout*) throws JposException;**

Parameter	Description
<i>timeout</i>	The <i>timeout</i> parameter gives the number of milliseconds before failing the method.

If zero, the method tries to begin insertion mode, then returns the appropriate status immediately. If JPOS\_FOREVER (-1), the method tries to begin insertion mode, then waits as long as needed until either the form is inserted or an error occurs.

**Remarks**      Initiates slip processing.

When called, the slip station is made ready to receive a form by opening the form's handling "jaws" or activating a form insertion mode. This method is paired with the **endInsertion** method for controlling form insertion.

If the printer device cannot be placed into insertion mode, a JposException is thrown. Otherwise, the Device continues to monitor form insertion until either:

- The form is successfully inserted.
- The form is not inserted before *timeout* milliseconds have elapsed, or an error is reported by the printer device. In this case, a JposException is thrown with an *ErrorCode* of JPOS\_E\_TIMEOUT or another value. The printer device remains in form insertion mode. This allows an application to perform some user interaction and reissue the **beginInsertion** method without altering the form handling mechanism.

**Errors**      A JposException may be thrown when this method is invoked. For further information, see "Exceptions" on page 15.

Some possible values of the exception's *ErrorCode* property are:

Value	Meaning
JPOS_E_ILLEGAL	The slip station does not exist (see the <b>CapSlpPresent</b> property) or an invalid <i>timeout</i> parameter was specified.
JPOS_E_TIMEOUT	The specified time has elapsed without the form being properly inserted.

**See Also**      **endInsertion** Method, **beginRemoval** Method, **endRemoval** Method

**beginItemList Method**

**Syntax**      **void beginItemList (int vatID) throws JposException;**

Parameter	Description
-----------	-------------

<i>vatID</i>	Vat identifier for reporting.
--------------	-------------------------------

**Remarks**      Initiates a validation report of items belonging to a particular VAT class.

This method is only supported if **CapItemList** is true. If this method is successful, **PrinterState** will be changed to FPTR\_PS\_ITEM\_LIST. After this method, only **verifyItem** and **endItemList** methods may be called.

**Errors**      A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
-------	---------

JPOS_E_ILLEGAL	The printer does not support an item list report (see the <b>CapItemList</b> property) or the printer does not support VAT tables (see the <b>CapHasVatTable</b> property).
----------------	---

JPOS\_E\_EXTENDED: *ErrorCodeExtended* =  
 JPOS\_EFPTR\_WRONG\_STATE:  
 The printer’s current state does not allow this state transition.

JPOS\_EFPTR\_BAD\_VAT:  
 The *vatID* parameter is invalid.

**See Also**      **endItemList** Method, **verifyItem** Method



## beginNonFiscal Method

**Syntax**      **void beginNonFiscal () throws JposException;**

**Remarks**      Initiates non-fiscal operations on the printer.

This method is only supported if **CapNonFiscalMode** is true. Output in this mode is accomplished using the **printNormal** method. This method can be successfully called only if the current value of the **PrinterState** property is FPTR\_PS\_MONITOR. If this method is successful, the **PrinterState** property will be changed to FPTR\_PS\_NONFISCAL. In order to stop non fiscal modality **endNonFiscal** method should be called.

**Errors**      A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
JPOS_E_ILLEGAL	The printer does not support non-fiscal output (see the <b>CapNonFiscalMode</b> property).
JPOS_E_EXTENDED:	<i>ErrorCodeExtended</i> =
	JPOS_EFPTR_WRONG_STATE:
	The printer’s current state does not allow this state transition.

**See Also**      **endNonFiscal** Method, **printNormal** Method

## beginRemoval Method

**Syntax**      **void beginRemoval (int *timeout*) throws JposException;**

Parameter	Description
<i>timeout</i>	The <i>timeout</i> parameter gives the number of milliseconds before failing the method.

If zero, the method tries to begin removal mode, then returns the appropriate status immediately. If JPOS\_FOREVER (-1), the method tries to begin removal mode, then waits as long as needed until either the form is removed or an error occurs.

**Remarks**      Initiates form removal processing.

When called, the printer is made ready to remove a form by opening the form handling “jaws” or activating a form ejection mode. This method is paired with the **endRemoval** method for controlling form removal.

If the printer device cannot be placed into removal or ejection mode, a JposException is thrown. Otherwise, the Device continues to monitor form removal until either:

- The form is successfully removed.
- The form is not removed before *timeout* milliseconds have elapsed, or an error is reported by the printer device. In this case, a JposException is thrown with an *ErrorCode* of JPOS\_E\_TIMEOUT or another value. The printer device remains in form removal mode. This allows an application to perform some user interaction and reissue the **beginRemoval** method without altering the form handling mechanism.

**Errors**      A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_ILLEGAL	The printer does not have a slip station (see the <b>CapSlpPresent</b> property) or an invalid <i>timeout</i> parameter was specified.
JPOS_E_TIMEOUT	The specified time has elapsed without the form being properly removed.

**See Also**      **beginInsertion** Method, **endInsertion** Method, **endRemoval** Method

## beginTraining Method

**Syntax**      **void beginTraining () throws JposException;**

**Remarks**      Initiates training operations.

This method is only supported if **CapTrainingMode** is true. Output in this mode is accomplished using the **printRec...** methods in order to print a receipt or other methods to print reports. This method can be successfully called only if the current value of the **PrinterState** property is FPTR\_PS\_MONITOR. If this method is successful, the **TrainingModeActive** property will be changed to true.

**Errors**      A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
JPOS_E_ILLEGAL	The printer does not support training mode (see the <b>CapTrainingMode</b> property).
JPOS_E_EXTENDED:	<i>ErrorCodeExtended</i> =
JPOS_EFPTR_WRONG_STATE:	The printer’s current state does not allow this state transition.

**See Also**      **endTraining** Method, **printRec...** Methods

**clearError Method**

- Syntax**      **void clearError () throws JposException;**
- Remarks**      Clears all printer error conditions. This method is always performed synchronously.
- Errors**          A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
JPOS_E_FAILURE	Error recovery failed.

**endFiscalDocument Method**

- Syntax**      **void endFiscalDocument () throws JposException;**
- Remarks**      Terminates fiscal printing to the slip station.
- This method is only supported if **CapSlpFiscalDocument** is true. If this method is successful, the **PrinterState** property will be changed to FPTR\_PS\_MONITOR.
- Errors**          A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
JPOS_E_ILLEGAL	The printer does not support fiscal output to the slip station (see the <b>CapSlpFiscalDocument</b> property).
JPOS_E_EXTENDED:	<i>ErrorCodeExtended</i> =
JPOS_E_FPTR_WRONG_STATE:	The printer is not currently in the Fiscal Document state.

- See Also**      **beginFiscalDocument** Method, **printFiscalDocumentLine** Method

## endFiscalReceipt Method

**Syntax**      **void endFiscalReceipt (boolean *printHeader*) throws JposException;**

Parameter	Description
-----------	-------------

<i>printHeader</i>	Indicates if the header lines are to be printed at this time.
--------------------	---

**Remarks**      Terminates fiscal printing to the receipt station.

If *printHeader* is false, this method will close the current fiscal receipt, cut it, and print the trailer lines and fiscal logotype, if they were not already printed after the total lines. All functions carried out by this method will be completed before this call returns. If this method is successful, the **PrinterState** property will be changed to FPTR\_PS\_MONITOR.

**Errors**      A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
-------	---------

JPOS_E_EXTENDED:	<i>ErrorCodeExtended</i> = JPOS_EFPTR_WRONG_STATE: The printer is not currently in the Fiscal Receipt Ending state.
------------------	---

**See Also**      **beginFiscalReceipt** Method, **printRec...** Methods

## endFixedOutput Method

**Syntax**      **void endFixedOutput () throws JposException;**

**Remarks**      Terminates non-fiscal fixed text printing on a printer station.

This method is only supported if **CapFixedOutput** is true. If this method is successful, the **PrinterState** property will be changed to FPTR\_PS\_MONITOR.

**Errors**      A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorsCode* property are:

Value	Meaning
-------	---------

JPOS_E_ILLEGAL	The printer does not support fixed output (see the <b>CapFixedOutput</b> property).
----------------	---

JPOS_E_EXTENDED:	<i>ErrorCodeExtended</i> = JPOS_EFPTR_WRONG_STATE: The printer is not currently in the Fixed Output state.
------------------	--

**See Also**      **beginFixedOutput** Method, **printFixedOutput** Method

**endInsertion Method**

**Syntax**      **void endInsertion () throws JposException;**

**Remarks**      Ends form insertion processing.

When called, the printer is taken out of form insertion mode. If the slip device has forms “jaws,” they are closed by this method. If no form is present, a *JposException* is thrown with its *ErrorCodeExtended* property set to *JPOS\_EFPTR\_SLP\_EMPTY*.

This method is paired with the **beginInsertion** method for controlling form insertion. The application may choose to call this method immediately after a successful **beginInsertion** if it wants to use the printer sensors to determine when a form is positioned within the slip printer. Alternatively, the application may prompt the user and wait for a key press before calling this method.

**Errors**      A *JposException* may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
<i>JPOS_E_ILLEGAL</i>	The printer is not in slip insertion mode.
<i>JPOS_E_EXTENDED</i>	<i>ErrorCodeExtended</i> = <i>JPOS_EFPTR_COVER_OPEN</i> : The device was taken out of insertion mode while the printer cover was open.  <i>JPOS_EFPTR_SLP_EMPTY</i> : The device was taken out of insertion mode without a form being inserted.

**See Also**      **beginInsertion** Method, **beginRemoval** Method, **endRemoval** Method

**endItemList Method**

**Syntax**      **void endItemList () throws JposException;**

**Remarks**      Terminates a validation report of items belonging to a particular VAT class.  
 This method is only supported if **CapItemList** is true and **CapHasVatTable** is true.

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

This method can be successfully called only if current value of PrinterState property is equal to FPTR\_PS\_ITEM\_LIST.

If this method is successful, the **PrinterState** property will be changed to FPTR\_PS\_MONITOR.

**Errors**      A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
JPOS_E_ILLEGAL	The printer does not support fixed output (see the <b>CapItemList</b> property) or the printer does not support VAT tables (see the <b>CapHasVatTable</b> property).
JPOS_E_EXTENDED:	<i>ErrorCodeExtended</i> =
	JPOS_EFPTR_WRONG_STATE: The printer’s current state does not allow this state transition.

**See Also**      **beginItemList** Method, **verifyItem** Method

## endNonFiscal Method

**Syntax**      **void endNonFiscal () throws JposException;**

**Remarks**      Terminates non-fiscal operations on one printer station.

This method is only supported if **CapNonFiscalMode** is true. If this method is successful, the **PrinterState** property will be changed to FPTR\_PS\_MONITOR.

**Errors**      A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
JPOS_E_ILLEGAL	The printer does not support non-fiscal output (see the <b>CapNonFiscalMode</b> property).
JPOS_E_EXTENDED:	<i>ErrorCodeExtended</i> =
	JPOS_EFPTR_WRONG_STATE:
	The printer is not currently in the Non-Fiscal state.

**See Also**      **beginNonFiscal** Method, **printNormal** Method



## endRemoval Method

**Syntax**      **void endRemoval () throws JposException;**

**Remarks**      Ends form removal processing.

When called, the printer is taken out of form removal or ejection mode. If a form is present, a `JposException` is thrown with the *ErrorCodeExtended* property set to `JPOS_EFPTR_SLP_FORM`.

This method is paired with the **beginRemoval** method for controlling form removal. The application may choose to call this method immediately after a successful **beginRemoval** if it wants to use the printer sensors to determine when the form has been removed. Alternatively, the application may prompt the user and wait for a key press before calling this method.

**Errors**          A `JposException` may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
<code>JPOS_E_ILLEGAL</code>	The printer is not in slip removal mode.
<code>JPOS_E_EXTENDED</code>	<i>ErrorCodeExtended</i> = <code>JPOS_EFPTR_SLP_FORM</code> : The device was taken out of removal mode while a form was still present.

**See Also**      **beginInsertion** Method, **endInsertion** Method, **beginRemoval** Method

**endTraining Method**

**Syntax**      **void endTraining () throws JposException;**

**Remarks**      Terminates training operations on either the receipt or the slip station.

This method is only supported if **CapTrainingMode** is true. If this method is successful, the **TrainingModeActive** property will be changed to false.

**Errors**      A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
JPOS_E_ILLEGAL	The printer does not support training mode (see the <b>CapTrainingMode</b> property).
JPOS_E_EXTENDED:	<i>ErrorCodeExtended</i> =
	JPOS_EFPTR_WRONG_STATE: The printer is not currently in the Training state.

**See Also**      **beginTraining** Method, **printRec...** Methods

**getData Method**

**Syntax**      **void getData (int *dataItem*, int [] *optArgs*, String[] *data*) throws JposException;**

Parameter	Description
<i>dataItem</i>	The specific data item to retrieve.
<i>optArgs</i>	For some countries, this additional argument may be needed. Consult the Service vendor's documentation for details.
<i>data</i>	Character string to hold the data retrieved.

The *dataItem* parameter has one of the following values:

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

**Remarks**      Retrieves data from the printer's fiscal module.

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

**Errors**      A JposException may be thrown when this method is invoked. For further information, see "Exceptions" on page 15.

Some possible values of the exception's *ErrorCode* property are:

Value	Meaning
JPOS_E_ILLEGAL	The <i>dataItem</i> specified is invalid.

**getDate Method**

**Syntax**      **void getDate (String[] date) throws JposException;**

Parameter	Description
-----------	-------------

<i>date</i>	Date and time returned as a string.
-------------	-------------------------------------

**Remarks**      Gets the printer's date and time.

The date and time are returned as a string in the format "ddmmyyyhhmm":

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

**Errors**      A JposException may be thrown when this method is invoked. For further information, see "Exceptions" on page 15.

Some possible values of the exception's *ErrorCode* property are:

Value	Meaning
-------	---------

JPOS_E_ILLEGAL	Retrieval of the date and time is not valid at this time.
----------------	---

**getTotalizer Method**

**Syntax**      **void getTotalizer (int vatID, int optArgs, String[] data) throws JposException;**

Parameter	Description
-----------	-------------

<i>vatID</i>	VAT identifier of the required totalizer.
--------------	---

<i>optArgs</i>	For some countries, this additional argument may be needed. Consult the JavaPOS Fiscal Printer Service vendor's documentation for details.
----------------	--

<i>data</i>	Totalizer returned as a string.
-------------	---------------------------------

**Remarks**      Gets the totalizer associated with the given VAT rate.

If **CapSetVatTable** is false, then only one totalizer is present.

**Errors**      A JposException may be thrown when this method is invoked. For further information, see "Exceptions" on page 15.

Some possible values of the exception's *ErrorCode* property are:

Value	Meaning
-------	---------

JPOS_E_ILLEGAL	The <i>vatID</i> parameter is invalid.
----------------	--

**getVatEntry Method**

**Syntax**      **void getVatEntry (int vatID, int optArgs, int[] vatRate) throws JposException;**

Parameter	Description
<i>vatID</i>	VAT identifier of the required rate.
<i>optArgs</i>	For some countries, this additional argument may be needed. Consult the JavaPOS Fiscal Printer Service vendor's documentation for details.
<i>vatRate</i>	The rate associated with the VAT identifier.

**Remarks**      Gets the rate associated with a given VAT identifier.

This method is only supported if **CapSetVatTable** is true.

**Errors**      A JposException may be thrown when this method is invoked. For further information, see "Exceptions" on page 15.

Some possible values of the exception's *ErrorCode* property are:

Value	Meaning
JPOS_E_ILLEGAL	The <i>vatID</i> parameter is invalid.

**printDuplicateReceipt Method**

**Syntax**      **void printDuplicateReceipt () throws JposException;**

**Remarks**      Prints a duplicate of a buffered transaction.

This method is only supported if **CapDuplicateReceipt** is true. This method will succeed if both the **CapDuplicateReceipt** and **DuplicateReceipt** properties are true. This method resets the **DuplicateReceipt** property to false.

**Errors**      A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
JPOS_E_BUSY	Cannot perform while output is in progress. (Only applies if <b>AsyncMode</b> is false.)
JPOS_E_ILLEGAL	The printer does not support duplicate receipts (see the <b>CapDuplicateReceipt</b> property) or there is no buffered transaction to print (see <b>DuplicateReceipt</b> property).
JPOS_E_EXTENDED:	<i>ErrorCodeExtended</i> =
	JPOS_EFPTR_WRONG_STATE: The printer is not currently in the Monitor state.
	JPOS_EFPTR_JRN_EMPTY: The journal station is out of paper.
	JPOS_EFPTR_REC_EMPTY: The receipt station is out of paper.

## printFiscalDocumentLine Method

**Syntax**      **void printFiscalDocumentLine (String *documentLine*) throws JposException;**

Parameter	Description
<i>documentLine</i>	String to be printed on the fiscal slip.

**Remarks**      Prints a line of fiscal text to the slip station.

This method is only supported if **CapSlpFiscalDocument** is true. This method is performed synchronously if **AsyncMode** is false, and asynchronously if **AsyncMode** is true.

**Errors**      A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_BUSY	Cannot perform while output is in progress. (Only applies if <b>AsyncMode</b> is false.)
JPOS_E_ILLEGAL	The printer does not support fiscal documents (see the <b>CapSlpFiscalDocument</b> property).
JPOS_E_EXTENDED:	<i>ErrorCodeExtended</i> =
	JPOS_EFPTR_WRONG_STATE: The printer is not currently in the Fiscal Document state.
	JPOS_EFPTR_COVER_OPEN: The printer cover is open. (Only applies if <b>AsyncMode</b> is false.)
	JPOS_EFPTR_SLP_EMPTY: The slip station was specified, but a form is not inserted. (Only applies if <b>AsyncMode</b> is false.)

**See Also**      **beginFiscalDocument** Method, **endFiscalDocument** Method





**printNormal Method**

**Syntax**      **void printNormal (int *station*, String *data*) throws JposException;**

Parameter	Description
<i>station</i>	The printer station to be used. May be FPTR_S_RECEIPT, FPTR_S_JOURNAL, or FPTR_S_SLIP.
<i>data</i>	The characters to be printed. May consist mostly of printable characters, escape sequences, carriage returns (13 decimal), and newline / line feeds (10 decimal) but in many cases these are not supported.

**Remarks**      Performs non-fiscal printing. Prints *data* on the printer *station*.

This method is performed synchronously if **AsyncMode** is false, and asynchronously if **AsyncMode** is true.

Special character values within *Data* are:

Value	Meaning
Newline / Line Feed (\n)	Print any data in the line buffer, and feed to the next print line. (A Carriage Return is not required in order to cause the line to be printed.)
Carriage Return (\r)	If a Carriage Return immediately precedes a Line Feed, or if the line buffer is empty, then it is ignored.  Otherwise, the line buffer is printed and the printer does not feed to the next print line. On some printers, print without feed may be directly supported. On others, a print may always feed to the next line, in which case the Device will print the line buffer and perform a reverse line feed if supported. If the printer does not support either of these features, then Carriage Return acts like a Line Feed.

**Errors**      A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_ILLEGAL	The specified <i>station</i> does not exist. (See the <b>CapJrnPresent</b> , <b>CapSlpPresent</b> and <b>CapRecPresent</b> properties.)
JPOS_E_BUSY	Cannot perform while output is in progress. (Only applies if <b>AsyncMode</b> is false.)

JPOS\_E\_EXTENDED: *ErrorCodeExtended* =  
JPOS\_EFPTR\_WRONG\_STATE:  
The printer is not currently in the Non-Fiscal state.

JPOS\_EFPTR\_COVER\_OPEN:  
The printer cover is open.  
(Only applies if **AsyncMode** is false.)

JPOS\_EFPTR\_JRN\_EMPTY:  
The journal station was specified but is out of paper.  
(Only applies if **AsyncMode** is false.)

JPOS\_EFPTR\_REC\_EMPTY:  
The receipt station was specified but is out of paper.  
(Only applies if **AsyncMode** is false.)

JPOS\_EFPTR\_SLP\_EMPTY:  
The slip station was specified, but a form is not inserted.  
(Only applies if **AsyncMode** is false.)

**See Also**     **beginNonFiscal** Method, **endNonFiscal** Method, **AsyncMode** property



**printPowerLossReport Method**

- Syntax**      **void printPowerLossReport () throws JposException;**
- Remarks**      Prints on the receipt a report of a power failure that resulted in a loss of data stored in the CMOS of the printer.
- This method is only supported if **CapPowerLossReport** is true. This method is always performed synchronously.
- Errors**            A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
JPOS_E_ILLEGAL	The printer does not support power loss reports (see the <b>CapPowerLossReport</b> property).
JPOS_E_EXTENDED:	<i>ErrorCodeExtended</i> =
	JPOS_EFPTR_WRONG_STATE: The printer’s current state does not allow this state transition.
	JPOS_EFPTR_COVER_OPEN: The printer cover is open.
	JPOS_EFPTR_JRN_EMPTY: The journal station is out of paper.
	JPOS_EFPTR_REC_EMPTY: The receipt station is out of paper.

## printRecItem Method

**Syntax**      **void printRecItem (String description, long price, int quantity, int vatInfo, long unitPrice, String unitName) throws JposException;**

Parameter	Description
<i>description</i>	Text describing the item sold.
<i>price</i>	Price of the line item.
<i>quantity</i>	Number of items. If zero, a single item is assumed.
<i>vatInfo</i>	VAT rate identifier or amount. If not used a zero is to be transferred.
<i>unitPrice</i>	Price of each item. If not used a zero is to be transferred.
<i>unitName</i>	Name of the unit i.e. "kg" or "ltr" or "pcs". If not used an empty string ("") is to be transferred

**Remarks**      Prints a receipt item for a sold item. If the *quantity* parameter is zero, then a single item quantity will be assumed.

Minimum parameters are *description* and *price* or *description*, *price*, *quantity*, and *unitPrice*. Most countries require *quantity* and *vatInfo* and some countries also require *unitPrice* and *unitName*. *VatInfo* parameter contains a VAT table identifier if **CapHasVatTable** is true. Otherwise, it contains a VAT amount. This method is performed synchronously if **AsyncMode** is false, and asynchronously if **AsyncMode** is true.

**Errors**          A JposException may be thrown when this method is invoked. For further information, see "Exceptions" on page 15.

Some possible values of the exception's *ErrorCode* property are:

Value	Meaning
JPOS_E_BUSY	Cannot perform while output is in progress. (Only applies if <b>AsyncMode</b> is false.)
JPOS_E_EXTENDED:	<i>ErrorCodeExtended</i> =
	JPOS_EFPTR_WRONG_STATE: The printer is not currently in the Fiscal Receipt state.
	JPOS_EFPTR_COVER_OPEN: The printer cover is open. (Only applies if <b>AsyncMode</b> is false.)
	JPOS_EFPTR_JRN_EMPTY: The journal station is out of paper. (Only applies if <b>AsyncMode</b> is false.)
	JPOS_EFPTR_REC_EMPTY: The receipt station is out of paper. (Only applies if <b>AsyncMode</b> is false.)

*ErrorCodeExtended* =  
JPOS\_EFPTR\_BAD\_ITEM\_QUANTITY:  
The quantity is invalid.  
(Only applies if **AsyncMode** is false.)

JPOS\_EFPTR\_BAD\_PRICE:  
The unit price is invalid.  
(Only applies if **AsyncMode** is false.)

JPOS\_EFPTR\_BAD\_ITEM\_DESCRIPTION:  
The discount description is too long or contains a reserved word.  
(Only applies if **AsyncMode** is false.)

JPOS\_EFPTR\_BAD\_VAT:  
The VAT parameter is invalid.  
(Only applies if **AsyncMode** is false.)

JPOS\_EFPTR\_RECEIPT\_TOTAL\_OVERFLOW:  
The receipt total has overflowed.  
(Only applies if **AsyncMode** is false.)

**See Also**    **beginFiscalReceipt** Method, **endFiscalReceipt** Method, **printRec...** Methods, **AmountDecimalPlaces** Property

## printRecItemAdjustment Method

**Syntax**      **void printRecItemAdjustment (int *adjustmentType*, String *description*, long *amount*, int *vatInfo*) throws JposException;**

Parameter	Description
<i>adjustmentType</i>	Type of discount. See below for values.
<i>description</i>	Text describing the discount.
<i>amount</i>	Amount of the discount.
<i>vatInfo</i>	VAT rate identifier or amount.

The *adjustmentType* parameter has one of the following values (*Note: If currency value, four decimal places are used*):

Value	Meaning
FPTR_AT_AMOUNT_DISCOUNT	Fixed amount discount. The <i>amount</i> parameter contains a currency value.
FPTR_AT_AMOUNT_SURCHARGE	Fixed amount surcharge. The <i>amount</i> parameter contains a currency value.
FPTR_AT_PERCENTAGE_DISCOUNT	Percentage discount. The <i>amount</i> parameter contains a percentage value.
FPTR_AT_PERCENTAGE_SURCHARGE	Percentage surcharge. The <i>amount</i> parameter contains a percentage value.

**Remarks**      Applies and prints a discount or a surcharge to the last receipt item sold. This discount may be either a fixed currency amount or a percentage amount relating to the last item.

If **CapOrderAdjustmentFirst** is true, the method must be called before the corresponding **printRecItem** method. If **CapOrderAdjustmentFirst** is false, the method must be called after the **printRecItem**. This method is performed synchronously if **AsyncMode** is false, and asynchronously if **AsyncMode** is true.

This discount/surcharge may be either a fixed currency amount or a percentage amount relating to the last item. If the discount amount is greater than the receipt subtotal, an error occurs since the subtotal can never be negative. In many countries discount operations cause the printing of a fixed line of text expressing the kind of operation that has been performed. Fixed amount discounts/surcharges are only supported if **CapAmountAdjustment** is true. Percentage discounts are only supported if **CapPercentAdjustment** is true.

The *VatInfo* parameter contains a VAT table identifier if **CapHasVatTable** is true. Otherwise, it contains a VAT amount.

**Errors** A `JposException` may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_BUSY	Cannot perform while output is in progress. (Only applies if <b>AsyncMode</b> is false.)
JPOS_E_ILLEGAL	One of the following errors occurred:  The printer does not support fixed amount adjustments (see the <b>CapAmountAdjustment</b> property).  The printer does not support percentage discounts (see the <b>CapPercentAdjustment</b> property).  The <i>adjustmentType</i> parameter is invalid.
JPOS_E_EXTENDED:	<i>ErrorCodeExtended</i> = JPOS_EFPTR_WRONG_STATE: The printer is not currently in the Fiscal Receipt state.  JPOS_EFPTR_COVER_OPEN: The printer cover is open. (Only applies if <b>AsyncMode</b> is false.)  JPOS_EFPTR_JRN_EMPTY: The journal station is out of paper. (Only applies if <b>AsyncMode</b> is false.)  JPOS_EFPTR_REC_EMPTY: The receipt station is out of paper. (Only applies if <b>AsyncMode</b> is false.)  JPOS_EFPTR_BAD_ITEM_AMOUNT: The discount amount is invalid. (Only applies if <b>AsyncMode</b> is false.)  JPOS_EFPTR_BAD_ITEM_DESCRIPTION: The discount description is too long or contains a reserved word. (Only applies if <b>AsyncMode</b> is false.)  JPOS_EFPTR_BAD_VAT: The VAT parameter is invalid. (Only applies if <b>AsyncMode</b> is false.)

**See Also** **beginFiscalReceipt** Method, **endFiscalReceipt** Method, **printRec...** Methods, **AmountDecimalPlaces** Property



## printRecMessage Method

**Syntax**      **void printRecMessage (String *message*) throws JposException;**

Parameter	Description
-----------	-------------

<i>message</i>	Text message to print.
----------------	------------------------

**Remarks**      Prints a message on the fiscal receipt. The length of an individual message is limited to the number of characters given in the **MessageLength** property.

This method is only supported if **CapAdditionalLines** is true. This method is only supported when the printer is in the Fiscal Receipt Ending state. This method is performed synchronously if **AsyncMode** is false, and asynchronously if **AsyncMode** is true.

**Errors**      A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
-------	---------

JPOS_E_BUSY	Cannot perform while output is in progress. (Only applies if <b>AsyncMode</b> is false.)
-------------	--

JPOS_E_EXTENDED:	<i>ErrorCodeExtended</i> =
JPOS_EFPTR_WRONG_STATE:	The printer is not in the Fiscal Receipt Ending state.

JPOS_EFPTR_COVER_OPEN:	The printer cover is open. (Only applies if <b>AsyncMode</b> is false.)
------------------------	--

JPOS_EFPTR_JRN_EMPTY:	The journal station is out of paper. (Only applies if <b>AsyncMode</b> is false.)
-----------------------	--

JPOS_EFPTR_REC_EMPTY:	The receipt station is out of paper. (Only applies if <b>AsyncMode</b> is false.)
-----------------------	--

JPOS_EFPTR_BAD_ITEM_DESCRIPTION:	The message is too long or contains a reserved word. (Only applies if <b>AsyncMode</b> is false.)
----------------------------------	--

**See Also**      **beginFiscalReceipt** Method, **endFiscalReceipt** Method, **printRec...** Methods, **MessageLength** property, **CapAdditionalLines** property



(Only applies if **AsyncMode** is false.)

See Also **beginFiscalReceipt** Method, **endFiscalReceipt** Method, **printRec...** Methods, **AmountDecimalPlaces** Property

## printRecRefund Method

Syntax **void printRecRefund (String *description*, long *amount*, int *vatInfo*)  
throws **JposException**;**

Parameter	Description
<i>description</i>	Text describing the refund.
<i>amount</i>	Amount of the refund.
<i>vatInfo</i>	VAT rate identifier or amount.

Remarks Processes a refund. The *amount* is positive, but it is printed as a negative number and the totals registers are decremented.

Some fixed text, along with the *description*, will be printed on the receipt and journal to indicate that a refund has occurred. The *vatInfo* parameter contains a VAT table identifier if **CapHasVatTable** is true. Otherwise it, contains a VAT amount. This method is performed synchronously if **AsyncMode** is false, and asynchronously if **AsyncMode** is true.

Errors A **JposException** may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_BUSY	Cannot perform while output is in progress. (Only applies if <b>AsyncMode</b> is false.)
JPOS_E_EXTENDED:	<i>ErrorCodeExtended</i> =
	JPOS_EFPTR_WRONG_STATE: The printer is not currently in the Fiscal Receipt state.
	JPOS_EFPTR_COVER_OPEN: The printer cover is open. (Only applies if <b>AsyncMode</b> is false.)
	JPOS_EFPTR_JRN_EMPTY: The journal station is out of paper. (Only applies if <b>AsyncMode</b> is false.)
	JPOS_EFPTR_REC_EMPTY: The receipt station is out of paper. (Only applies if <b>AsyncMode</b> is false.)
	JPOS_EFPTR_BAD_ITEM_DESCRIPTION:

The *description* is too long or contains a reserved word.  
(Only applies if **AsyncMode** is false.)

JPOS\_EFPTR\_BAD\_ITEM\_AMOUNT:  
The *amount* is invalid.  
(Only applies if **AsyncMode** is false.)

JPOS\_EFPTR\_BAD\_VAT:  
The VAT information is invalid.  
(Only applies if **AsyncMode** is false.)

**See Also** **beginFiscalReceipt** Method, **endFiscalReceipt** Method, **printRec...** Methods,  
**AmountDecimalPlaces** Property

## printRecSubtotal Method

**Syntax**      **void printRecSubtotal (long amount) throws JposException;**

Parameter	Description
<i>amount</i>	Amount of the subtotal.

**Remarks**      Checks and prints the current receipt subtotal. If **CapCheckTotal** is true, the *Amount* is compared to the subtotal calculated by the printer. If the subtotals match, the subtotal is printed on both the receipt and journal. If the results do not match, the receipt is automatically canceled. If **CapCheckTotal** is false, then the subtotal is printed on the receipt and journal and the parameter is never compared to the subtotal computed by the printer.

This method is performed synchronously if **AsyncMode** is false, and asynchronously if **AsyncMode** is true.

If this method compares the application's subtotal with the printer's subtotal and they do not match, the **PrinterState** property will be changed to **FPTR\_PS\_FISCAL\_RECEIPT\_ENDING**.

**Errors**      A *JposException* may be thrown when this method is invoked. For further information, see "Exceptions" on page 15.

Some possible values of the exception's *ErrorCode* property are:

Value	Meaning
<b>JPOS_E_BUSY</b>	Cannot perform while output is in progress. (Only applies if <b>AsyncMode</b> is false.)
<b>JPOS_E_EXTENDED:</b>	<i>ErrorCodeExtended</i> =
	<b>JPOS_EFPTR_WRONG_STATE:</b> The printer is not currently in the Fiscal Receipt state.
	<b>JPOS_EFPTR_COVER_OPEN:</b> The printer cover is open. (Only applies if <b>AsyncMode</b> is false.)
	<b>JPOS_EFPTR_JRN_EMPTY:</b> The journal station is out of paper. (Only applies if <b>AsyncMode</b> is false.)
	<b>JPOS_EFPTR_REC_EMPTY:</b> The receipt station is out of paper. (Only applies if <b>AsyncMode</b> is false.)
	<b>JPOS_EFPTR_BAD_ITEM_AMOUNT:</b> The subtotal from the application does not match the subtotal computed by the printer. (Only applies if <b>AsyncMode</b> is false.)

*ErrorCodeExtended* =  
 JPOS\_EFPTR\_NEGATIVE\_TOTAL:  
 The total computed by the printer is less than zero.  
 (Only applies if **AsyncMode** is false.)

See Also **beginFiscalReceipt** Method, **endFiscalReceipt** Method, **printRec...** Methods, **AmountDecimalPlaces** Property

## printRecSubtotalAdjustment Method

Syntax **void printRecSubtotalAdjustment (int *adjustmentType*, String *description*, long *amount*) throws JposException;**

Parameter	Description
<i>adjustmentType</i>	Type of discount. See below for values.
<i>description</i>	Text describing the discount.
<i>amount</i>	Amount of the discount.

The *adjustmentType* parameter has one of the following values (*Note: If currency value, four decimal places are used*):

Value	Meaning
FPTR_AT_AMOUNT_DISCOUNT	Fixed amount discount. The <i>amount</i> parameter contains a currency value.
FPTR_AT_AMOUNT_SURCHARGE	Fixed amount surcharge. The <i>amount</i> parameter contains a currency value.
FPTR_AT_PERCENTAGE_DISCOUNT	Percentage discount. The <i>amount</i> parameter contains a percentage value.
FPTR_AT_PERCENTAGE_SURCHARGE	Percentage surcharge. The <i>amount</i> parameter contains a percentage value.

**Remarks** Applies and prints a discount/surcharge to the current receipt subtotal. This discount/surcharge may be either a fixed currency amount or a percentage amount relating to the current receipt subtotal. If the discount/surcharge amount is greater than the receipt subtotal, an error occurs since the subtotal can never be negative. In many countries discount/surcharge operations cause the printing of a fixed line of text expressing the kind of operation that has been performed. Fixed amount discounts are only supported if **CapSubAmountAdjustment** is true. Percentage discounts are only supported if **CapSubPercentAdjustment** is true.

This method is performed synchronously if **AsyncMode** is false, and asynchronously if **AsyncMode** is true.

**Errors** A `JposException` may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s `ErrorCode` property are:

Value	Meaning
JPOS_E_BUSY	Cannot perform while output is in progress. (Only applies if <b>AsyncMode</b> is false.)
JPOS_E_ILLEGAL	One of the following errors occurred:  The printer does not support fixed amount discounts (see the <b>CapSubAmountAdjustment</b> property).  The printer does not support percentage discounts (see the <b>CapSubPercentAdjustment</b> property).  The <code>adjustmentType</code> parameter is invalid.
JPOS_E_EXTENDED:	<code>ErrorCodeExtended =</code> JPOS_EFPTR_WRONG_STATE: The printer is not currently in the Fiscal Receipt state.  JPOS_EFPTR_COVER_OPEN: The printer cover is open. (Only applies if <b>AsyncMode</b> is false.)  JPOS_EFPTR_JRN_EMPTY: The journal station is out of paper. (Only applies <b>AsyncMode</b> is false.)  JPOS_EFPTR_REC_EMPTY: The receipt station is out of paper. (Only applies if <b>AsyncMode</b> is false.)  JPOS_EFPTR_BAD_ITEM_AMOUNT: The discount <i>amount</i> is invalid. (Only applies if <b>AsyncMode</b> is false.)  JPOS_EFPTR_BAD_ITEM_DESCRIPTION: The discount <i>description</i> is too long or contains a reserved word. (Only applies if <b>AsyncMode</b> is false.)

**See Also** **beginFiscalReceipt** Method, **endFiscalReceipt** Method, **printRec...** Methods, **AmountDecimalPlaces** Property





**Errors** A `JposException` may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s `ErrorCode` property are:

Value	Meaning
JPOS_E_BUSY	Cannot perform while output is in progress. (Only applies if <b>AsyncMode</b> is false.)
JPOS_E_EXTENDED:	<i>ErrorCodeExtended</i> =
JPOS_EFPTR_WRONG_STATE:	The printer is not currently in the Fiscal Receipt state.
JPOS_EFPTR_COVER_OPEN:	The printer cover is open. (Only applies if <b>AsyncMode</b> is false.)
JPOS_EFPTR_JRN_EMPTY:	The journal station is out of paper. (Only applies if <b>AsyncMode</b> is false.)
JPOS_EFPTR_REC_EMPTY:	The receipt station is out of paper. (Only applies if <b>AsyncMode</b> is false.)
JPOS_EFPTR_BAD_ITEM_AMOUNT:	One of the following errors occurred:  The application computed total does not match the printer computed total.  The <i>total</i> parameter is invalid.  The <i>payment</i> parameter is invalid (Only applies if <b>AsyncMode</b> is false.)
JPOS_EFPTR_BAD_ITEM_DESCRIPTION:	The <i>description</i> is too long or contains a reserved word. (Only applies if <b>AsyncMode</b> is false.)
JPOS_EFPTR_NEGATIVE_TOTAL:	The total computed by the printer is less than zero. (Only applies if <b>AsyncMode</b> is false.)
JPOS_EFPTR_WORD_NOT_ALLOWED:	The description contains the reserved word.

**See Also** **beginFiscalReceipt** Method, **endFiscalReceipt** Method, **printRec...** Methods, **PredefinedPaymentLines** Property, **AmountDecimalPlaces** Property

**printRecVoid Method**

**Syntax**      **void printRecVoid (String *description*) throws JposException;**

**ParameterDescription**


---

*description*                      Text describing the void.

**Remarks**      Cancels the current receipt. The receipt is annulled but it is not physically canceled from the printer's fiscal memory since fiscal receipts are printed with an increasing serial number and totals are accumulated in registers. When a receipt is canceled, its subtotal is subtracted from the totals registers, but it is added to the canceled receipt register.

Some fixed text, along with the *description*, will be printed on the receipt and journal to indicate that the receipt has been canceled. This method is performed synchronously if **AsyncMode** is false, and asynchronously if **AsyncMode** is true. If this method is successful, the **PrinterState** property will be changed to FPTR\_PS\_FISCAL\_RECEIPT\_ENDING.

**Errors**          A JposException may be thrown when this method is invoked. For further information, see "Exceptions" on page 15.

Some possible values of the exception's *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
JPOS_E_BUSY	Cannot perform while output is in progress. (Only applies if <b>AsyncMode</b> is false.)
JPOS_E_EXTENDED:	<i>ErrorCodeExtended</i> =
	JPOS_EFPTR_WRONG_STATE: The printer is not currently in the Fiscal Receipt state.
	JPOS_EFPTR_COVER_OPEN: The printer cover is open. (Only applies if <b>AsyncMode</b> is false.)
	JPOS_EFPTR_JRN_EMPTY: The journal station is out of paper. (Only applies if <b>AsyncMode</b> is false.)
	JPOS_EFPTR_REC_EMPTY: The receipt station is out of paper. (Only applies if <b>AsyncMode</b> is false.)
	JPOS_EFPTR_BAD_ITEM_DESCRIPTION: The description is too long or contains a reserved word. (Only applies if <b>AsyncMode</b> is false.)

**See Also**      **beginFiscalReceipt** Method, **endFiscalReceipt** Method, **printRec...** Methods

**printRecVoidItem Method**

**Syntax**      **void printRecVoidItem** (*String description*, **long amount**, **int quantity**,  
**int adjustmentType**, **long adjustment**, **int vatInfo**)  
**throws JposException;**

<b>Parameter</b>	<b>Description</b>
<i>description</i>	Text description of the item void.
<i>amount</i>	Amount of item to be voided.
<i>quantity</i>	Quantity of item to be voided.
<i>adjustmentType</i>	Type of discount. See below for values.
<i>adjustment</i>	Amount of the discount/surcharge
<i>vatInfo</i>	VAT rate identifier or amount.

The *adjustmentType* parameter has one of the following values (*Note: If currency value, four decimal places are used*):

<b>Value</b>	<b>Meaning</b>
FPTR_AT_AMOUNT_DISCOUNT	Fixed amount discount. The <i>adjustment</i> parameter contains a currency value.
FPTR_AT_AMOUNT_SURCHARGE	Fixed amount surcharge. The <i>adjustment</i> parameter contains a currency value.
FPTR_AT_PERCENTAGE_DISCOUNT	Percentage discount. The <i>adjustment</i> parameter contains a percentage value.
FPTR_AT_PERCENTAGE_SURCHARGE	Percentage surcharge. The <i>adjustment</i> parameter contains a percentage value.

**Remarks**      Cancels an item that has been added to the receipt and print a void description. *Amount* is a positive number, it will be printed as a negative and will be decremented from the totals registers.

The *vatInfo* parameter contains a VAT table identifier if **CapHasVatTable** is true. Otherwise, it contains a VAT amount. Fixed amount discounts/surcharges are only supported if **CapAmountAdjustment** is true. Percentage discounts are only supported if **CapPercentAdjustment** is true. This method is performed synchronously if **AsyncMode** is false, and asynchronously if **AsyncMode** is true.

**Errors**          A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_BUSY	Cannot perform while output is in progress. (Only applies if <b>AsyncMode</b> is false.)
JPOS_E_ILLEGAL	One of the following errors occurred:  The printer does not support fixed amount adjustments (see the <b>CapAmountAdjustment</b> property).  The printer does not support percentage discounts (see the <b>CapPercentAdjustment</b> property).  The <i>adjustmentType</i> parameter is invalid.
JPOS_E_EXTENDED	<i>ErrorCodeExtended</i> = JPOS_EFPTR_WRONG_STATE: The printer is not currently in the Fiscal Receipt state.  JPOS_EFPTR_COVER_OPEN: The printer cover is open. (Only applies if <b>AsyncMode</b> is false.)  JPOS_EFPTR_JRN_EMPTY: The journal station is out of paper. (Only applies if <b>AsyncMode</b> is false.)  JPOS_EFPTR_REC_EMPTY: The receipt station is out of paper. (Only applies if <b>AsyncMode</b> is false.)  JPOS_EFPTR_BAD_ITEM_AMOUNT: The <i>amount</i> is invalid. (Only applies if <b>AsyncMode</b> is false.)  JPOS_EFPTR_BAD_ITEM_QUANTITY: The <i>quantity</i> is invalid. (Only applies if <b>AsyncMode</b> is false.)  JPOS_EFPTR_BAD_VAT: The VAT information is invalid. (Only applies if <b>AsyncMode</b> is false.)  JPOS_EFPTR_BAD_ITEM_DESCRIPTION: The <i>description</i> is too long or contains a reserved word. (Only applies if <b>AsyncMode</b> is false.)  <i>ErrorCodeExtended</i> = JPOS_EFPTR_NEGATIVE_TOTAL: The total computed by the printer is less than zero. (Only applies if <b>AsyncMode</b> is false.)

See Also **beginFiscalReceipt** Method, **endFiscalReceipt** Method, **printRec...** Methods,



**Remarks** Prints a report of the fiscal EPROM contents on the receipt that occurred between two end points.

This method is always performed synchronously.

**Errors** A `JposException` may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_BUSY	Cannot perform while output is in progress.
JPOS_E_ILLEGAL	One of the following errors occurred:  The <i>reportType</i> parameter is invalid.  One or both of <i>startNum</i> and <i>endNum</i> are invalid.  <i>startNum</i> > <i>endNum</i> .
JPOS_E_EXTENDED:	<i>ErrorCodeExtended</i> =
	JPOS_EFPTR_WRONG_STATE: The printer’s current state does not allow this state transition.
	JPOS_EFPTR_COVER_OPEN: The printer cover is open.
	JPOS_EFPTR_JRN_EMPTY: The journal station is out of paper.
	JPOS_EFPTR_REC_EMPTY: The receipt station is out of paper.

## printXReport Method

**Syntax**      **void printXReport () throws JposException;**

**Remarks**      Prints a report of all the daily fiscal activities on the receipt. No data will be written to the fiscal EPROM as a result of this method invocation.

This method is only supported if **CapXReport** is true. This method is always performed synchronously.

**Errors**      A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
JPOS_E_ILLEGAL	The printer does not support X reports (see the <b>CapXReport</b> property).
JPOS_E_EXTENDED:	<i>ErrorCodeExtended</i> =
	JPOS_EFPTR_WRONG_STATE: The printer’s current state does not allow this state transition.
	JPOS_EFPTR_COVER_OPEN: The printer cover is open.
	JPOS_EFPTR_JRN_EMPTY: The journal station is out of paper.
	JPOS_EFPTR_REC_EMPTY: The receipt station is out of paper.

## printZReport Method

<b>Syntax</b>	<b>void printZReport () throws JposException;</b>
<b>Remarks</b>	Prints a report of all the daily fiscal activities on the receipt. Data will be written to the fiscal EPROM as a result of this method invocation.  This method is always performed synchronously.
<b>Errors</b>	A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception's *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
JPOS_E_EXTENDED:	<i>ErrorCodeExtended</i> =
JPOS_EFPTR_WRONG_STATE:	The printer's current state does not allow this state transition.
JPOS_EFPTR_COVER_OPEN:	The printer cover is open.
JPOS_EFPTR_JRN_EMPTY:	The journal station is out of paper.
JPOS_EFPTR_REC_EMPTY:	The receipt station is out of paper.



## resetPrinter Method

**Syntax**      **void resetPrinter () throws JposException;**

**Remarks**      Forces the printer to return to Monitor state. This forces any interrupted operations to be canceled and closed. This method must be invoked when the printer is not in a Monitor state after a successful call to the **claim** method and successful setting of the **DeviceEnabled** property to true. This typically happens if a power failure occurs during a fiscal operation.

Calling this method does not close the printer, i.e. does not force a Z report to be printed.

The Device will handle this command as follows:

- If the printer was in either Fiscal Receipt, Fiscal Receipt Total or Fiscal Receipt Ending state, the receipt will be ended without updating any registers.
- If the printer was in a non-fiscal state, the printer will exit that state.
- If the printer was in the training state, the printer will exit the training state.

This method is always performed synchronously.

**Errors**      A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

**setDate Method**

**Syntax**      **void setDate (String date) throws JposException;**

Parameter	Description
<i>date</i>	Date and time as a string.

**Remarks**      Sets the printer's date and time.

The date and time is passed as a string in the format "ddmmyyyhhmm", where:

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

This method can only be called while **DayOpened** is false.

**Errors**      A JposException may be thrown when this method is invoked. For further information, see "Exceptions" on page 15.

Some possible values of the exception's *ErrorCode* property are:

Value	Meaning
JPOS_E_ILLEGAL	The printer has already begun the fiscal day (see the <b>DayOpened</b> property).
JPOS_E_EXTENDED:	<i>ErrorCodeExtended</i> = JPOS_EFPTR_BAD_DATE: One of the date parameters is invalid.



**setPOSID Method**

**Syntax**      **void setPOSID (String *POSID*, String *cashierID*) throws JposException;**

<b>Parameter</b>	<b>Description</b>
<i>POSID</i>	Identifier for the POS system.
<i>cashierID</i>	Identifier of the current cashier.

**Remarks**      Sets the POS and cashier identifiers. These values will be printed when each fiscal receipt is closed.

This method is only supported if **CapSetPOSID** is true. This method can only be called while **DayOpened** is false.

**Errors**      A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
JPOS_E_ILLEGAL	One of the following errors occurred:  The printer does not support setting the POS identifier (see the CapSetPOSID property).  The printer has already begun the fiscal day (see the DayOpened property).  Either the <i>POSID</i> or <i>cashierID</i> parameter is invalid.

**setStoreFiscalID Method**

**Syntax**      **void setStoreFiscalID (String *ID*) throws JposException;**

---

**ParameterDescription**


---

*ID*                      Fiscal identifier.

**Remarks**      Sets the store fiscal ID. This value is retained by the printer even after power failures. This *ID* is automatically printed by the printer after the fiscal receipt header lines.

This method is only supported if **CapSetStoreFiscalID** is true. This method can only be called while **DayOpened** is false.

**Errors**          A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
JPOS_E_ILLEGAL	One of the following errors occurred:  The printer does not support setting the store fiscal identifier (see the <b>CapSetStoreFiscalID</b> property).  The printer has already begun the fiscal day (see the <b>DayOpened</b> property).  The <i>ID</i> parameter was invalid.



## setVatTable Method

**Syntax**      **void setVatTable () throws JposException;**

**Remarks**      Sends the VAT table built inside the Service to the printer. The VAT table is built one entry at a time using the **setVatValue** method.

This method is only supported if **CapHasVatTable** is true. This method can only be called while **DayOpened** is false.

**Errors**      A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
JPOS_E_ILLEGAL	The printer has already begun the fiscal day (see the <b>DayOpened</b> property).

**See Also**      **setVatValue** Method

**setVatValue Method**

**Syntax**      **void setVatValue (int vatID, String vatValue) throws JposException;**

Parameter	Description
<i>vatID</i>	Index of the VAT table entry to set.
<i>vatValue</i>	Tax value as a percentage.

**Remarks**      Sets the value of a specific VAT class in the VAT table. The VAT table is built one entry at a time in the Service using this method. The entire table is then sent to the printer at one time using the **setVatTable** method.

This method is only supported if **CapHasVatTable** is true. This method can only be called while **DayOpened** is false.

**Errors**      A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_ILLEGAL	One of the following errors occurred: The printer does not support VAT tables (see the <b>CapHasVatTable</b> property). The printer has already begun the fiscal day (see the <b>DayOpened</b> property). The printer does not support changing an existing VAT value.

**See Also**      **setVatTable** Method



**verifyItem Method**

**Syntax**      **void verifyItem (String *itemName*, int *vatID*) throws JposException;**

<b>Parameter</b>	<b>Description</b>
<i>itemName</i>	Item to be verified.
<i>vatID</i>	VAT identifier of the item.

**Remarks**      Compares *itemName* and its *vatID* with the values stored in the printer.  
 This method is only supported if **CapHasVatTable** is true. This method can only be called while the printer is in the Item List state.

**Errors**      A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
JPOS_E_ILLEGAL	The printer does not support VAT tables (see the <b>CapHasVatTable</b> property).
JPOS_E_EXTENDED:	<i>ErrorCodeExtended</i> =
	JPOS_EFPTR_WRONG_STATE: The printer is not currently in the Item List state.
	JPOS_EFPTR_BAD_ITEM_DESCRIPTION: The item name is too long or contains a reserved word. (Only applies if <b>AsyncMode</b> is false.)
	JPOS_EFPTR_BAD_VAT: The VAT parameter is invalid. (Only applies if <b>AsyncMode</b> is false.)

**See Also**      **setVatTable** Method

## Events

### DirectIOEvent

**Interface** `jpos.events.DirectIOListener`

**Method** `directIOOccurred (DirectIOEvent e);`

**Description** Provides Device Service information directly to the application. This event provides a means for a vendor-specific Fiscal Printer Device Service to provide events to the application that are not otherwise supported by the Device Control.

**Properties** This event contains the following properties:

Property	Type	Description
<i>EventNumber</i>	<i>int</i>	Event number whose specific values are assigned by the Device Service.
<i>Data</i>	<i>int</i>	Additional numeric data. Specific values vary by the <i>EventNumber</i> and the Device Service. This property is settable.
<i>Object</i>	<i>Object</i>	Additional data whose usage varies by the <i>EventNumber</i> and Device Service. This property is settable.

**Remarks** This event is to be used only for those types of vendor specific functions that are not otherwise described as part of the JavaPOS standard. Use of this event may restrict the application program from being used with other vendor's Fiscal Printer devices which may not have any knowledge of the Device Service's need for this event.

**See Also** "Events" on page 18, **directIO** Method

**ErrorEvent**

**Interface** `jpos.events.ErrorListener`

**Method** `errorOccured(ErrorEvent e);`

**Description** Notifies the application that a printer error has been detected and a suitable response by the application is necessary to process the error condition.

**Properties** This event contains the following properties:

Property	Type	Description
<i>ErrorCode</i>	<i>int</i>	Error code causing the <b>ErrorEvent</b> . See list of <i>ErrorCodes</i> on Page .
<i>ErrorCodeExtended</i>	<i>int</i>	Extended error code causing the <b>ErrorEvent</b> . If <i>ErrorCode</i> is JPOS_E_XTENDED, then see values below. Otherwise, it may contain a Device Service-specific value.
<i>ErrorLocus</i>	<i>int</i>	Location of the error, and is set to JPOS_E_OUTPUT indicating the error occurred while processing asynchronous output. See values below.
<i>ErrorResponse</i>	<i>int</i>	<b>ErrorEvent</b> response, whose default value may be overwritten by the application (i.e., this property is settable). See values below.

If *ErrorCode* is JPOS\_E\_EXTENDED, then *ErrorCodeExtended* has one of the following values:

Value	Meaning
JPOS_EFPTR_COVER_OPEN	The printer cover is open.
JPOS_EFPTR_JRN_EMPTY	The journal station is out of paper.
JPOS_EFPTR_REC_EMPTY	The receipt station is out of paper.
JPOS_EFPTR_SLP_EMPTY	A form is not inserted in the slip station.
JPOS_EFPTR_WRONG_STATE	The requested method could not be executed in the printer's current state.
JPOS_EFPTR_TECHNICAL_ASSISTANCE	The printer has encountered a severe error condition. Calling for printer technical assistance is required.

JPOS_EFPTR_CLOCK_ERROR	The printer's internal clock has failed.
JPOS_EFPTR_FISCAL_MEMORY_FULL	The printer's fiscal memory has been exhausted.
JPOS_EFPTR_FISCAL_MEMORY_DISCONNECTED	The printer's fiscal memory has been disconnected.
JPOS_EFPTR_FISCAL_TOTALS_ERROR	The Grand Total in working memory does not match the one in the EPROM.
JPOS_EFPTR_BAD_ITEM_QUANTITY	The Quantity parameter is invalid.
JPOS_EFPTR_BAD_ITEM_AMOUNT	The Amount parameter is invalid.
JPOS_EFPTR_BAD_ITEM_DESCRIPTION	The Description parameters is either too long, contains illegal characters or contains the reserved word.
JPOS_EFPTR_RECEIPT_TOTAL_OVERFLOW	The receipt total has overflowed.
JPOS_EFPTR_BAD_VAT	The Vat parameter is invalid.
JPOS_EFPTR_BAD_PRICE	The Price parameter is invalid.
JPOS_EFPTR_NEGATIVE_TOTAL	The printer's computed total or subtotal is less than zero.
JPOS_EFPTR_MISSING_DEVICES	Some of the other devices which according to the local fiscal legislation are to be connected has been disconnected. In some countries in order to use a fiscal printer a full set of peripheral devices are to be connected to the POS (such as cash drawer and customer display). In case one of these devices is not present sales are not allowed.

The application's **ErrorEvent** listener may change the value of *ErrorResponse* to one of the following:

<b>Value</b>	<b>Meaning</b>
JPOS_ER_RETRY	Retry the asynchronous output. The error state is exited.
JPOS_ER_CLEAR	Clear the asynchronous output or buffered input data. The error state is exited.

**Remarks** Enqueued when an error is detected and the Device state transitions into the error state. This event is not delivered until **DataEventEnabled** is true, so that proper application sequencing occurs.

**See Also** “Device Output Models” on page 25 , “Device States” on page 30.

## OutputCompleteEvent

**Interface** `jpos.events.OutputCompleteListener`

**Method** `outputCompleteOccurred (OutputCompleteEvent e);`

**Description** Notifies the application that the queued output request associated with the *OutputID* property has completed successfully.

**Properties** This event contains the following property:

Property	Type	Description
<i>OutputID</i>	<i>int</i>	The ID number of the asynchronous output request that is complete.

**Remarks** This event is enqueued after the request’s data has been both sent and the Device Service has confirmation that it was processed by the device successfully.

**See Also** “Device Output Models” on page 25

## StatusUpdateEvent

**Interface** `jpos.events.StatusUpdateListener`

**Method** `statusUpdateOccurred(StatusUpdateEvent e);`

**Description** Notifies the application that a printer has had an operation status change.

**Properties** This event contains the following properties:

Property	Type	Description
<i>status</i>	<i>int</i>	Indicates the status change and has one of the following values:
Value	Meaning	
FPTR_SUE_COVER_OPEN	Printer cover is open.	
FPTR_SUE_COVER_OK	Printer cover is closed.	
FPTR_SUE_JRN_EMPTY	No journal paper.	
FPTR_SUE_JRN_NEAREMPTY	Journal paper is low.	
FPTR_SUE_JRN_PAPEROK	Journal paper is ready.	
FPTR_SUE_REC_EMPTY	No receipt paper.	
FPTR_SUE_REC_NEAREMPTY	Receipt paper is low.	
FPTR_SUE_REC_PAPEROK	Receipt paper is ready.	
FPTR_SUE_SLP_EMPTY	No slip form.	
FPTR_SUE_SLP_NEAREMPTY	Almost at the bottom of the slip form.	
FPTR_SUE_SLP_PAPEROK	Slip form is inserted.	
FPTR_SUE_IDLE	All asynchronous output has finished, either successfully or because output has been cleared. The printer State is now JPOS_S_IDLE. The <b>FlagWhenIdle</b> property must be true for this event to be delivered, and the property is set to false just before delivering the event.	

*Note that Release 1.3* added Power State Reporting with additional *Power reporting* **StatusUpdateEvent** values. See “StatusUpdateEvent” description on page 78.

**Remarks**      Enqueued when a significant status event has occurred.

**See Also**      “Events” on page 18.





# Hard Totals

## Summary

### Properties

<i>Common</i>	<i>Ver</i>	<i>Type</i>	<i>Access</i>	<i>May Use After</i>
<b>AutoDisable</b>		boolean	R/W	<i>Not Supported</i>
<b>CapPowerReporting</b>	1.3	int	R	open
<b>CheckHealthText</b>		String	R	open
<b>Claimed</b>		boolean	R	open
<b>DataCount</b>		int	R	<i>Not Supported</i>
<b>DataEventEnabled</b>		boolean	R/W	<i>Not Supported</i>
<b>DeviceEnabled</b>		boolean	R/W	open
<b>FreezeEvents</b>		boolean	R/W	open
<b>OutputID</b>		int	R	<i>Not Supported</i>
<b>PowerNotify</b>	1.3	int	R/W	open
<b>PowerState</b>	1.3	int	R	open
<b>State</b>		int	R	--
<b>DeviceControlDescription</b>		String	R	--
<b>DeviceControlVersion</b>		int	R	--
<b>DeviceServiceDescription</b>		String	R	open
<b>DeviceServiceVersion</b>		int	R	open
<b>PhysicalDeviceDescription</b>		String	R	open
<b>PhysicalDeviceName</b>		String	R	open

---

<i>Specific</i>	<i>Ver</i>	<i>Type</i>	<i>Access</i>	<i>May Use After</i>
<b>CapErrorDetection</b>		boolean	R	open
<b>CapSingleFile</b>		boolean	R	open
<b>CapTransactions</b>		boolean	R	open
<b>FreeData</b>		int	R	open & enable
<b>TotalsSize</b>		int	R	open & enable
<b>NumberOfFiles</b>		int	R	open & enable
<b>TransactionInProgress</b>		boolean	R	open

**Methods**

<i>Common</i>	<i>Ver</i>	<i>May Use After</i>
<b>open</b>		--
<b>close</b>		open
<b>claim</b>		open
<b>release</b>		open & claim
<b>checkHealth</b>		open & enable; <i>Note 1</i>
<b>clearInput</b>		<i>Not Supported</i>
<b>clearOutput</b>		<i>Not Supported</i>
<b>directIO</b>		open
 <i>Specific</i>		
<b>claimFile</b>		open & enable; <i>Note 2</i>
<b>releaseFile</b>		open & enable
<b>read</b>		open & enable; <i>Note 2</i>
<b>write</b>		open & enable; <i>Note 2</i>
<b>setAll</b>		open & enable; <i>Note 2</i>
<b>validateData</b>		open & enable; <i>Note 2</i>
<b>recalculateValidationData</b>		open & enable; <i>Note 2</i>
<b>create</b>		open & enable; <i>Note 1</i>
<b>find</b>		open & enable; <i>Note 1</i>
<b>findByIndex</b>		open & enable; <i>Note 1</i>
<b>delete</b>		open & enable; <i>Note 2</i>
<b>rename</b>		open & enable; <i>Note 2</i>
<b>beginTrans</b>		open & enable
<b>commitTrans</b>		open & enable
<b>rollback</b>		open & enable

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

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

**Events**

<i>Name</i>	<i>Ver</i>	<i>May Occur After</i>
<b>DataEvent</b>		<i>Not Supported</i>
<b>DirectIOEvent</b>	1.3	open & claim
<b>ErrorEvent</b>		<i>Not Supported</i>
<b>OutputCompleteEvent</b>		<i>Not Supported</i>
<b>StatusUpdateEvent</b>	1.3	open, claim, & enable

## General Information

The Hard Totals Control's class name is "jpos.HardTotals".  
The device constants are contained in the class "jpos.HardTotalsConst".  
See "Package Structure" on page 40.

### Capabilities

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

- Supports at least one totals file with the name "" (the empty string) in an area of totals memory. Each totals file is read and written as if it were a sequence of byte data.
- Creates each totals file with a fixed size and may be deleted, initialized, and claimed for exclusive use.

The Hard Totals device may have the following additional capabilities:

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

### Model

Totals memory is frequently a limited but secure resource - perhaps of only several thousand bytes of storage. The following is the general model of the Hard Totals:

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

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

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

- The following operations may be performed on a totals file:
  - **read**: Read a series of data bytes.
  - **write**: Write a series of data bytes.
  - **setAll**: Set all the data in a totals file to a value.
  - **find**: Locate an existing totals file by name, and return a file handle and size.
  - **findByIndex**: Enumerate all of the files in the Hard Totals area.
  - **delete**: Delete a totals file by name.
  - **rename**: Rename an existing totals file.
  - **claimFile**: Gain exclusive access to a specific file for use by the claiming application. A timeout value may be specified in case another application maintains access for a period a time.  
The common **claim** method may also be used to claim the entire Hard Totals device.
  - **releaseFile**: Release exclusive access to the file.
- The **FreeData** property holds the current number of unassigned data bytes.
- The **TotalsSize** property holds the totals memory size.
- The **NumberOfFiles** property holds the number of totals files that exist in the hard totals device.

- Transaction operations are optionally supported. A transaction is defined as a series of data writes to be applied as an atomic operation to one or more Hard Totals files.

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

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

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

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

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

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

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

When a validation error is reported, it is recommended that the application read all of the data in the totals file. The application will want to determine which portions of data are invalid, and take action based on the results of the reads.

- **recalculateValidationData** may be called to cause recalculation of all validation data within a totals file. This may be called after recovery has been performed as in the previous paragraph.

- **validateData** may be called to verify that all data within a totals file passes validation.
- Data **writes** automatically cause recalculation of validation data for the validation block or blocks in which the written data resides.
- Since advanced error detection usually imposes a performance penalty, the application may choose to select this feature when each totals file is created.

## Device Sharing

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

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



## Properties

### CapErrorDetection Property R

Type	<b>boolean</b>
Remarks	If true, then advanced error detection is supported.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapSingleFile Property R

Type	<b>boolean</b>
Remarks	If true, then only a single file, identified by the empty string (“”), is supported.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapTransactions Property R

Type	<b>boolean</b>
Remarks	If true, then transactions are supported.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

## FreeData Property R

Type	<b>int</b>
Remarks	<p>Holds the number of bytes of unallocated data in the Hard Totals device.</p> <p>It is initialized to an appropriate value when the device is enabled and is updated as files are <b>created</b> and <b>deleted</b>. If creating a file requires some overhead to support the file information, then this overhead is not included in what is reported by this property. This guarantees that a new file of size <b>FreeData</b> may be created.</p> <p>Data writes within a transaction may temporarily reduce what's reported by this property, since some Hard Totals space may need to be allocated to prepare for the transaction commit. Therefore, the application should ensure that sufficient <b>FreeData</b> is maintained to allow its maximally sized transactions to be performed.</p>
Errors	A JposException may be thrown when this property is accessed. For further information, see "Exceptions" on page 15.
See Also	<b>create</b> Method, <b>write</b> Method

## NumberOfFiles Property R

Type	<b>int</b>
Remarks	<p>Holds the number of totals file currently in the Hard Totals device.</p> <p>This property is initialized and kept current while the device is enabled.</p>
Errors	A JposException may be thrown when this property is accessed. For further information, see "Exceptions" on page 15.
See Also	<b>FreeData</b> Property

## TotalsSize Property R

Type	<b>int</b>
Remarks	<p>Holds the size of the Hard Totals area. This size is equal to the largest totals file that can be created if no other files exist.</p> <p>This property is initialized when the device is enabled.</p>
Errors	A JposException may be thrown when this property is accessed. For further information, see "Exceptions" on page 15.
See Also	<b>FreeData</b> Property

## TransactionInProgress Property R

<b>Type</b>	<b>boolean</b>
<b>Remarks</b>	If true, then the application is within a transaction.  This property is initialized to false by the <b>open</b> method.
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
<b>See Also</b>	<b>beginTrans</b> Method

## Methods

### beginTrans Method

<b>Syntax</b>	<b>void beginTrans () throws JposException;</b>				
<b>Remarks</b>	Marks the beginning of a series of Hard Totals writes that must either be applied as a group or not at all.				
<b>Errors</b>	A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.  Some possible values of the exception’s <i>ErrorCode</i> property are:				
	<table> <thead> <tr> <th><b>Value</b></th> <th><b>Meaning</b></th> </tr> </thead> <tbody> <tr> <td>JPOS_E_ILLEGAL</td> <td>Transactions are not supported by this device.</td> </tr> </tbody> </table>	<b>Value</b>	<b>Meaning</b>	JPOS_E_ILLEGAL	Transactions are not supported by this device.
<b>Value</b>	<b>Meaning</b>				
JPOS_E_ILLEGAL	Transactions are not supported by this device.				
<b>See Also</b>	<b>commitTrans</b> Method, <b>rollback</b> Method				

### claim Method (Common)

<b>Syntax</b>	<b>void claim (int <i>timeout</i>) throws JposException;</b>						
	The <i>timeout</i> parameter gives the maximum number of milliseconds to wait for exclusive access to be satisfied. If zero, the method attempts to claim the device, then returns the appropriate status immediately. If JPOS_FOREVER (-1), the method waits as long as needed until exclusive access is satisfied.						
<b>Remarks</b>	Requests exclusive access to the device.  If any other application has claimed exclusive access to any of the hard totals files by using <b>claimFile</b> , then this <b>claim</b> cannot be satisfied until those files are released by <b>releaseFile</b> .  When successful, the <b>claimed</b> property is changed to true.						
<b>Errors</b>	A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.  Some possible values of the exception’s <i>ErrorCode</i> property are:						
	<table> <thead> <tr> <th><b>Value</b></th> <th><b>Meaning</b></th> </tr> </thead> <tbody> <tr> <td>JPOS_E_ILLEGAL</td> <td>An invalid <i>timeout</i> parameter was specified.</td> </tr> <tr> <td>JPOS_E_TIMEOUT</td> <td>Another application has exclusive access to the device or one or more of its files and did not relinquish control before <i>timeout</i> milliseconds expired.</td> </tr> </tbody> </table>	<b>Value</b>	<b>Meaning</b>	JPOS_E_ILLEGAL	An invalid <i>timeout</i> parameter was specified.	JPOS_E_TIMEOUT	Another application has exclusive access to the device or one or more of its files and did not relinquish control before <i>timeout</i> milliseconds expired.
<b>Value</b>	<b>Meaning</b>						
JPOS_E_ILLEGAL	An invalid <i>timeout</i> parameter was specified.						
JPOS_E_TIMEOUT	Another application has exclusive access to the device or one or more of its files and did not relinquish control before <i>timeout</i> milliseconds expired.						
<b>See Also</b>	“Device Sharing Model” on page 12, <b>release</b> Method, <b>claimFile</b> Method, <b>releaseFile</b> Method						

**claimFile Method**

**Syntax**      **void claimFile (int hTotalsFile, int timeout) throws JposException;**

<b>Parameter</b>	<b>Description</b>
<i>hTotalsFile</i>	Handle to the totals file that is to be claimed.
<i>timeout</i>	The time in milliseconds to wait for the file to become available. If zero, the method attempts to claim the file, then returns the appropriate status immediately. If JPOS_FOREVER (-1), the method waits as long as needed until exclusive access is satisfied.

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

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

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

**Errors**      A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
JPOS_E_ILLEGAL	The handle is invalid, or an invalid <i>timeout</i> parameter was specified.
JPOS_E_TIMEOUT	The <i>timeout</i> value expired before another application released exclusive access of either the requested totals file or the entire totals area.

**See Also**      **claim** Method, **releaseFile** Method

**commitTrans Method**

- Syntax**      **void commitTrans () throws JposException;**
- Remarks**      Ends the current transaction. All writes between the previous **beginTrans** method and this method are saved to the Hard Totals areas.
- Errors**          A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_ILLEGAL	Transactions are not supported by this device, or no transaction is in progress.

- See Also**      **beginTrans** Method, **rollback** Method

**create Method**

- Syntax**      **void create (String fileName, int[] hTotalsFile, int size, boolean errorDetection) throws JposException;**

Parameter	Description
<i>fileName</i>	The name to be assigned to the file. Must be no longer than 10 characters. All displayable ASCII characters (0x20 through 0x7F) are valid.
<i>hTotalsFile</i>	Handle of the newly created totals file. Set by the method.
<i>size</i>	The byte array size for the data. Once created, the array size and therefore the file size used to store the array cannot be changed – totals files are fixed-length files.
<i>errorDetection</i>	The level of error detection desired for this file: If true, then the Device Service will enable advanced error detection if supported. If false, then higher performance access is required, so advanced error detection need not be enabled for this file.

- Remarks**      Creates a totals file with the specified name, size, and error detection level. The data area is initialized to binary zeros.
- If **CapSingleFile** is true, then only one file may be created, and its name must be the empty string (“”). Otherwise, the number of totals files that may be created is limited only by the free space available in the Hard Totals area.

**Errors** A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_CLAIMED	Cannot create because the entire totals file area is claimed by another application.
JPOS_E_ILLEGAL	The <i>fileName</i> is too long or contains invalid characters.
JPOS_E_EXISTS	<i>fileName</i> already exists.
JPOS_E_EXTENDED	<i>ErrorCodeExtended</i> = JPOS_ETOT_NOROOM: There is insufficient room in the totals area to create the file.

**See Also** **find** Method, **delete** Method, **rename** Method

## delete Method

**Syntax** **void delete (String *fileName*) throws JposException;**

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

**Remarks** Deletes the named file.

**Errors** A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_CLAIMED	Cannot delete because either the totals file or the entire totals area is claimed by another application.
JPOS_E_ILLEGAL	The <i>fileName</i> is too long or contains invalid characters.
JPOS_E_NOEXIST	<i>fileName</i> was not found.

**See Also** **create** Method, **find** Method, **rename** Method

**find Method**

**Syntax**      **void find (String *fileName*, int[] *hTotalsFile*, int[] *size*) throws JposException;**

Parameter	Description
<i>fileName</i>	The totals file name to be located.
<i>hTotalsFile</i>	Handle of the totals file. Set by the method.
<i>size</i>	The length of the file in bytes. Set by the method.

**Remarks**      Locates an existing totals file.

**Errors**          A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_CLAIMED	Cannot find because the entire totals file area is claimed by another application.
JPOS_E_ILLEGAL	The <i>fileName</i> contains invalid characters.
JPOS_E_NOEXIST	<i>fileName</i> was not found.

**See Also**      **create Method, delete Method, rename Method**

**findByIndex Method**

**Syntax**      **void findByIndex (int *index*, String[] *fileName*) throws JposException;**

Parameter	Description
<i>index</i>	The index of the totals file name to be found.
<i>fileName</i>	The file name associated with <i>index</i> . Set by the method.

**Remarks**      Determines the totals file name currently associated with the given index.

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

The creation and deletion of files may change the relationship between indices and the file names; the data areas used to manage file names and attributes may be compacted or rearranged as a result. Therefore, the application may need to **claim** the device to ensure that all file names are retrieved successfully.



**Errors** A `JposException` may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s `ErrorCode` property are:

Value	Meaning
<code>JPOS_E_CLAIMED</code>	Cannot find because the entire totals file area is claimed by another application.
<code>JPOS_E_ILLEGAL</code>	The <i>index</i> is greater than the largest file index that is currently defined.

**See Also** `create` Method, `find` Method

## read Method

**Syntax** `void read (int hTotalsFile, byte[] data, int offset, int count) throws JposException;`

Parameter	Description
<i>hTotalsFile</i>	Totals file handle returned from a <code>create</code> or <code>find</code> method.
<i>data</i>	The data buffer in which the totals data will be placed. Array length must be at least <i>count</i> .
<i>offset</i>	Starting offset for the data to be read.
<i>count</i>	Number of bytes of data to read.

**Remarks** Reads data from a totals file.

**Errors** A `JposException` may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s `ErrorCode` property are:

Value	Meaning
<code>JPOS_E_CLAIMED</code>	Cannot read because either the totals file or the entire totals area is claimed by another application.
<code>JPOS_E_ILLEGAL</code>	The handle is invalid, part of the data range is outside the bounds of the totals file, or <i>data</i> array length is less than <i>count</i> .
<code>JPOS_E_EXTENDED</code>	<code>ErrorCodeExtended = JPOS_ETOT_VALIDATION</code> : A validation error has occurred while reading data.

**See Also** `write` Method

**recalculateValidationData Method**

**Syntax**      **void recalculateValidationData (int hTotalsFile) throws JposException;**

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

**Remarks**      Recalculates validation data for the specified totals file.

**Errors**        A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
JPOS_E_CLAIMED	Cannot recalculate because either the totals file or the entire totals area is claimed by another application.
JPOS_E_ILLEGAL	The handle is invalid, or advanced error detection is either not supported by the Device Service or by this file.

**release Method (Common)**

**Syntax**      **void release () throws JposException;**

**Remarks**      Releases exclusive access to the device.

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

**Errors**        A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
JPOS_E_ILLEGAL	The application does not have exclusive access to the device.

**See Also**      “Device Sharing Model” on page 12, **claim** Method, **claimFile** Method

**releaseFile Method**

**Syntax**      **void releaseFile (int *hTotalsFile*) throws JposException;**  
 The *hTotalsFile* parameter contains the handle of the totals file to be released.

**Remarks**      Releases exclusive access to a specific file.

**Errors**        A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
JPOS_E_ILLEGAL	The handle is invalid, or the specified file is not claimed by this application.

**See Also**      **claim** Method, **claimFile** Method

**rename Method**

**Syntax**      **void rename (int *hTotalsFile*, String *fileName*) throws JposException;**

<b>Parameter</b>	<b>Description</b>
<i>hTotalsFile</i>	The handle of the totals file to be renamed.
<i>fileName</i>	The new name to be assigned to the file. Must be no longer than 10 characters. All displayable ASCII characters (0x20 through 0x7F) are valid.

**Remarks**      Renames a totals file.  
 If **CapSingleFile** is true, then this method will fail.

**Errors**        A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
JPOS_E_CLAIMED	Cannot rename because either the totals file or the entire totals area is claimed by another application.
JPOS_E_ILLEGAL	The handle is invalid, the <i>fileName</i> contains invalid characters, or the <b>CapSingleFile</b> property is true.
JPOS_E_EXISTS	<i>fileName</i> already exists.

**rollback Method**

**Syntax**      **void rollback () throws JposException;**

**Remarks**      Ends the current transaction. All writes between the previous **beginTrans** and this method are discarded; they are not saved to the Hard Totals areas.

**Errors**      A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
JPOS_E_ILLEGAL	Transactions are not supported by this device, or no transaction is in progress.

**See Also**      **beginTrans** Method, **commitTrans** Method

**setAll Method**

**Syntax**      **void setAll (int hTotalsFile, byte value) throws JposException;**

<b>Parameter</b>	<b>Description</b>
<i>hTotalsFile</i>	Handle of a totals file.
<i>value</i>	Value to set all locations to in totals file.

**Remarks**      Sets all the data in a totals file to the specified value.

**Errors**      A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
JPOS_E_CLAIMED	Cannot set because either the totals file or the entire totals area is claimed by another application.

## validateData Method

**Syntax**      **void validateData (int *hTotalsFile*) throws JposException;**

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

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

**Errors**          A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_CLAIMED	Cannot validate because either the totals file or the entire totals area is claimed by another application.
JPOS_E_ILLEGAL	The handle is invalid, or advanced error detection is either not supported by the Device Service or by this file.

## write Method

**Syntax**      **void write (int *hTotalsFile*, byte[] *data*, int *offset*, int *count*) throws JposException;**

Parameter	Description
<i>hTotalsFile</i>	Totals file handle returned from a <b>create</b> or <b>find</b> method.
<i>data</i>	Data buffer containing the totals data to be written.
<i>offset</i>	Starting offset for the data to be written.
<i>count</i>	Number of bytes of <i>data</i> to write.

**Remarks**      Writes data to a totals file.

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

**Errors**          A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_CLAIMED	Cannot write because either the totals file or the entire totals area is claimed by another application.
JPOS_E_ILLEGAL	The handle is invalid, or part of all of the data range is outside the bounds of the totals file.

JPOS\_E\_EXTENDED    *ErrorCodeExtended* = JPOS\_ETOT\_NOROOM:  
Cannot write because a transaction is in progress, and  
there is not enough free space to prepare for the  
transaction commit.

*ErrorCodeExtended* = JPOS\_ETOT\_VALIDATION:  
A validation error has occurred while reading data.

**See Also**    **read** Method, **beginTrans** Method, **commitTrans** Method, **rollback** Method,  
**FreeData** Property

## Events

### DirectIOEvent

**Interface** `jpos.events.DirectIOListener`

**Method** `directIOOccurred (DirectIOEvent e);`

**Description** Provides Device Service information directly to the application. This event provides a means for a vendor-specific Hard Totals Device Service to provide events to the application that are not otherwise supported by the Device Control.

**Properties** This event contains the following properties:

Property	Type	Description
<i>EventNumber</i>	<i>int</i>	Event number whose specific values are assigned by the Device Service.
<i>Data</i>	<i>int</i>	Additional numeric data. Specific values vary by the <i>EventNumber</i> and the Device Service. This property is settable.
<i>Object</i>	<i>Object</i>	Additional data whose usage varies by the <i>EventNumber</i> and Device Service. This property is settable.

**Remarks** This event is to be used only for those types of vendor specific functions that are not otherwise described as part of the JavaPOS standard. Use of this event may restrict the application program from being used with other vendor's Hard Totals devices which may not have any knowledge of the Device Service's need for this event.

**See Also** "Events" on page 18, **directIO** Method

## StatusUpdateEvent

**Interface**    `jpos.events.StatusUpdateListener`

**Method**      `statusUpdateOccurred (StatusUpdateEvent e);`

**Description**    Notifies the application that there is a change in the power status of a Hard Totals device.

**Properties**     This event contains the following property:

<b>Property</b>	<b>Type</b>	<b>Description</b>
<i>Status</i>	<i>int</i>	Reports a change in the power state of a Hard Totals device.

*Note that Release 1.3* added Power State Reporting with additional *Power reporting StatusUpdateEvent values*. See “StatusUpdateEvent” description on page 78.

**Remarks**      Enqueued when the Hard Totals device detects a power state change.

**See Also**      “Events” on page 18



# Keylock

## Summary

### Properties

<i>Common</i>	<i>Ver</i>	<i>Type</i>	<i>Access</i>	<i>May Use After</i>
<b>AutoDisable</b>		boolean	R/W	<i>Not Supported</i>
<b>CapPowerReporting</b>	1.3	int	R	open
<b>CheckHealthText</b>		String	R	open
<b>Claimed</b>		boolean	R	open
<b>DataCount</b>		int	R	<i>Not Supported</i>
<b>DataEventEnabled</b>		boolean	R/W	<i>Not Supported</i>
<b>DeviceEnabled</b>		boolean	R/W	open
<b>FreezeEvents</b>		boolean	R/W	open
<b>OutputID</b>		int	R	<i>Not Supported</i>
<b>PowerNotify</b>	1.3	int	R/W	open
<b>PowerState</b>	1.3	int	R	open
<b>State</b>		int	R	--
<b>DeviceControlDescription</b>		String	R	--
<b>DeviceControlVersion</b>		int	R	--
<b>DeviceServiceDescription</b>		String	R	open
<b>DeviceServiceVersion</b>		int	R	open
<b>PhysicalDeviceDescription</b>		String	R	open
<b>PhysicalDeviceName</b>		String	R	open

### Properties

<i>Specific</i>	<i>Ver</i>	<i>Type</i>	<i>Access</i>	<i>May Use After</i>
<b>KeyPosition</b>		int	R	open & enable
<b>PositionCount</b>		int	R	open

**Methods***Common*

	<i>Ver</i>	<i>May Use After</i>
<b>open</b>		--
<b>close</b>		open
<b>claim</b>		open
<b>release</b>		open & claim
<b>checkHealth</b>		open & enable
<b>clearInput</b>		<i>Not Supported</i>
<b>clearOutput</b>		<i>Not Supported</i>
<b>directIO</b>		open

*Specific*

<b>waitForKeylockChange</b>		open & enable
-----------------------------	--	---------------

**Events**

<i>Name</i>	<i>Ver</i>	<i>May Occur After</i>
<b>DataEvent</b>		<i>Not Supported</i>
<b>DirectIOEvent</b>	1.3	open & claim
<b>ErrorEvent</b>		<i>Not Supported</i>
<b>OutputCompleteEvent</b>		<i>Not Supported</i>
<b>StatusUpdateEvent</b>		open & enable

## General Information

The Key Lock Control's class name is "jpos.Keylock".

The device constants are contained in the class "jpos.KeylockConst".

See "Package Structure" on page 40.

### Capabilities

The keylock has the following minimal set of capabilities:

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

### Model

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

- LOCK\_KP\_LOCK            1
- LOCK\_KP\_NORM           2
- LOCK\_KP\_SUPR            3

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

### Device Sharing

The keylock is a sharable device. Its device sharing rules are:

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

## Properties

### KeyPosition Property R

**Type**      **int**

**Remarks**      Holds a value which indicates the keylock position.

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

This property has one of the following values:

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

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

**Errors**      A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### PositionCount Property R

**Syntax**      **int**

**Remarks**      Holds the total number of keylock positions that are present on the keylock device.

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

**Errors**      A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.



## Events

### DirectIOEvent

**Interface** `jpos.events.DirectIOListener`

**Method** `directIOOccurred (DirectIOEvent e);`

**Description** Provides Device Service information directly to the application. This event provides a means for a vendor-specific Keylock Device Service to provide events to the application that are not otherwise supported by the Device Control.

**Properties** This event contains the following properties:

Property	Type	Description
<i>EventNumber</i>	<i>int</i>	Event number whose specific values are assigned by the Device Service.
<i>Data</i>	<i>int</i>	Additional numeric data. Specific values vary by the <i>EventNumber</i> and the Device Service. This property is settable.
<i>Object</i>	<i>Object</i>	Additional data whose usage varies by the <i>EventNumber</i> and Device Service. This property is settable.

**Remarks** This event is to be used only for those types of vendor specific functions that are not otherwise described as part of the JavaPOS standard. Use of this event may restrict the application program from being used with other vendor's Keylock devices which may not have any knowledge of the Device Service's need for this event.

**See Also** "Events" on page 18, **directIO** Method

## StatusUpdateEvent

**Interface** `jpos.events.StatusUpdateListener`

**Method** `statusUpdateOccurred (StatusUpdateEvent e);`

**Description** Notifies the application when the keylock position changes.

**Properties** This event contains the following property:

Property	Type	Description
<i>Status</i>	<i>int</i>	The key position in the Keylock.

The *Status* property has one of the following values:

Value	Description
LOCK_KP_LOCK	Keylock is in the “locked” position. Value is one (1).
LOCK_KP_NORM	Keylock is in the “normal” position. Value is two (2).
LOCK_KP_SUPR	Keylock is in the “supervisor” position. Value is three (3).

*Other Values* Keylock is in one of the auxiliary positions. This value may range from four (4) to the total number of keylock positions indicated by the **PositionCount** property.

*Note that Release 1.3* added Power State Reporting with additional *Power reporting StatusUpdateEvent values*. See “StatusUpdateEvent” description on page 78.

**Remarks** This event is enqueued when a keylock switch position undergoes a change or if Power State Reporting is enabled and a change in the power state is detected.

**See Also** “Events” on page 18





# Line Display

## Summary

### Properties

<i>Common</i>	<i>Ver</i>	<i>Type</i>	<i>Access</i>	<i>May Use After</i>
<b>AutoDisable</b>		boolean	R/W	<i>Not Supported</i>
<b>CapPowerReporting</b>	1.3	int	R	open
<b>CheckHealthText</b>		String	R	open
<b>Claimed</b>		boolean	R	open
<b>DataCount</b>		int	R	<i>Not Supported</i>
<b>DataEventEnabled</b>		boolean	R/W	<i>Not Supported</i>
<b>DeviceEnabled</b>		boolean	R/W	open & claim
<b>FreezeEvents</b>		boolean	R/W	open
<b>OutputID</b>		int	R	<i>Not Supported</i>
<b>PowerNotify</b>	1.3	int	R/W	open
<b>PowerState</b>	1.3	int	R	open
<b>State</b>		int	R	--
<b>DeviceControlDescription</b>		String	R	--
<b>DeviceControlVersion</b>		int	R	--
<b>DeviceServiceDescription</b>		String	R	open
<b>DeviceServiceVersion</b>		int	R	open
<b>PhysicalDeviceDescription</b>		String	R	open
<b>PhysicalDeviceName</b>		String	R	open

<i>Specific</i>	<i>Ver</i>	<i>Type</i>	<i>Access</i>	<i>May Use After</i>
<b>CapBlink</b>		int	R	open
<b>CapBrightness</b>		boolean	R	open
<b>CapCharacterSet</b>		int	R	open
<b>CapDescriptors</b>		boolean	R	open
<b>CapHMarquee</b>		boolean	R	open
<b>CapICharWait</b>		boolean	R	open
<b>CapVMarquee</b>		boolean	R	open
<b>DeviceWindows</b>		int	R	open
<b>DeviceRows</b>		int	R	open
<b>DeviceColumns</b>		int	R	open
<b>DeviceDescriptors</b>		int	R	open
<b>DeviceBrightness</b>		int	R/W	open, claim, & enable
<b>CharacterSet</b>		int	R/W	open, claim, & enable
<b>CharacterSetList</b>		String	R	open
<b>CurrentWindow</b>		int	R/W	open
<b>Rows</b>		int	R	open
<b>Columns</b>		int	R	open
<b>CursorRow</b>		int	R/W	open
<b>CursorColumn</b>		int	R/W	open
<b>CursorUpdate</b>		boolean	R/W	open
<b>MarqueeType</b>		int	R/W	open
<b>MarqueeFormat</b>		int	R/W	open
<b>MarqueeUnitWait</b>		int	R/W	open
<b>MarqueeRepeatWait</b>		int	R/W	open
<b>InterCharacterWait</b>		int	R/W	open

**Methods**

<i>Common</i>	<i>Ver</i>	<i>May Use After</i>
<b>open</b>		--
<b>close</b>		open
<b>claim</b>		open
<b>release</b>		open & claim
<b>checkHealth</b>		open, claim, & enable
<b>clearInput</b>		<i>Not Supported</i>
<b>clearOutput</b>		<i>Not Supported</i>
<b>directIO</b>		open
 <i>Specific</i>		
<b>displayText</b>		open, claim, & enable
<b>displayTextAt</b>		open, claim, & enable
<b>clearText</b>		open, claim, & enable
<b>scrollText</b>		open, claim, & enable
 <b>setDescription</b>		 open, claim, & enable
<b>clearDescriptors</b>		open, claim, & enable
 <b>createWindow</b>		 open, claim, & enable
<b>destroyWindow</b>		open, claim, & enable
<b>refreshWindow</b>		open, claim, & enable

**Events**

<i>Name</i>	<i>Ver</i>	<i>May Occur After</i>
<b>DataEvent</b>		<i>Not Supported</i>
<b>DirectIOEvent</b>	1.3	open & claim
<b>ErrorEvent</b>		<i>Not Supported</i>
<b>OutputCompleteEvent</b>		<i>Not Supported</i>
<b>StatusUpdateEvent</b>	1.3	open, claim, & enable

## General Information

The Line Display Control's class name is "jpos.LineDisplay".

The device constants are contained in the class "jpos.LineDisplayConst".

See "Package Structure" on page 40.

### Capabilities

The Line Display has the following capability:

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

The line display may also have the following additional capabilities:

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

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

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

### Model

The general model of a line display consists of:

- One or more rows containing one or more columns of characters. The rows and columns are numbered beginning with (0, 0) at the upper-left corner of the window. The characters in the default character set will include at least one of the following, with a capability defining the character set:
  - The digits '0' through '9' plus space, minus ('-'), and period ('.').
  - The above set plus uppercase 'A' through 'Z.'
  - All ASCII characters from 0x20 through 0x7F, which includes space, digits, uppercase, lowercase, and some special characters.

- Window 0, which is always defined as follows:
  - Its “viewport” — the portion of the display that is updated by the window — covers the entire display.
  - The size of the window matches the entire display.

Therefore, window 0, which is also called the “device window,” maps directly onto the display.

- Option to create additional windows. A created window has the following characteristics:
  - Its viewport covers part or all of the display.
  - The window may either match the size of the viewport, or it may be larger than the viewport in either the horizontal or vertical direction. In the second case, marquee scrolling of the window can be set.
  - The window maintains its own values for rows and columns, current cursor row and column, cursor update flag, scroll type and format, and timers.
  - All viewports behave transparently. If two viewports overlap, then the last character displayed at a position by either of the windows will be visible.

## Display Modes

- **Immediate Mode**  
In effect when **MarqueeType** is DISP\_MT\_NONE and **InterCharacterWait** is zero.

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

- **Teletype Mode**  
In effect when **MarqueeType** is DISP\_MT\_NONE and **InterCharacterWait** is not zero.

Calls to **displayText** and **displayTextAt** are enqueued and processed in the order they are received. **InterCharacterWait** specifies the time to wait between outputting each character. **InterCharacterWait** only applies to those characters within the viewport.

- **Marquee Mode**  
In effect when **MarqueeType** is not DISP\_MT\_NONE.

The window must be bigger than the viewport.

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

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

When the mode is changed from **Marquee On Mode** to **Marquee Init Mode**,

the marquee stops. Changes may be made to the window, then the window may be returned to *Marquee On Mode* to restart the marquee with the new data.

It is illegal to use **displayText**, **displayTextAt**, **clearText**, **refreshWindow** and **scrollText** unless in *Marquee Init Mode* or *Marquee Off Mode*.

## Device Sharing

The line display is an exclusive-use device, as follows:

- The application must claim the device before enabling it.
- The application must claim and enable the device before accessing some properties or calling methods that update the device.
- See the “Summary” table for precise usage prerequisites.

## Properties

### CapBlink Property R

<b>Type</b>	<b>int</b>								
<b>Remarks</b>	Holds the character blink capability of the device. It has one of the following values:								
	<table> <thead> <tr> <th><b>Value</b></th> <th><b>Meaning</b></th> </tr> </thead> <tbody> <tr> <td>DISP_CB_NOBLINK</td> <td>Blinking is not supported. Value is 0.</td> </tr> <tr> <td>DISP_CB_BLINKALL</td> <td>Blinking is supported. The entire contents of the display are either blinking or in a steady state.</td> </tr> <tr> <td>DISP_CB_BLINKEACH</td> <td>Blinking is supported. Each character may be individually set to blink or to be in a steady state.</td> </tr> </tbody> </table>	<b>Value</b>	<b>Meaning</b>	DISP_CB_NOBLINK	Blinking is not supported. Value is 0.	DISP_CB_BLINKALL	Blinking is supported. The entire contents of the display are either blinking or in a steady state.	DISP_CB_BLINKEACH	Blinking is supported. Each character may be individually set to blink or to be in a steady state.
<b>Value</b>	<b>Meaning</b>								
DISP_CB_NOBLINK	Blinking is not supported. Value is 0.								
DISP_CB_BLINKALL	Blinking is supported. The entire contents of the display are either blinking or in a steady state.								
DISP_CB_BLINKEACH	Blinking is supported. Each character may be individually set to blink or to be in a steady state.								
	This property is initialized by the <b>open</b> method.								
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.								

### CapBrightness Property R

<b>Type</b>	<b>boolean</b>
<b>Remarks</b>	If true, then the brightness control is supported.
	This property is initialized by the <b>open</b> method.
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

**CapCharacterSet Property R**

<b>Type</b>	<b>int</b>														
<b>Remarks</b>	<p>Holds the default character set capability. It has one of the following values:</p> <table border="1"> <thead> <tr> <th><b>Value</b></th> <th><b>Meaning</b></th> </tr> </thead> <tbody> <tr> <td>DISP_CCS_NUMERIC</td> <td>The default character set supports numeric data, plus space, minus, and period.</td> </tr> <tr> <td>DISP_CCS_ALPHA</td> <td>The default character set supports uppercase alphabetic plus numeric, space, minus, and period.</td> </tr> <tr> <td>DISP_CCS_ASCII</td> <td>The default character set supports all ASCII characters 0x20 through 0x7F.</td> </tr> <tr> <td>DISP_CCS_KANA</td> <td>The default character set supports partial code page 932, including ASCII characters 0x20 through 0x7F and the Japanese Kana characters 0xA1 through 0xDF, but excluding the Japanese Kanji characters.</td> </tr> <tr> <td>DISP_CCS_KANJI</td> <td>The default character set supports code page 932, including the Shift-JIS Kanji characters, Levels 1 and 2.</td> </tr> <tr> <td>DISP_CCS_UNICODE</td> <td>The default character set supports UNICODE.</td> </tr> </tbody> </table> <p>The default character set may contain a superset of these ranges. The initial <b>CharacterSet</b> property may be examined for additional information.</p> <p>This property is initialized by the <b>open</b> method.</p>	<b>Value</b>	<b>Meaning</b>	DISP_CCS_NUMERIC	The default character set supports numeric data, plus space, minus, and period.	DISP_CCS_ALPHA	The default character set supports uppercase alphabetic plus numeric, space, minus, and period.	DISP_CCS_ASCII	The default character set supports all ASCII characters 0x20 through 0x7F.	DISP_CCS_KANA	The default character set supports partial code page 932, including ASCII characters 0x20 through 0x7F and the Japanese Kana characters 0xA1 through 0xDF, but excluding the Japanese Kanji characters.	DISP_CCS_KANJI	The default character set supports code page 932, including the Shift-JIS Kanji characters, Levels 1 and 2.	DISP_CCS_UNICODE	The default character set supports UNICODE.
<b>Value</b>	<b>Meaning</b>														
DISP_CCS_NUMERIC	The default character set supports numeric data, plus space, minus, and period.														
DISP_CCS_ALPHA	The default character set supports uppercase alphabetic plus numeric, space, minus, and period.														
DISP_CCS_ASCII	The default character set supports all ASCII characters 0x20 through 0x7F.														
DISP_CCS_KANA	The default character set supports partial code page 932, including ASCII characters 0x20 through 0x7F and the Japanese Kana characters 0xA1 through 0xDF, but excluding the Japanese Kanji characters.														
DISP_CCS_KANJI	The default character set supports code page 932, including the Shift-JIS Kanji characters, Levels 1 and 2.														
DISP_CCS_UNICODE	The default character set supports UNICODE.														
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.														

**CapDescriptors Property R**

<b>Type</b>	<b>boolean</b>
<b>Remarks</b>	<p>If true, then the display supports descriptors.</p> <p>This property is initialized by the <b>open</b> method.</p>
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.



### CapHMarquee Property R

Type	<b>boolean</b>
Remarks	If true, the display supports horizontal marquee windows. This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapICharWait Property R

Type	<b>boolean</b>
Remarks	If true, the display supports intercharacter wait. This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapVMarquee Property R

Type	<b>boolean</b>
Remarks	If true, the display supports vertical marquee windows. This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

**CharacterSet Property R/W**

<b>Type</b>	<b>int</b>												
<b>Remarks</b>	Contains the character set for displaying characters. It has one of the following values:												
	<table> <thead> <tr> <th><b>Value</b></th> <th><b>Meaning</b></th> </tr> </thead> <tbody> <tr> <td>Range 101 - 199</td> <td>A device-specific character set that does not match a code page, nor the ASCII or ANSI character sets.</td> </tr> <tr> <td>Range 400 - 990</td> <td>Code page; matches one of the standard values.</td> </tr> <tr> <td>DISP_CS_UNICODE</td> <td>The character set supports UNICODE. The value of this constant is 997.</td> </tr> <tr> <td>DISP_CS_ASCII</td> <td>The ASCII character set, supporting the ASCII characters 0x20 through 0x7F. The value of this constant is 998.</td> </tr> <tr> <td>DISP_CS_ANSI</td> <td>The ANSI character set. The value of this constant is 999.</td> </tr> </tbody> </table>	<b>Value</b>	<b>Meaning</b>	Range 101 - 199	A device-specific character set that does not match a code page, nor the ASCII or ANSI character sets.	Range 400 - 990	Code page; matches one of the standard values.	DISP_CS_UNICODE	The character set supports UNICODE. The value of this constant is 997.	DISP_CS_ASCII	The ASCII character set, supporting the ASCII characters 0x20 through 0x7F. The value of this constant is 998.	DISP_CS_ANSI	The ANSI character set. The value of this constant is 999.
<b>Value</b>	<b>Meaning</b>												
Range 101 - 199	A device-specific character set that does not match a code page, nor the ASCII or ANSI character sets.												
Range 400 - 990	Code page; matches one of the standard values.												
DISP_CS_UNICODE	The character set supports UNICODE. The value of this constant is 997.												
DISP_CS_ASCII	The ASCII character set, supporting the ASCII characters 0x20 through 0x7F. The value of this constant is 998.												
DISP_CS_ANSI	The ANSI character set. The value of this constant is 999.												
	<p>This property is initialized to an appropriate value when the device is first enabled following the <b>open</b> method. This value is guaranteed to support at least the set of characters specified by <b>CapCharacterSet</b>.</p>												
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.												
<b>See Also</b>	<b>CharacterSetList</b> Property, <b>CapCharacterSet</b> Property												

**CharacterSetList Property R**

<b>Type</b>	<b>String</b>
<b>Remarks</b>	<p>Holds the character set numbers supported. It consists of ASCII numeric set numbers separated by commas.</p> <p>For example, if the string is “101,850,999”, then the device supports a device-specific character set, code page 850, and the ANSI character set.</p> <p>This property is initialized by the <b>open</b> method.</p>
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
<b>See Also</b>	<b>CharacterSet</b> Property

## Columns Property R

Type	int
Remarks	<p>Holds the number of columns for this window.</p> <p>For window 0, this property is the same as <b>DeviceColumns</b>. For other windows, it may be less or greater than <b>DeviceColumns</b>.</p> <p>This property is initialized to <b>DeviceColumns</b> by the <b>open</b> method, and is updated when <b>CurrentWindow</b> is set and when <b>createWindow</b> or <b>destroyWindow</b> are called.</p>
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>Rows</b> Property

## CurrentWindow Property R/W

Type	int
Remarks	<p>Holds the current window to which text is displayed.</p> <p>Several properties are associated with each window: <b>Rows</b>, <b>Columns</b>, <b>CursorRow</b>, <b>CursorColumn</b>, <b>CursorUpdate</b>, <b>MarqueeType</b>, <b>MarqueeUnitWait</b>, <b>MarqueeRepeatWait</b>, and <b>InterCharacterWait</b>.</p> <p>When set, this property changes the current window and sets the associated properties to their values for this window.</p> <p>Setting a window does not refresh its viewport. If this window and another window’s viewports overlap, and the other window has changed the viewport, then <b>refreshWindow</b> may be called to restore this window’s viewport contents.</p> <p>This property is initialized to zero – the device window – by the <b>open</b> method, and is updated when <b>createWindow</b> or <b>destroyWindow</b> are called.</p>
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CursorPosition Property R/W

Type	int
Remarks	<p>Holds the column in the current window to which the next displayed character will be output.</p> <p>Legal values range from zero through <b>Columns</b>. (See <b>displayText</b> for a note on the interpretation of <b>CursorPosition = Columns</b>.)</p> <p>This property is initialized to zero by the <b>open</b> and <b>createWindow</b> methods, and is updated when <b>CurrentWindow</b> is set or <b>clearText</b>, <b>displayTextAt</b> or <b>destroyWindow</b> is called. It is also updated when <b>displayText</b> is called if <b>CursorPosition</b> is true.</p>
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>CursorPosition</b> Property, <b>displayText</b> Method

### CursorPosition Property R/W

Type	int
Remarks	<p>Holds the row in the current window to which the next displayed character will be output.</p> <p>Legal values range from zero through one less than <b>Rows</b>.</p> <p>This property is initialized to zero by the <b>open</b> and <b>createWindow</b> methods, and is updated when <b>CurrentWindow</b> is set or <b>clearText</b>, <b>displayTextAt</b> or <b>destroyWindow</b> is called. It is also updated when <b>displayText</b> is called if <b>CursorPosition</b> is true.</p>
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>CursorPosition</b> Property, <b>displayText</b> Method

## CursorUpdate Property R/W

<b>Type</b>	<b>boolean</b>
<b>Remarks</b>	<p>When true, <b>CursorRow</b> and <b>CursorColumn</b> will be updated to point to the character beyond the last character output when characters are displayed using the <b>displayText</b> or <b>displayTextAt</b> method.</p> <p>When false, the cursor properties will not be updated when characters are displayed.</p> <p>This property is maintained for each window. It initialized to true by the <b>open</b> and <b>createWindow</b> methods, and is updated when <b>CurrentWindow</b> is set or <b>destroyWindow</b> is called.</p>
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
<b>See Also</b>	<b>CursorRow</b> Property, <b>CursorColumn</b> Property

## DeviceBrightness Property R/W

<b>Type</b>	<b>int</b>				
<b>Remarks</b>	<p>Holds the device brightness value, expressed as a percentage between 0 and 100.</p> <p>Any device can support 0% (blank) and 100% (full intensity). Blanking can, at a minimum, be supported by sending spaces to the device. If <b>CapBrightness</b> is true, then the device also supports one or more levels of dimming.</p> <p>If a device does not support the specified brightness value, then the Device Service will choose an appropriate substitute.</p> <p>This property is initialized to 100 when the device is first enabled following the <b>open</b> method.</p>				
<b>Errors</b>	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p> <p>Some possible values of the exception’s <i>ErrorCode</i> property are:</p> <table border="1"> <thead> <tr> <th><b>Value</b></th> <th><b>Meaning</b></th> </tr> </thead> <tbody> <tr> <td>JPOS_E_ILLEGAL</td> <td>An invalid value was used: Not in the range 0 - 100.</td> </tr> </tbody> </table>	<b>Value</b>	<b>Meaning</b>	JPOS_E_ILLEGAL	An invalid value was used: Not in the range 0 - 100.
<b>Value</b>	<b>Meaning</b>				
JPOS_E_ILLEGAL	An invalid value was used: Not in the range 0 - 100.				

## DeviceColumns Property R

Type	<b>int</b>
Remarks	Holds the number of columns on this device.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>DeviceRows</b> Property

## DeviceDescriptors Property R

Type	<b>int</b>
Remarks	Holds the number of descriptors on this device. If <b>CapDescriptors</b> is true, then this property is non-zero.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>setDescriptor</b> Method, <b>clearDescriptors</b> Method

## DeviceRows Property R

Type	<b>int</b>
Remarks	Holds the number of rows on this device.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>DeviceColumns</b> Property

## DeviceWindows Property R

Type	int
Remarks	<p>Holds the maximum window number supported by this device. A value of zero indicates that only the device window is supported and that no windows may be created.</p> <p>This property is initialized by the <b>open</b> method.</p>
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>CurrentWindow</b> Property

## InterCharacterWait Property R/W

Type	int
Remarks	<p>Holds the wait time between displaying each character with the <b>displayText</b> and <b>displayTextAt</b> methods. This provides a “teletype” appearance when displaying text.</p> <p>This property is only used if the window is not in <i>Marquee Mode</i> — that is, <b>MarqueeType</b> must be DISP_MT_NONE.</p> <p>When non-zero and the window is not in <i>Marquee Mode</i>, the window is in <i>Teletype Mode</i>: <b>displayText</b> and <b>displayTextAt</b> requests are enqueued and processed in the order they are received. This property specifies the time to wait between outputting each character into the viewport. The wait time is the specified number of milliseconds. (Note that the system timer resolution may reduce the precision of the wait time.) If <b>CursorUpdate</b> is true, <b>CursorRow</b> and <b>CursorColumn</b> are updated to their final values before <b>displayText</b> or <b>displayTextAt</b> returns, even though all of its data may not yet be displayed.</p> <p>When this property is zero and the window is not in <i>Marquee Mode</i>, <i>Immediate Mode</i> is in effect, so that characters are processed as quickly as possible. If some display requests are enqueued at the time this property is set to zero, the requests are completed as quickly as possible.</p> <p>If <b>CapICharWait</b> is false, then intercharacter waiting is not supported, and the value of this property is not used.</p> <p>This property is initialized to zero by the <b>open</b> and <b>createWindow</b> methods, and is updated when <b>CurrentWindow</b> is set or <b>destroyWindow</b> is called.</p>
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>displayText</b> Method

**MarqueeFormat Property R/W**

Type	int
Remarks	Holds the marquee format for the current window.

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

This property is initialized to DISP\_MF\_WALK by the **open** and **createWindow** methods, and is updated when **CurrentWindow** is set or **destroyWindow** is called.

This property is read when a transition is made to *Marquee On Mode*. It is not used when not in *Marquee Mode*.

When this property is DISP\_MF\_WALK, and a transition is made from *Marquee Init Mode* to *Marquee On Mode*, the following occurs:

1. Map the window to the viewport as follows:

<u>Marquee TypeWindow</u>		<u>Viewport</u>
LeftFirst Column	=	Last Column
UpFirst Row	=	Last Row
RightLast Column	=	First Column
DownLast Row	=	First Row

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

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



When this property is DISP\_MF\_PLACE, and a transition is made from *Marquee Init Mode* to *Marquee On Mode*, the following occurs:

1. Map the window to the viewport as follows:

<u>Marquee TypeWindow</u>		<u>Viewport</u>
LeftFirst Column	=	First Column
UpFirst Row	=	First Row
RightLast Column	=	Last Column
DownLast Row	=	Last Row

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

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

**Errors** A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_ILLEGAL	An invalid value was used, or attempted to change window 0.

**See Also** **MarqueeType** Property, **MarqueeUnitWait** Property, **MarqueeRepeatWait** Property

**Example 1** Marquee Walk format.  
 - Assume a 2x20 display.  
 - An application has a line display instance named myLD.  
 - The application has performed:  
 myLD.createWindow(0, 3, 2, 3, 2, 5); // 2x3 viewport of 2x5 window  
 myLD.displayText(“0123456789”, DISP\_DT\_NORMAL);

The window contains:

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



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

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

The marquee has scrolled to the end of the window.

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

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

**Example 2** Marquee Place format.

- Assume a 2x20 display.
- An application has a line display instance named myLD.
- The application has performed:
 

```
myLD.createWindow(0, 3, 2, 3, 2, 5); // 2x3 viewport of 2x5 window
myLD.displayText("0123456789", DISP_DT_NORMAL);
```

The window contains:

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

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

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

If the application performs the sequence:

```
myLD.setMarqueeType(DISP_MT_INIT);
myLD.setMarqueeFormat(DISP_MF_PLACE);
myLD.displayTextAt(0, 4, "AB", DISP_DT_NORMAL);
```

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

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



**MarqueeRepeatWait Property R/W**

<b>Type</b>	<b>int</b>
<b>Remarks</b>	<p>Holds the wait time between scrolling the final character or row of the window into its viewport and restarting the marquee with the first or last character or row.</p> <p>The wait time is the specified number of milliseconds. (Note that the timer resolution may reduce the precision of the wait time.)</p> <p>This property is initialized to zero by the <b>open</b> and <b>createWindow</b> methods, and is updated when <b>CurrentWindow</b> is set or <b>destroyWindow</b> is called.</p> <p>This property is not used if not in <i>Marquee Mode</i>.</p>
<b>Errors</b>	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>
<b>See Also</b>	<p><b>MarqueeType</b> Property, <b>MarqueeFormat</b> Property, <b>MarqueeUnitWait</b> Property</p>

**MarqueeType Property R/W**

**Type**            **int**

**Remarks**       Holds the marquee type for the current window. When not DISP\_MT\_NONE, the window is in *Marquee Mode*. This property has one of the following values:

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

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

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

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

This property is always DISP\_MT\_NONE for window 0 – the device window.

This property is initialized to DISP\_MT\_NONE by the **open** and **createWindow** methods, and is updated when **CurrentWindow** is set or **destroyWindow** is called.

**Errors** A `JposException` may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s `ErrorCode` property are:

Value	Meaning
JPOS_E_ILLEGAL	An invalid value was used, or attempted to change window 0.

**See Also** **MarqueeFormat** Property, **MarqueeUnitWait** Property, **MarqueeRepeatWait** Property

### MarqueeUnitWait Property R/W

**Type** `int`

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

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

This property is not used if **MarqueeType** is `DISP_MT_NONE`.

This property is initialized to zero by the **open** and **createWindow** methods, and is updated when **CurrentWindow** is set or **destroyWindow** is called.

**Errors** A `JposException` may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

**See Also** **MarqueeType** Property, **MarqueeFormat** Property, **MarqueeRepeatWait** Property

### Rows Property R

**Type** `int`

**Remarks** Holds the number of rows for this window.

For window 0, this property is the same as **DeviceRows**.  
For other windows, it may be less or greater than **DeviceRows**.

This property is initialized to **DeviceRows** by the **open** method, and is updated when **CurrentWindow** is set or **createWindow** or **destroyWindow** are called.

**Errors** A `JposException` may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

**See Also** **Columns** Property

## Methods

### clearDescriptors Method

**Syntax**      **void clearDescriptors () throws JposException;**

**Remarks**      Turns off all descriptors.

This function is illegal if **CapDescriptors** is false.

**Errors**      A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_ILLEGAL	The device does not support descriptors.

**See Also**      **setDescriptor** Method, **DeviceDescriptors** Property

### clearText Method

**Syntax**      **void clearText () throws JposException;**

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

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

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

If in *Marquee On Mode*, this method is illegal.

**Errors**      A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_ILLEGAL	In <i>Marquee On Mode</i> .

**See Also**      **displayText** Method





**Errors** A `JposException` may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_ILLEGAL	One or more parameters are out of their valid ranges, or all available windows are already in use.

**See Also** `destroyWindow` Method, `CurrentWindow` Property

### **destroyWindow Method**

**Syntax** `void destroyWindow () throws JposException;`

**Remarks** Destroys the current window. The characters displayed in its viewport are not changed.

`CurrentWindow` is set to window 0. The device window and the associated window properties are updated.

**Errors** A `JposException` may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_ILLEGAL	The current window is 0. This window may not be destroyed.

**See Also** `createWindow` Method, `CurrentWindow` Property

## displayText Method

**Syntax**      **void displayText (String *data*, int *attribute*) throws JposException;**

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

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

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

---

### Note

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

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

---

The operation of **displayText** (and **displayTextAt**) varies for each mode:

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

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

Special character values within *data* are:

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

**Errors** A `JposException` may be thrown when this method is invoked. For further information, see "Exceptions" on page 15.

Some possible values of the exception's *ErrorCode* property are:

Value	Meaning
<code>JPOS_E_ILLEGAL</code>	<i>attribute</i> is illegal, or the display is in <b>Marquee On Mode</b> .

**See Also** **displayTextAt** Method, **clearText** Method, **InterCharacterWait** Property



## refreshWindow Method

**Syntax**      **void refreshWindow (int *window*) throws JposException;**

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

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

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

**Errors**      A JposException may be thrown when this method is invoked. For further information, see "Exceptions" on page 15.

Some possible values of the exception's *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
JPOS_E_ILLEGAL	<i>window</i> is larger than <b>DeviceWindows</b> or has not been created, or in <i>Marquee On Mode</i> .

## scrollText Method

**Syntax**      **void scrollText (int direction, int units) throws JposException;**

The *direction* parameter indicates the scrolling direction, and is one of the following values:

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

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

**Remarks**      Scrolls the current window.

This function is only legal in *Immediate Mode*.

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

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

**Errors**      A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_ILLEGAL	<i>direction</i> is illegal, or in <i>Teletype Mode</i> or <i>Marquee Mode</i> .

**See Also**      **displayText Method**







**setDescriptor Method**

**Syntax**      **void setDescriptor (int *descriptor*, int *attribute*) throws JposException;**

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

The *attribute* parameter indicates the attribute for the descriptor. It has one of the following values:

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

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

This function is illegal if **CapDescriptors** is false.

The device and its Device Service determine the mapping of *descriptor* to its descriptors.

**Errors**      A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_ILLEGAL	The device does not support descriptors, or one of the parameters contained an illegal value.

**See Also**      **clearDescriptors** Method, **DeviceDescriptors** Property

## Events

### DirectIOEvent

**Interface** `jpos.events.DirectIOListener`

**Method** `directIOOccurred (DirectIOEvent e);`

**Description** Provides Device Service information directly to the application. This event provides a means for a vendor-specific Line Display Device Service to provide events to the application that are not otherwise supported by the Device Control.

**Properties** This event contains the following properties:

Property	Type	Description
<i>EventNumber</i>	<i>int</i>	Event number whose specific values are assigned by the Device Service.
<i>Data</i>	<i>int</i>	Additional numeric data. Specific values vary by the <i>EventNumber</i> and the Device Service. This property is settable.
<i>Object</i>	<i>Object</i>	Additional data whose usage varies by the <i>EventNumber</i> and Device Service. This property is settable.

**Remarks** This event is to be used only for those types of vendor specific functions that are not otherwise described as part of the JavaPOS standard. Use of this event may restrict the application program from being used with other vendor's Line Display devices which may not have any knowledge of the Device Service's need for this event.

**See Also** "Events" on page 18, **directIO** Method

## StatusUpdateEvent

**Interface**    `jpos.events.StatusUpdateListener`

**Method**      `statusUpdateOccurred (StatusUpdateEvent e);`

**Description**    Notifies the application that there is a change in the power status of a Line Display.

**Properties**     This event contains the following property:

<b>Property</b>	<b>Type</b>	<b>Description</b>
<i>Status</i>	<i>int</i>	Reports a change in the power state of a display.

*Note that Release 1.3* added Power State Reporting with additional *Power reporting* **StatusUpdateEvent** values. See “StatusUpdateEvent” description on page 78.

**Remarks**      Enqueued when the Line Display detects a power state change.

**See Also**      “Events” on page 18

# MICR – Magnetic Ink Character Recognition Reader

## Summary

### Properties

<i>Common</i>	<i>Ver</i>	<i>Type</i>	<i>Access</i>	<i>May Use After</i>
<b>AutoDisable</b>		boolean	R/W	open
<b>CapPowerReporting</b>	1.3	int	R	open
<b>CheckHealthText</b>		String	R	open
<b>Claimed</b>		boolean	R	open
<b>DataCount</b>		int	R	open
<b>DataEventEnabled</b>		boolean	R/W	open
<b>DeviceEnabled</b>		boolean	R/W	open & claim
<b>FreezeEvents</b>		boolean	R/W	open
<b>OutputID</b>		int	R	<i>Not Supported</i>
<b>PowerNotify</b>	1.3	int	R/W	open
<b>PowerState</b>	1.3	int	R	open
<b>State</b>		int	R	--
<b>DeviceControlDescription</b>		String	R	--
<b>DeviceControlVersion</b>		int	R	--
<b>DeviceServiceDescription</b>		String	R	open
<b>DeviceServiceVersion</b>		int	R	open
<b>PhysicalDeviceDescription</b>		String	R	open
<b>PhysicalDeviceName</b>		String	R	open

---

<i>Specific</i>	<i>Ver</i>	<i>Type</i>	<i>Access</i>	<i>May Use After</i>
<b>CapValidationDevice</b>		boolean	R	open
<b>RawData</b>		String	R	open
<b>AccountNumber</b>		String	R	open
<b>Amount</b>		String	R	open
<b>BankNumber</b>		String	R	open
<b>EPC</b>		String	R	open
<b>SerialNumber</b>		String	R	open
<b>TransitNumber</b>		String	R	open
<b>CheckType</b>		int	R	open
<b>CountryCode</b>		int	R	open

**Methods**

<i>Common</i>	<i>Ver</i>	<i>May Use After</i>
<b>open</b>		--
<b>close</b>		open
<b>claim</b>		open
<b>release</b>		open & claim
<b>checkHealth</b>		open, claim, & enable
<b>clearInput</b>		open & claim
<b>clearOutput</b>		<i>Not Supported</i>
<b>directIO</b>		open
 <i>Specific</i>		
<b>beginInsertion</b>		open, claim, & enable
<b>endInsertion</b>		open, claim, & enable
<b>beginRemoval</b>		open, claim, & enable
<b>endRemoval</b>		open, claim, & enable

**Events**

<i>Name</i>	<i>Ver</i>	<i>May Occur After</i>
<b>DataEvent</b>		open, claim, & enable
<b>DirectIOEvent</b>	1.3	open & claim
<b>ErrorEvent</b>		open, claim, & enable
<b>OutputCompleteEvent</b>		<i>Not Supported</i>
<b>StatusUpdateEvent</b>	1.3	open, claim, & enable

## General Information

The MICR Control's class name is "jpos.MICR".

The device constants are contained in the class "jpos.MICRConst".

See "Package Structure" on page 40.

### Capabilities

The MICR Control has the following minimal set of capabilities:

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

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

Some MICR devices support exception tables, which cause non-standard parsing of the serial number for specific check routing numbers. Exception tables are not directly supported by this JavaPOS release. However, a Device Service may choose to support them, and could assign JposEntryRegistry entries under its device name to define the exception entries.

This release of JavaPOS does not define any parsing of partial MICR check data if an error occurs while reading a check. This is done intentionally since any Device Service that implements such functionality cannot guarantee that fields parsed from partial data are correct. For example, it is possible to get MICR data that contains no '?' characters, but fails its checksum. This would indicate that one or more characters in the data are incorrect, but there is no way to determine which characters they are. If an application wishes to try and parse the partial data itself, the **RawData** property is filled in with whatever was read even in error cases.



## Model

In general, the MICR Device follows the JavaPOS model for input devices. One point of difference is that the MICR Device requires the execution of methods to insert and remove the check for processing. Therefore, this Device requires more than simply setting the **DataEventEnabled** property to true in order to receive data. The basic model is as follows:

- The MICR Control is opened, claimed, and enabled.
- When an application wishes to perform a MICR read, the application calls **beginInsertion**, specifying a timeout value. This results in the device being made ready to have a check inserted. If the check is not inserted before the timeout limit expires, a `JposException` is thrown.

In the event of a timeout, the MICR device will remain in a state allowing a check to be inserted while the application provides any additional prompting required and then reissues the **beginInsertion** method.

- Once a check is inserted, the method returns and the application calls **endInsertion**, which results in the MICR device being taken out of check insertion mode and the check, if present, actually being read.
  - If the check is successfully read, a **DataEvent** is enqueued.
  - If the **AutoDisable** property is true, then the Device automatically disables itself when a **DataEvent** is enqueued.
  - A queued **DataEvent** can be delivered to the application when **DataEventEnabled** is true. Just before delivering this event, data is copied into properties, and further data events are disabled by setting **DataEventEnabled** to false. This causes subsequent input data to be enqueued while the application processes the current input and associated properties. When the application has finished the current input and is ready for more data, it reenables events by setting **DataEventEnabled** to true.
  - An **ErrorEvent** (or events) are enqueued if an error occurs while reading the check, and is delivered to the application when **DataEventEnabled** is true.
  - The **DataCount** property may be read to obtain the number of queued **DataEvents**.
  - All queued input may be deleted by calling **clearInput**.
- After processing a **DataEvent**, the application should query the **CapValidationDevice** property to determine if validation printing can be performed on the check prior to check removal. If this property is true, the application may call the Printer Control's **beginInsertion** and **endInsertion** methods. This positions the check for validation printing. The POS Printer's validation printing methods can then be used to perform validation printing. When validation printing is complete, the application should call the Printer Control's removal methods to remove the check.

- Once the check is no longer needed in the device, the application must call **beginRemoval**, also specifying a timeout value. This method will throw a `JposException` if the check is not removed in the timeout period. In this case, the application may perform any additional prompting prior to calling the method again. Once the check is removed, the application should call **endRemoval** to take the MICR device out of removal mode.

Many models of MICR devices do not require any check handling processing from the application. Such devices may always be capable of receiving a check and require no commands to actually read and eject the check. For these types of MICR devices, the **beginInsertion**, **endInsertion**, **beginRemoval** and **endRemoval** methods simply return, and input data will be enqueued until the **DataEventEnabled** property is set to true. However, applications should still use these methods to ensure application portability across different MICR devices.





### Device Sharing

The MICR is an exclusive-use device. Its device sharing rules are:

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

## MICR Character Substitution

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

MICR Character	Name	Substitute Character
	Transit	t
	Amount	a
	On-U's	o
	Dash	-

## Properties

### AccountNumber Property R

Type	<b>String</b>
Remarks	<p>Holds the account number parsed from the most recently read MICR data.</p> <p>This account number will not include a check serial number if a check serial number is able to be separately parsed, even if the check serial number is embedded in the account number portion of the 'On Us' field. If the account number cannot be identified, the string will be empty ("").</p> <p>Its value is set prior to a <b>DataEvent</b> being sent to the application.</p>
Errors	A JposException may be thrown when this property is accessed. For further information, see "Exceptions" on page 15.
See Also	<b>RawData</b> Property, <b>DataEvent</b>

### Amount Property R

Type	<b>String</b>
Remarks	<p>Holds the amount field parsed from the most recently read MICR data.</p> <p>The amount field on a check consists of ten digits bordered by Amount symbols. All non space digits will be represented in the test string including leading 0's. If the amount is not present, the string will be empty ("").</p> <p>Its value is set prior to a <b>DataEvent</b> being sent to the application.</p>
Errors	A JposException may be thrown when this property is accessed. For further information, see "Exceptions" on page 15.
See Also	<b>RawData</b> Property, <b>DataEvent</b>

## BankNumber Property R

<b>Type</b>	<b>String</b>
<b>Remarks</b>	<p>Holds the bank number portion of the transit field parsed from the most recently read MICR data.</p> <p>The bank number is contained in digits 5 through 8 of the transit field. If the bank number or transit field is not present or successfully identified, the string will be empty (“”).</p> <p>Its value is set prior to a <b>DataEvent</b> being sent to the application.</p>
<b>Errors</b>	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>
<b>See Also</b>	<b>RawData</b> Property, <b>TransitNumber</b> Property, <b>DataEvent</b>

## CapValidationDevice Property R

<b>Type</b>	<b>boolean</b>
<b>Remarks</b>	<p>If true, the device also performs validation printing via the POS Printer’s slip station, and a check does not have to be removed from the MICR device prior to performing validation printing.</p> <p>For devices that are both a MICR device as well as a POS Printer, the device will automatically position the check for validation printing after successfully performing a MICR read. Either the MICR’s or the POS Printer’s <b>beginRemoval</b> and <b>endRemoval</b> methods may be called to remove the check once processing is complete.</p> <p>This property is initialized by the <b>open</b> method.</p>
<b>Errors</b>	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>

**CheckType Property R**

<b>Type</b>	<b>int</b>								
<b>Remarks</b>	Holds the type of check parsed from the most recently read MICR data. It has one of the following values:  <table> <thead> <tr> <th><b>Value</b></th> <th><b>Meaning</b></th> </tr> </thead> <tbody> <tr> <td>MICR_CT_PERSONAL</td> <td>The check is a personal check.</td> </tr> <tr> <td>MICR_CT_BUSINESS</td> <td>The check is a business or commercial check.</td> </tr> <tr> <td>MICR_CT_UNKNOWN</td> <td>Unknown type of check.</td> </tr> </tbody> </table> <p>Its value is set prior to a <b>DataEvent</b> being sent to the application.</p>	<b>Value</b>	<b>Meaning</b>	MICR_CT_PERSONAL	The check is a personal check.	MICR_CT_BUSINESS	The check is a business or commercial check.	MICR_CT_UNKNOWN	Unknown type of check.
<b>Value</b>	<b>Meaning</b>								
MICR_CT_PERSONAL	The check is a personal check.								
MICR_CT_BUSINESS	The check is a business or commercial check.								
MICR_CT_UNKNOWN	Unknown type of check.								
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.								
<b>See Also</b>	<b>RawData</b> Property, <b>DataEvent</b>								

**CountryCode Property R**

<b>Type</b>	<b>int</b>										
<b>Remarks</b>	Holds the country of origin of the check parsed from the most recently read MICR data. It has one of the following values:  <table> <thead> <tr> <th><b>Value</b></th> <th><b>Meaning</b></th> </tr> </thead> <tbody> <tr> <td>MICR_CC_USA</td> <td>The check is from America.</td> </tr> <tr> <td>MICR_CC_CANADA</td> <td>The check is from Canada.</td> </tr> <tr> <td>MICR_CC_MEXICO</td> <td>The check is from Mexico.</td> </tr> <tr> <td>MICR_CC_UNKNOWN</td> <td>Check origination is unknown.</td> </tr> </tbody> </table> <p>Its value is set prior to a <b>DataEvent</b> being sent to the application.</p>	<b>Value</b>	<b>Meaning</b>	MICR_CC_USA	The check is from America.	MICR_CC_CANADA	The check is from Canada.	MICR_CC_MEXICO	The check is from Mexico.	MICR_CC_UNKNOWN	Check origination is unknown.
<b>Value</b>	<b>Meaning</b>										
MICR_CC_USA	The check is from America.										
MICR_CC_CANADA	The check is from Canada.										
MICR_CC_MEXICO	The check is from Mexico.										
MICR_CC_UNKNOWN	Check origination is unknown.										
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.										
<b>See Also</b>	<b>RawData</b> Property, <b>DataEvent</b>										

## EPC Property R

Type	<b>String</b>
Remarks	<p>Holds the Extended Processing Code (“EPC”) field parsed from the most recently read MICR data. It will contain a single character 0 though 9 if the field is present. If not, the string will be empty (“”).</p> <p>Its value is set prior to a <b>DataEvent</b> being sent to the application.</p>
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>RawData</b> Property, <b>DataEvent</b>

## RawData Property R

Type	<b>String</b>
Remarks	<p>Holds the MICR data from the most recent MICR read. It contains any of the 15 MICR characters with appropriate substitution to represent non-ASCII characters (see “MICR Character Substitution”, page 377). No parsing or special processing is done to the data returned in this property. A sample value may look like the following:</p> <pre>“2t123456789t123 4 567890o 123 a0000001957a”</pre> <p>Note that spaces are used to represent spaces in the MICR data.</p> <p>Its value is set prior to a <b>DataEvent</b> being sent to the application.</p>
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>AccountNumber</b> Property, <b>Amount</b> Property, <b>BankNumber</b> Property, <b>CheckType</b> Property, <b>CountryCode</b> Property, <b>EPC</b> Property, <b>SerialNumber</b> Property, <b>TransitNumber</b> Property, <b>DataEvent</b>

## SerialNumber Property R

<b>Type</b>	<b>String</b>
<b>Remarks</b>	<p>Holds the serial number of the check parsed from the most recently read MICR data.</p> <p>If the serial number cannot be successfully parsed, the string will be empty (“”).</p> <p>Its value is set prior to a <b>DataEvent</b> being sent to the application.</p>
<b>Errors</b>	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>
<b>See Also</b>	<b>RawData</b> Property, <b>DataEvent</b>

## TransitNumber Property R

<b>Type</b>	<b>String</b>
<b>Remarks</b>	<p>Holds the transit field of the check parsed from the most recently read MICR data. It consists of all the characters read between the ‘Transit’ symbols on the check. It is a nine character string.</p> <p>Its value is set prior to a <b>DataEvent</b> being sent to the application.</p>
<b>Errors</b>	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>
<b>See Also</b>	<b>RawData</b> Property, <b>DataEvent</b>



## Methods

### beginInsertion Method

**Syntax**      **void beginInsertion (int *timeout*) throws JposException;**

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

If zero, the method tries to begin insertion mode, then returns the appropriate status immediately. If JPOS\_FOREVER (-1), the method initiates the begin insertion mode, then waits as long as needed until either the check is inserted or an error occurs.

**Remarks**      Initiates check insertion processing.

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

If the MICR device cannot be placed into insertion mode, a JposException is thrown. Otherwise, check insertion is monitored until either:

- The check is successfully inserted.
- The check is not inserted before *timeout* milliseconds have elapsed, or an error is reported by the MICR device. In this case, a JposException is thrown. The MICR device remains in check insertion mode. This allows an application to perform some user interaction and reissue the **beginInsertion** method without altering the MICR check handling mechanism.

**Errors**      A JposException may be thrown when this method is invoked. For further information, see "Exceptions" on page 15.

Some possible values of the exception's *ErrorCode* property are:

Value	Meaning
JPOS_E_BUSY	If the MICR is a combination device, the peer device may be busy.
JPOS_E_ILLEGAL	An invalid <i>timeout</i> parameter was specified.
JPOS_E_TIMEOUT	The specified time has elapsed without the check being properly inserted.

**See Also**      **endInsertion** Method, **beginRemoval** Method, **endRemoval** Method

**beginRemoval Method**

<b>Syntax</b>	<p><b>void beginRemoval (int <i>timeout</i>) throws JposException;</b></p> <p>The <i>timeout</i> parameter gives the number of milliseconds before failing the method.</p> <p>If zero, the method tries to begin removal mode, then returns the appropriate status immediately. If JPOS_FOREVER (-1), the method initiates the begin removal mode, then waits as long as needed until either the check is removed or an error occurs.</p>								
<b>Remarks</b>	<p>Initiates check removal processing.</p> <p>When called, the MICR is made ready to remove a check, by opening the MICR's check handling "jaws" or activating a MICR's check ejection mode. This method is paired with the <b>endRemoval</b> method for controlling check removal. Although some MICR devices do not require this sort of processing, the application should still use these methods to ensure application portability across different MICR devices.</p> <p>If the MICR device cannot be placed into removal or ejection mode, a JposException is thrown. Otherwise, check removal is monitored until either:</p> <ul style="list-style-type: none"> <li>• The check is successfully removed.</li> <li>• The check is not removed before <i>timeout</i> milliseconds have elapsed, or an error is reported by the MICR device. In this case, a JposException is thrown. The MICR device remains in check removal mode. This allows an application to perform some user interaction and reissue the <b>beginRemoval</b> method without altering the MICR check handling mechanism.</li> </ul>								
<b>Errors</b>	<p>A JposException may be thrown when this method is invoked. For further information, see "Exceptions" on page 15.</p> <p>Some possible values of the exception's <i>ErrorCode</i> property are:</p> <table border="1"> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>JPOS_E_BUSY</td> <td>If the MICR is a combination device, the peer device may be busy.</td> </tr> <tr> <td>JPOS_E_ILLEGAL</td> <td>An invalid <i>timeout</i> parameter was specified.</td> </tr> <tr> <td>JPOS_E_TIMEOUT</td> <td>The specified time has elapsed without the check being properly removed.</td> </tr> </tbody> </table>	Value	Meaning	JPOS_E_BUSY	If the MICR is a combination device, the peer device may be busy.	JPOS_E_ILLEGAL	An invalid <i>timeout</i> parameter was specified.	JPOS_E_TIMEOUT	The specified time has elapsed without the check being properly removed.
Value	Meaning								
JPOS_E_BUSY	If the MICR is a combination device, the peer device may be busy.								
JPOS_E_ILLEGAL	An invalid <i>timeout</i> parameter was specified.								
JPOS_E_TIMEOUT	The specified time has elapsed without the check being properly removed.								
<b>See Also</b>	<b>beginInsertion</b> Method, <b>endInsertion</b> Method, <b>endRemoval</b> Method								

## endInsertion Method

**Syntax**      **void endInsertion () throws JposException;**

**Remarks**      Ends check insertion processing.

When called, the MICR is taken out of check insertion mode. If a check is not detected in the device, a `JposException` is thrown with an extended error code of `JPOS_EMICR_NOCHECK`. Upon completion of this method, the check will be read by the MICR device, and data will be available as soon as the **DataEventEnabled** property is set to true. This allows an application to prompt the user prior to calling this method to ensure that the form is correctly positioned.

This method is paired with the **beginInsertion** method for controlling check insertion. Although some MICR devices do not require this sort of processing, the application should still use these methods to ensure application portability across different MICR devices.

**Errors**      A `JposException` may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
<code>JPOS_E_ILLEGAL</code>	The printer is not in check insertion mode.
<code>JPOS_E_EXTENDED</code>	<i>ErrorCodeExtended</i> = <code>JPOS_EMICR_NOCHECK</code> : The device was taken out of insertion mode without a check being inserted.

**See Also**      **beginInsertion** Method, **beginRemoval** Method, **endRemoval** Method

**endRemoval Method**

**Syntax**      **void endRemoval () throws JposException;**

**Remarks**      Ends check removal processing.

When called, the MICR is taken out of check removal or ejection mode. If a check is detected in the device, a `JposException` is thrown with an extended error code of `JPOS_EMICR_CHECK`.

This method is paired with the **beginRemoval** method for controlling check removal. Although some MICR devices do not require this sort of processing, the application should still use these methods to ensure application portability across different MICR devices. For further information see “Events” on page 18

**Errors**      A `JposException` may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
<code>JPOS_E_ILLEGAL</code>	The printer is not in check removal mode.
<code>JPOS_E_EXTENDED</code>	<i>ErrorCodeExtended</i> = <code>JPOS_EMICR_CHECK</code> : The device was taken out of removal mode while a check is still present.

**See Also**      **beginInsertion** Method, **endInsertion** Method, **beginRemoval** Method

## Events

### DataEvent

**Interface**     **jpos.events.DataListener**

**Method**        **dataOccurred (DataEvent e)**

**Description**    Notifies the application when MICR data is read from a check and is available to be read.

**Properties**      This event contains the following property:

Parameter	Type	Description
-----------	------	-------------

<i>Status</i>	<i>int</i>	Set to zero.
---------------	------------	--------------

Before delivering this event, the **RawData** property is updated and the data is parsed (if possible) into the MICR data fields.

**See Also**        “Device Input Model” on page 22, “Events” on “Events” on page 18, **RawData** Property, **AccountNumber** Property, **Amount** Property, **BankNumber** Property, **CheckType** Property, **CountryCode** Property, **EPC** Property, **SerialNumber** Property, **TransitNumber** Property

**DirectIOEvent****Interface** `jpos.events.DirectIOListener`**Method** `directIOOccurred (DirectIOEvent e);`**Description** Provides Device Service information directly to the application. This event provides a means for a vendor-specific MICR Device Service to provide events to the application that are not otherwise supported by the Device Control.**Properties** This event contains the following properties:

<b>Property</b>	<b>Type</b>	<b>Description</b>
<i>EventNumber</i>	<i>int</i>	Event number whose specific values are assigned by the Device Service.
<i>Data</i>	<i>int</i>	Additional numeric data. Specific values vary by the <i>EventNumber</i> and the Device Service. This property is settable.
<i>Object</i>	<i>Object</i>	Additional data whose usage varies by the <i>EventNumber</i> and Device Service. This property is settable.

**Remarks** This event is to be used only for those types of vendor specific functions that are not otherwise described as part of the JavaPOS standard. Use of this event may restrict the application program from being used with other vendor's MICR devices which may not have any knowledge of the Device Service's need for this event.**See Also** "Events" on page 18, **directIO** Method

## ErrorEvent

Interface	<b>jpos.events.ErrorListener</b>
Method	<b>errorOccurred (ErrorEvent e);</b>
Description	Notifies the application that an error has been detected when reading MICR data.
Properties	This event contains the following properties:

Parameter	Type	Description
<i>ErrorCode</i>	<i>int</i>	Error Code causing the error event. See list of <i>ErrorCodes</i> on “ErrorCode” on page 16.
<i>ErrorCodeExtended</i>	<i>int</i>	Extended Error Code causing the error event. It may contain a Service-specific value.
<i>ErrorLocus</i>	<i>int</i>	Location of the error. See values below.
<i>ErrorResponse</i>	<i>int</i>	Error response, whose default value may be overridden by the application (i.e., this property is settable). See values below.

The *ErrorLocus* property has one of the following values:

Value	Meaning
JPOS_EL_INPUT	Error occurred while gathering or processing event-driven input. No input data is available.
JPOS_EL_INPUT_DATA	Error occurred while gathering or processing event-driven input, and some previously buffered data is available.

The application’s error event listener may change *ErrorResponse* to one of the following values:

Value	Meaning
JPOS_ER_CLEAR	Clear the buffered input data. The error state is exited. Default when locus is JPOS_EL_INPUT.
JPOS_ER_CONTINUEINPUT	Use only when locus is JPOS_EL_INPUT_DATA. Acknowledges the error and directs the Device to continue processing. The Device remains in the error state and will deliver additional <b>DataEvents</b> as directed by the <b>DataEventEnabled</b> property. When all input has been delivered and <b>DataEventEnabled</b> is again set to true, then another <b>ErrorEvent</b> is delivered with locus JPOS_EL_INPUT. Default when locus is JPOS_EL_INPUT_DATA.

- Remarks** This event is not delivered until **DataEventEnabled** is true, so that proper application sequencing occurs.
- See Also** “Device Input Model” on page 22, “Device States” on page 30

## StatusUpdateEvent

**Interface** `jpos.events.StatusUpdateListener`

**Method** `statusUpdateOccurred (StatusUpdateEvent e);`

**Description** Notifies the application that there is a change in the power status of a MICR device.

**Properties** This event contains the following property:

Property	Type	Description
<i>Status</i>	<i>int</i>	Reports a change in the power state of a MICR device.

*Note that Release 1.3* added Power State Reporting with additional *Power reporting StatusUpdateEvent values*. See “StatusUpdateEvent” description on page 78.

**Remarks** Enqueued when the MICR device detects a power state change.

**See Also** “Events” on page 18



# MSR – Magnetic Stripe Reader

## Summary

### Properties

<i>Common</i>	<i>Ver</i>	<i>Type</i>	<i>Access</i>	<i>May Use After</i>
<b>AutoDisable</b>		boolean	R/W	open
<b>CapPowerReporting</b>	1.3	int	R	open
<b>CheckHealthText</b>		String	R	open
<b>Claimed</b>		boolean	R	open
<b>DataCount</b>		int	R	open
<b>DataEventEnabled</b>		boolean	R/W	open
<b>DeviceEnabled</b>		boolean	R/W	open & claim
<b>FreezeEvents</b>		boolean	R/W	open
<b>OutputID</b>		int	R	<i>Not Supported</i>
<b>PowerNotify</b>	1.3	int	R/W	open
<b>PowerState</b>	1.3	int	R	open
<b>State</b>		int	R	--
<b>DeviceControlDescription</b>		String	R	--
<b>DeviceControlVersion</b>		int	R	--
<b>DeviceServiceDescription</b>		String	R	open
<b>DeviceServiceVersion</b>		int	R	open
<b>PhysicalDeviceDescription</b>		String	R	open
<b>PhysicalDeviceName</b>		String	R	open

<i>Specific</i>	<i>Ver</i>	<i>Type</i>	<i>Access</i>	<i>May Use After</i>
<b>CapISO</b>		boolean	R	open
<b>CapJISOne</b>		boolean	R	open
<b>CapJISTwo</b>		boolean	R	open
<b>CapTransmitSentinels</b>	1.5	boolean	R	open
<b>TracksToRead</b>		int	R/W	open
<b>DecodeData</b>		boolean	R/W	open
<b>ParseDecodeData</b>		boolean	R/W	open
<b>ErrorReportingType</b>		int	R/W	open
<b>Track1Data</b>		byte[]	R	open
<b>Track2Data</b>		byte[]	R	open
<b>Track3Data</b>		byte[]	R	open
<b>Track4Data</b>	1.5	byte[]	R	open
<b>AccountNumber</b>		String	R	open
<b>ExpirationDate</b>		String	R	open
<b>Title</b>		String	R	open
<b>FirstName</b>		String	R	open
<b>MiddleInitial</b>		String	R	open
<b>Surname</b>		String	R	open
<b>Suffix</b>		String	R	open
<b>ServiceCode</b>		String	R	open
<b>Track1DiscretionaryData</b>		byte[]	R	open
<b>Track2DiscretionaryData</b>		byte[]	R	open
<b>TransmitSentinels</b>	1.5	boolean	R/W	open

**Methods**

<i>Common</i>	<i>Ver</i>	<i>May Use After</i>
<b>open</b>		--
<b>close</b>		open
<b>claim</b>		open
<b>release</b>		open & claim
<b>checkHealth</b>		open, claim, & enable
<b>clearInput</b>		open & claim
<b>clearOutput</b>		<i>Not Supported</i>
<b>directIO</b>		open

**Events**

<i>Name</i>	<i>Ver</i>	<i>May Occur After</i>
<b>DataEvent</b>		open, claim, & enable
<b>DirectIOEvent</b>	1.3	open & claim
<b>ErrorEvent</b>		open, claim, & enable
<b>OutputCompleteEvent</b>		<i>Not Supported</i>
<b>StatusUpdateEvent</b>	1.3	open, claim, & enable

## General Information

The MSR Control's class name is "jpos.MSR".  
The device constants are contained in the class "jpos.MSRConst".  
See "Package Structure" on page 40.

### Capabilities

The MSR Control has the following minimal set of capabilities:

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

The MSR Control may have the following additional capabilities:

- Support for specific card types: ISO, JIS Type I and/or JIS Type II. Note: for the purpose of this standard, the following convention is assumed:
  - Track 1 is ISO or JIS-I Track 1
  - Track 2 is ISO or JIS-I Track 2
  - Track 3 is ISO or JIS-I Track 3
  - Track 4 is JIS-II data
  - Determination of the type of card is based on the type of content the card tracks are expected to hold.

Support for optionally returning the track sentinels with track data.

### Clarifications for JIS-II data handling

Prior to Version 1.5 of this specification, it was not clearly stated how the Control should treat JIS-II data and into which of the **Track*n*Data** properties the data should be stored. This version of the specification defines **Track4Data**, which the Control should use to store JIS-II data. However, in order to maintain application backward compatibility with previous versions, the Control may also store the JIS-II data into the previously used **Track*n*Data** property. In such cases, the **DataEvent Status** and the **ErrorEvent ErrorCodeExtended** attributes should be set to reflect both **Track4Data** and **Track*n*Data**. Note that applications that use this particular method of accessing JIS-II data may not be portable across Controls.

## Device Behavior Model

Four writable properties control MSR data handling:

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

The MSR Device follows the JavaPOS model for input devices:

- When input is received by the Device Service a **DataEvent** is enqueued.
- If the **AutoDisable** property is true, then the Device automatically disables itself when a **DataEvent** is enqueued.
- A queued **DataEvent** can be delivered to the application when the **DataEventEnabled** property is true. Just before delivering this event, data is copied into properties, and further data events are disabled by setting **DataEventEnabled** to false. This causes subsequent input data to be enqueued while the application processes the current input and associated properties. When the application has finished the current input and is ready for more data, it reenables events by setting **DataEventEnabled** to true.
- An **ErrorEvent** (or events) is enqueued if an error occurs while gathering or processing input, and is delivered to the application when **DataEventEnabled** is true.
- The **DataCount** property may be read to obtain the number of queued **DataEvents**.
- All queued input may be deleted by calling **clearInput**.

## Device Sharing

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

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

## Properties

### AccountNumber Property R

Type	<b>String</b>
Remarks	<p>Holds the account number obtained from the most recently swiped card.</p> <p>Set to the empty string if:</p> <ul style="list-style-type: none"><li>• The field was not included in the track data obtained, or,</li><li>• The track data format was not one of those listed in the <b>ParseDecodeData</b> property description, or,</li><li>• <b>ParseDecodeData</b> is false.</li></ul>
Errors	<p>A <code>JposException</code> may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>

### CapISO Property R

Type	<b>boolean</b>
Remarks	<p>If true, the MSR device supports ISO cards.</p> <p>This property is initialized by the <b>open</b> method.</p>
Errors	<p>A <code>JposException</code> may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>

### CapJISOne Property R

Type	<b>boolean</b>
Remarks	<p>If true, the MSR device supports JIS Type-I cards.</p> <p>JIS-I cards are a superset of ISO cards. Therefore, if <b>CapJISOne</b> is true, then it is implied that <b>CapISO</b> is also true.</p> <p>This property is initialized by the <b>open</b> method.</p>
Errors	<p>A <code>JposException</code> may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>

**CapJISTwo Property R**

<b>Type</b>	<b>boolean</b>
<b>Remarks</b>	If true, the MSR device supports JIS Type-II cards.  This property is initialized by the <b>open</b> method.
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

**CapTransmitSentinels Property R*****Added in Release 1.5***

<b>Type</b>	<b>boolean</b>
<b>Remarks</b>	If true, the device is able to transmit the start and end sentinels. If false, these characters cannot be returned to the application.  This property is initialized by the <b>open</b> method.
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
<b>See Also</b>	<b>TransmitSentinels</b> Property.

**DecodeData Property R/W**

<b>Type</b>	<b>boolean</b>
<b>Remarks</b>	If false, the <b>Track1Data</b> , <b>Track2Data</b> , <b>Track3Data</b> and <b>Track4Data</b> properties contain the original encoded bit sequences, known as “raw data format”.  If true, each byte of track data contained within the <b>Track1Data</b> , <b>Track2Data</b> , <b>Track3Data</b> , and <b>Track4Data</b> properties is mapped from its original encoded bit sequence (as it exists on the magnetic card) to its corresponding decoded ASCII bit sequence. This conversion is mainly of relevance for data that is NOT of the 7-bit format, since 7-bit data needs no decoding to decipher its corresponding alphanumeric and/or Katakana characters.  The decoding that takes place is as follows for each card type, track, and track data format:

Card Type	Track Data Property	Data Format	Raw Bytes	Decoded Bytes
ISO	<b>Track1Data</b>	6-Bit	0x00 - 0x3F	0x20 - 0x5F
	<b>Track2Data</b>	4-Bit	0x00 - 0x0F	0x30 - 0x3F
	<b>Track3Data</b>	4-Bit	0x00 - 0x0F	0x30 - 0x3F
JIS-I	<b>Track1Data</b>	6-Bit	0x00 - 0x3F	0x20 - 0x5F
	<b>Track1Data</b>	7-Bit	0x00 - 0x7F	Data Unchanged
	<b>Track2Data</b>	4-Bit	0x00 - 0x0F	0x30 - 0x3F
	<b>Track3Data</b>	4-Bit	0x00 - 0x0F	0x30 - 0x3F
	<b>Track3Data</b>	7-Bit	0x00 - 0x7F	Data Unchanged
JIS-II	<b>Track4Data</b>	7-Bit	0x00 - 0x7F	Data Unchanged



This property is initialized to true by the **open** method.

Setting this property to false automatically sets **ParseDecodeData** to false.

**Errors** A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

**See Also** **ParseDecodeData** Property

## ErrorReportingType Property R/W

**Type** **int**

**Remarks** Holds the type of errors to report via **ErrorEvents**. This property has one of the following values:

Value	Meaning
MSR_ERT_CARD	Report errors at a card level.
MSR_ERT_TRACK	Report errors at a track level.

An error is reported by an **ErrorEvent** when a card is swiped, and one or more of the tracks specified by the **TracksToRead** property contains data with errors. When the **ErrorEvent** is delivered to the application, two types of error reporting are supported:

- **Card level:** A general error status is given, with no data returned. This level should be used when a simple pass/fail of the card data is sufficient.
- **Track level:** The Control can return an extended status with a separate status for each of the tracks. Also, for those tracks that contain valid data or no data, the track’s properties are updated as with a **DataEvent**. For those tracks that contain invalid data, the track’s properties are set to empty. This level should be used when the application may be able to utilize a successfully read track or tracks when another of the tracks contains errors. For example, suppose **TracksToRead** is MSR\_TR\_1\_2\_3, and a swiped card contains good track 1 and 2 data, but track 3 contains “random noise” that is flagged as an error by the MSR. With track level error reporting, the **ErrorEvent** sets the track 1 and 2 properties with the valid data, sets the track 3 properties to empty, and returns an error code indicating the status of each track.

This property is initialized to MSR\_ERT\_CARD by the **open** method.

**Errors** A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

**See Also** **ErrorEvent**

## ExpirationDate Property R

<b>Type</b>	<b>String</b>
<b>Remarks</b>	<p>Holds the expiration date obtained from the most recently swiped card, as four ASCII decimal characters in the form YYMM. For example, February 1998 is “9802” and August 2018 is “1808”.</p> <p>Set to the empty string if:</p> <ul style="list-style-type: none"><li>• The field was not included in the track data obtained, or,</li><li>• The track data format was not one of those listed in the <b>ParseDecodeData</b> property description, or,</li><li>• <b>ParseDecodeData</b> is false.</li></ul>
<b>Errors</b>	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>

## FirstName Property R

<b>Type</b>	<b>String</b>
<b>Remarks</b>	<p>Holds the first name obtained from the most recently swiped card.</p> <p>Set to the empty string if:</p> <ul style="list-style-type: none"><li>• The field was not included in the track data obtained, or,</li><li>• The track data format was not one of those listed in the <b>ParseDecodeData</b> property description, or,</li><li>• <b>ParseDecodeData</b> is false.</li></ul>
<b>Errors</b>	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>

## MiddleInitial Property R

Type	String
Remarks	<p>Holds the middle initial obtained from the most recently swiped card.</p> <p>Set to the empty string if:</p> <ul style="list-style-type: none"><li>• The field was not included in the track data obtained, or,</li><li>• The track data format was not one of those listed in the <b>ParseDecodeData</b> property description, or,</li><li>• <b>ParseDecodeData</b> is false.</li></ul>
Errors	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>

## ParseDecodeData Property R/W

Type	boolean
Remarks	<p>When true, the decoded data contained within the <b>Track1Data</b> and <b>Track2Data</b> properties is further separated into fields for access via various other properties. <b>Track3Data</b> is not parsed because its data content is of an open format defined by the card issuer. JIS-I Track 1 Format C and ISO Track 1 Format C data are not parsed for similar reasons. <b>Track4Data</b> is also not parsed.</p> <p>The parsed data properties are the defined possible fields for cards with data consisting of the following formats:</p> <ul style="list-style-type: none"><li>• JIS-I / ISO Track 1 Format A</li><li>• JIS-I / ISO Track 1 Format B</li><li>• JIS-I / ISO Track 1 VISA Format (a de-facto standard)</li><li>• JIS-I / ISO Track 2 Format</li></ul> <p>This property is initialized to true by the <b>open</b> method.</p> <p>Setting this property to true automatically sets <b>DecodeData</b> to true.</p>
Errors	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>
See Also	<p><b>DecodeData</b> Property, <b>Surname</b> Property, <b>Suffix</b> Property, <b>AccountNumber</b> Property, <b>FirstName</b> Property, <b>MiddleInitial</b> Property, <b>Title</b> Property, <b>ExpirationDate</b> Property, <b>ServiceCode</b> Property, <b>Track1DiscretionaryData</b> Property, <b>Track2DiscretionaryData</b> Property</p>

## ServiceCode Property R

Type	<b>String</b>
Remarks	<p>Holds the service code obtained from the most recently swiped card.</p> <p>Set to the empty string if:</p> <ul style="list-style-type: none"><li>• The field was not included in the track data obtained, or,</li><li>• The track data format was not one of those listed in the <b>ParseDecodeData</b> property description, or,</li><li>• <b>ParseDecodeData</b> is false.</li></ul>
Errors	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>

## Suffix Property R

Type	<b>String</b>
Remarks	<p>Holds the suffix obtained from the most recently swiped card.</p> <p>Set to the empty string if:</p> <ul style="list-style-type: none"><li>• The field was not included in the track data obtained, or,</li><li>• The track data format was not one of those listed in the <b>ParseDecodeData</b> property description, or,</li><li>• <b>ParseDecodeData</b> is false.</li></ul>
Errors	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>

## Surname Property R

Type	<b>String</b>
Remarks	<p>Holds the surname obtained from the most recently swiped card.</p> <p>Set to the empty string if:</p> <ul style="list-style-type: none"><li>• The field was not included in the track data obtained, or,</li><li>• The track data format was not one of those listed in the <b>ParseDecodeData</b> property description, or,</li><li>• <b>ParseDecodeData</b> is false.</li></ul>
Errors	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>

## Title Property R

Type	<b>String</b>
Remarks	<p>Holds the title obtained from the most recently swiped card.</p> <p>Set to the empty string if:</p> <ul style="list-style-type: none"><li>• The field was not included in the track data obtained, or,</li><li>• The track data format was not one of those listed in the <b>ParseDecodeData</b> property description, or,</li><li>• <b>ParseDecodeData</b> is false.</li></ul>
Errors	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>

## Track1Data Property R

Type	<b>byte[]</b>
Remarks	<p>Holds the track 1 data from the most recently swiped card or a zero-length array.</p> <p>If <b>TransmitSentinels</b> is false, this property contains track data between but not including the start and end sentinels. If <b>TransmitSentinels</b> is true, then the start and end sentinels are included.</p> <p>If <b>DecodeData</b> is true, then the data returned by this property has been decoded from the “raw” format. The data may also be parsed into other properties when the <b>ParseDecodeData</b> property is set.</p> <p>A zero-length array indicates that the track was not accessible.</p>
Errors	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>
See Also	<b>TracksToRead</b> Property

## Track1DiscretionaryData Property R

<b>Type</b>	<b>byte[]</b>
<b>Remarks</b>	<p>Holds the track 1 discretionary data obtained from the most recently swiped card.</p> <p>The array will be of zero-length if:</p> <ul style="list-style-type: none"><li>• The field was not included in the track data obtained, or,</li><li>• The track data format was not one of those listed in the <b>ParseDecodeData</b> property description, or,</li><li>• <b>ParseDecodeData</b> is false.</li></ul> <p>The amount of data contained in this property varies widely depending upon the format of the track 1 data.</p>
<b>Errors</b>	<p>A <code>JposException</code> may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>

## Track2Data Property R

<b>Type</b>	<b>byte[]</b>
<b>Remarks</b>	<p>Holds the track 2 data from the most recently swiped card or a zero-length array.</p> <p>If <b>TransmitSentinels</b> is false, this property contains track data between but not including the start and end sentinels. If <b>TransmitSentinels</b> is true, then the start and end sentinels are included.</p> <p>If <b>DecodeData</b> is true, then the data returned by this property has been decoded from the “raw” format. The data may also be parsed into other properties when the <b>ParseDecodeData</b> property is set.</p> <p>A zero-length array indicates that the track was not accessible.</p>
<b>Errors</b>	<p>A <code>JposException</code> may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>
<b>See Also</b>	<b>TracksToRead</b> Property

## Track2DiscretionaryData Property R

Type	byte[]
Remarks	<p>Holds the track 2 discretionary data obtained from the most recently swiped card.</p> <p>The array will be of zero-length if:</p> <ul style="list-style-type: none"><li>• The field was not included in the track data obtained, or,</li><li>• The track data format was not one of those listed in the <b>ParseDecodeData</b> property description, or,</li><li>• <b>ParseDecodeData</b> is false.</li></ul>
Errors	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>

## Track3Data Property R

Type	byte[]
Remarks	<p>Holds the track 3 data from the most recently swiped card or a zero-length array.</p> <p>If <b>TransmitSentinels</b> is false, this property contains track data between but not including the start and end sentinels. If <b>TransmitSentinels</b> is true, then the start and end sentinels are included.</p> <p>If <b>DecodeData</b> is true, then the data returned by this property has been decoded from the “raw” format. The data may also be parsed into other properties when the <b>ParseDecodeData</b> property is set.</p> <p>A zero-length array indicates that the track was not accessible.</p>
Errors	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>
See Also	<b>TracksToRead</b> Property

**Track4Data Property R*****Added in Release 1.5***

<b>Type</b>	<b>binary</b>
<b>Remarks</b>	<p>Holds the track 4 data (JIS-II) obtained from the most recently swiped card or a zero length array.</p> <p>If <b>TransmitSentinels</b> is false, this property contains track data between but not including the start and end sentinels. If <b>TransmitSentinels</b> is true, then the start and end sentinels are included.</p> <p>If <b>DecodeData</b> is true, then the data returned by this property has been decoded from the “raw” format.</p> <p>A zero length array indicates that the track was not accessible.</p> <p>To maintain compatibility with previous versions, the Control may also continue to store the JIS-II data in another <b>TracknData</b> property. However, it should be noted that to insure application portability, <b>Track4Data</b> should be used to access JIS-II data.</p>
<b>Errors</b>	<p>A JposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.</p>
<b>See Also</b>	<p><b>Track1Data</b> Property, <b>Track2Data</b> Property, <b>Track3Data</b> Property, <b>TransmitSentinels</b> Property.</p>



**TracksToRead Property R/W**

**Type**        **int**

**Remarks**    Holds the track data that the application wishes to have placed into the **Track1Data**, **Track2Data**, and **Track3Data** properties following a card swipe. It has one of the following values:

<b>Value</b>	<b>Meaning</b>
MSR_TR_1	Obtain Track 1.
MSR_TR_2	Obtain Track 2.
MSR_TR_3	Obtain Track 3.
MSR_TR_1_2	Obtain Tracks 1 and 2.
MSR_TR_1_3	Obtain Tracks 1 and 3.
MSR_TR_2_3	Obtain Tracks 2 and 3.
MSR_TR_1_2_3	Obtain Tracks 1, 2, and 3.
MSR_TR_4	Obtain track 4.
MSR_TR_1_4	Obtain tracks 1 and 4.
MSR_TR_2_4	Obtain tracks 2 and 4.
MSR_TR_3_4	Obtain tracks 3 and 4.
MSR_TR_1_2_4	Obtain tracks 1, 2, and 4.
MSR_TR_1_3_4	Obtain tracks 1, 3, and 4.
MSR_TR_2_3_4	Obtain tracks 2, 3, and 4.
MSR_TR_1_2_3_4	Obtain tracks 1, 2, 3, and 4.

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

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

This property is initialized to MSR\_TR\_1\_2\_3 by the **open** method.

**Errors**        A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

**TransmitSentinels Property R*****Added in Release 1.5***

<b>Type</b>	<b>boolean</b>				
<b>Remarks</b>	<p>If true, the <b>Track1Data</b>, <b>Track2Data</b>, <b>Track3Data</b>, and <b>Track4Data</b> properties contain start and end sentinel values.</p> <p>If false, then these properties contain only the track data between these sentinels.</p> <p>This property is initialized to false by the <b>open</b> method.</p>				
<b>Errors</b>	<p>A JposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.</p> <p>Some possible values of the exception’s <i>ErrorCode</i> property are:</p> <table> <thead> <tr> <th><b>Value</b></th> <th><b>Meaning</b></th> </tr> </thead> <tbody> <tr> <td>JPOS_E_ILLEGAL</td> <td>The <b>CapTransmitSentinels</b> property is false.</td> </tr> </tbody> </table>	<b>Value</b>	<b>Meaning</b>	JPOS_E_ILLEGAL	The <b>CapTransmitSentinels</b> property is false.
<b>Value</b>	<b>Meaning</b>				
JPOS_E_ILLEGAL	The <b>CapTransmitSentinels</b> property is false.				
<b>See Also</b>	<b>CapTransmitSentinels</b> Property, <b>Track1Data</b> Property, <b>Track2Data</b> Property, <b>Track3Data</b> Property, <b>Track4Data</b> Property.				

## Events

### DataEvent

**Interface** `jpos.events.DataListener`

**Method** `dataOccurred (DataEvent e);`

**Description** Notifies the application when input data from the MSR device is available.

**Properties** This event contains the following property:

Property	Type	Description
----------	------	-------------

<i>Status</i>	<i>int</i>	See below.
---------------	------------	------------

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

High Word		Low Word	
High Byte	Low Byte	High Byte	Low Byte
Track 4	Track 3	Track 2	Track 1

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

A value greater than zero indicates the length in bytes of the corresponding **TrackxData** property.

**Remarks** Before this event is delivered, the swiped data is placed into **Track1Data**, **Track2Data**, **Track3Data**, and **Track4Data**. If **DecodeData** is true, then this track is decoded. If **ParseDecodeData** is true, then the data is parsed into several additional properties.

**DirectIOEvent****Interface** `jpos.events.DirectIOListener`**Method** `directIOOccurred (DirectIOEvent e);`**Description** Provides Device Service information directly to the application. This event provides a means for a vendor-specific MSR Device Service to provide events to the application that are not otherwise supported by the Device Control.**Properties** This event contains the following properties:

<b>Property</b>	<b>Type</b>	<b>Description</b>
<i>EventNumber</i>	<i>int</i>	Event number whose specific values are assigned by the Device Service.
<i>Data</i>	<i>int</i>	Additional numeric data. Specific values vary by the <i>EventNumber</i> and the Device Service. This property is settable.
<i>Object</i>	<i>Object</i>	Additional data whose usage varies by the <i>EventNumber</i> and Device Service. This property is settable.

**Remarks** This event is to be used only for those types of vendor specific functions that are not otherwise described as part of the JavaPOS standard. Use of this event may restrict the application program from being used with other vendor's MSR devices which may not have any knowledge of the Device Service's need for this event.**See Also** "Events" on page 18, **directIO** Method

## ErrorEvent

**Interface** `jpos.events.ErrorListener`

**Method** `errorOccurred (ErrorEvent e);`

**Description** Notifies the application that an error occurred at the MSR device.

**Properties** This event contains the following properties:

Name	Type	Description
<i>ErrorCode</i>	<i>int</i>	Error Code causing the error event. See list of <i>ErrorCodes</i> on page 16.
<i>ErrorCodeExtended</i>	<i>int</i>	Extended Error Code causing the error event. If <i>ErrorCode</i> is JPOS_E_EXTENDED, then see values below. Otherwise, it may contain a Service-specific value.
<i>ErrorLocus</i>	<i>int</i>	Location of the error. See values below.
<i>ErrorResponse</i>	<i>int</i>	Error response, whose default value may be overridden by the application (i.e., this property is settable). See values below.

If the **ErrorReportingType** property is MSR\_ERT\_CARD and *ErrorCode* is JPOS\_E\_EXTENDED, then *ErrorCodeExtended* has one of the following values:

Value	Meaning
JPOS_EMSR_START	Start sentinel error.
JPOS_EMSR_END	End sentinel error.
JPOS_EMSR_PARITY	Parity error.
JPOS_EMSR_LRC	LRC error.

If the **ErrorReportingType** property is MSR\_ERT\_TRACK, and *ErrorCode* is JPOS\_E\_EXTENDED, then *ErrorCodeExtended* contains Track-level status, broken down as follows:

High Word		Low Word	
High Byte	Low Byte	High Byte	Low Byte
Track4	Track 3	Track 2	Track 1

Where Each of the track status bytes has one of the following values:

Value	Meaning
JPOS_SUCCESS	No error.
JPOS_EMSR_START	Start sentinel error.
JPOS_EMSR_END	End sentinel error.

JPOS\_EMSR\_PARITY Parity error.

JPOS\_EMSR\_LRC LRC error.

JPOS\_E\_FAILURE Other or general error.

The *ErrorLocus* parameter has one of the following values:

Value	Meaning
JPOS_EL_INPUT	Error occurred while gathering or processing event-driven input. No input data is available.
JPOS_EL_INPUT_DATA	Error occurred while gathering or processing event-driven input, and some previously buffered data is available.

The application's error event listener may change *ErrorResponse* to one of the following values:

Value	Meaning
JPOS_ER_CLEAR	Clear the buffered input data. The error state is exited. Default when locus is JPOS_EL_INPUT.
JPOS_ER_CONTINUEINPUT	Use only when locus is JPOS_EL_INPUT_DATA. Acknowledges the error and directs the Device to continue processing. The Device remains in the error state and will deliver additional <b>DataEvents</b> as directed by the <b>DataEventEnabled</b> property. When all input has been delivered and <b>DataEventEnabled</b> is again set to true, then another <b>ErrorEvent</b> is delivered with locus JPOS_EL_INPUT. Default when locus is JPOS_EL_INPUT_DATA.

**Remarks** Enqueued when an error is detected while trying to read MSR data. This event is not delivered until **DataEventEnabled** is true, so that proper application sequencing occurs.

If the **ErrorReportingType** property is MSR\_ERT\_CARD, then the track that caused the fault cannot be determined. The track data properties are not changed.

If the **ErrorReportingType** property is MSR\_ERT\_TRACK, then the *ErrorCode* and *ErrorCodeExtended* properties may indicate the track-level status. Also, the track data properties are updated as with **DataEvent**, with the properties for the track or tracks in error set to empty strings.

Unlike **DataEvent**, individual track lengths are not reported. However, the application can determine their lengths by getting the length of each of the **TrackData** properties.

Also, since this is an `ErrorEvent` (even though it is reporting partial data), the `DataCount` property is not incremented and the Control remains enabled, regardless of the `AutoDisable` property value.

**See Also** “Device Behavior Model” on page 395, **ErrorReportingType** Property

## StatusUpdateEvent

**Interface** `jpos.events.StatusUpdateListener`

**Method** `statusUpdateOccurred (StatusUpdateEvent e);`

**Description** Notifies the application that there is a change in the power status of a MSR device.

**Properties** This event contains the following property:

Property	Type	Description
<code>Status</code>	<code>int</code>	Reports a change in the power state of a MSR device.

*Note that Release 1.3* added Power State Reporting with additional *Power reporting* **StatusUpdateEvent** values. See “StatusUpdateEvent” description on page 78.

**Remarks** Enqueued when the MSR device detects a power state change.

**See Also** “Events” on page 18





# PIN Pad

## Summary

### Properties

<i>Common</i>	<i>Ver</i>	<i>Type</i>	<i>Access</i>	<i>May Use After</i>
<b>AutoDisable</b>	1.3	boolean	R/W	<i>Not Supported</i>
<b>CapPowerReporting</b>	1.3	int	R	open
<b>CheckHealthText</b>	1.3	String	R	open
<b>Claimed</b>	1.3	boolean	R	open
<b>DataCount</b>	1.3	int	R	open
<b>DataEventEnabled</b>	1.3	boolean	R/W	open
<b>DeviceEnabled</b>	1.3	boolean	R/W	open & claim
<b>FreezeEvents</b>	1.3	boolean	R/W	open
<b>OutputID</b>	1.3	int	R	open
<b>PowerNotify</b>	1.3	int	R/W	open
<b>PowerState</b>	1.3	int	R	open
<b>State</b>	1.3	int	R	--
<b>DeviceControlDescription</b>	1.3	String	R	--
<b>DeviceControlVersion</b>	1.3	int	R	--
<b>DeviceServiceDescription</b>	1.3	String	R	open
<b>DeviceServiceVersion</b>	1.3	int	R	open
<b>PhysicalDeviceDescription</b>	1.3	String	R	open
<b>PhysicalDeviceName</b>	1.3	String	R	open

**Properties (Continued)**

<i>Specific</i>	<i>Ver</i>	<i>Type</i>	<i>Access</i>	<i>May Use After</i>
<b>CapMACCalculation</b>	1.3	boolean	R	open
<b>CapDisplay</b>	1.3	int	R	open
<b>CapLanguage</b>	1.3	int	R	open
<b>CapKeyboard</b>	1.3	boolean	R	open
<b>CapTone</b>	1.3	boolean	R	open
<b>AvailablePromptsList</b>	1.3	String	R	open
<b>Prompt</b>	1.3	int	R/W	open
<b>AvailableLanguagesList</b>	1.3	String	R	open
<b>PromptLanguage</b>	1.3	String	R/W	open
<b>AccountNumber</b>	1.3	String	R/W	open
<b>Amount</b>	1.3	long	R/W	open
<b>MerchantID</b>	1.3	String	R/W	open
<b>TerminalID</b>	1.3	String	R/W	open
<b>Track1Data</b>	1.3	byte[]	R/W	open
<b>Track2Data</b>	1.3	byte[]	R/W	open
<b>Track3Data</b>	1.3	byte[]	R/W	open
<b>Track4Data</b>	1.5	byte[]	R/W	open
<b>TransactionType</b>	1.3	String	R/W	open
<b>MinimumPINLength</b>	1.3	int	R/W	open
<b>MaximumPINLength</b>	1.3	int	R/W	open
<b>PINEntryEnabled</b>	1.3	boolean	R	open
<b>EncryptedPIN</b>	1.3	String	R	open
<b>AdditionalSecurityInformaion</b>	1.3	String	R	open

**Methods***Common*

	<i>Ver</i>	<i>May Use After</i>
<b>open</b>	1.3	--
<b>close</b>	1.3	open
<b>claim</b>	1.3	open
<b>release</b>	1.3	open & claim
<b>checkHealth</b>	1.3	open, claim, & enable
<b>clearInput</b>	1.3	open, claim, & enable
<b>clearOutput</b>	1.3	<i>Not Supported</i>
<b>directIO</b>	1.3	open

*Specific*

<b>beginEFTTransaction</b>	1.3	open, claim, & enable
<b>endEFTTransaction</b>	1.3	beginEFTTransaction
<b>enablePINEntry</b>	1.3	beginEFTTransaction
<b>computeMAC</b>	1.3	beginEFTTransaction
<b>verifyMAC</b>	1.3	beginEFTTransaction
<b>updateKey</b>	1.3	beginEFTTransaction

**Events**

<i>Name</i>	<i>Ver</i>	<i>May Occur After</i>
<b>DataEvent</b>	1.3	open, claim, & enable
<b>DirectIOEvent</b>	1.3	open & claim
<b>ErrorEvent</b>	1.3	open, claim, & enable
<b>OutputCompleteEvent</b>	1.3	<i>Not Supported</i>
<b>StatusUpdateEvent</b>	1.3	open, claim, & enable

## General Information

The PIN Pad Control's class name is "jpos.PINpad".  
The device constants are contained in the class "jpos.PINpadConst".  
See "Package Structure" on page 40.

*This device was added in JavaPOS Release 1.3.*

A PIN Pad

- Provides a mechanism for customers to perform PIN Entry
- Acts as a cryptographic engine for communicating with an EFT Transaction Host.

A PIN Pad will perform these functions by implementing one or more PIN Pad Management Systems. A PIN Pad Management System defines the manner in which the PIN Pad will perform functions such as PIN Encryption, Message Authentication Code calculation, and Key Updating. Examples of PIN Pad Management Systems include: Master-Session, DUKPT, APACS40, HGEPOS, AS2805, and JDEBIT2 along with many others.

### Capabilities

The PIN Pad Control has the following minimal capability:

- Accepts a PIN Entry at its keyboard and provide an Encrypted PIN to the application.

The PIN Pad Control may have the following additional capabilities:

- Computes Message Authentication Codes.
- Performs Key Updating in accordance with the selected PIN Pad Management System.
- Supports multiple PIN Pad Management Systems.
- Allows use of the PIN Pad Keyboard, Display, & Tone Generator for application usage. If one or more of these features are available, then the application opens and uses the associated POS Keyboard, Line Display, or Tone Indicator Device Controls.

## Features Not Supported

This specification does not include support for the following:

- **Initial Key Loading.** This operation usually requires downloading at least one key in the clear and must be done in a secure location (typically either the factory or at a Financial Institution). Thus, support for initial key loading is outside the scope of this specification. However, this specification does include support for updating keys while a PIN Pad unit is installed at a retail site.
- **Full EFT functionality.** This specification addresses the functionality of a PIN Pad that is used solely as a peripheral device by an Electronic Funds Transfer application. It specifically does not define the functionality of an Electronic Funds Transfer application that might execute within an intelligent PIN Pad. This specification does not include support for applications in which the PIN Pad application determines that a message needs to be transmitted to the EFT Transaction Host. *Consequently, this specification will not apply in Canada, Germany, Netherlands, and possibly other countries. It also does not apply to PIN Pads in which the vendor has chosen to provide EFT Functionality in the PIN Pad.*

**Smartcard Reader.** Some PIN Pad devices will include a Smartcard reader. Support for this device may be included in a future revision of this specification. In the interim, the **directIO** method could be used to control such added functionality.

## Note on Terminology

For the PIN Pad device, clarification of the terminology used to describe the data exchange with the device is necessary. “Hex-ASCII” is used to indicate that the “standard” representation of bytes as hexadecimal ASCII characters is used. For instance, the byte stream {0x15, 0xC7, 0xF0} would be represented in hex-ASCII as “15C7F0”.

## Model

A PIN Pad performs encryption functions under control of a PIN Pad Management System. Some PIN Pads will support multiple PIN Pad Management Systems. Some PIN Pad Management Systems support multiple keys (sets) for different EFT Transaction Hosts. Thus, for each EFT transaction, the application will need to select the PIN Pad Management System and EFT Transaction Host to be used. Depending on the PIN Pad Management System, one or more EFT transaction parameters will need to be provided to the PIN Pad for use in the encryption functions. The application should set the value of **ALL** EFT Transaction parameter properties to enable easier migration to EFT Transaction Hosts that require a different PIN Pad Management System.

After opening, claiming, and enabling the Device Control, an application should use the following general scenario for each EFT Transaction.

- Set the EFT transaction properties (**AccountNumber**, **Amount**, **MerchantID**, **TerminalID**, **Track1Data**, **Track2Data**, **Track3Data**, **Track4data** and **TransactionType**) and then call the **beginEFTTransaction** method. This will initialize the Device to perform the encryption functions for the EFT transaction.
- If PIN Entry is required, call the **enablePINEntry** method. Then set the **DataEventEnabled** property and wait for the **DataEvent**.
- If Message Authentication Codes are required, call the **computeMAC** and **verifyMAC** methods as needed.
- Call the **endEFTTransaction** method to notify the Device that all operations for the EFT transaction have been completed.

This specification supports two models of usage of the PIN Pad display. The **CapDisplay** property indicates one of the following models:

- an application has complete control of the text that is to be displayed. For this model, there is an associated Line Display Control that is used by the application to interact with the display.
- an application cannot supply the text to be displayed. Instead, it can only select from a list of pre-defined messages to be displayed. For this model, there is a set of PIN Pad properties that are used to control the display.

## Device Sharing

The PIN Pad is an exclusive-use device, as follows:

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

## Properties

### AccountNumber Property R/W

<b>Type</b>	<b>String</b>
<b>Remarks</b>	Holds the account number to be used for the current EFT transaction. The application must set this property before calling the <code>beginEFTTransaction</code> method.
<b>Errors</b>	A <code>JposException</code> may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s **ErrorCode** property are:

<b>Value</b>	<b>Meaning</b>
JPOS_E_ILLEGAL	An attempt was made to change this property after the <code>beginEFTTransaction</code> method has been called.

### AdditionalSecurityInformation Property R

<b>Type</b>	<b>String</b>
<b>Remarks</b>	Holds additional security/encryption information when a <b>DataEvent</b> is delivered. This property will be formatted as a Hex-ASCII string. The information content and internal format of this string will vary among PIN Pad Management Systems. For example, if the PIN Pad Management System is DUKPT, then this property will contain the “PIN Pad sequence number”. If the PIN Entry was cancelled, this property will contain the empty string.
<b>Errors</b>	A <code>JposException</code> may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.



## Amount Property R/W

<b>Type</b>	<b>long</b>
<b>Remarks</b>	Holds the amount of the current EFT transaction. The application must set this property before calling the <b>beginEFTTransaction</b> method. This property is a monetary value stored using an implied four decimal places. For example, an actual value of 12345 represents 1.2345.
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s **ErrorCode** property are:

<b>Value</b>	<b>Meaning</b>
JPOS_E_ILLEGAL	An attempt was made to change this property after the <b>beginEFTTransaction</b> method has been called.

## AvailableLanguagesList Property R

<b>Type</b>	<b>String</b>
<b>Remarks</b>	<p>Holds a semi-colon separated list of a set of “language definitions” that are supported by the pre-defined prompts in the PIN Pad. A “language definition” consists of an ISO-639 language code and an ISO-3166 country code (as also used in the Java Locale class). The two codes are comma separated.</p> <p>For example,. the string ”EN,US;FR,CAN,” represents two supported language definitions: US English and Canadian French where the variant of French used will be dependent on what is available on the device</p> <p>If <b>CapLanguage</b> is PPAD_LANG_NONE, then this property will be the empty string.</p> <p>This property is initialized by the <b>open</b> method.</p>
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
<b>See Also</b>	<b>PromptLanguage</b> Property

## AvailablePromptsList Property R

<b>Type</b>	<b>String</b>
<b>Remarks</b>	Holds a comma-separated string representation of the supported values for the <b>Prompt</b> property.

The full set of supported **Prompt** values are shown below:

Value Name (Value)	Meaning
PPAD_MSG_ENTERPIN (1)	Enter pin number on the PIN Pad.
PPAD_MSG_PLEASEWAIT (2)	The system is processing. Wait.
PPAD_MSG_ENTERVALIDPIN (3)	The pin that was entered is not correct. Enter the correct pin number.
PPAD_MSG_RETRIESEXCEEDED (4)	The user has failed to enter the correct pin number and the maximum number of attempts has been exceeded.
PPAD_MSG_APPROVED (5)	The request has been approved.
PPAD_MSG_DECLINED (6)	The EFT Transaction Host has declined to perform the requested function.
PPAD_MSG_CANCELED (7)	The request is cancelled.
PPAD_MSG_AMOUNTOK (8)	Enter Yes/No to approve the amount.
PPAD_MSG_NOTREADY (9)	PIN Pad is not ready for use.
PPAD_MSG_IDLE (10)	The System is Idle.
PPAD_MSG_SLIDE_CARD (11)	Slide card through the integrated MSR.
PPAD_MSG_INSERTCARD (12)	Insert (smart)card.
PPAD_MSG_SELECTCARDTYPE (13)	Select the card type (typically credit or debit).

Values 1000 and above are reserved for Device Service defined values.

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

**Errors** A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

## CapDisplay Property R

Type        **int**

Remarks    Defines the operations that the application may perform on the PIN Pad display.

Value	Meaning
PPAD_DISP_UNRESTRICTED	The application can use the PIN Pad display in an unrestricted manner to display messages. In this case, an associated Line Display Device Control is the interface to the PIN Pad display. The application must call Line Display methods to manipulate the display.
PPAD_DISP_PINRESTRICTED	The application can use the PIN Pad display in an unrestricted manner except during PIN Entry. The PIN Pad will display a pre-defined message during PIN Entry. If an attempt is made to use the associated Line Display Device Control while PIN Entry is enabled, the Line Display Control will throw a JposException with an associated <i>ErrorCode</i> of JPOS_E_BUSY.
PPAD_DISP_RESTRICTED_LIST	The application cannot specify the text of messages to display. It can only select from a list of pre-defined messages. There is no associated Line Display Device Control.
PPAD_DISP_RESTRICTED_ORDER	The application cannot specify the text of messages to display. It can only select from a list of pre-defined messages. The selections must occur in a pre-defined acceptable order. There is no associated Line Display Device Control.
PPAD_DISP_NONE	The PIN Pad does not have the PIN Pad display.

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

**Errors**        A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

## CapLanguage Property R

**Type**        **int**

**Remarks**    Defines the capabilities that the application has to select the language of pre-defined messages (e.g. English, French, Arabic etc.).

This property has one of the following values:

<b>Value</b>	<b>Meaning</b>
PPAD_LANG_NONE	The PIN Pad supports no pre-defined prompt messages. The property will be set to this value if <b>CapDisplay</b> = PPAD_DISP_UNRESTRICTED. Any attempt to set the value of the <b>PromptLanguage</b> property will cause a JposException to be thrown with the associated <i>ErrorCode</i> of JPOS_E_ILLEGAL.
PPAD_LANG_ONE	The PIN Pad supports pre-defined prompt messages in one language. Any attempt to set the value of the <b>PromptLanguage</b> property to other than the default value will cause JposException to be thrown with the associated <i>ErrorCode</i> of JPOS_E_ILLEGAL.
PPAD_LANG_PINRESTRICTED	The PIN Pad cannot change prompt languages during PIN Entry. The application must set the desired value into the <b>PromptLanguage</b> property before calling enablePINEntry. Any attempt to set the value of the <b>PromptLanguage</b> while <b>PINEntryEnabled</b> is true will cause a JposException to be thrown with the associated <i>ErrorCode</i> of JPOS_E_BUSY.
PPAD_LANG_UNRESTRICTED	The application can change the language of pre-defined prompt messages at anytime. The currently displayed message will change immediately.

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

**Errors**        A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

**See Also**     **PromptLanguage** Property.

### CapMACCalculation Property R

Type	<b>boolean</b>
Remarks	If true, the PIN Pad supports MAC calculation.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapKeyboard Property R

Type	<b>boolean</b>
Remarks	If true, the application can use the PIN Pad to obtain input. The application will use an associated POS Keyboard Device Control as the interface to the PIN Pad keyboard. Note that the associated POS Keyboard Control is effectively disabled while <b>PINEntryEnabled</b> is true.  If false, the application cannot obtain input directly from the PIN Pad keyboard.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapTone Property R

Type	<b>boolean</b>
Remarks	If true, the PIN Pad has a Tone Indicator. The Tone Indicator may be accessed by use of an associated Tone Indicator Control. If false, there is no Tone Indicator.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

## EncryptedPIN Property R

<b>Type</b>	<b>String</b>
<b>Remarks</b>	<p>Holds the value of the Encrypted PIN after a DataEvent. This property will be formatted as a hexadecimal ASCII string. Each character is in the ranges ‘0’ through ‘9’ or ‘A’ through ‘F’. Each pair of characters is the hexadecimal representation for a byte.</p> <p>For example, if the first four characters are “12FA”, then the first two bytes of the PIN are 12 hexadecimal (18) and FA hexadecimal (250).</p> <p>If the PIN Entry was cancelled, this property will contain the empty string.</p>
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

## MaximumPINLength Property R/W

<b>Type</b>	<b>int</b>
<b>Remarks</b>	<p>Holds the maximum acceptable number of digits in a PIN. This property will be set to a default value by the <b>open</b> method. If the application wishes to change this property, it should be set before the <b>enablePINEntry</b> method is called. Note that in some implementations, this value cannot be changed by the application.</p>
<b>Errors</b>	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>

Some possible values of the exception’s *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
JPOS_E_ILLEGAL	An attempt was made to change this property after the <b>enablePINEntry</b> method has been called.

**MerchantID Property R/W**

<b>Type</b>	<b>String</b>
<b>Remarks</b>	Holds the Merchant ID, as it is known to the EFT Transaction Host. The application must set this property before calling the <b>beginEFTTransaction</b> method.
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
JPOS_E_ILLEGAL	An attempt was made to change this property after the <b>beginEFTTransaction</b> method has been called.

**MinimumPINLength Property R/W**

<b>Type</b>	<b>int</b>
<b>Remarks</b>	Holds the minimum acceptable number of digits in a PIN. This property will be set to a default value by the open method. If the application wishes to change this property, it should be set before the <b>enablePINEntry</b> method is called. Note that in some implementations, this value cannot be changed by the application.
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
JPOS_E_ILLEGAL	An attempt was made to change this property after the <b>enablePINEntry</b> method has been called.

**PINEntryEnabled Property R**

<b>Type</b>	<b>boolean</b>
<b>Remarks</b>	If true, the PIN entry operation is enabled. It is set when the <b>enablePINEntry</b> method is called. It will be set to false when the user has completed the PIN Entry operation or when the <b>endEFTTransaction</b> method has completed.
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

**Prompt Property R/W**

<b>Type</b>	<b>int</b>
<b>Remarks</b>	<p>Holds the identifier of a pre-defined message to be displayed on the PIN Pad. This property is used if <b>CapDisplay</b> is PPAD_DISP_RESTRICTED_LIST or PPAD_DISP_RESTRICTED_ORDER. It is also used during PIN Entry if <b>CapDisplay</b> has a value of PPAD_DISP_PINRESTRICTED. The <b>AvailablePromptsList</b> property lists the possible values for this property.</p> <p>This property is initialized by the <b>open</b> method.</p>
<b>Errors</b>	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>

Some possible values of the exception’s *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
JPOS_E_ILLEGAL	<p>One of the following has occurred:</p> <p>An attempt was made to set the property to a value that is not supported by the PIN Pad Device Service.</p> <p>An attempt was made to select prompt messages in an unacceptable order (if <b>CapDisplay</b> is PPAD_DISP_RESTRICTED_ORDER).</p>

**See Also**     **PromptLanguage** Property



## PromptLanguage Property R/W

<b>Type</b>	<b>String</b>						
<b>Remarks</b>	<p>Holds the “language definition” for the message to be displayed (as specified by the <b>Prompt</b> property). This property is used if the <b>Prompt</b> property is being used. The exact effect of changing this property depends on the value of <b>CapLanguage</b>.</p> <p>A “language definition” consists of an ISO-639 language code and an ISO-3166 country code (as also used in the Java Locale class). The two codes are comma separated.</p> <p>The country code is optional and implies that the application does not care which country variant of the language is used.</p> <p>For example, the string ”EN,US” represents a US English language definition, the string ”FR,” represents a French language definition where the variant of French used will be dependent on what is available on the device.</p> <p>The property is initialized to a default value by the open method.</p>						
<b>Errors</b>	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p> <p>Some possible values of the exception’s <i>ErrorCode</i> property are:</p> <table> <thead> <tr> <th><b>Value</b></th> <th><b>Meaning</b></th> </tr> </thead> <tbody> <tr> <td>JPOS_E_ILLEGAL</td> <td> <p>One of the following errors occurred:</p> <p>An attempt was made to set the property to a value that is not supported by the PIN Pad Device Service.</p> <p><b>CapLanguage</b> is PPAD_LANG_NONE. and an attempt was made to set the value of this property.</p> <p><b>CapLanguage</b> is PPAD_LANG_ONE and an attempt to was made to set the value of this property to other than the default value.</p> </td> </tr> <tr> <td>JPOS_E_BUSY</td> <td> <p><b>CapLanguage</b> is PPAD_LANG_PINRESTRICTED and <b>PINEntryEnabled</b> is true.</p> </td> </tr> </tbody> </table>	<b>Value</b>	<b>Meaning</b>	JPOS_E_ILLEGAL	<p>One of the following errors occurred:</p> <p>An attempt was made to set the property to a value that is not supported by the PIN Pad Device Service.</p> <p><b>CapLanguage</b> is PPAD_LANG_NONE. and an attempt was made to set the value of this property.</p> <p><b>CapLanguage</b> is PPAD_LANG_ONE and an attempt to was made to set the value of this property to other than the default value.</p>	JPOS_E_BUSY	<p><b>CapLanguage</b> is PPAD_LANG_PINRESTRICTED and <b>PINEntryEnabled</b> is true.</p>
<b>Value</b>	<b>Meaning</b>						
JPOS_E_ILLEGAL	<p>One of the following errors occurred:</p> <p>An attempt was made to set the property to a value that is not supported by the PIN Pad Device Service.</p> <p><b>CapLanguage</b> is PPAD_LANG_NONE. and an attempt was made to set the value of this property.</p> <p><b>CapLanguage</b> is PPAD_LANG_ONE and an attempt to was made to set the value of this property to other than the default value.</p>						
JPOS_E_BUSY	<p><b>CapLanguage</b> is PPAD_LANG_PINRESTRICTED and <b>PINEntryEnabled</b> is true.</p>						
<b>See Also</b>	<b>CapLanguage</b> Property, <b>AvailableLanguagesList</b> Property						

**TerminalID Property R/W**

<b>Type</b>	<b>String</b>
<b>Remarks</b>	Holds the terminal ID, as it is known to the EFT Transaction Host. The application must set this property before calling the <b>beginEFTTransaction</b> method.
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

Some possible values of the exception's *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
JPOS_E_ILLEGAL	An attempt was made to change this property after the <b>beginEFTTransaction</b> method has been called.

**Track1Data Property R/W**

<b>Type</b>	<b>byte[]</b>
<b>Remarks</b>	Holds either the decoded track 1 data from the previous card swipe or an empty array. An empty array indicates that the track was not physically read. The application must set this property before calling the <b>beginEFTTransaction</b> method.
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

Some possible values of the exception's *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
JPOS_E_ILLEGAL	An attempt was made to change this property after the <b>beginEFTTransaction</b> method has been called.

## Track2Data Property R/W

Type	byte[]
Remarks	Holds either the decoded track 2 data from the previous card swipe or an empty array. An empty array indicates that the track was not physically read. The application must set this property before calling the <b>beginEFTTransaction</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_ILLEGAL	An attempt was made to change this property after the <b>beginEFTTransaction</b> method has been called.

**Track3Data Property R/W**

<b>Type</b>	<b>byte[]</b>
<b>Remarks</b>	Holds either the decoded track 3 data from the previous card swipe or an empty array. An empty array indicates that the track was not physically read. The application must set this property before calling the <b>beginEFTTransaction</b> method
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
JPOS_E_ILLEGAL	An attempt was made to change this property after the <b>beginEFTTransaction</b> method has been called.

**Track4Data Property R/W****Added in Release 1.5**

<b>Type</b>	<b>byte[]</b>
<b>Remarks</b>	Holds either the decoded track 4 (JIS-II) data from the previous card swipe or an empty array. An empty array indicates that the track was not physically read. The application must set this property before calling the <b>beginEFTTransaction</b> method.  To maintain compatibility with previous versions, the Control may also continue to store the JIS-II data in another <b>TracknData</b> property. However, it should be noted that to ensure application portability, <b>Track4Data</b> should be used to access JIS-II data.
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
JPOS_E_ILLEGAL	An attempt was made to change this property after the <b>beginEFTTransaction</b> method has been called.

**TransactionType Property R/W**

**Type**      **int**

**Remarks**      Holds the type of the current EFT transaction. The application must set this property before calling the **beginEFTTransaction** method.

This property has one of the following values:

<b>Value</b>	<b>Meaning</b>
PPAD_TRANS_DEBIT	Debit (decrease) the specified account
PPAD_TRANS_CREDIT	Credit (increase) the specified account.
PPAD_TRANS_INQ	(Balance) Inquiry
PPAD_TRANS_RECONCILE	Reconciliation/Settlement
PPAD_TRANS_ADMIN	Administrative Transaction

**Errors**      A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
JPOS_E_ILLEGAL	An attempt was made to change this property after the <b>beginEFTTransaction</b> method has been called.

## Methods

### beginEFTTransaction Method

**Syntax**      **void beginEFTTransaction (String PINPadSystem, int transactionHost) throws JposException ;**

Parameter	Description
<i>PINPadSystem</i>	Name of the desired PIN Pad Management System.(see below). The Device Service may support other PIN Pad Management systems.
<i>transactionHost</i>	Identifies the particular EFT Transaction Host to be used for this transaction.

The *PINPadSystem* parameter has one of the following values:

Value	Meaning
“M/S”	Master/Session. (USA, Latin America)
“DUKPT”	Derived Unique Key Per Transaction (USA, Latin America)
“APACS40”	Standard 40 (UK and other countries)
“AS2805”	Australian Standard 2805
“HGEPOS”	(Italian)
“JDEBIT2”	Japan Debit 2

**Remarks**      Initiates the beginning of an EFT Transaction. The Device will perform initialization functions (such as computing session keys). No other PIN Pad functions can be performed until this method is called.

**Errors**          A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

### computeMAC Method

**Syntax**      **void computeMAC (String inMsg, String[] outMsg) throws JposException;**

Parameter	Description
<i>inMsg</i>	The message that the application intends to send to an EFT Transaction Host
<i>outMsg</i>	Contains the result of applying the MAC calculation to <i>inMsg</i> . This output parameter will contain a reformatted message that may actually be transmitted to an EFT Transaction Host.

**Remarks** Computes a MAC value and appends it to the designated message. Depending on the selected PIN Pad Management System, the PIN Pad may also insert other fields into the message. Note that this method cannot be used while PIN Pad input (PIN Entry) is enabled.

**Errors** A `JposException` may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_DISABLED	A <b>beginEFTTransaction</b> method has not been performed.
JPOS_E_BUSY	PINEntryEnabled is true. The PIN Pad cannot perform a MAC calculation during PIN Entry.

### enablePINEntry Method

**Syntax** `void enablePINEntry () throws JposException;`

**Remarks** Enables PIN Entry at the PIN Pad device. When this method is called, the **PINEntryEnabled** property will be changed to true. If the PIN Pad uses pre-defined prompts for PIN Entry, then the Prompt property will be changed to `PPAD_MSG_ENTERPIN`.

When the user has completed the PIN entry operation (either by entering their PIN or by hitting Cancel), the **PINEntryEnabled** property will be changed to false. A `DataEvent` will be delivered to provide the encrypted PIN to the application when **DataEventEnabled** is set to true. Note that any data entered at the PIN Pad while **PINEntryEnabled** is true will be supplied in encrypted form and will NOT be provided to any associated Keyboard Device Control.

**Errors** A `JposException` may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_DISABLED	A <b>beginEFTTransaction</b> method has not been performed.

**endEFTTransaction Method**

**Syntax**      **void endEFTTransaction (int *completionCode*) throws JposException;**

*completionCode* has one of the following values:

Value	Meaning
PPAD_EFT_NORMAL	The EFT transaction completed normally. Note that this does not mean that the EFT transaction was approved. It merely means that the proper sequence of messages was transmitted and received.
PPAD_EFT_ABNORMAL	The proper sequence of messages was not transmitted & received.

**Remarks**      Ends an EFT Transaction. The Device will perform termination functions (such as computing next transaction keys).

**Errors**        A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

**updateKey Method**

**Syntax**      **void updateKey (int *keyNum*, String *key*) throws JposException;**

Parameter	Description
<i>keyNum</i>	A key number.
<i>key</i>	A Hex-ASCII value for a new key.

**Remarks**      Provides a new encryption key to the PIN Pad. It is used only for those PIN Pad Management Systems in which new key values are sent to the terminal as a field in standard messages from the EFT Transaction Host.

**Errors**        A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_ILLEGAL	One of the following conditions occurred:  The selected PIN Pad Management System does not support this function.  The <i>keyNum</i> specifies an unacceptable key number.  The <i>key</i> contains a bad key (not Hex-ASCII or wrong length or bad parity).



## verifyMAC Method

**Syntax**      **void verifyMAC (String *message*) throws JposException;**

The *message* contains a message received from an EFT Transaction Host.

**Remarks**      Verifies the MAC value in a message received from an EFT Transaction Host. This method throws a JposException if it cannot verify the message. Note that this method cannot be used while PIN Entry is enabled.

**Errors**          A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
JPOS_E_DISABLED	A <b>beginEFTTransaction</b> method has not been performed.
JPOS_E_BUSY	PINEntryEnabled is true. The PIN Pad cannot perform a MAC verification during PIN Entry.

## Events

### DataEvent

**Interface** `jpos.events.DataListener`

**Method** `dataOccurred (DataEvent e);`

**Description** Notifies the application when a PIN Entry operation has completed.

**Properties** This event contains the following property:

Property	Type	Description
<i>Status</i>	<i>int</i>	See below.

The *Status* property has one of the following values:

Value	Meaning
PPAD_SUCCESS	PIN Entry has occurred and values have been stored into the <b>EncryptedPIN</b> and <b>AdditionalSecurityInformation</b> properties.
PPAD_CANCEL	The user hit the cancel button on the PIN Pad.
PPAD_TIMEOUT	A timeout condition occurred in the PIN Pad. (Not all PIN Pads will report this condition)

### DirectIOEvent

**Interface** `jpos.events.DirectIOListener`

**Method** `directIOOccurred (DirectIOEvent e);`

**Description** Provides Device Service information directly to the application. This event provides a means for a vendor-specific PIN Pad Device Service to provide events to the application that are not otherwise supported by the Device Control.

**Properties** This event contains the following properties:

Property	Type	Description
<i>EventNumber</i>	<i>int</i>	Event number whose specific values are assigned by the Device Service event.
<i>Data</i>	<i>int</i>	Additional numeric data. Specific alues vary by the <i>EventNumber</i> and the Device Service. This property is settable.
<i>Object</i>	<i>Object</i>	Additional data whose usage varies by the <i>EventNumber</i> and Device Service. This property is settable.

- Remarks** This event is to be used only for those types of vendor specific functions that are not otherwise described as part of the JavaPOS standard. Use of this event may restrict the application program from being used with other vendor's PIN Pad devices which may not have any knowledge of the Device Service's need for this event.
- See Also** "Events" on page 18

## ErrorEvent

- Interface** **jpos.events.ErrorListener**
- Method** **errorOccurred (ErrorEvent e);**
- Description** Notifies the application that an error was detected while trying to perform a PIN encryption function.
- Properties** This event contains the following properties:
- | Property                 | Type       | Description   |
|--------------------------|------------|---|
| <i>ErrorCode</i>         | <i>int</i> | Error code causing the error event. See list of <i>ErrorCodes</i> on page 16.   |
| <i>ErrorCodeExtended</i> | <i>int</i> | Extended Error Code causing the error event. If <i>ErrorCode</i> is JPOS_E_EXTENDED, then see value below. Otherwise it may contain a Service-specific value. |
| <i>ErrorLocus</i>        | <i>int</i> | Location of the error. See values below.  |
| <i>ErrorResponse</i>     | <i>int</i> | Error response, whose default values may be overridden by the application (i.e. this property is settable). See values below.                                 |

The *ErrorCodeExtended* property has the following value:

Value	Meaning
PPAD_BAD_KEY	An Encryption Key is corrupted or missing.

The *ErrorLocus* property has the following value:

Value	Meaning
JPOS_EL_INPUT	Error occurred while gathering or processing event-driven input. No input data is available.

The application's error event listener may change *ErrorResponse* to the following value:

Value	Meaning
JPOS_ER_CLEAR	Clear the buffered input data. The error state is exited. Default when locus is JPOS_EL_INPUT.

**Remarks** More detailed diagnostic information may optionally be obtained using the **checkHealth** or **directIO** methods.

## StatusUpdateEvent

**Interface** `jpos.events.StatusUpdateListener`

**Method** `statusUpdateOccurred (StatusUpdateEvent e);`

**Description** Notifies the application that there is a change in the power status of a PIN Pad.

**Properties** This event contains the following property:

Property	Type	Description
<i>Status</i>	<i>int</i>	Reports a change in the power state of a PIN Pad.

*Note that Release 1.3* added Power State Reporting with additional *Power reporting StatusUpdateEvent values*. See "StatusUpdateEvent" description on page 78.

**Remarks** Enqueued when the PIN Pad detects a power state change.

**See Also** "Events" on page 18

# Pointcard Reader Writer

## Summary

### Properties

<i>Common</i>	<i>Ver<sup>a</sup></i>	<i>Type</i>	<i>Access</i>	<i>May Use After</i>
<b>AutoDisable</b>	1.5	boolean	R/W	<i>Not Supported</i>
<b>CapPowerReporting</b>	1.5	int	R	open
<b>CheckHealthText</b>	1.5	string	R	open
<b>Claimed</b>	1.5	boolean	R	open
<b>DataCount</b>	1.5	int	R	open
<b>DataEventEnabled</b>	1.5	boolean	R/W	open
<b>DeviceEnabled</b>	1.5	boolean	R/W	open & claim
<b>FreezeEvents</b>	1.5	boolean	R/W	open
<b>OutputID</b>	1.5	int	R	open
<b>PowerNotify</b>	1.5	int	R/W	open
<b>PowerState</b>	1.5	int	R	open
<b>State</b>	1.5	int	R	--
<b>DeviceControlDescription</b>	1.5	string	R	--
<b>DeviceControlVersion</b>	1.5	int	R	--
<b>DeviceServiceDescription</b>	1.5	string	R	open
<b>DeviceServiceVersion</b>	1.5	int	R	open
<b>PhysicalDeviceDescription</b>	1.5	string	R	open
<b>PhysicalDeviceName</b>	1.5	string	R	open

- a. The Point Card Reader Writer definition was introduced in the JavaPOS version 1.5.

**Properties (Continued)**

<i>Specific</i>	<i>Ver</i>	<i>Type</i>	<i>Access</i>	<i>May Use After</i>
<b>CapBold</b>	1.5	boolean	R	open
<b>CapCardEntranceSensor</b>	1.5	boolean	R	open
<b>CapCharacterSet</b>	1.5	int	R	open
<b>CapCleanCard</b>	1.5	boolean	R	open
<b>CapClearPrint</b>	1.5	boolean	R	open
<b>CapDhigh</b>	1.5	boolean	R	open
<b>CapDwide</b>	1.5	boolean	R	open
<b>CapDwideDhigh</b>	1.5	boolean	R	open
<b>CapItalic</b>	1.5	boolean	R	open
<b>CapLeft90</b>	1.5	boolean	R	open
<b>CapPrint</b>	1.5	boolean	R	open
<b>CapPrintMode</b>	1.5	boolean	R	open
<b>CapRight90</b>	1.5	boolean	R	open
<b>CapRotate180</b>	1.5	boolean	R	open
<b>CapTracksToRead</b>	1.5	int	R	open
<b>CapTracksToWrite</b>	1.5	int	R	open
<b>CardState</b>	1.5	int	R	open
<b>CharacterSet</b>	1.5	int	R/W	open, claim, & enable
<b>CharacterSetList</b>	1.5	string	R	open
<b>FontTypeFaceList</b>	1.5	string	R	open
<b>LineChars</b>	1.5	int	R	open, claim, & enable
<b>LineCharsList</b>	1.5	string	R	open
<b>LineHeight</b>	1.5	int	R	open, claim, & enable
<b>LineSpacing</b>	1.5	int	R	open, claim, & enable
<b>LineWidth</b>	1.5	int	R	open, claim, & enable
<b>MapMode</b>	1.5	int	R	open, claim, & enable
<b>MaxLine</b>	1.5	int	R	open, claim, & enable
<b>PrintHeight</b>	1.5	int	R	open, claim, & enable
<b>ReadState1</b>	1.5	int	R	open
<b>ReadState2</b>	1.5	int	R	open
<b>RecvLength1</b>	1.5	int	R	open, claim, & enable
<b>RecvLength2</b>	1.5	int	R	open, claim, & enable
<b>SidewaysMaxChars</b>	1.5	int	R	open
<b>SidewaysMaxLines</b>	1.5	int	R	open

**Properties (Continued)**

<i>Specific</i>	<i>Ver</i>	<i>Type</i>	<i>Access</i>	<i>May Use After</i>
<b>TracksToRead</b>	1.5	int	R/W	open, claim, & enable
<b>TracksToWrite</b>	1.5	int	R/W	open, claim, & enable
<b>Track1Data</b>	1.5	byte[]	R	open
<b>Track2Data</b>	1.5	byte[]	R	open
<b>Track3Data</b>	1.5	byte[]	R	open
<b>Track4Data</b>	1.5	byte[]	R	open
<b>Track5Data</b>	1.5	byte[]	R	open
<b>Track6Data</b>	1.5	byte[]	R	open
<b>WriteState1</b>	1.5	int	R	open
<b>WriteState2</b>	1.5	int	R	open
<b>Write1Data</b>	1.5	byte[]	R/W	open
<b>Write2Data</b>	1.5	byte[]	R/W	open
<b>Write3Data</b>	1.5	byte[]	R/W	open
<b>Write4Data</b>	1.5	byte[]	R/W	open
<b>Write5Data</b>	1.5	byte[]	R/W	open
<b>Write6Data</b>	1.5	byte[]	R/W	open

**Methods**

<i>Common</i>	<i>Ver</i>	<i>May Use After</i>
<b>open</b>	1.5	--
<b>close</b>	1.5	open
<b>claim</b>	1.5	open
<b>release</b>	1.5	open & claim
<b>checkHealth</b>	1.5	open, claim, & enable
<b>clearInput</b>	1.5	open & claim
<b>clearOutput</b>	1.5	open & claim
<b>directIO</b>	1.5	open
 <i>Specific</i>		
<b>beginInsertion</b>	1.5	open, claim, & enable
<b>beginRemoval</b>	1.5	open, claim, & enable
<b>cleanCard</b>	1.5	open, claim, & enable
<b>clearPrintWrite</b>	1.5	open, claim, & enable
<b>endInsertion</b>	1.5	open, claim, & enable
<b>endRemoval</b>	1.5	open, claim, & enable
<b>printWrite</b>	1.5	open, claim, & enable
<b>rotatePrint</b>	1.5	open, claim, & enable
<b>validateData</b>	1.5	open, claim, & enable

**Events**

<i>Name</i>	<i>Ver</i>	<i>May Occur After</i>
<b>DataEvent</b>	1.5	open, claim, & enable
<b>DirectIOEvent</b>	1.5	open & claim
<b>ErrorEvent</b>	1.5	open, claim, & enable
<b>OutputCompleteEvent</b>	1.5	open, claim, & enable
<b>StatusUpdateEvent</b>	1.3	open, claim, & enable



## General Information

The Point Card Reader Writer class name is “jpos.PointCardRW”.  
The device constants are contained in the class “jpos.PointCardRWConst”.  
See “Package Structure” on page 40.

*This device was added in JavaPOS Release 1.5.*

## Capabilities

The Point Card Reader Writer has the following capabilities.

- Both reading and writing of the point card magnetic data are possible.
- Supports reading and writing of data from up to 6 tracks.
- The data on the tracks is in a device specific format, see the device manual for specific definition. The data is usually in ASCII format.
- Supports point cards with or without a printing area. Actual printing support depends upon the capabilities of the device.
- Supports both card insertion and ejection.
- No special security capabilities (e.g., encryption) are supported.

## Model

The general model of Point Card Reader Writer is as follows:

- The Point Card Reader Writer reads all the magnetic stripes on a point card. The data length and reading information are placed in the property corresponding to the track.
- The Point Card Reader Writer follows the input model of event driven input during the card insertion processing. Also, writing to the printing area and the magnetic stripe follows the output model.

## Input Model

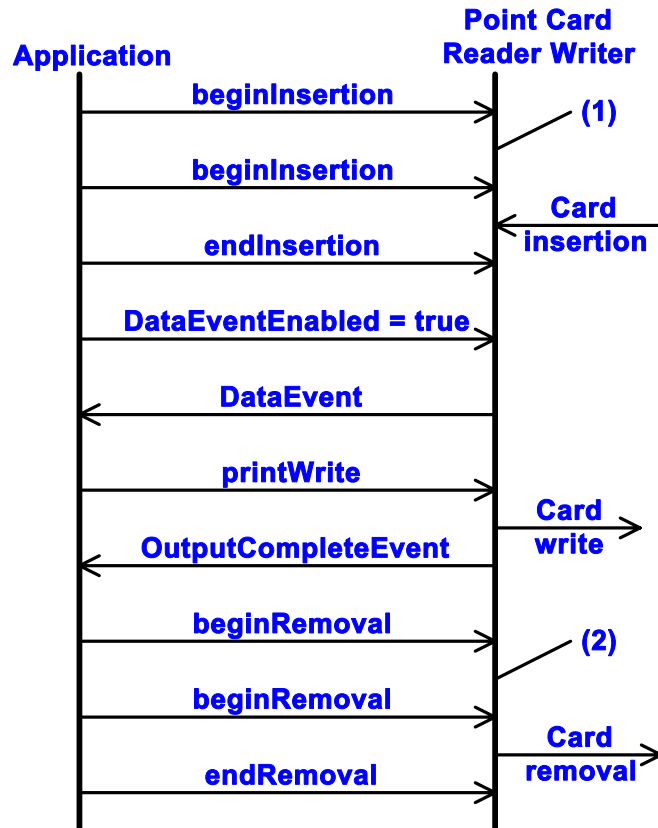
- An application must call **open** and **claim**, then set **DeviceEnabled** to true.
- When an application wants a card inserted, it calls the **beginInsertion** method, specifying a timeout value.
- If a card is not inserted before the timeout period elapses, the Point Card Reader Writer fires an exception.
- Even if a timeout occurs, the Point Card Reader Writer remains in insertion mode. If the application still wants a card inserted, it must call the **beginInsertion** method again.
- To exit insertion mode, either after a card was inserted or the application wishes to abort insertion, the application calls the **endInsertion** method.
- If there is a point card in the Point Card Reader Writer when **endInsertion** is called, the point card's data tracks are automatically read and a **DataEvent** is enqueued. When the application sets the **DataEventEnabled** property to true, the **DataEvent** will be delivered.
- If an error occurs while reading the point card's data tracks, an **ErrorEvent** is enqueued instead of a **DataEvent**. When the application sets the **DataEventEnabled** property to true, the **ErrorEvent** will be delivered.
- The application can obtain the current number of enqueued data events by reading the **DataCount** property.
- All enqueued but undelivered input may be deleted by calling the **clearInput** method.

## Output Model

- To write data to a card, the application calls the **printWrite** method. The ability to write data depends upon the capabilities of the device.
- The **printWrite** method is always performed asynchronously. All asynchronous output is performed on a first-in, first-out basis.
- When the application calls **printWrite**, the Point Card Reader Writer assigns a unique identification number for this request. This ID is stored in the property **OutputID**. The Point Card Reader Writer then either queues the request or starts its processing. Either way, the Point Card Reader Writer returns to the application quickly.
- When the **printWrite** method completes, an **OutputCompleteEvent** is delivered to the application. The **OutputID** associated with the completed request is passed in the **OutputCompleteEvent**.
- If the **printWrite** method fails during its processing, an **ErrorEvent** will be delivered to the application. If the application had multiple outstanding output requests, the **OutputID** of the request that failed can be determined by watching which requests have successfully completed by monitoring **OutputCompleteEvents**. The request that failed is the one that was issued immediately after the last request that successfully completed.
- All incomplete output requests may be deleted by calling the **clearOutput** method. This method also stops any output that is in progress, if possible. No **OutputCompleteEvents** will be delivered for output requests terminated in this manner.
- When done accessing the point card, the application calls the **beginRemoval** method, specifying a timeout value.
- If the card is not removed before the timeout period elapses, the Point Card Reader Writer fires an exception.
- Even if a timeout occurs, the Point Card Reader Writer remains in removal mode. If the application still wants the card removed, it must call the **beginRemoval** method again.
- To exit removal mode, either after the card was physically removed or the application wishes to abort removal, the application calls the **endRemoval** method.

## Card Insertion Diagram

The processing from card insertion to card removal is shown below. All methods, other than **printWrite**, are performed synchronously.

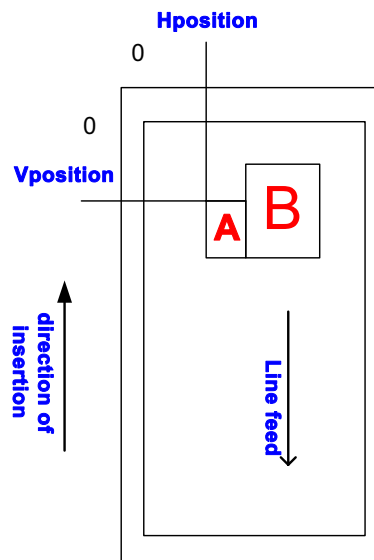


- (1) If the card is not inserted into the Point Card Reader Writer before the application specified timeout elapses, an exception is thrown. The application needs to call **beginInsertion** again to confirm that a point card has been inserted or call **endInsertion** to cancel the card insertion. After a successful **beginInsertion**, the application must call **endInsertion** to cause the Point Card Reader Writer to exit insertion mode and to read the magnetic stripe data from the point card.
- (2) If the card is not removed from the Point Card Reader Writer before the application specified timeout elapses, an exception is thrown. The application needs to call **beginRemoval** again to confirm that the point card has been removed, or call **endRemoval** to cancel the card removal. After a successful **beginRemoval**, the application must call **endRemoval** to cause the Point Card Reader Writer to exit removal mode.

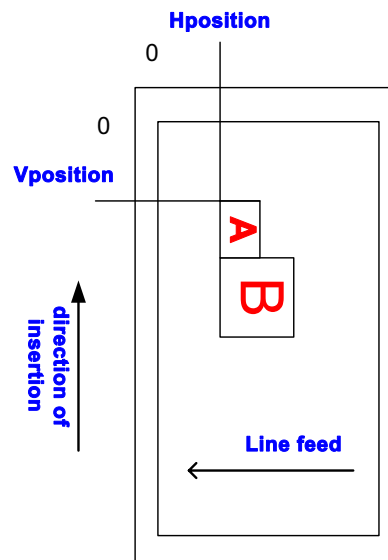
## Printing Capability

- The Point Card Reader Writer supports devices that allow for rewriting the print area of a card.
- The Point Card Reader Writer supports printing specified either by dot units or by line units. When **CapPrintMode** is true, the unit type is determined by the value of the **MapMode** property. When **CapPrintMode** is false, the unit type is defined as lines.
- The data to print is passed to the **printWrite** method as the *data* parameter. Special character modifications, such as double height, are dependent upon the capabilities of the device. The starting print location is specified by the *vposition* and *hposition* parameters respectively indicating the vertical and horizontal start position expressed in units defined by the **MapMode** property value.
- When using line units, the start position for lines containing both single and double high characters is the top of a single high character for horizontal printing and the bottom of all characters for vertical printing. See the diagram below for further clarification.

**Horizontal writing**



**Vertical writing**



## Cleaning Capability

- Cleaning of the Point Card Reader Writer is necessary to prevent errors caused by dirt build up inside the device.
- A special cleaning card is used. There are two types of cleaning card: a wet card (such as a card wet with ethanol before use) and a dry card.
- Cleaning is carried out by having the inserted cleaning card make several passes over the read heads inside the device.
- Some Point Card Reader Writers perform the cleaning operation by use of a switch on the device. Others perform the cleaning operation entirely under control of the application.

## Initialization of Magnetic Stripe Data

- Some Point Card Reader Writers can initialize the magnetic stripe data to prevent the illegal use of a point card.
- There are three initialization techniques in use for Point Card Reader Writers:
  - Initialize all of the data, including the start sentinel, end sentinel, and a correct LRC.
  - Write an application specific code into the data area using no sentinels.
  - Initialize all tracks to empty by just writing start and end sentinels.
- Initialization of the magnetic stripe is dependent upon the capability of the device.

## Device Sharing

The Point Card Reader Writer is an exclusive-use device, as follows:

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

## Data Characters and Escape Sequences

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

Every escape sequence begins with the escape character ESC, whose value is 27 decimal, followed by a vertical bar ('|'). This is followed by zero or more digits and/or lowercase alphabetic characters. The escape sequence is terminated by an uppercase alphabetic character. Sequences that do not begin with ESC "|" are passed through to Point Card Reader Writer. Also, sequences that begin with ESC "|" but which are not valid UnifiedPOS escape sequences are passed through to Point Card Reader Writer.

To determine if escape sequences or data can be performed on Point Card Reader Writer, the application can call the **validateData** method. (For some escape sequences, corresponding capability properties can also be used.)

The following escape sequences are recognized. If an escape sequence specifies an operation that is not supported by the Point Card Reader Writer, then it is ignored.

**Print Mode** Characteristics that are remembered until explicitly changed.

Name	Data	Remarks
Font typeface selection	ESC #T	Selects a new typeface for the following data. Values for the character '#' are:  0 = Default typeface. 1 = Select first typeface from the <b>FontTypefaceList</b> property. 2 = Select second typeface from the <b>FontTypefaceList</b> property.  And so on.

**Print Line** Characteristics that are reset at the end of each print method or by a “Normal” sequence.

Name	Data	Remarks
Bold	ESC  bC	Prints in bold or double-strike.
Underline	ESC  #uC	Prints with underline. The character '#' is replaced by an ASCII decimal string telling the thickness of the underline in printer dot units. If '#' is omitted, then a printer-specific default thickness is used.
Italic	ESC  iC	Prints in italics.
Reverse video	ESC  rvC	Prints in a reverse video format.
Single high & wide	ESC  1C	Prints normal size.
Double wide	ESC  2C	Prints double-wide characters.
Double high	ESC  3C	Prints double-high characters.
Double high & wide	ESC  4C	Prints double-high/double-wide characters.
Scale horizontally	ESC  #hC	Prints with the width scaled '#' times the normal size, where '#' is replaced by an ASCII decimal string.
Scale vertically	ESC  #vC	Prints with the height scaled '#' times the normal size, where '#' is replaced by an ASCII decimal string.
Center	ESC  cA	Aligns following text in the center.
Right justify	ESC  rA	Aligns following text at the right.
Normal	ESC  N	Restores printer characteristics to normal condition.



## Properties

### CapBold PropertyType R

Type	<b>boolean</b>
Remarks	If true, then the Point Card Reader Writer can print bold characters, false if it cannot.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapCardEntranceSensor Property R

Type	<b>boolean</b>
Remarks	If true, then the Point Card Reader Writer has an entrance sensor, false if it does not.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapCharacterSet Property R

Type	<b>int</b>												
Remarks	Holds the default character set capability. It may be one of the following: <table border="1"> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>PCRW_CCS_ALPHA</td> <td>The default character set supports upper case alphabetic plus numeric, space, minus, and period.</td> </tr> <tr> <td>PCRW_CCS_ASCII</td> <td>The default character set supports all ASCII characters between 20-hex and 7F-hex.</td> </tr> <tr> <td>PCRW_CCS_KANA</td> <td>The default character set supports partial code page 932, including ASCII characters 20-hex through 7F-hex and the Japanese Kana characters A1-hex through DF-hex, but excluding the Japanese Kanji characters.</td> </tr> <tr> <td>PCRW_CCS_KANJI</td> <td>The default character set supports code page 932, including the Shift-JIS Kanji characters, Levels 1 and 2.</td> </tr> <tr> <td>PCRW_CCS_UNICODE</td> <td>The default character set supports UNICODE.</td> </tr> </tbody> </table> <p>The default character set may contain a superset of these ranges. The initial <b>CharacterSet</b> property may be examined for additional information.</p> <p>This property is initialized by the <b>open</b> method.</p>	Value	Meaning	PCRW_CCS_ALPHA	The default character set supports upper case alphabetic plus numeric, space, minus, and period.	PCRW_CCS_ASCII	The default character set supports all ASCII characters between 20-hex and 7F-hex.	PCRW_CCS_KANA	The default character set supports partial code page 932, including ASCII characters 20-hex through 7F-hex and the Japanese Kana characters A1-hex through DF-hex, but excluding the Japanese Kanji characters.	PCRW_CCS_KANJI	The default character set supports code page 932, including the Shift-JIS Kanji characters, Levels 1 and 2.	PCRW_CCS_UNICODE	The default character set supports UNICODE.
Value	Meaning												
PCRW_CCS_ALPHA	The default character set supports upper case alphabetic plus numeric, space, minus, and period.												
PCRW_CCS_ASCII	The default character set supports all ASCII characters between 20-hex and 7F-hex.												
PCRW_CCS_KANA	The default character set supports partial code page 932, including ASCII characters 20-hex through 7F-hex and the Japanese Kana characters A1-hex through DF-hex, but excluding the Japanese Kanji characters.												
PCRW_CCS_KANJI	The default character set supports code page 932, including the Shift-JIS Kanji characters, Levels 1 and 2.												
PCRW_CCS_UNICODE	The default character set supports UNICODE.												
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.												

### CapCleanCard Property R

<b>Type</b>	<b>boolean</b>
<b>Remarks</b>	If true, then the Point Card Reader Writer supports cleaning under application control, false if it does not.  This property is initialized by the <b>open</b> method.
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapClearPrint Property R

<b>Type</b>	<b>boolean</b>
<b>Remarks</b>	If true, then the Point Card Reader Writer supports clearing (erasing) the printing area, false if it does not.  This property is initialized by the <b>open</b> method.
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapDhigh Property R

<b>Type</b>	<b>boolean</b>
<b>Remarks</b>	If true, then the Point Card Reader Writer can print double high characters, false if it cannot.  This property is initialized by the <b>open</b> method.
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapDwide Property R

<b>Type</b>	<b>boolean</b>
<b>Remarks</b>	If true, then the Point Card Reader Writer can print double wide characters, false if it cannot.  This property is initialized by the <b>open</b> method.
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapDwideDhigh Property R

Type	<b>boolean</b>
Remarks	If true, then the Point Card Reader Writer can print double high / double wide characters, false if it cannot.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### Capltalic Property R

Type	<b>boolean</b>
Remarks	If true, then the Point Card Reader Writer can print italic characters, false if it cannot.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapLeft90 Property R

Type	<b>boolean</b>
Remarks	If true, then the Point Card Reader Writer can print in rotated 90° left mode, false if it cannot.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapPrint Property R

Type	<b>boolean</b>
Remarks	If true, then the Point Card Reader Writer has printing capability; false if it does not.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapPrintMode Property R

Type	<b>boolean</b>
Remarks	If true, then the Point Card Reader Writer can designate a printing start position with the <b>MapMode</b> property, false if it cannot.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

**CapRight90 Property R**

<b>Type</b>	<b>boolean</b>
<b>Remarks</b>	If true, then the Point Card Reader Writer can print in a rotated 90° right mode, false if it cannot.  This property is initialized by the <b>open</b> method.
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

**CapRotate180 Property R**

<b>Type</b>	<b>boolean</b>
<b>Remarks</b>	If true, then the Point Card Reader Writer can print in a rotated upside down mode, false if it cannot.  This property is initialized by the <b>open</b> method.
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

**CapTracksToRead Property R**

<b>Type</b>	<b>int</b>														
<b>Remarks</b>	A bitmask indicating which magnetic tracks are accessible on the inserted point card. The value contained in this property is a bitwise OR of the constants PCRW_TRACK1 through PCRW_TRACK6.  For example, access to track 1 is possible when PCRW_TRACK1 is set.  This property is initialized by the <b>open</b> method.														
	<table> <thead> <tr> <th><b>Value</b></th> <th><b>Meaning</b></th> </tr> </thead> <tbody> <tr> <td>PCRW_TRACK1</td> <td>Track1</td> </tr> <tr> <td>PCRW_TRACK2</td> <td>Track2</td> </tr> <tr> <td>PCRW_TRACK3</td> <td>Track3</td> </tr> <tr> <td>PCRW_TRACK4</td> <td>Track4</td> </tr> <tr> <td>PCRW_TRACK5</td> <td>Track5</td> </tr> <tr> <td>PCRW_TRACK6</td> <td>Track6</td> </tr> </tbody> </table>	<b>Value</b>	<b>Meaning</b>	PCRW_TRACK1	Track1	PCRW_TRACK2	Track2	PCRW_TRACK3	Track3	PCRW_TRACK4	Track4	PCRW_TRACK5	Track5	PCRW_TRACK6	Track6
<b>Value</b>	<b>Meaning</b>														
PCRW_TRACK1	Track1														
PCRW_TRACK2	Track2														
PCRW_TRACK3	Track3														
PCRW_TRACK4	Track4														
PCRW_TRACK5	Track5														
PCRW_TRACK6	Track6														
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.														

## CapTracksToWrite Property R

**Type**      **int**

**Remarks**      A bitmask indicating which magnetic tracks are writable on the inserted point card. The value contained in this property is a bitwise OR of the constants PCRW\_TRACK1 through PCRW\_TRACK6.

For example, access to track 1 is possible when PCRW\_TRACK1 is set.

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

Value	Meaning
PCRW_TRACK1	Track1
PCRW_TRACK2	Track2
PCRW_TRACK3	Track3
PCRW_TRACK4	Track4
PCRW_TRACK5	Track5
PCRW_TRACK6	Track6

**Errors**      A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

## CardState Property R

**Type**      **int**

**Remarks**      If **CapCardEntranceSensor** is true, the current card entrance sensor status is stored in this property. The value will be one of the following.

Value	Meaning
PCRW_STATE_NOCARD	No card or card sensor position indeterminate
PCRW_STATE_REMAINING	Card remaining at the entrance
PCRW_STATE_INRW	There is a card in the device

If **CapCardEntranceSensor** is false, then **CardState** will always be set to PCRW\_STATE\_NOCARD.

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

**Errors**      A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

**See Also**      **CapCardEntranceSensor** Property.

**CharacterSet Property R/W****Type**      **int****Remarks**      The character set for printing characters.

<b>Value</b>	<b>Meaning</b>
Range 101 - 199	Device-specific character sets that do not match a code page or the ASCII or ANSI character sets.
Range 400 - 990	Code page; matches one of the standard values.
PCRW_CS_UNICODE	The character set supports UNICODE. The value of this constant is 997.
PCRW_CS_ASCII	The ASCII character set, supporting the ASCII characters between 0x20 and 0x7F. The value of this constant is 998.
PCRW_CS_ANSI	The ANSI character set. The value of this constant is 999.
Range 1000 and higher	Windows code page; matches one of the standard values.

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

**Errors**      A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
JPOS_E_ILLEGAL	An invalid property value was specified.

**See Also**      **CharacterSetList** Property.

**CharacterSetList Property R****Type**      **String****Remarks**      Holds the string of character set numbers. The string consists of an ASCII numeric set numbers separated by commas.

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

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

**Errors**      A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

**See Also**      **CharacterSet** Property.

## FontTypefaceList Property R

<b>Type</b>	<b>String</b>
<b>Remarks</b>	<p>A string that specifies the fonts and/or typefaces that are supported by the Point Card Reader Writer.</p> <p>The string consists of font or typeface names separated by commas. The application selects a font or typeface for the Point Card Reader Writer by using the font typeface selection escape sequence (ES C #fT). The “#” character is replaced by the number of the font or typeface within the list: 1, 2, and so on.</p> <p>In Japan, this property will frequently include the fonts “Mincho” and “Gothic”. Other fonts or typefaces may be commonly supported in other countries.</p> <p>An empty string indicates that only the default typeface is supported.</p> <p>This property is initialized by the <b>open</b> method.</p>
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
<b>See Also</b>	“Data Characters and Escape Sequences” on page 453.

## LineChars Property R/W

<b>Type</b>	<b>int</b>				
<b>Remarks</b>	<p>The number of characters that may be printed on a line on the Point Card Reader Writer.</p> <p>If changed to a line character width that can be supported, then the width is set to the specified value. If the exact width cannot be supported, then subsequent lines will be printed with a character size that most closely supports the specified characters per line. (For example, if set to 36 and the Point Card Reader Writer can print either 30 or 40 characters per line, then the Service should select the character size “40” and print up to 36 characters on each line.)</p> <p>If the character width cannot be supported, then an exception is thrown. (For example, if set to 42 and Point Card Reader Writer can print either 30 or 40 characters per line, then the Service cannot support the request.)</p> <p>Setting <b>LineChars</b> may also update <b>LineWidth</b>, <b>LineHeight</b>, and <b>LineSpacing</b>, since the character pitch or font may be changed.</p> <p>The value of <b>LineChars</b> is initialized to the Point Card Reader Writer’s default line character width when the device is first enabled following the <b>open</b> method.</p>				
<b>Errors</b>	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p> <p>Some possible values of the exception’s <i>ErrorCode</i> property are:</p> <table> <thead> <tr> <th><b>Value</b></th> <th><b>Meaning</b></th> </tr> </thead> <tbody> <tr> <td>JPOS_E_ILLEGAL</td> <td>An invalid line character width was specified.</td> </tr> </tbody> </table>	<b>Value</b>	<b>Meaning</b>	JPOS_E_ILLEGAL	An invalid line character width was specified.
<b>Value</b>	<b>Meaning</b>				
JPOS_E_ILLEGAL	An invalid line character width was specified.				
<b>See Also</b>	<b>LineCharsList</b> Property.				

**LineCharsList Property R**

<b>Type</b>	<b>String</b>
<b>Remarks</b>	<p>A string containing the line character widths supported by the Point Card Reader Writer.</p> <p>The string consists of an ASCII numeric set numbers separated by commas. For example, if the string is “32,36,40”, then the station supports line widths of 32, 36, and 40 characters.</p> <p>This property is initialized by the <b>open</b> method.</p>
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
<b>See Also</b>	<b>LineChars</b> Property.

**LineHeight Property R/W**

<b>Type</b>	<b>int</b>
<b>Remarks</b>	<p>The Point Card Reader Writer print line height. If <b>CapPrintMode</b> is true, this is expressed in the unit of measure given by <b>MapMode</b>.</p> <p>If changed to a height that can be supported with the current character width, then the line height is set to this value. If the exact height cannot be supported, then the height is set to the closest supported value.</p> <p>When <b>LineChars</b> is changed, <b>LineHeight</b> is updated to the default line height for the selected width.</p> <p>The value of <b>LineHeight</b> is initialized to the Point Card Reader Writer’s default line height when the device is first enabled following the <b>open</b> method.</p>
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

**LineSpacing Property R/W**

<b>Type</b>	<b>int</b>
<b>Remarks</b>	<p>The spacing of each single-high print line, including both the printed line height plus the white space between each pair of lines. Depending upon the Point Card Reader Writer and the current line spacing, a multi-high print line might exceed this value. If <b>CapPrintMode</b> is true, line spacing is expressed in the unit of measure given by <b>MapMode</b>.</p> <p>If changed to a spacing that can be supported by the Point Card Reader Writer, then the line spacing is set to this value. If the spacing cannot be supported, then the spacing is set to the closest supported value.</p> <p>When <b>LineChars</b> or <b>LineHeight</b> is changed, <b>LineSpacing</b> is updated to the default line spacing for the selected width or height.</p> <p>The value of <b>LineSpacing</b> is initialized to the Point Card Reader Writer’s default line spacing when the device is first enabled following the <b>open</b> method.</p>
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.



**LineWidth Property R**

Type	<b>int</b>
Remarks	The width of a line of <b>LineChars</b> characters. If <b>CapPrintMode</b> is true, expressed in the unit of measure given by <b>MapMode</b> . Setting <b>LineChars</b> may also update <b>LineWidth</b> . The value of <b>LineWidth</b> is initialized to the Point Card Reader Writer's default line width when the device is first enabled following the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see "Exceptions" on page 15.

**MapMode Property R/W**

Type	<b>int</b>										
Remarks	Contains the mapping mode of the Point Card Reader Writer. The mapping mode defines the unit of measure used for other properties, such as line heights and line spacings. The following map modes are supported:										
	<table> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>PCRW_MM_DOTS</td> <td>The Point Card Reader Writer's dot width. This width may be different for each Point Card Reader Writer.</td> </tr> <tr> <td>PCRW_MM_TWIPS</td> <td>1/1440 of an inch.</td> </tr> <tr> <td>PCRW_MM_ENGLISH</td> <td>0.001 inch.</td> </tr> <tr> <td>PCRW_MM_METRIC</td> <td>0.01 millimeter.</td> </tr> </tbody> </table>	Value	Meaning	PCRW_MM_DOTS	The Point Card Reader Writer's dot width. This width may be different for each Point Card Reader Writer.	PCRW_MM_TWIPS	1/1440 of an inch.	PCRW_MM_ENGLISH	0.001 inch.	PCRW_MM_METRIC	0.01 millimeter.
Value	Meaning										
PCRW_MM_DOTS	The Point Card Reader Writer's dot width. This width may be different for each Point Card Reader Writer.										
PCRW_MM_TWIPS	1/1440 of an inch.										
PCRW_MM_ENGLISH	0.001 inch.										
PCRW_MM_METRIC	0.01 millimeter.										
	Setting <b>MapMode</b> may also change <b>LineHeight</b> , <b>LineSpacing</b> , and <b>LineWidth</b> . The value of <b>MapMode</b> is initialized to PCRW_MM_DOTS when the device is first enabled following the <b>open</b> method.										
Errors	A JposException may be thrown when this property is accessed. For further information, see "Exceptions" on page 15. Some possible values of the exception's <i>ErrorCode</i> property are:										
	<table> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>JPOS_E_ILLEGAL</td> <td>An invalid mapping mode value was specified.</td> </tr> </tbody> </table>	Value	Meaning	JPOS_E_ILLEGAL	An invalid mapping mode value was specified.						
Value	Meaning										
JPOS_E_ILLEGAL	An invalid mapping mode value was specified.										

**MaxLine Property R**

Type	<b>int</b>
Remarks	When the <b>CapPrintMode</b> property is false, <b>MaxLine</b> contains the maximum printable line number. In the case where there is a double-high character in the same line, this is dependent upon the capability of the device. When the <b>LineHeight</b> property and/or the <b>LineSpacing</b> property change, the <b>MaxLine</b> property may be changed.
Errors	A JposException may be thrown when this property is accessed. For further information, see "Exceptions" on page 15.
See Also	<b>LineHeight</b> Property.

**PrintHeight Property R**

- Type**      **int**
- Remarks**      When the **CapPrintMode** property is true, the height of the largest character in the character set is stored in this property expressed in **MapMode** units.
- When the **MapMode** property is changed the value of the **PrintHeight** property changes.
- Errors**      A **JposException** may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
- See Also**      **CapPrintMode** Property, **MapMode** Property.

**ReadState1 Property R**

- Type**      **int**
- Remarks**      The property is divided into four bytes with each byte containing status information about the first four tracks. The diagram below indicates how the property value is divided:
- The Control sets a value to this property immediately before it enqueues the **ErrorEvent** or **DataEvent**.

High Word		Low Word	
High Byte	Low Byte	High Byte	Low Byte
Track4	Track 3	Track 2	Track 1

The following values can be set:

<b>Value</b>	<b>Meaning</b>
JPOS_SUCCESS	Successful read of the data.
JPOS_E_PCRW_START	It is a start sentinel error.
JPOS_E_PCRW_END	It is a end sentinel error.
JPOS_E_PCRW_PARITY	It is a parity error.
JPOS_E_PCRW_ENCODE	There is no encoding.
JPOS_E_PCRW_LRC	It is a LRC error.
JPOS_E_PCRW_VERIFY	It is a verify error.
JPOS_E_FAILURE	It is other error.

- Errors**      A **JposException** may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
- See Also**      **ReadState2** Property.

**ReadState2 Property R****Type**      **int**

**Remarks**      The property is divided into four bytes with two bytes containing status information about the fifth and sixth tracks. The diagram below indicates how the property value is divided:

The Point Card Reader Writer sets a value to this property immediately before it enqueues the **ErrorEvent** or **DataEvent**.

High Word		Low Word	
High Byte	Low Byte	High Byte	Low Byte
Unused	Unused	Track 6	Track 5

The following values can be set.

<b>Value</b>	<b>Meaning</b>
JPOS_SUCCESS	Successful read of the data.
JPOS_E_PCRW_START	It is a start sentinel error.
JPOS_E_PCRW_END	It is an end sentinel error.
JPOS_E_PCRW_PARITY	It is a parity error.
JPOS_E_PCRW_ENCODE	There is no encoding.
JPOS_E_PCRW_LRC	It is a LRC error.
JPOS_E_PCRW_VERIFY	It is a verify error.
JPOS_E_FAILURE	It is other error.

**Errors**      A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

**See Also**      **ReadState1** Property.

## RecvLength1 Property R

Type **int**

Remarks The property is divided into four bytes with each of the bytes representing information about the first four tracks. The diagram below indicates how the value is divided:

High Word		Low Word	
High Byte	Low Byte	High Byte	Low Byte
Track4	Track 3	Track 2	Track 1

A value of zero for a track byte means that no data was obtained from the swipe for that particular track. This might be due to the hardware device simply not having a read head for the track, or STX, ETX and LRC only was obtained from the swipe for that particular track, or reading of data without being made with some errors, or perhaps the application intentionally precluded incoming data from the track via the **TracksToRead** property.

A value greater than zero indicates the length in bytes of the corresponding **TrackxData** property.

Errors A `JposException` may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

See Also **CapTracksToRead** property, **TracksToRead** property, **RecvLength2** Property.

## RecvLength2 Property R

Type **int**

Remarks The property is divided into four bytes with two of the bytes representing information about the fifth and sixth tracks, while the third and fourth bytes are unused. The diagram below indicates how the value is divided:

High Word		Low Word	
High Byte	Low Byte	High Byte	Low Byte
Unused	Unused	Track 6	Track 5

A value of zero for a track byte means that no data was obtained from the swipe for that particular track. This might be due to the hardware device simply not having a read head for the track, or STX, ETX, and LRC only was obtained from the swipe for that particular track, or reading of data without being made with some errors, or perhaps the application intentionally precluded incoming data from the track via the **TracksToRead** property.

A value greater than zero indicates the length in bytes of the corresponding **TrackxData** property.

Errors A `JposException` may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

See Also **CapTracksToRead** property, **TracksToRead** property, **RecvLength1** Property.

### SidewaysMaxChars Property R

Type	int
Remarks	<p>Holds the maximum number of characters that may be printed on each line in sideways mode.</p> <p>If the capabilities <b>CapLeft90</b> and <b>CapRight90</b> are both false, then <b>SidewaysMaxChars</b> is zero.</p> <p>Changing the properties <b>LineHeight</b>, <b>LineSpacing</b>, and <b>LineChars</b> may cause this property to change.</p> <p>This property is initialized when the device is first enabled following the <b>open</b> method.</p>
Errors	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>
See Also	<b>SidewaysMaxLines</b> Property.

### SidewaysMaxLines Property R

Type	int
Remarks	<p>Holds the maximum number of lines that may be printed in sideways mode.</p> <p>If the capabilities <b>CapLeft90</b> and <b>CapRight90</b> are both false, then <b>SidewaysMaxLines</b> is zero.</p> <p>Changing the properties <b>LineHeight</b>, <b>LineSpacing</b>, and <b>LineChars</b> may cause this property to change.</p> <p>This property is initialized when the device is first enabled following the <b>open</b> method.</p>
Errors	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>
See Also	<b>SidewaysMaxChars</b> Property.

**TracksToRead Property R/W**

<b>Type</b>	<b>int</b>						
<b>Remarks</b>	<p>Holds the tracks that are to be read from the point card. It contains a bitwise OR of the constants PCRW_TRACK1 through PCRW_TRACK6. It may only contain values that are marked as allowable by the <b>CapTracksToRead</b> property. For example, to read tracks 1, 2, and 3, this property should be set to: PCRW_TRACK1   PCRW_TRACK2   PCRW_TRACK3.</p> <p>This property is initialized when the device is first enabled following the <b>open</b> method.</p>						
<b>Errors</b>	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p> <p>Some possible values of the exception’s <i>ErrorCode</i> property are:</p> <table> <thead> <tr> <th><b>Value</b></th> <th><b>Meaning</b></th> </tr> </thead> <tbody> <tr> <td>JPOS_E_BUSY</td> <td>This operation cannot be performed because asynchronous output is in progress.</td> </tr> <tr> <td>JPOS_E_ILLEGAL</td> <td>An illegal track was defined. The track is not available for reading. Refer to <b>CapTracksToRead</b>.</td> </tr> </tbody> </table>	<b>Value</b>	<b>Meaning</b>	JPOS_E_BUSY	This operation cannot be performed because asynchronous output is in progress.	JPOS_E_ILLEGAL	An illegal track was defined. The track is not available for reading. Refer to <b>CapTracksToRead</b> .
<b>Value</b>	<b>Meaning</b>						
JPOS_E_BUSY	This operation cannot be performed because asynchronous output is in progress.						
JPOS_E_ILLEGAL	An illegal track was defined. The track is not available for reading. Refer to <b>CapTracksToRead</b> .						
<b>See Also</b>	<b>CapTracksToRead</b> Property.						

**TracksToWrite Property R/W**

<b>Type</b>	<b>int</b>						
<b>Remarks</b>	<p>Holds the tracks that are to be written to the point card. It contains a bitwise OR of the constants PCRW_TRACK1 through PCRW_TRACK6. It may only contain values that are marked as allowable by the <b>CapTracksToWrite</b> property. For example, to write tracks 1, 2, and 3, this property should be set to: PCRW_TRACK1   PCRW_TRACK2   PCRW_TRACK3.</p> <p>This property is initialized when the device is first enabled following the <b>open</b> method.</p>						
<b>Errors</b>	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p> <p>Some possible values of the exception’s <i>ErrorCode</i> property are:</p> <table> <thead> <tr> <th><b>Value</b></th> <th><b>Meaning</b></th> </tr> </thead> <tbody> <tr> <td>JPOS_E_BUSY</td> <td>This operation cannot be performed because asynchronous output is in progress.</td> </tr> <tr> <td>JPOS_E_ILLEGAL</td> <td>An illegal track was defined. The track is not available for writing. Refer to <b>CapTracksToWrite</b>.</td> </tr> </tbody> </table>	<b>Value</b>	<b>Meaning</b>	JPOS_E_BUSY	This operation cannot be performed because asynchronous output is in progress.	JPOS_E_ILLEGAL	An illegal track was defined. The track is not available for writing. Refer to <b>CapTracksToWrite</b> .
<b>Value</b>	<b>Meaning</b>						
JPOS_E_BUSY	This operation cannot be performed because asynchronous output is in progress.						
JPOS_E_ILLEGAL	An illegal track was defined. The track is not available for writing. Refer to <b>CapTracksToWrite</b> .						
<b>See Also</b>	<b>CapTracksToWrite</b> Property, <b>printWrite</b> Method.						

### Track1Data Property R

Type	byte[]
Remarks	Contains the track 1 data from the point card.  This property contains track data between but not including the start and end sentinels.  An empty string indicates that the track was not accessible.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### Track2Data Property R

Type	byte[]
Remarks	Contains the track 2 data from the point card.  This property contains track data between but not including the start and end sentinels.  An empty string indicates that the track was not accessible.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### Track3Data Property R

Type	byte[]
Remarks	Contains the track 3 data from the point card.  This property contains track data between but not including the start and end sentinels.  An empty string indicates that the track was not accessible.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### Track4Data Property

Type	byte[]
Remarks	Contains the track 4 data from the point card.  This property contains track data between but not including the start and end sentinels.  An empty string indicates that the track was not accessible.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

**Track5Data Property R**

<b>Type</b>	<b>byte[]</b>
<b>Remarks</b>	Contains the track 5 data from the point card.  This property contains track data between but not including the start and end sentinels.  An empty string indicates that the track was not accessible.
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

**Track6Data Property R**

<b>Type</b>	<b>byte[]</b>
<b>Remarks</b>	Contains the track 6 data from the point card.  This property contains track data between but not including the start and end sentinels.  An empty string indicates that the track was not accessible.
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.



## WriteState1 Property R

**Type**      **int**

**Remarks**      The property is divided into four bytes with each byte containing status information about the first four tracks. The diagram below indicates how the property is divided:

The Control sets a value to this property immediately before it enqueues the **ErrorEvent**.

High Word		Low Word	
High Byte	Low Byte	High Byte	Low Byte
Track4	Track 3	Track 2	Track 1

The following value is set.

<b>Value</b>	<b>Meaning</b>
JPOS_E_SUCCESS	Successful write of the data.
JPOS_E_PCRW_START	It is a start sentinel error.
JPOS_E_PCRW_END	It is a end sentinel error.
JPOS_E_PCRW_PARITY	It is a parity error.
JPOS_E_PCRW_ENCODE	There is not encoding.
JPOS_E_PCRW_LRC	It is a LRC error.
JPOS_E_PCRW_VERIFY	It is a verify error.
JPOS_E_FAILURE	It is other error.

**Errors**      A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

**See Also**      **WriteState2** Property.

**WriteState2 Property R****Type**      **int****Remarks**      The property is divided into four bytes with each byte containing status information about the fifth and sixth tracks. The diagram below indicates how the property is divided:

The Control sets a value to this property immediately before it enqueues the **ErrorEvent**.

High Word		Low Word	
High Byte	Low Byte	High Byte	Low Byte
Unused	Unused	Track 6	Track 5

The following value is set.

Value	Meaning
JPOS_SUCCESS	Successful write of the data.
JPOS_E_PCRW_START	It is a start sentinel error.
JPOS_E_PCRW_END	It is a end sentinel error.
JPOS_E_PCRW_PARITY	It is a parity error.
JPOS_E_PCRW_ENCODE	There is not encoding.
JPOS_E_PCRW_LRC	It is a LRC error.
JPOS_E_PCRW_VERIFY	It is a verify error.
JPOS_E_FAILURE	It is other error.

**Errors**      A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.**See Also**      **WriteState1** Property.**Write1Data Property R/W****Type**      **byte[]****Remarks**      The **printWrite** method writes this data to track 1 of a point card.

This property contains track data between but not including the start and end sentinels.

**Errors**      A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### Write2Data Property R/W

Type	byte[]
Remarks	The <b>printWrite</b> method writes this data to track 2 of a point card.  This property contains track data between but not including the start and end sentinels.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### Write3Data Property R/W

Type	byte[]
Remarks	The <b>printWrite</b> method writes this data to track 3 of a point card.  This property contains track data between but not including the start and end sentinels.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### Write4Data Property R/W

Type	byte[]
Remarks	The <b>printWrite</b> method writes this data to track 4 of a point card.  This property contains track data between but not including the start and end sentinels.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### Write5Data Property R/W

Type	byte[]
Remarks	The <b>printWrite</b> method writes this data to track 5 of a point card.  This property contains track data between but not including the start and end sentinels.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### Write6Data Property R/W

Type	byte[]
Remarks	The <b>printWrite</b> method writes this data to track 6 of a point card.  This property contains track data between but not including the start and end sentinels.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

## Methods

### beginInsertion Method

**Syntax**      **void beginInsertion (int *timeout*) throws JposException;**

Parameter	Description
<i>timeout</i>	The number of milliseconds before failing the method

If zero, the method initiates insertion mode and either returns immediately if successful, or raises an exception. If FOREVER (-1), the method initiates the begin insertion mode, then waits as long as needed until either the point card is inserted or an error occurs.

**Remarks**      Called to initiate point card insertion processing.

When called, Point Card Reader Writer state is changed to allow the insertion of a point card and the point card insertion mode is entered. This method is paired with the **endInsertion** method for controlling point card insertion.

If the Point Card Reader Writer device cannot be placed into insertion mode an exception is raised. Otherwise, the Control continues to monitor point card insertion until either the point card is not inserted before *timeout* milliseconds have elapsed, or an error is reported by the Point Card Reader Writer device. In the latter case, the Control raises an exception with the appropriate error code. The Point Card Reader Writer device remains in point card insertion mode. This allows an application to perform some user interaction and reissue the **beginInsertion** method without altering the point card handling mechanism.

**Errors**      A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_BUSY	This operation cannot be performed because asynchronous output is in progress.
JPOS_E_ILLEGAL	The Point Card Reader Writer does not exist or an invalid <i>timeout</i> parameter was specified.
JPOS_E_TIMEOUT	The specified time has elapsed without the point card being properly inserted.
JPOS_E_EXTENDED	Refer to the definitions for <i>ErrorCodeExtended</i> in the Events section “ErrorEvent” on page 483.

**See Also**      **endInsertion** Method, **beginRemoval** Method, **endRemoval** Method.

## beginRemoval Method

**Syntax**      `void beginRemoval (int timeout) throws JposException;`

Parameter	Description
<i>timeout</i>	The number of milliseconds before failing the method

If zero, the method initiates the begin removal mode and either returns immediately or raises an exception. If FOREVER (-1), the method initiates the begin removal mode, then waits as long as needed until either the form is removed or an error occurs.

**Remarks**      Called to initiate point card removal processing.

When called, the Point Card Reader Writer is made ready to eject a point card or activating a point card ejection mode. This method is paired with the **endRemoval** method for controlling point card removal.

The model that has the sensor in the entrance ends normally when a card is ejected from Point Card Reader Writer. The model without the sensor ends normally when that ejection processing is implemented.

If the Point Card Reader Writer cannot be placed into removal or ejection mode, an exception is raised. Otherwise, the Control continues to monitor point card removal until either the point card is not ejected before *timeout* milliseconds have elapsed, or an error is reported by the Point Card Reader Writer. In this case, the Control raises an exception with the appropriate error code. The Point Card Reader Writer remains in point card ejection mode. This allows an application to perform some user interaction and reissue the **beginRemoval** method without altering the point card handling mechanism.

**Errors**      A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_BUSY	This operation cannot be performed because asynchronous output is in progress.
JPOS_E_ILLEGAL	The Point Card Reader Writer does not exist or an invalid <i>timeout</i> parameter was specified.
JPOS_E_TIMEOUT	The specified time has elapsed without the point card being properly inserted.
JPOS_E_EXTENDED	Refer to the definitions for <i>ErrorCodeExtended</i> in the Events section “ErrorEvent” on page 483.

**See Also**      **CapCardEntranceSensor** Property, **CardState** Property, **beginInsertion** Method, **endInsertion** Method, **endRemoval** Method.

## cleanCard Method

**Syntax**      `void cleanCard() throws JposException;`

**Remarks**      This method is used to clean the read/write heads of the Point Card Reader Writer. This method is only supported if the **CapCleanCard** property is true.

**Errors**          A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
JPOS_E_ILLEGAL	The Point Card Reader Writer does not exist or <b>CapCleanCard</b> is false.
JPOS_E_EXTENDED	Refer to the definitions for <i>ErrorCodeExtended</i> in the Events section “ErrorEvent” on page 483.

**See Also**      **CapCleanCard** Property.

## clearPrintWrite Method

**Syntax**      **void clearPrintWrite ( int *kind*, int *hposition*, int *vposition*, int *width*,  
int *height* )  
throws **JposException**;**

Parameter	Description
<i>kind</i>	Defines the parts of the point card that will be cleared. 1: Printing area 2: Magnetic tracks 3: Both printing area and magnetic tracks
<i>hposition</i>	The horizontal start position for erasing the printing area. The value is in <b>MapMode</b> units if <b>CapPrintMode</b> is true.
<i>vposition</i>	The vertical start position for erasing the printing area. The value is in <b>MapMode</b> units if <b>CapPrintMode</b> is true.
<i>width</i>	The width used for erasing the printing area. The value is in <b>MapMode</b> units if <b>CapPrintMode</b> is true.
<i>height</i>	The height used for erasing the printing area. The value is in <b>MapMode</b> units if <b>CapPrintMode</b> is true.

**Remarks**      Used to erase the printing area of a point card and/or erase the magnetic track data on a point card.

When the **CapPrint** and **CapClearPrint** properties are both true, this method can be used to clear the printing area of a point card. The *hposition*, *vposition*, *width*, and *height* parameters define the rectangle that will be cleared. If these parameters are 0, 0, -1, -1 respectively, this method will erase the entire printing area.

The initialization of the magnetic track data relies upon the capability of the device.

**Errors**      A **JposException** may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_BUSY	This operation cannot be performed because asynchronous output is in progress.
JPOS_E_EXTENDED	Refer to the definitions for <i>ErrorCodeExtended</i> in the Events section “ErrorEvent” on page 483.

**See Also**      **CapClearPrint** Property, **CapPrint** Property, **CapPrintMode** Property, **MapMode** Property.

**endInsertion Method**

**Syntax**      **void endInsertion ( ) throws JposException;**

**Remarks**      Called to end point card insertion processing.  
When called, the Point Card Reader Writer is taken out of point card insertion mode. If no point card is present, an exception is raised.  
This method is paired with the **beginInsertion** method for controlling point card insertion.

**Errors**          A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_ILLEGAL	The Point Card Reader Writer is not in point card insertion mode.
JPOS_E_FAILURE	A card is not inserted in the Point Card Reader Writer.
JPOS_E_EXTENDED	Refer to the definitions for <i>ErrorCodeExtended</i> in the Events section “ErrorEvent” on page 483.

**See Also**      **beginInsertion** Method, **beginRemoval** Method, **endRemoval** Method.

**endRemoval Method**

**Syntax**      **void endRemoval ( ) throws JposException;**

**Remarks**      Called to end point card removal processing.  
When called, the Point Card Reader Writer is taken out of point card removal or ejection mode. If a point card is present, an exception is raised. This method is paired with the **beginRemoval** method for controlling point card removal.  
The application may choose to call this method immediately after a successful **beginRemoval** if it wants to use the Point Card Reader Writer sensors to determine when the point card has been ejected. Alternatively, the application may prompt the user and wait for a key being pressed before calling this method.

**Errors**          A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_ILLEGAL	The Point Card Reader Writer is not in point card removal mode.
JPOS_E_FAILURE	There is a card in the Point Card Reader Writer.
JPOS_E_EXTENDED	Refer to the definitions for <i>ErrorCodeExtended</i> in the Events section “ErrorEvent” on page 483.

**See Also**      **beginInsertion** Method, **beginRemoval** Method, **endInsertion** Method.





**rotatePrint Method**

**Syntax**      **void rotatePrint (int rotation) throws JposException;**

Parameter	Description
<i>rotation</i>	Direction of rotation. See values below.
Value	Meaning
PCRW_RP_RIGHT90	Rotate printing 90° to the right (clockwise).
PCRW_RP_LEFT90	Rotate printing 90° to the left (counter-clockwise).
PCRW_RP_ROTATE180	Rotate printing 180°, that is print upside-down.
PCRW_RP_NORMAL	End rotated printing.

**Remarks**      Enters or exits rotated print mode.

The **rotatePrint** method designates the rotation of the printing area. After calling this method, the application calls the **printWrite** method and the print data is printed in the direction specified by the **rotatePrint** call. If *rotation* is PCRW\_RP\_NORMAL, then rotated print mode is exited.

Changing the rotation mode may also change the Point Card Reader Writer's line height, line spacing, line width, and other metrics.

**Errors**      A JposException may be thrown when this method is invoked. For further information, see "Exceptions" on page 15.

Some possible values of the exception's *ErrorCode* property are:

Value	Meaning
JPOS_E_BUSY	This operation cannot be performed because asynchronous output is in progress.
JPOS_E_ILLEGAL	The Point Card Reader Writer does not support the specified rotation.
JPOS_E_EXTENDED	Refer to the definitions for <i>ErrorCodeExtended</i> in the Events section "ErrorEvent" on page 483.

**See Also**      "Data Characters and Escape Sequences" on page 453, **printWrite** Method.

**validateData Method**

**Syntax**      **void validateData ( String *data* ) throws JposException;**

<b>Parameter</b>	<b>Description</b>
<i>data</i>	The data to be validated. May include printable data and escape sequences.

**Remarks**      Called to determine whether a data sequence, possibly including one or more escape sequences, is valid for printing, prior to calling the **printWrite** method. This method does not cause any printing, but is used to determine the capabilities of the Point Card Reader Writer.

**Errors**          A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
JPOS_E_ILLEGAL	Some of the data is not precisely supported by the device, but the Control can select valid alternatives.
JPOS_E_FAILURE	Some of the data is not supported. No alternatives can be selected.

Cases which cause *ErrorCode* of JPOS\_E\_ILLEGAL:

<b>Escape Sequence</b>	<b>Condition</b>
Underline	The thickness ‘#’ is not precisely supported: Control will select the closest supported value.
Shading	The percentage ‘#’ is not precisely supported: Control will select the closest supported value.
Scale horizontally	The scaling factor ‘#’ is not supported. Control will select the closest supported value.
Scale vertically	The scaling factor ‘#’ is not supported. Control will select the closest supported value.

Cases which will cause JPOS\_E\_FAILURE to be returned are:

<b>Escape Sequence</b>	<b>Condition</b>
(General)	The escape sequence format is not valid
Font typeface	The typeface ‘#’ is not supported:
Bold	Not supported.
Underline	Not supported.
Italic	Not supported.
Reverse video	Not supported.
Single high & wide	Not supported.
Double wide	Not supported.
Double high	Not supported.
Double high & wide	Not supported.

**See Also**      “Data Characters and Escape Sequences” on page 453, **printWrite** Method.

## Events

### DataEvent

<b>Interface</b>	jpos.events.DataListner	
<b>Method</b>	<b>dataOccurred (DataEvent e);</b>	
<b>Description</b>	Notifies the application that there is input data available from the device.	
<b>Attributes</b>	This event contains the following attribute:	
	<b>Attributes</b>	<b>Type</b> <b>Description</b>
	<i>Status</i>	<i>int</i> The <i>Status</i> parameter contains zero.
<b>Remarks</b>	The point card data is placed in each property before this event is delivered.	

### DirectIOEvent

<b>Interface</b>	jpos.events.DirectIOListner	
<b>Method</b>	<b>directIOOccurred (DirectIOEvent e);</b>	
<b>Description</b>	Provides Service information directly to the application. This event provides a means for a vendor-specific PointCard Service to provide events to the application that are not otherwise supported by the Control.	
<b>Attributes</b>	This event contains the following attributes:	
	<b>Attributes</b>	<b>Type</b> <b>Description</b>
	<i>EventNumber</i>	<i>int</i> Event number whose specific values are assigned by the Service.
	<i>Data</i>	<i>int</i> Additional numeric data. Specific values vary by the <i>EventNumber</i> and the Service. This property is settable.
	<i>Obj</i>	<i>object</i> Additional data whose usage varies by the <i>EventNumber</i> and Service. This property is settable.
<b>Remarks</b>	This event is to be used only for those types of vendor specific functions that are not otherwise described. Use of this event may restrict the application program from being used with other vendor's point card devices which may not have any knowledge of the Service's need for this event.	
<b>See Also</b>	"Exceptions" on page 15, <b>directIO</b> Method.	

## ErrorEvent

<b>Interface</b>	jpos.events.ErrorEventListner	
<b>Method</b>	<b>errorEventOccurred (ErrorEvent e);</b>	
<b>Description</b>	Notifies the application that a PointCard error has been detected and a suitable response by the application is necessary to process the error condition.	
<b>Attributes</b>	This event contains the following properties:	
	<b>Attributes</b>	<b>Type</b> <b>Description</b>
	<i>ErrorCode</i>	<i>int</i> Result code causing the error event. See a list of Error Codes on page 16.
	<i>ErrorCodeExtended</i>	<i>int</i> Extended Error code causing the error event. If <i>ErrorCode</i> is JPOS_E_EXTENDED, then see values below. Otherwise, it may contain a Service-specific value.
	<i>ErrorLocus</i>	<i>int</i> Location of the error. See values below.
	<i>ErrorResponse</i>	<i>int</i> Error response, whose default value may be overridden by the application. (i.e., this property is settable). See values below.

If *ErrorCode* is JPOS\_E\_EXTENDED, then *ErrorCodeExtended* has one of the following values:

<b>Value</b>	<b>Meaning</b>
JPOS_EPCRW_READ	There was a read error.
JPOS_EPCRW_WRITE	There was a write error.
JPOS_EPCRW_JAM	There was a card jam.
JPOS_EPCRW_MOTOR	There was a conveyance motor error.
JPOS_EPCRW_COVER	The conveyance motor cover was open.
JPOS_EPCRW_PRINTER	The printer has an error.
JPOS_EPCRW_RELEASE	There is a card remaining in the entrance.
JPOS_EPCRW_DISPLAY	There was a display indicator error.
JPOS_EPCRW_NOCARD	There is no card in the reader.

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

Value	Meaning
JPOS_EL_OUTPUT	Error occurred while processing asynchronous output.
JPOS_EL_INPUT	Error occurred while gathering or processing event-driven input. No input data is available.
JPOS_EL_INPUT_DATA	Error occurred while gathering or processing event-driven input, and some previously buffered data is available.

The contents of the *ErrorResponse* property are preset to a default value, based on the *ErrorLocus*. The application's error processing may change *ErrorResponse* to one of the following values:

Value	Meaning
JPOS_ER_RETRY	Typically valid only when locus is JPOS_EL_OUTPUT. Retry the asynchronous output. The error state is exited. May be valid when locus is JPOS_EL_INPUT. Default when locus is JPOS_EL_OUTPUT.
JPOS_ER_CLEAR	Clear the asynchronous output or buffered input data. The error state is exited. Default when locus is EL_INPUT.
JPOS_ER_CONTINUEINPUT	Use only when locus is JPOS_EL_INPUT_DATA. Acknowledges the error and directs the Control to continue processing. The Control remains in the error state and will deliver additional <b>DataEvents</b> as directed by the <b>DataEventEnabled</b> property. When all input has been delivered and the <b>DataEventEnabled</b> property is again set to true, then another <b>ErrorEvent</b> is delivered with locus JPOS_EL_INPUT. Default when locus is JPOS_EL_INPUT_DATA.

**Remarks** Input error events are generated when errors occur while reading the magnetic track data from a newly inserted card. These error events are not delivered until the **DataEventEnabled** property is set to true so as to allow proper application sequencing. All error information is placed into the **ReadStateX** properties before this event is delivered. The **RecvLengthX** property is set to 0 for each track that had an error and the **TrackXData** property is set to empty for each track that had an error.

Output error events are generated and delivered when an error occurs during asynchronous **printWrite** processing. The errors are placed into the **WriteStateX** properties before the event is delivered.

**See Also** **ReadStatex** Property, **RecvLengthx** Property, **TrackxData** Property, **WriteStatex** Property.

## OutputCompleteEvent

<b>Interface</b>	<code>jpos.events.OutputCompleteEventListner</code>						
<b>Method</b>	<code>outputCompleteEventOccurred (ErrorEvent e);</code>						
<b>Description</b>	Notifies the application that the queued output request associated with the <i>OutputID</i> attribute has completed successfully.						
<b>Attributes</b>	This event contains the following attribute:						
	<table> <thead> <tr> <th>Attributes</th> <th>Type</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td><i>OutputID</i></td> <td><i>int</i></td> <td>The ID number of the asynchronous output request that is complete.</td> </tr> </tbody> </table>	Attributes	Type	Description	<i>OutputID</i>	<i>int</i>	The ID number of the asynchronous output request that is complete.
Attributes	Type	Description					
<i>OutputID</i>	<i>int</i>	The ID number of the asynchronous output request that is complete.					
<b>Remarks</b>	This event is enqueued after the request's data has been both sent and the Service has confirmation that it was processed by the device successfully.						
<b>See Also</b>	"Device Output Models" on page 25.						

## StatusUpdateEvent

<b>Interface</b>	<code>jpos.events.StatusUpdateEventListner</code>														
<b>Method</b>	<code>statusUpdateEventOccurred (ErrorEvent e);</code>														
<b>Description</b>	Notifies the application that there is a change in the status of the PointCard device.														
<b>Attributes</b>	This event contains the following attribute:														
	<table> <thead> <tr> <th>Attributes</th> <th>Type</th> <th>Description</th> </tr> </thead> <tbody> <tr> <td><i>Status</i></td> <td><i>int</i></td> <td>Indicates a change in the power status of the unit.</td> </tr> </tbody> </table> <p>If <i>Status</i> parameter has one of the following values:</p> <table> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>PCRW_SUE_NOCARD</td> <td>No card or card sensor position indeterminate.</td> </tr> <tr> <td>PCRW_SUE_REMAINING</td> <td>Card remaining in the entrance.</td> </tr> <tr> <td>PCRW_SUE_INRW</td> <td>There is a card in the device.</td> </tr> </tbody> </table> <p><i>Note that Release 1.3</i> added Power State Reporting with additional <i>Power reporting StatusUpdateEvent values</i>. See description of "StatusUpdateEvent" on page 485.</p>	Attributes	Type	Description	<i>Status</i>	<i>int</i>	Indicates a change in the power status of the unit.	Value	Meaning	PCRW_SUE_NOCARD	No card or card sensor position indeterminate.	PCRW_SUE_REMAINING	Card remaining in the entrance.	PCRW_SUE_INRW	There is a card in the device.
Attributes	Type	Description													
<i>Status</i>	<i>int</i>	Indicates a change in the power status of the unit.													
Value	Meaning														
PCRW_SUE_NOCARD	No card or card sensor position indeterminate.														
PCRW_SUE_REMAINING	Card remaining in the entrance.														
PCRW_SUE_INRW	There is a card in the device.														
<b>Remarks</b>	Fired when the entrance sensor status of the Point Card Reader Writer changes. If the capability <b>CapCardEntranceSensor</b> is false, then the device does not support status reporting, and this event will never be fired to report card insertion state changes.														
<b>See Also</b>	"Events" on page 18, <b>CapCardEntranceSensor</b> Property.														





# POS Keyboard

## Summary

### Properties

<i>Common</i>	<i>Ver</i>	<i>Type</i>	<i>Access</i>	<i>May Use After</i>
<b>AutoDisable</b>		boolean	R/W	open
<b>CapPowerReporting</b>	1.3	int	R	open
<b>CheckHealthText</b>		String	R	open
<b>Claimed</b>		boolean	R	open
<b>DataCount</b>		int	R	open
<b>DataEventEnabled</b>		boolean	R/W	open
<b>DeviceEnabled</b>		boolean	R/W	open & claim
<b>FreezeEvents</b>		boolean	R/W	open
<b>OutputID</b>		int	R	<i>Not Supported</i>
<b>PowerNotify</b>	1.3	int	R/W	open
<b>PowerState</b>	1.3	int	R	open
<b>State</b>		int	R	--
<b>DeviceControlDescription</b>		String	R	--
<b>DeviceControlVersion</b>		int	R	--
<b>DeviceServiceDescription</b>		String	R	open
<b>DeviceServiceVersion</b>		int	R	open
<b>PhysicalDeviceDescription</b>		String	R	open
<b>PhysicalDeviceName</b>		String	R	open

<i>Specific</i>	<i>Ver</i>	<i>Type</i>	<i>Access</i>	<i>May Use After</i>
<b>CapKeyUp</b>		boolean	R	open
<b>EventTypes</b>		int	R/W	open
<b>POSKeyData</b>		int	R	open
<b>POSKeyEventType</b>		int	R	open

**Methods**

<i>Common</i>	<i>Ver</i>	<i>May Use After</i>
<b>open</b>		--
<b>close</b>		open
<b>claim</b>		open
<b>release</b>		open & claim
<b>checkHealth</b>		open, claim, & enable
<b>clearInput</b>		open & claim
<b>clearOutput</b>		<i>Not Supported</i>
<b>directIO</b>		open

**Events**

<i>Name</i>	<i>Ver</i>	<i>May Occur After</i>
<b>DataEvent</b>		open, claim, & enable
<b>DirectIOEvent</b>	1.3	open & claim
<b>ErrorEvent</b>		open, claim, & enable
<b>OutputCompleteEvent</b>		<i>Not Supported</i>
<b>StatusUpdateEvent</b>	1.3	open, claim, & enable

## General Information

The POS Keyboard Control's class name is "jpos.POSKeyboard".  
The device constants are contained in the class "jpos.POSKeyboardConst".  
See "Package Structure" on page 40.

### Capabilities

The POS Keyboard Control has the following capability:

- Reads keys from a POS keyboard. A POS keyboard may be an auxiliary keyboard, or it may be a virtual keyboard consisting of some or all of the keys on the system keyboard.

### Model

The POS Keyboard Control follows the JavaPOS model for input devices:

- When input is received by the Device Service a **DataEvent** is enqueued.
- If the **AutoDisable** property is true, then the Device automatically disables itself when a **DataEvent** is enqueued.
- A queued **DataEvent** can be delivered to the application when the **DataEventEnabled** property is true. Just before firing this event, data is copied into the properties, and further data events are disabled by setting **DataEventEnabled** to false. This causes subsequent input data to be enqueued while the application processes the current input and associated properties. When the application has finished the current input and is ready for more data, it reenables events by setting **DataEventEnabled** to true.
- An **ErrorEvent** (or events) are enqueued if an error occurs while gathering or processing input, and is delivered to the application when **DataEventEnabled** is true.
- The **DataCount** property may be read to obtain the number of queued **DataEvents**.
- All queued input may be deleted by calling **clearInput**.

### Keyboard Translation

The POS Keyboard Control must supply a mechanism for translating its internal key codes into user-defined codes which are returned by the data events. Note that this translation *must* be end-user configurable.

### Device Sharing

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

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

## Properties

### CapKeyUp Property R

<b>Type</b>	<b>boolean</b>
<b>Remarks</b>	If true, then the device is able to generate both key down and key up events, depending upon the setting of the <b>EventTypes</b> . If false, then the device is only able to generate the key down event.
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### EventTypes Property R/W

<b>Type</b>	<b>int</b>						
<b>Remarks</b>	Holds the type of events that the application wants to receive. It has one of the following values:						
	<table> <thead> <tr> <th><b>Value</b></th> <th><b>Meaning</b></th> </tr> </thead> <tbody> <tr> <td>KBD_ET_DOWN</td> <td>Generate key down events.</td> </tr> <tr> <td>KBD_ET_DOWN_UP</td> <td>Generate key down and key up events.</td> </tr> </tbody> </table> <p>This property is initialized to KBD_ET_DOWN by the <b>open</b> method.</p>	<b>Value</b>	<b>Meaning</b>	KBD_ET_DOWN	Generate key down events.	KBD_ET_DOWN_UP	Generate key down and key up events.
<b>Value</b>	<b>Meaning</b>						
KBD_ET_DOWN	Generate key down events.						
KBD_ET_DOWN_UP	Generate key down and key up events.						
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.						

### POSKeyData Property R

<b>Type</b>	<b>int</b>
<b>Remarks</b>	Holds the value of the key from the last <b>DataEvent</b> . The application may treat this value as device independent, assuming that the system installer has configured the Device Service to translate internal key codes to the codes expected by the application. Such configuration is inherently Device Service-specific.
	This property is set just before delivering the <b>DataEvent</b> .
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

## POSKeyEventType Property R

**Type**        **int**

**Remarks**    Holds the type of the last keyboard event: Is the key being pressed or released? It has one of the following values:

<b>Value</b>	<b>Meaning</b>
--------------	----------------

---

KBD\_KET\_KEYDOWN

The key in **POSKeyData** was pressed.

KBD\_KET\_KEYUP

The key in **POSKeyData** was released.

This property is set just before delivering the **DataEvent**.

**Errors**        A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

## Events

### DataEvent

<b>Interface</b>	<b>jpos.events.DataListener</b>	
<b>Method</b>	<b>dataOccurred (DataEvent e);</b>	
<b>Description</b>	Notifies the application that input data is available from the POS Keyboard device.	
<b>Properties</b>	This event contains the following property:	
	<b>Parameter</b>	<b>Type</b> <b>Description</b>
	<i>Status</i>	<i>int</i> Contains zero.
<b>Remarks</b>	The logical key number is placed in the <b>POSKeyData</b> property and the event type is placed in the <b>POSKeyEvent</b> property before this event is delivered.	
<b>See Also</b>	“Events” on page 18	

### DirectIOEvent

<b>Interface</b>	<b>jpos.events.DirectIOListener</b>	
<b>Method</b>	<b>directIOOccurred (DirectIOEvent e);</b>	
<b>Description</b>	Provides Device Service information directly to the application. This event provides a means for a vendor-specific POS Keyboard Device Service to provide events to the application that are not otherwise supported by the Device Control.	
<b>Properties</b>	This event contains the following properties:	
	<b>Property</b>	<b>Type</b> <b>Description</b>
	<i>EventNumber</i>	<i>int</i> Event number whose specific values are assigned by the Device Service.
	<i>Data</i>	<i>int</i> Additional numeric data. Specific values vary by the <i>EventNumber</i> and the Device Service. This property is settable.
	<i>Object</i>	<i>Object</i> Additional data whose usage varies by the <i>EventNumber</i> and Device Service. This property is settable.
<b>Remarks</b>	This event is to be used only for those types of vendor specific functions that are not otherwise described as part of the JavaPOS standard. Use of this event may restrict the application program from being used with other vendor’s POS Keyboard devices which may not have any knowledge of the Device Service’s need for this event.	
<b>See Also</b>	“Events” on page 18, <b>directIO</b> Method	

## ErrorEvent

Interface	<b>jpos.events.ErrorListener</b>
Method	<b>errorOccurred (ErrorEvent e);</b>
Description	Notifies the application that an error was detected trying to read POS Keyboard data.
Properties	This event contains the following properties:

Parameter	Type	Description
<i>ErrorCode</i>	<i>int</i>	Error Code causing the error event. See list of <i>ErrorCodes</i> on page 16.
<i>ErrorCodeExtended</i>	<i>int</i>	Extended Error Code causing the error event. It may contain a Service-specific value.
<i>ErrorLocus</i>	<i>int</i>	Location of the error. See values below.
<i>ErrorResponse</i>	<i>int</i>	Error response, whose default value may be overridden by the application (i.e., this property is settable). See values below.

The *ErrorLocus* property has one of the following values:

Value	Meaning
JPOS_EL_INPUT	Error occurred while gathering or processing event-driven input. No input data is available.
JPOS_EL_INPUT_DATA	Error occurred while gathering or processing event-driven input, and some previously buffered data is available.

The application's error event listener may change *ErrorResponse* to one of the following values:

Value	Meaning
JPOS_ER_CLEAR	Clear the buffered input data. The error state is exited. Default when locus is JPOS_EL_INPUT.
JPOS_ER_CONTINUEINPUT	Use only when locus is JPOS_EL_INPUT_DATA. Acknowledges the error and directs the Device to continue processing. The Device remains in the error state, and will deliver additional <b>DataEvents</b> as directed by the <b>DataEventEnabled</b> property. When all input has been delivered and <b>DataEventEnabled</b> is again set to true, then another <b>ErrorEvent</b> is delivered with locus JPOS_EL_INPUT. Default when locus is JPOS_EL_INPUT_DATA.

- Remarks** This event is not delivered until **DataEventEnabled** is true, so that proper application sequencing occurs.
- See Also** “Device Input Model” on page 22, “Device States” on page 30

## StatusUpdateEvent

- Interface** `jpos.events.StatusUpdateListener`
- Method** `statusUpdateOccurred (StatusUpdateEvent e);`
- Description** Notifies the application when the working status of the POS Keyboard changes.
- Properties** This event contains the following property:

Property	Type	Description
<i>Status</i>	<i>int</i>	The status reported from the POS Keyboard.  <i>Note that Release 1.3</i> added Power State Reporting with additional <i>Power reporting</i> <b>StatusUpdateEvent</b> values. See “StatusUpdateEvent” description on page 78.

- Remarks** Enqueued when the POS Keyboard needs to alert the application of a device state change.
- See Also** “Events” on page 18



# POS Power

## Summary

### Properties

<i>Common</i>	<i>Ver<sup>a</sup></i>	<i>Type</i>	<i>Access</i>	<i>May Use After</i>
<b>AutoDisable</b>	1.5	boolean	R/W	<i>Not Supported</i>
<b>CapPowerReporting</b>	1.5	int	R	open
<b>CheckHealthText</b>	1.5	string	R	open
<b>Claimed</b>	1.5	boolean	R	open
<b>DataCount</b>	1.5	int	R	<i>Not Supported</i>
<b>DataEventEnabled</b>	1.5	boolean	R/W	<i>Not Supported</i>
<b>DeviceEnabled</b>	1.5	boolean	R/W	open
<b>FreezeEvents</b>	1.5	boolean	R/W	open
<b>OutputID</b>	1.5	int	R	<i>Not Supported</i>
<b>PowerNotify</b>	1.5	int	R/W	open
<b>PowerState</b>	1.5	int	R	open
<b>State</b>	1.5	int	R	--
<b>DeviceControlDescription</b>	1.5	string	R	--
<b>DeviceControlVersion</b>	1.5	int	R	--
<b>DeviceServiceDescription</b>	1.5	string	R	open
<b>DeviceServiceVersion</b>	1.5	int	R	open
<b>PhysicalDeviceDescription</b>	1.5	string	R	open
<b>PhysicalDeviceName</b>	1.5	string	R	open

a. This POSPower definition was introduced in JavaPOS version 1.5.

**Properties (Continued)**

<i>Specific</i>	<i>Ver</i>	<i>Type</i>	<i>Access</i>	<i>May Use After</i>
<b>CapFanAlarm</b>	1.5	boolean	R	open
<b>CapHeatAlarm</b>	1.5	boolean	R	open
<b>CapQuickCharge</b>	1.5	boolean	R	open
<b>CapShutdownPOS</b>	1.5	boolean	R	open
<b>CapUPSChargeState</b>	1.5	int	R	open
<b>EnforcedShutdownDelayTime</b>	1.5	int	R/W	open
<b>PowerFailDelayTime</b>	1.5	int	R	open
<b>QuickChargeMode</b>	1.5	boolean	R	open
<b>QuickChargeTime</b>	1.5	int	R	open
<b>UPSChargeState</b>	1.5	int	R	open & enable

**Methods**

<i>Common</i>	<i>Ver</i>	<i>May Use After</i>
<b>open</b>	1.5	--
<b>close</b>	1.5	open
<b>claim</b>	1.5	open
<b>release</b>	1.5	open & claim
<b>checkHealth</b>	1.5	open, claim, & enable
<b>clearInput</b>	1.5	<i>Not Supported</i>
<b>clearOutput</b>	1.5	<i>Not Supported</i>
<b>directIO</b>	1.5	open
 <i>Specific</i>		
<b>shutdownPOS</b>	1.5	Open & Enable

**Events**

<i>Name</i>	<i>Ver</i>	<i>May Occur After</i>
<b>DataEvent</b>	1.5	<i>Not Supported</i>
<b>DirectIOEvent</b>	1.5	open & claim
<b>ErrorEvent</b>	1.5	<i>Not Supported</i>
<b>OutputCompleteEvent</b>	1.5	<i>Not Supported</i>
<b>StatusUpdateEvent</b>	1.3	open & enable

## General Information

The POS Power class name is “jpos.POSPower”.  
The device constants are contained in the class “jpos.POSPowerConst”.  
See “Package Structure” on page 40.

*This device was added in JavaPOS Release 1.5.*

## Capabilities

The POSPower device class has the following capabilities:

- Supports a command to “shut down” the system.
- Supports accessing a power handling mechanism of the underlying operating system and hardware.
- Informs the application if a power fail situation has occurred.
- Informs the application if the UPS charge state has changed.
- Informs the application about high CPU temperature.
- Informs the application about stopped CPU fan.
- Informs the application if an operating system dependant enforced shutdown mechanism is processed.
- Allows the application after saving application data locally or transferring application data to a server to shut down the POS terminal.
- Informs the application about an initiated shutdown.

## Device Sharing

The POSPower is a sharable device. Its device sharing rules are:

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

## Model

The general model of POSPower is based on the power model of each device in version 1.3 or later. The same common properties are used but all states relate to the POS terminal itself and not to a peripheral device.

There are three states of the POSPower:

- **ONLINE.** The POS terminal is powered on and ready for use. This is the “operational” state.
- **OFF.** The POS terminal is powered off or detached from the power supplying net. The POS terminal runs on battery power support. This is the powerfail situation.
- **OFFLINE.** The POS terminal is powered on but is running in a “lower-power-consumption” mode. It may need to be placed online by pressing a button or key or something else which may wake up the system.

Power reporting only occurs while the device is open, enabled and power notification is switched on.

In a powerfail situation - that means the POSPower is in the state OFF - the POS terminal will be shut down automatically after the last application has closed the POSPower device or the time specified by the **EnforcedShutdownDelayTime** property has been elapsed.

A call to the **shutdownPOS** method will always shut down the POS terminal independent of the system power state.

## Properties

### CapFanAlarm Property R

Type	<b>boolean</b>
Remarks	If true the device is able to detect whether the CPU fan is stopped. Otherwise it is false.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapHeatAlarm Property R

Type	<b>boolean</b>
Remarks	If true the device is able to detect whether the CPU is running at too high of a temperature. Otherwise it is false.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapQuickCharge Property R

Type	<b>boolean</b>
Remarks	If true the power management allows the charging of the battery in quick mode. The time for charging the battery is shorter than usual. Otherwise it is false.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>QuickChargeMode</b> Property, <b>QuickChargeTime</b> Property.

### CapShutdownPOS Property R

Type	<b>boolean</b>
Remarks	If true the device is able to explicitly shut down the POS. Otherwise it is false.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>shutdownPOS</b> Method.

## CapUPSChargeState Property R

<b>Type</b>	<b>int</b>										
<b>Remarks</b>	If not equal to zero, the UPS can deliver one or more charge states. It can contain any of the following values logically ORed together.										
	<table border="1"> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>PWR_UPS_FULL</td> <td>UPS battery is near full charge.</td> </tr> <tr> <td>PWR_UPS_WARNING</td> <td>UPS battery is near 50% charge.</td> </tr> <tr> <td>PWR_UPS_LOW</td> <td>UPS battery is near empty. Application shutdown should be started to ensure that it can be completed before the battery charge is depleted. A minimum of 2 minutes of normal system operation can be assumed when this state is entered unless this is the first state reported upon entering the “Off” power state.</td> </tr> <tr> <td>PWR_UPS_CRITICAL</td> <td>UPS battery is in a critical state and could be disconnected at any time without further warning.</td> </tr> </tbody> </table>	Value	Meaning	PWR_UPS_FULL	UPS battery is near full charge.	PWR_UPS_WARNING	UPS battery is near 50% charge.	PWR_UPS_LOW	UPS battery is near empty. Application shutdown should be started to ensure that it can be completed before the battery charge is depleted. A minimum of 2 minutes of normal system operation can be assumed when this state is entered unless this is the first state reported upon entering the “Off” power state.	PWR_UPS_CRITICAL	UPS battery is in a critical state and could be disconnected at any time without further warning.
Value	Meaning										
PWR_UPS_FULL	UPS battery is near full charge.										
PWR_UPS_WARNING	UPS battery is near 50% charge.										
PWR_UPS_LOW	UPS battery is near empty. Application shutdown should be started to ensure that it can be completed before the battery charge is depleted. A minimum of 2 minutes of normal system operation can be assumed when this state is entered unless this is the first state reported upon entering the “Off” power state.										
PWR_UPS_CRITICAL	UPS battery is in a critical state and could be disconnected at any time without further warning.										
	This property is initialized by the <b>open</b> method.										
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.										
<b>See Also</b>	<b>UPSChargeState</b> Property.										

## EnforcedShutdownDelayTime Property R/W

<b>Type</b>	<b>int</b>
<b>Remarks</b>	<p>If not equal to zero the system has a built-in mechanism to shut down the POS terminal after a determined time in a power fail situation. This property contains the time in milliseconds when the system will shut down automatically after a power failure. A power failure is the situation when the POS terminal is powered off or detached from the power supplying net and runs on battery power support. If zero no automatic shutdown is performed and the application has to call itself the <b>shutdownPOS</b> method.</p> <p>Applications will be informed about an initiated automatic shutdown.</p> <p>This property is initialized by the <b>open</b> method.</p>
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
<b>See Also</b>	<b>shutdownPOS</b> Method.

## PowerFailDelayTime Property R

Type	<b>int</b>
Remarks	<p>This property contains the time in milliseconds for power fail intervals which will not create a power fail situation. In some countries the power has sometimes short intervals where the power supply is interrupted. Those short intervals are in the range of milliseconds up to a few seconds and are handled by batteries or other electric equipment and should not cause a power fail situation. The power fail interval starts when the POS terminal is powered off or detached from the power supplying net and runs on battery power support. The power fail interval ends when the POS terminal is again powered on or attached to the power supplying net. However, if the power fail interval is longer than the time specified in the <b>PowerFailDelayTime</b> property a power fail situation is created.</p> <p>Usually this parameter is a configuration parameter of the underlying power management. So, the application can only read this property.</p> <p>This property is initialized by the <b>open</b> method.</p>
Errors	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>

## QuickChargeMode Property R

Type	<b>boolean</b>
Remarks	<p>If true, the battery is being recharged in a quick charge mode. If false, it is being charged in a normal mode.</p> <p>This property is only set if <b>CapQuickCharge</b> is true.</p>
Errors	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>
See Also	<b>CapQuickCharge</b> Property, <b>QuickChargeTime</b> Property.

## QuickChargeTime Property R

Type	<b>int</b>
Remarks	<p>This time specifies the remaining time for loading the battery in quick charge mode. After the time has elapsed, the battery loading mechanism of power management usually switches into normal mode.</p> <p>This time is specified in milliseconds.</p> <p>This property is only set if <b>CapQuickCharge</b> is true.</p>
Errors	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>
See Also	<b>CapQuickCharge</b> Property, <b>QuickChargeMode</b> Property.



**UPSChargeState Property R****Type**      **int****Remarks**      This property holds the actual UPS charge state.

It has one of the following values:

<b>Value</b>	<b>Meaning</b>
PWR_UPS_FULL	UPS battery is near full charge.
PWR_UPS_WARNING	UPS battery is near 50% charge.
PWR_UPS_LOW	UPS battery is near empty. Application shutdown should be started to ensure that is can be completed before the battery charge is depleted. A minimum of 2 minutes of normal system operation can be assumed when this state is entered unless this is the first state reported upon entering the “Off” power state.
PWR_UPS_CRITICAL	UPS battery is in a critical state and could be disconnected at any time without further warning.

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

**Errors**      A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15**See Also**      **CapUPSChargeState** Property.

## Methods

### shutdownPOS Method

**Syntax**      **void shutdownPOS ( ) throws JposException;**

**Remarks**      Call to shut down the POS terminal. This method will always shut down the system independent of the system power state.

If the POSPower is claimed, only the application which claimed the device is able to shut down the POS terminal.

Applications will be informed about an initiated shutdown.

It is recommended that in a power fail situation an application has to call this method after saving all data and setting the application to a defined state. If the **EnforcedShutdownDelayTime** property specifies a time greater than zero and the application did not call the **shutdownPOS** method within the time specified in **EnforcedShutdownDelayTime**, the system will be shut down automatically. This mechanism may be provided by an underlying operating system to prevent the battery from being emptied before the system is shut down. This method is only supported if **CapShutdownPOS** is true.

**Errors**      A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_ILLEGAL	This method is not supported (see the <b>CapShutdownPOS</b> property)

**See Also**      **CapShutdownPOS** Property, **EnforcedShutdownDelayTime** Property.

## Events

### DirectIOEvent

**Interface** jpos.events.DirectIOListner

**Method** **directIOOccurred (DirectIOEvent e);**

**Description** Provides Service information directly to the application. This event provides a means for a vendor-specific POSPower Service to provide events to the application that are not otherwise supported by the Control.

**Attributes** This event contains the following attributes:

Attributes	Type	Description
<i>EventNumber</i>	<i>int</i>	Event number whose specific values are assigned by the Device Service.
<i>Data</i>	<i>int</i>	Additional numeric data. Specific values vary by the <i>EventNumber</i> and the Device Service. This property is settable.
<i>Obj</i>	<i>object</i>	Additional data whose usage varies by the <i>EventNumber</i> and Device Service. This property is settable.

**Remarks** This event is to be used only for those types of vendor specific functions that are not otherwise described. Use of this event may restrict the application program from being used with other vendor's POSPower devices which may not have any knowledge of the Service's need for this event.

**See Also** "Events" on page 18, **directIO** Method.

## StatusUpdateEvent

**Interface** `jpos.events.StatusUpdateEventListener`

**Method** `statusUpdateEventOccurred (StatusUpdateEvent e);`

**Description** Delivered when **UPSChargeState** changes or an alarm situation occurs.

**Attributes** This event contains the following attribute:

Attributes	Type	Description
<i>Status</i>	<i>int</i>	See below.

The *Status* property contains the updated power status or alarm status.

Value	Meaning
PWR_SUE_UPS_FULL	UPS battery is near full charge. Can be returned if <b>CapUPSChargeState</b> contains PWR_UPS_FULL.
PWR_SUE_UPS_WARNING	UPS battery is near 50% charge. Can be returned if <b>CapUPSChargeState</b> contains PWR_UPS_WARNING.
PWR_SUE_UPS_LOW	UPS battery is near empty. Application shutdown should be started to ensure that it can be completed before the battery charge is depleted. A minimum of 2 minutes of normal system operation can be assumed when this state is entered unless this is the first charge state reported upon entering the “Off” state. Can be returned if <b>CapUPSChargeState</b> contains PWR_UPS_LOW.
PWR_SUE_UPS_CRITICAL	UPS is in critical state, and will in short time be disconnected. Can be returned if <b>CapUPSChargeState</b> contains PWR_UPS_CRITICAL.
PWR_SUE_FAN_STOPPED	The CPU fan is stopped. Can be returned if <b>CapFanAlarm</b> is true.
PWR_SUE_FAN_RUNNING	The CPU fan is running. Can be returned if <b>CapFanAlarm</b> is true.
PWR_SUE_TEMPERATURE_HIGH	The CPU is running on high temperature. Can be returned if <b>CapHeatAlarm</b> is true.
PWR_SUE_TEMPERATURE_OK	The CPU is running on normal temperature. Can be returned if <b>CapHeatAlarm</b> is true.
PWR_SUE_SHUTDOWN	The system will shutdown immediately.

*Note that Release 1.3* added Power State Reporting with additional *Power reporting StatusUpdateEvent values*. See “StatusUpdateEvent” description on page 78.

**See Also** **CapFanAlarm**, **CapHeatAlarm**, **CapUPSChargeState**, and **UPSChargeState** Properties.

# POS Printer

## Summary

### Properties

<i>Common</i>	<i>Ver</i>	<i>Type</i>	<i>Access</i>	<i>May Use After</i>
<b>AutoDisable</b>		boolean	R/W	<i>Not Supported</i>
<b>CapPowerReporting</b>	1.3	int	R	open
<b>CheckHealthText</b>		String	R	open
<b>Claimed</b>		boolean	R	open
<b>DataCount</b>		int	R	<i>Not Supported</i>
<b>DataEventEnabled</b>		boolean	R/W	<i>Not Supported</i>
<b>DeviceEnabled</b>		boolean	R/W	open & claim
<b>FreezeEvents</b>		boolean	R/W	open
<b>OutputID</b>		int	R	open
<b>PowerNotify</b>	1.3	int	R/W	open
<b>PowerState</b>	1.3	int	R	open
<b>State</b>		int	R	--
<b>DeviceControlDescription</b>		String	R	--
<b>DeviceControlVersion</b>		int	R	--
<b>DeviceServiceDescription</b>		String	R	open
<b>DeviceServiceVersion</b>		int	R	open
<b>PhysicalDeviceDescription</b>		String	R	open
<b>PhysicalDeviceName</b>		String	R	open
<i>Specific</i>	<i>Ver</i>	<i>Type</i>	<i>Access</i>	<i>May Use After</i>
<b>CapCharacterSet</b>		int	R	open
<b>CapConcurrentJrnRec</b>		boolean	R	open
<b>CapConcurrentJrnSlp</b>		boolean	R	open
<b>CapConcurrentRecSlp</b>		boolean	R	open
<b>CapCoverSensor</b>		boolean	R	open
<b>CapTransaction</b>		boolean	R	open

<i>Specific (continued)</i>	<i>Ver</i>	<i>Type</i>	<i>Access</i>	<i>May Use After</i>
<b>CapJrnPresent</b>		boolean	R	open
<b>CapJrn2Color</b>		boolean	R	open
<b>CapJrnBold</b>		boolean	R	open
<b>CapJrnCartridgeSensor</b>	1.5	int	R	open
<b>CapJrnColor</b>	1.5	int	R	open
<b>CapJrnDhigh</b>		boolean	R	open
<b>CapJrnDwide</b>		boolean	R	open
<b>CapJrnDwideDhigh</b>		boolean	R	open
<b>CapJrnEmptySensor</b>		boolean	R	open
<b>CapJrnItalic</b>		boolean	R	open
<b>CapJrnNearEndSensor</b>		boolean	R	open
<b>CapJrnUnderline</b>		boolean	R	open
<b>CapRecPresent</b>		boolean	R	open
<b>CapRec2Color</b>		boolean	R	open
<b>CapRecBarCode</b>		boolean	R	open
<b>CapRecBitmap</b>		boolean	R	open
<b>CapRecBold</b>		boolean	R	open
<b>CapRecCartridgeSensor</b>	1.5	int	R	open
<b>CapRecColor</b>	1.5	int	R	open
<b>CapRecDhigh</b>		boolean	R	open
<b>CapRecDwide</b>		boolean	R	open
<b>CapRecDwideDhigh</b>		boolean	R	open
<b>CapRecEmptySensor</b>		boolean	R	open
<b>CapRecItalic</b>		boolean	R	open
<b>CapRecLeft90</b>		boolean	R	open
<b>CapRecMarkFeed</b>	1.5	int	R	open
<b>CapRecNearEndSensor</b>		boolean	R	open
<b>CapRecPapercut</b>		boolean	R	open
<b>CapRecRight90</b>		boolean	R	open
<b>CapRecRotate180</b>		boolean	R	open
<b>CapRecStamp</b>		boolean	R	open
<b>CapRecUnderline</b>		boolean	R	open

<i>Specific (continued)</i>	<i>Ver</i>	<i>Type</i>	<i>Access</i>	<i>May Use After</i>
<b>CapSlpPresent</b>		boolean	R	open
<b>CapSlpFullslip</b>		boolean	R	open
<b>CapSlp2Color</b>		boolean	R	open
<b>CapSlpBarCode</b>		boolean	R	open
<b>CapSlpBitmap</b>		boolean	R	open
<b>CapSlpBothSidesPrint</b>	1.5	boolean	R	open
<b>CapSlpCartridgeSensor</b>	1.5	int	R	open
<b>CapSlpColor</b>	1.5	int	R	open
<b>CapSlpBold</b>		boolean	R	open
<b>CapSlpDhigh</b>		boolean	R	open
<b>CapSlpDwide</b>		boolean	R	open
<b>CapSlpDwideDhigh</b>		boolean	R	open
<b>CapSlpEmptySensor</b>		boolean	R	open
<b>CapSlpItalic</b>		boolean	R	open
<b>CapSlpLeft90</b>		boolean	R	open
<b>CapSlpNearEndSensor</b>		boolean	R	open
<b>CapSlpRight90</b>		boolean	R	open
<b>CapSlpRotate180</b>		boolean	R	open
<b>CapSlpUnderline</b>		boolean	R	open
<b>AsyncMode</b>		boolean	R/W	open
<b>CartridgeNotify</b>	1.5	int	R/W	open
<b>CharacterSet</b>		int	R/W	open, claim, & enable
<b>CharacterSetList</b>		String	R	open
<b>CoverOpen</b>		boolean	R	open, claim, & enable
<b>ErrorLevel</b>		int	R	open
<b>ErrorStation</b>		int	R	open
<b>ErrorString</b>		String	R	open
<b>FontTypefaceList</b>		String	R	open
<b>FlagWhenIdle</b>		boolean	R/W	open
<b>MapMode</b>		int	R/W	open
<b>RotateSpecial</b>		int	R/W	open

<i>Specific (continued)</i>	<i>Ver</i>	<i>Type</i>	<i>Access</i>	<i>May Use After</i>
<b>JrnLineChars</b>		int	R/W	open, claim, & enable
<b>JrnLineCharsList</b>		String	R	open
<b>JrnLineHeight</b>		int	R/W	open, claim, & enable
<b>JrnLineSpacing</b>		int	R/W	open, claim, & enable
<b>JrnLineWidth</b>		int	R	open, claim, & enable
<b>JrnLetterQuality</b>		boolean	R/W	open, claim, & enable
<b>JrnEmpty</b>		boolean	R	open, claim, & enable
<b>JrnNearEnd</b>		boolean	R	open, claim, & enable
<b>JrnCartridgeState</b>	1.5	int	R	open, claim, & enable
<b>JrnCurrentCartridge</b>	1.5	int	R/W	open, claim, & enable
<b>RecLineChars</b>		int	R/W	open, claim, & enable
<b>RecLineCharsList</b>		String	R	open
<b>RecLineHeight</b>		int	R/W	open, claim, & enable
<b>RecLineSpacing</b>		int	R/W	open, claim, & enable
<b>RecLineWidth</b>		int	R	open, claim, & enable
<b>RecLetterQuality</b>		boolean	R/W	open, claim, & enable
<b>RecEmpty</b>		boolean	R	open, claim, & enable
<b>RecNearEnd</b>		boolean	R	open, claim, & enable
<b>RecSidewaysMaxLines</b>		int	R	open, claim, & enable
<b>RecSidewaysMaxChars</b>		int	R	open, claim, & enable
<b>RecLinesToPaperCut</b>		int	R	open, claim, & enable
<b>RecBarCodeRotationList</b>		String	R	open
<b>RecCartridgeState</b>	1.5	int	R	open, claim, & enable
<b>RecCurrentCartridge</b>	1.5	int	R/W	open, claim, & enable
<b>SlpLineChars</b>		int	R/W	open, claim, & enable
<b>SlpLineCharsList</b>		String	R	open
<b>SlpLineHeight</b>		int	R/W	open, claim, & enable
<b>SlpLineSpacing</b>		int	R/W	open, claim, & enable
<b>SlpLineWidth</b>		int	R	open, claim, & enable
<b>SlpLetterQuality</b>		boolean	R/W	open, claim, & enable
<b>SlpEmpty</b>		boolean	R	open, claim, & enable
<b>SlpNearEnd</b>		boolean	R	open, claim, & enable
<b>SlpSidewaysMaxLines</b>		int	R	open, claim, & enable



<b>SlpSidewaysMaxChars</b>		int	R	open, claim, & enable
<b>SlpMaxLines</b>		int	R	open, claim, & enable
<b>SlpLinesNearEndToEnd</b>		int	R	open, claim, & enable
<b>SlpBarCodeRotationList</b>		String	R	open
<b>SlpPrintSide</b>	1.5	int	R	open, claim, & enable
<b>SlpCartridgeState</b>	1.5	int	R	open, claim, & enable
<b>SlpCurrentCartridge</b>	1.5	int	R/W	open, claim, & enable

### Methods

#### *Common*

	<i>Ver</i>	<i>May Use After</i>
<b>open</b>		--
<b>close</b>		open
<b>claim</b>		open
<b>release</b>		open & claim
<b>checkHealth</b>		open, claim, & enable
<b>clearInput</b>		<i>Not Supported</i>
<b>clearOutput</b>		open & claim
<b>directIO</b>		open

#### *Specific*

<b>printNormal</b>		open, claim, & enable
<b>printTwoNormal</b>		open, claim, & enable
<b>printImmediate</b>		open, claim, & enable
<b>beginInsertion</b>		open, claim, & enable
<b>endInsertion</b>		open, claim, & enable
<b>beginRemoval</b>		open, claim, & enable
<b>endRemoval</b>		open, claim, & enable
<b>cutPaper</b>		open, claim, & enable
<b>rotatePrint</b>		open, claim, & enable
<b>printBarCode</b>		open, claim, & enable
<b>printBitmap</b>		open, claim, & enable
<b>transactionPrint</b>		open, claim, & enable
<b>validateData</b>		open, claim, & enable
<b>setBitmap</b>		open, claim, & enable

*Specific (continued)*

<b>setLogo</b>		open, claim, & enable
<b>changePrintSide</b>	1.5	open, claim, & enable
<b>markFeed</b>	1.5	open, claim, & enable

**Events**

<i>Name</i>	<i>Ver</i>	<i>May Occur After</i>
<b>DataEvent</b>		<i>Not Supported</i>
<b>DirectIOEvent</b>	1.3	open & claim
<b>ErrorEvent</b>		open, claim, & enable
<b>OutputCompleteEvent</b>		open, claim, & enable
<b>StatusUpdateEvent</b>		open, claim, & enable

## General Information

The POS Printer Control's class name is "jpos.POSPrinter".  
The device constants are contained in the class "jpos.POSPrinterConst".  
See "Package Structure" on page 40.

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

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

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

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

### Capabilities

The POS printer has the following capability:

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

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

The following capabilities are not addressed in this version of the JavaPOS specification. A device service may choose to support them through the **directIO** mechanism.

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

## Model

The POS Printer follows the JavaPOS model for output devices, with some enhancements:

- The following methods are always performed synchronously: **beginInsertion**, **endInsertion**, **beginRemoval**, **endRemoval**, **changePrintSide** and **checkHealth**. These methods will fail if asynchronous output is outstanding.
- The **printImmediate** method is also always performed synchronously: This method tries to print its data immediately (that is, as the very next printer operation). It may be called when asynchronous output is outstanding. This method is primarily intended for use in exception conditions when asynchronous output is outstanding.
- The following methods are performed either synchronously or asynchronously, depending on the value of the **AsyncMode** property: **printNormal**, **printTwoNormal**, **cutPaper**, **rotatePrint**, **printBarCode**, **printBitmap**, **transactionPrint**, and **markFeed**. When **AsyncMode** is false, then these methods print synchronously.
- When **AsyncMode** is true, then these methods operate as follows:
  - The Device buffers the request, sets the **OutputID** property to an identifier for this request, and returns as soon as possible. When the request completes successfully, an **OutputCompleteEvent** is enqueued. A property of this event contains the **OutputID** of the completed request.
  - Asynchronous printer methods will not throw an exception due to a printing problem, such as out of paper or printer fault. These errors will only be reported by an **ErrorEvent**. An exception is thrown only if the printer is not claimed and enabled, a parameter is invalid, or the request cannot be enqueued. The first two error cases are due to an application error, while the last is a serious system resource error exception.
  - If an error occurs while performing an asynchronous request, an **ErrorEvent** is enqueued. The **ErrorStation** property is set to the station or stations that were printing when the error occurred. The **ErrorLevel** and **ErrorString** properties are also set.
  - The event handler may call synchronous print methods (but not asynchronous methods), then can either retry the outstanding output or clear it.
  - All asynchronous output is performed on a first-in first-out basis.
  - All output buffered may be deleted by calling **clearOutput**. **OutputCompleteEvents** will not be delivered for cleared output. This method also stops any output that may be in progress (when possible).
  - The property **FlagWhenIdle** may be set to cause the a **StatusUpdateEvent** to be enqueued when all outstanding outputs have finished, whether successfully or because they were cleared.

- Transaction mode printing is supported. A transaction is a sequence of print operations that are printed to a station as a unit. Print operations which may be included in a transaction are **printNormal**, **cutPaper**, **rotatePrint**, **printBarCode**, **printBitmap**, and **markFeed**. During a transaction, the print operations are first validated. If valid, they are added to the transaction but not printed yet. Once the application has added as many operations as required, then the transaction print method is called.

If the transaction is printed synchronously and an exception is not thrown, then the entire transaction printing was successful. If the transaction is printed asynchronously, then the asynchronous print rules listed above are followed. If an error occurs and the Error Event handler causes a retry, the entire transaction is retried.

The printer error reporting model is as follows:

- Printer out-of-paper and cover open conditions are reported by setting the exception's (or `ErrorEvent`'s) *ErrorCode* to `JPOS_E_EXTENDED` and then setting the associated *ErrorCodeExtended* to one of the following error conditions:  
`JPOS_EPTR_JRN_EMPTY`,  
`JPOS_EPTR_REC_EMPTY`,  
`JPOS_EPTR_SLP_EMPTY`,  
`JPOS_EPTR_COVER_OPEN`,  
`JPOS_JRN_CARTRIDGE_REMOVED`,  
`JPOS_REC_CARTRIDGE_REMOVED`,  
`JPOS_SLP_CARTRIDGE_REMOVED`,  
`JPOS_JRN_CARTRIDGE_EMPTY`,  
`JPOS_REC_CARTRIDGE_EMPTY`,  
`JPOS_SLP_CARTRIDGE_EMPTY`,  
`JPOS_JRN_HEAD_CLEANING`,  
`JPOS_REC_HEAD_CLEANING`, or  
`JPOS_SLP_HEAD_CLEANING`.
- Other printer errors are reported by setting the exception's (or `ErrorEvent`'s) *ErrorCode* to `JPOS_E_FAILURE` or another standard error status. These failures are typically due to a printer fault or jam, or to a more serious error.

***Release 1.5 and later – Print cartridge support added***

The print cartridge model is as follows:

- The **CapJrnCartridgeSensor**, **CapRecCartridgeSensor**, and the **CapSlpCartridgeSensor** capabilities are used to determine whether the printer has the ability to detect the operating condition of the cartridge.
- Prior to determining a cartridge's operating condition, a cartridge is selected by using one of the following properties: **JrnCurrentCartridge**, **RecCurrentCartridge**, or **SlpCurrentCartridge**.
- The condition of the selected cartridge is set up using one of the **JrnCartridgeState**, **RecCartridgeState** or **SlpCartridgeState** properties. The values that these properties can take in order of high priority to low priority are as follows: PTR\_CART\_UNKNOWN, PTR\_CART\_REMOVED, PTR\_CART\_EMPTY, PTR\_CART\_CLEANING, PTR\_CART\_NEAREND, PTR\_CART\_OK.
- **CapJrnColor**, **CapRecColor**, and **CapSlpColor** capabilities are used to determine the color capabilities of the station.

**Mono Color**

- **CapJrnColor**, **CapRecColor**, and **CapSlpColor** capabilities are set to PTR\_COLOR\_PRIMARY.

**Two Color**

- **CapJrnColor**, **CapRecColor**, and **CapSlpColor** capabilities are a logical OR combination of PTR\_COLOR\_PRIMARY and PTR\_COLOR\_CUSTOM1.
- PTR\_COLOR\_CUSTOM1 refers to the secondary color, usually red.
- Secondary color printing can be done by using the ESC|rC escape sequence.

**Custom Color**

- **CapJrnColor**, **CapRecColor**, and **CapSlpColor** capabilities are a logical OR combination of PTR\_COLOR\_PRIMARY and any of the following bit values:  
PTR\_COLOR\_CUSTOM1, PTR\_COLOR\_CUSTOM2,  
PTR\_COLOR\_CUSTOM3, PTR\_COLOR\_CUSTOM4,  
PTR\_COLOR\_CUSTOM5, PTR\_COLOR\_CUSTOM6.
- Selection of a custom color can be done using the ESC|#rC escape sequence.

**Full Color**

- **CapJrnColor**, **CapRecColor**, and **CapSlpColor** capabilities are a logical OR combination of PTR\_COLOR\_FULL and the following values:  
PTR\_COLOR\_CYAN, PTR\_COLOR\_MAGENTA,  
PTR\_COLOR\_YELLOW.
- PTR\_COLOR\_FULL is not used to indicate that a print cartridge is currently installed in the printer. Rather, it is used to indicate that the printer has the ability to print in full color mode.
- Full color printing is accomplished by using the ESC|#fC escape sequence.

**Full Color with Custom Color(s)**

- **CapJrnColor**, **CapRecColor**, and **CapSlpColor** are a logical OR combination of the settings for **Custom Color** and **Full Color**.

***Release 1.5 and later – Cartridge State Reporting Requirements for DeviceEnabled***

The print cartridge state reporting model is:

- **CartridgeNotify** property: The application may set this property to enable cartridge state reporting via **StatusUpdateEvents** and **JrnCartridgeState**, **RecCartridgeState**, and **SlpCartridgeState** properties. This property may only be set before the device is enabled (that is, before **DeviceEnabled** is set to true). This restriction allows simpler implementation of cartridge status notification with no adverse effects on the application. The application is either prepared to receive notifications or doesn't want them, and has no need to switch between these cases. This property may be one of:

PTR\_CN\_DISABLED, or PTR\_CN\_ENABLED

The following semantics are added to **DeviceEnabled** when the **CapJrnCartridgeSensor**, **CapRecCartridgeSensor**, and **CapSlpCartridgeSensor** capabilities are not zero, and **CartridgeNotify** is set to PTR\_CN\_ENABLED:

- Monitoring the cartridge state begins when **DeviceEnabled** changes from false to true.
- When **DeviceEnabled** changes from true to false, the state of the cartridge is no longer valid. Therefore, **JrnCartridgeState**, **RecCartridgeState**, and **SlpCartridgeState** properties are set to PTR\_CART\_UNKNOWN.

## Device Sharing

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

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



## Data Characters and Escape Sequences

The default character set of all POS printers is assumed to support at least the ASCII characters 0x20 through 0x7F, which include spaces, digits, uppercase, lowercase, and some special characters. If the printer does not support lowercase characters, then the device service may translate them to uppercase.

Every escape sequence begins with the escape character ESC, whose value is 27 decimal, followed by a vertical bar ('|'). This is followed by zero or more digits and/or lowercase alphabetic characters. The escape sequence is terminated by an uppercase alphabetic character. Sequences that do not begin with ESC "|" are passed through to the printer. Also, sequences that begin with ESC "|" but which are not valid escape sequences are passed through to the printer.

To determine if escape sequences or data can be performed on a printer station, the application can call the **validateData** method. (For some escape sequences, corresponding capability properties can also be used.)

The following escape sequences are recognized. If an escape sequence specifies an operation that is not supported by the printer station, then it is ignored.

**Commands** Perform indicated action.

Name	Data	Remarks
Paper cut	ESC #P	Cuts receipt paper. The character '#' is replaced by an ASCII decimal string telling the percentage cut desired. If '#' is omitted, then a full cut is performed. For example: The C string "\x1B 75P" requests a 75% partial cut.
Feed and Paper cut	ESC #fp	Cuts receipt paper, after feeding the paper by the <b>RecLinesToPaperCut</b> lines. The character '#' is defined by the "Paper cut" escape sequence.
Feed, Paper cut, and Stamp	ESC #sP	Cuts and stamps receipt paper, after feeding the paper by the <b>RecLinesToPaperCut</b> lines. The character '#' is defined by the "Paper cut" escape sequence.
Fire stamp	ESC sL	Fires the stamp solenoid, which usually contains a graphical store emblem.
Print bitmap	ESC #B	Prints the pre-stored bitmap. The character '#' is replaced by the bitmap number. See <b>setBitmap</b> method.
Print top logo	ESC tL	Prints the pre-stored top logo.
Print bottom logo	ESC bL	Prints the pre-stored bottom logo.
Feed lines	ESC #lF	Feed the paper forward by lines. The character '#' is replaced by an ASCII decimal string telling the number of lines to be fed. If '#' is omitted, then one line is fed.
Feed units	ESC #uF	Feed the paper forward by mapping mode units. The character '#' is replaced by an ASCII decimal string telling the number of units to be fed. If '#' is omitted, then one unit is fed.
Feed reverse	ESC #rF	Feed the paper backward. The character '#' is replaced by an ASCII decimal string telling the number of lines to be fed. If '#' is omitted, then one line is fed.

**Print Mode** Characteristics that are remembered until explicitly changed.

Name	Data	Remarks
Font typeface selection	ESC #fT	Selects a new typeface for the following data. Values for the character '#' are: 0 = Default typeface. 1 = Select first typeface from the <b>FontTypefaceList</b> property. 2 = Select second typeface from the <b>FontTypefaceList</b> property. And so on.

**Print Line** Characteristics that are reset at the end of each print method or by a “Normal” sequence.

Name Here	Data	Remarks
Bold	ESC  bC	Prints in bold or double-strike.
Underline	ESC  #uC	Prints with underline. The character ‘#’ is replaced by an ASCII decimal string telling the thickness of the underline in printer dot units. If ‘#’ is omitted, then a printer-specific default thickness is used.
Italic	ESC  iC	Prints in italics.
Alternate color (Custom)	ESC  #rC	Prints using an alternate custom color. The character ‘#’ is replaced by an ASCII decimal string indicating the desired color. The value of the decimal string is equal to the value of the cartridge constant used in the printer device properties. If ‘#’ is omitted, then the secondary color (Custom Color 1) is selected. Custom Color 1 is usually red.
Reverse video	ESC  rvC	Prints in a reverse video format.
Shading	ESC  #sC	Prints in a shaded manner. The character ‘#’ is replaced by an ASCII decimal string telling the percentage shading desired. If ‘#’ is omitted, then a printer-specific default level of shading is used.
Single high & wide	ESC  1C	Prints normal size.
Double wide	ESC  2C	Prints double-wide characters.
Double high	ESC  3C	Prints double-high characters.
Double high & wide	ESC  4C	Prints double-high/double-wide characters.
Scale horizontally	ESC  #hC	Prints with the width scaled ‘#’ times the normal size, where ‘#’ is replaced by an ASCII decimal string.
Scale vertically	ESC  #vC	Prints with the height scaled ‘#’ times the normal size, where ‘#’ is replaced by an ASCII decimal string.
RGB Color	ESC  #fC	Prints in # color. The character ‘#’ is replaced by an ASCII decimal string indicating the additive amount of RGB to produce the desired color. There are 3 digits each of Red, Green, and Blue elements. Valid values range from “000” to “255”. (E.g., “255255000” represents yellow). Color Matching to the subtractive percentage of CMY (Cyan, Magenta and Yellow color components) to produce the desired color matching specified by RGB is up to the Service. If ‘#’ is omitted, then the primary color is used. Bitmap printing is not affected. ( <b>See Note below.</b> )
SubScript	ESC  tbC	Prints SubScript characters. ( <b>See Note below.</b> )
SuperScript	ESC  tpC	Prints SuperScript characters. ( <b>See Note below.</b> )
Center	ESC  cA	Aligns following text in the center.
Right justify	ESC  rA	Aligns following text at the right.
Normal	ESC  N	Restores printer characteristics to normal condition.

**Note:** These escape sequences are only available in Release 1.5 and later.

## Properties

### AsyncMode Property R/W

Type	<b>boolean</b>
Remarks	If true, then the print methods <b>printNormal</b> , <b>printTwoNormal</b> , <b>cutPaper</b> , <b>rotatePrint</b> , <b>printBarCode</b> , and <b>printBitmap</b> will be performed asynchronously. If false, they will be printed synchronously.  This property is initialized to false by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapCharacterSet Property R

Type	<b>int</b>												
Remarks	Holds the default character set capability. It has one of the following values: <table border="1"> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>PTR_CCS_ALPHA</td> <td>The default character set supports uppercase alphabetic plus numeric, space, minus, and period.</td> </tr> <tr> <td>PTR_CCS_ASCII</td> <td>The default character set supports all ASCII characters 0x20 through 0x7F.</td> </tr> <tr> <td>PTR_CCS_KANA</td> <td>The default character set supports partial code page 932, including ASCII characters 0x20 through 0x7F and the Japanese Kana characters 0xA1 through 0xDF, but excluding the Japanese Kanji characters.</td> </tr> <tr> <td>PTR_CCS_KANJI</td> <td>The default character set supports code page 932, including the Shift-JIS Kanji characters, Levels 1 and 2.</td> </tr> <tr> <td>PTR_CCS_UNICODE</td> <td>The default character set supports UNICODE.</td> </tr> </tbody> </table> <p>The default character set may contain a superset of these ranges. The initial <b>CharacterSet</b> property may be examined for additional information.</p> <p>This property is initialized by the <b>open</b> method.</p>	Value	Meaning	PTR_CCS_ALPHA	The default character set supports uppercase alphabetic plus numeric, space, minus, and period.	PTR_CCS_ASCII	The default character set supports all ASCII characters 0x20 through 0x7F.	PTR_CCS_KANA	The default character set supports partial code page 932, including ASCII characters 0x20 through 0x7F and the Japanese Kana characters 0xA1 through 0xDF, but excluding the Japanese Kanji characters.	PTR_CCS_KANJI	The default character set supports code page 932, including the Shift-JIS Kanji characters, Levels 1 and 2.	PTR_CCS_UNICODE	The default character set supports UNICODE.
Value	Meaning												
PTR_CCS_ALPHA	The default character set supports uppercase alphabetic plus numeric, space, minus, and period.												
PTR_CCS_ASCII	The default character set supports all ASCII characters 0x20 through 0x7F.												
PTR_CCS_KANA	The default character set supports partial code page 932, including ASCII characters 0x20 through 0x7F and the Japanese Kana characters 0xA1 through 0xDF, but excluding the Japanese Kanji characters.												
PTR_CCS_KANJI	The default character set supports code page 932, including the Shift-JIS Kanji characters, Levels 1 and 2.												
PTR_CCS_UNICODE	The default character set supports UNICODE.												
Errors	A JposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.												
See Also	<b>CharacterSet</b> Property.												

## CapConcurrentJrnRec Property R

Type	<b>boolean</b>
Remarks	<p>If true, then the Journal and Receipt stations can print at the same time. The <b>printTwoNormal</b> method may be used with the PTR_TWO_RECEIPT_JOURNAL and PTR_S_JOURNAL_RECEIPT station parameter. If false, the application should print to only one of the stations at a time, and minimize transitions between the stations. Non-concurrent printing may be required for reasons such as:</p> <ul style="list-style-type: none"><li>• Higher likelihood of error, such as greater chance of paper jams when moving between the stations.</li><li>• Higher performance when each station is printed separately.</li></ul> <p>This property is initialized by the <b>open</b> method.</p>
Errors	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>

## CapConcurrentJrnSlp Property R

Type	<b>boolean</b>
Remarks	<p>If true, then the Journal and Slip stations can print at the same time. The <b>printTwoNormal</b> method may be used with the PTR_TWO_RECEIPT_JOURNAL and PTR_S_JOURNAL_SLIP station parameter. If false, the application must use the sequence <b>beginInsertion/endInsertion</b> followed by print requests to the Slip followed by <b>beginRemoval/endRemoval</b> before printing on the Journal. Non-concurrent printing may be required for reasons such as:</p> <ul style="list-style-type: none"><li>• Physical constraints, such as the Slip form being placed in front of the Journal station.</li><li>• Higher likelihood of error, such as greater chance of paper jams when moving between the stations.</li><li>• Higher performance when each station is printed separately.</li></ul> <p>This property is initialized by the <b>open</b> method.</p>
Errors	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>

## CapConcurrentRecSlip Property R

Type	<b>boolean</b>
Remarks	<p>If true, then the Receipt and Slip stations can print at the same time. The <b>printTwoNormal</b> method may be used with the PTR_TWO_RECEIPT_JOURNAL and PTR_S_RECEIPT_SLIP station parameter. If false, the application must use the sequence <b>beginInsertion/endInsertion</b> followed by print requests to the Slip followed by <b>beginRemoval/endRemoval</b> before printing on the Receipt. Non-concurrent printing may be required for reasons such as:</p> <ul style="list-style-type: none"><li>• Physical constraints, such as the Slip form being placed in front of the Receipt station.</li><li>• Higher likelihood of error, such as greater chance of paper jams when moving between the stations.</li><li>• Higher performance when each station is printed separately.</li></ul> <p>This property is initialized by the <b>open</b> method.</p>
Errors	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>

## CapCoverSensor Property R

Type	<b>boolean</b>
Remarks	<p>If true, then the printer has a “cover open” sensor.</p> <p>This property is initialized by the <b>open</b> method.</p>
Errors	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>

## CapJrn2Color Property R

Type	<b>boolean</b>
Remarks	<p>If true, then the journal can print dark plus an alternate color.</p> <p>This property is initialized by the <b>open</b> method.</p>
Errors	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>

**CapJrnBold Property R**

<b>Type</b>	<b>boolean</b>
<b>Remarks</b>	If true, then the journal can print bold characters.  This property is initialized by the <b>open</b> method.
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

**CapJrnCartridgeSensor Property R** *Added in Release 1.5*

<b>Type</b>	<b>int</b>										
<b>Remarks</b>	This bit mapped parameter is used to indicate the presence of Journal Cartridge monitoring sensors.  If <b>CapJrnPresent</b> is false, this property is “0”. Otherwise it is a logical OR combination of any of the following values:										
	<table> <thead> <tr> <th><b>Value</b></th> <th><b>Meaning</b></th> </tr> </thead> <tbody> <tr> <td>PTR_CART_REMOVED</td> <td>There is a function to indicate that the Cartridge has been removed.</td> </tr> <tr> <td>PTR_CART_EMPTY</td> <td>There is a function to indicate that the Cartridge is empty.</td> </tr> <tr> <td>PTR_CART_CLEANING</td> <td>There is a function to indicate that the head is being cleaned.</td> </tr> <tr> <td>PTR_CART_NEAREND</td> <td>There is a function to indicate that the color Cartridge is near end.</td> </tr> </tbody> </table>	<b>Value</b>	<b>Meaning</b>	PTR_CART_REMOVED	There is a function to indicate that the Cartridge has been removed.	PTR_CART_EMPTY	There is a function to indicate that the Cartridge is empty.	PTR_CART_CLEANING	There is a function to indicate that the head is being cleaned.	PTR_CART_NEAREND	There is a function to indicate that the color Cartridge is near end.
<b>Value</b>	<b>Meaning</b>										
PTR_CART_REMOVED	There is a function to indicate that the Cartridge has been removed.										
PTR_CART_EMPTY	There is a function to indicate that the Cartridge is empty.										
PTR_CART_CLEANING	There is a function to indicate that the head is being cleaned.										
PTR_CART_NEAREND	There is a function to indicate that the color Cartridge is near end.										
	Note that the above mentioned values are arranged according to their priority level.  This property is initialized by the <b>open</b> method.										
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.										
<b>See Also</b>	<b>JrnCartridgeState</b> Property, <b>JrnCurrentCartridge</b> Property, <b>CartridgeNotify</b> Property.										

**CapJrnColor Property R*****Added in Release 1.5*****Type**      **int****Remarks**      This capability indicates the availability of Journal color cartridges.

If **CapJrnPresent** is false, this property is “0”. Otherwise, this property indicates the supported color cartridges.

**CapJrnColor** is a logical OR combination of any of the following values:

<b>Value</b>	<b>Meaning</b>
PTR_COLOR_PRIMARY	Supports Primary Color (Usually Black)
PTR_COLOR_CUSTOM1	Supports 1 <sup>st</sup> Custom Color (Secondary Color, usually Red)
PTR_COLOR_CUSTOM2	Supports 2 <sup>nd</sup> Custom Color
PTR_COLOR_CUSTOM3	Supports 3 <sup>rd</sup> Custom Color
PTR_COLOR_CUSTOM4	Supports 4 <sup>th</sup> Custom Color
PTR_COLOR_CUSTOM5	Supports 5 <sup>th</sup> Custom Color
PTR_COLOR_CUSTOM6	Supports 6 <sup>th</sup> Custom Color
PTR_COLOR_CYAN	Supports Cyan Color for full color printing
PTR_COLOR_MAGENTA	Supports Magenta Color for full color printing
PTR_COLOR_YELLOW	Supports Yellow Color for full color printing
PTR_COLOR_FULL	Supports Full Color.

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

**Errors**      A JposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.



### CapJrnDhigh Property R

Type	<b>boolean</b>
Remarks	If true, then the journal can print double high characters.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapJrnDwide Property R

Type	<b>boolean</b>
Remarks	If true, then the journal can print double wide characters.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapJrnDwideDhigh Property R

Type	<b>boolean</b>
Remarks	If true, then the journal can print double high / double wide characters.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapJrnEmptySensor Property R

Type	<b>boolean</b>
Remarks	If true, then the journal has an out-of-paper sensor.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapJrnItalic Property R

Type	<b>boolean</b>
Remarks	If true, then the journal can print italic characters.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapJrnNearEndSensor Property R

Type	<b>boolean</b>
Remarks	If true, then the journal has a low paper sensor.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapJrnPresent Property R

Type	<b>boolean</b>
Remarks	If true, then the journal print station is present.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapJrnUnderline Property R

Type	<b>boolean</b>
Remarks	If true, then the journal can underline characters  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapRec2Color Property R

Type	<b>boolean</b>
Remarks	If true, then the receipt can print dark plus an alternate color.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapRecBarCode Property R

Type	<b>boolean</b>
Remarks	If true, then the receipt has bar code printing capability.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapRecBitmap Property R

Type	<b>boolean</b>
Remarks	If true, then the receipt can print bitmaps.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapRecBold Property R

Type	<b>boolean</b>
Remarks	If true, then the receipt can print bold characters.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

**CapRecCartridgeSensor Property R** *Added in Release 1.5***Type** `int`**Remarks** This bit mapped parameter is used to indicate the presence of Receipt Cartridge monitoring sensors.If **CapRecPresent** is false, this property is “0”. Otherwise it is a logical OR combination of any of the following values:

<b>Value</b>	<b>Meaning</b>
PTR_CART_REMOVED	There is a function to indicate that the Cartridge has been removed.
PTR_CART_EMPTY	There is a function to indicate that the Cartridge is empty.
PTR_CART_CLEANING	There is a function to indicate that the head is being cleaned.
PTR_CART_NEAREND	There is a function to indicate that the color Cartridge is near end.

Note that the above mentioned values are arranged according to their priority level.

This property is initialized by the **open** method.**Errors** A JposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.**See Also** **RecCartridgeState** Property, **RecCurrentCartridge** Property, **CartridgeNotify** Property.

**CapRecColor Property R****Added in Release 1.5****Type**      **int****Remarks**      This capability indicates the availability of Receipt color cartridges.

If **CapRecPresent** is false, this property is “0”. Otherwise, this property indicates the supported color cartridges.

**CapRecColor** is a logical OR combination of any of the following values:

<b>Value</b>	<b>Meaning</b>
PTR_COLOR_PRIMARY	Supports Primary Color (Usually Black)
PTR_COLOR_CUSTOM1	Supports 1 <sup>st</sup> Custom Color (Secondary Color, usually Red)
PTR_COLOR_CUSTOM2	Supports 2 <sup>nd</sup> Custom Color
PTR_COLOR_CUSTOM3	Supports 3 <sup>rd</sup> Custom Color
PTR_COLOR_CUSTOM4	Supports 4 <sup>th</sup> Custom Color
PTR_COLOR_CUSTOM5	Supports 5 <sup>th</sup> Custom Color
PTR_COLOR_CUSTOM6	Supports 6 <sup>th</sup> Custom Color
PTR_COLOR_CYAN	Supports Cyan Color for full color printing
PTR_COLOR_MAGENTA	Supports Magenta Color for full color printing
PTR_COLOR_YELLOW	Supports Yellow Color for full color printing
PTR_COLOR_FULL	Supports Full Color.

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

**Errors**      A JposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

**CapRecDhigh Property R****Type**      **boolean****Remarks**      If true, then the receipt can print double high characters.

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

**Errors**      A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapRecDwide Property R

Type	<b>boolean</b>
Remarks	If true, then the receipt can print double wide characters.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapRecDwideDhigh Property R

Type	<b>boolean</b>
Remarks	If true, then the receipt can print double high / double wide characters.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapRecEmptySensor Property R

Type	<b>boolean</b>
Remarks	If true, then the receipt has an out-of-paper sensor.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapRecItalic Property R

Type	<b>boolean</b>
Remarks	If true, then the receipt can print italic characters.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

**CapRecLeft90 Property R**

<b>Type</b>	<b>boolean</b>
<b>Remarks</b>	If true, then the receipt can print in rotated 90° left mode.  This property is initialized by the <b>open</b> method.
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

**CapRecMarkFeed Property R *Added in Release 1.5***

<b>Type</b>	<b>int</b>										
<b>Remarks</b>	This parameter indicates the type of mark sensed paper handling available.  <b>CapRecMarkFeed</b> is a logical OR combination of the following values. (The values are identical to those used with the <b>markFeed</b> method.)										
	<table border="1"> <thead> <tr> <th><b>Value</b></th> <th><b>Meaning</b></th> </tr> </thead> <tbody> <tr> <td>PTR_MF_TO_TAKEUP</td> <td>Feed the Mark Sensed paper to the paper take-up position.</td> </tr> <tr> <td>PTR_MF_TO_CUTTER</td> <td>Feed the Mark Sensed paper to the autocutter cutting position.</td> </tr> <tr> <td>PTR_MF_TO_CURRENT_TOF</td> <td>Feed the Mark Sensed paper to the present paper’s top of form. (Reverse feed if required)</td> </tr> <tr> <td>PTR_MF_TO_NEXT_TOF</td> <td>Feed the Mark Sensed paper to the paper’s next top of form.</td> </tr> </tbody> </table>	<b>Value</b>	<b>Meaning</b>	PTR_MF_TO_TAKEUP	Feed the Mark Sensed paper to the paper take-up position.	PTR_MF_TO_CUTTER	Feed the Mark Sensed paper to the autocutter cutting position.	PTR_MF_TO_CURRENT_TOF	Feed the Mark Sensed paper to the present paper’s top of form. (Reverse feed if required)	PTR_MF_TO_NEXT_TOF	Feed the Mark Sensed paper to the paper’s next top of form.
<b>Value</b>	<b>Meaning</b>										
PTR_MF_TO_TAKEUP	Feed the Mark Sensed paper to the paper take-up position.										
PTR_MF_TO_CUTTER	Feed the Mark Sensed paper to the autocutter cutting position.										
PTR_MF_TO_CURRENT_TOF	Feed the Mark Sensed paper to the present paper’s top of form. (Reverse feed if required)										
PTR_MF_TO_NEXT_TOF	Feed the Mark Sensed paper to the paper’s next top of form.										
	If <b>CapRecMarkFeed</b> equals “0”, mark sensed paper handling is not supported.  This property is initialized by the <b>open</b> method.										
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.										
<b>See Also</b>	<b>markFeed</b> Method.										

### CapRecNearEndSensor Property R

Type	<b>boolean</b>
Remarks	If true, then the receipt has a low paper sensor.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapRecPapercut Property R

Type	<b>boolean</b>
Remarks	If true, then the receipt can perform paper cuts.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapRecPresent Property R

Type	<b>boolean</b>
Remarks	If true, then the receipt print station is present.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapRecRight90 Property R

Type	<b>boolean</b>
Remarks	If true, then the receipt can print in a rotated 90° right mode.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.



### CapRecRotate180 Property R

Type	<b>boolean</b>
Remarks	If true, then the receipt can print in a rotated upside down mode.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapRecStamp Property R

Type	<b>boolean</b>
Remarks	If true, then the receipt has a stamp capability.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapRecUnderline Property R

Type	<b>boolean</b>
Remarks	If true, then the receipt can underline characters.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapSlp2Color Property R

Type	<b>boolean</b>
Remarks	If true, then the slip can print dark plus an alternate color.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

**CapSlpBarCode Property R**

Type	<b>boolean</b>
Remarks	If true, then the slip has bar code printing capability.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

**CapSlpBitmap Property R**

Type	<b>boolean</b>
Remarks	If true, then the slip can print bitmaps.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

**CapSlpBold Property R**

Type	<b>boolean</b>
Remarks	If true, then the slip can print bold characters.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

**CapSlpBothSidesPrint Property R**      *Added in Release 1.5*

Type	<b>boolean</b>
Remarks	If true, then the slip station can automatically print on both sides of a check, either by flipping the check or through the use of dual print heads.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

**CapSlpCartridgeSensor Property R** *Added in Release 1.5***Type** int**Remarks** This bit mapped parameter is used to indicate the presence of Slip Cartridge monitoring sensors.

If **CapSlpPresent** is false, this property is “0”. Otherwise it is a logical OR combination of any of the following values:

<b>Value</b>	<b>Meaning</b>
PTR_CART_REMOVED	There is a function to indicate that the Cartridge has been removed.
PTR_CART_EMPTY	There is a function to indicate that the Cartridge is empty.
PTR_CART_CLEANING	There is a function to indicate that the head is being cleaned.
PTR_CART_NEAREND	There is a function to indicate that the color Cartridge is near end.

Note that the above mentioned values are arranged according to their priority level.

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

**Errors** A JposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.**See Also** **SlpCartridgeState** Property, **SlpCurrentCartridge** Property, **CartridgeNotify** Property.

**CapSlpColor Property R*****Added in Release 1.5*****Type**      **int****Remarks**      This capability indicates the availability of Slip color cartridges.

If **CapSlpPresent** is false, this property is “0”. Otherwise, this property indicates the supported color cartridges.

**CapSlpColor** is a logical OR combination of any of the following values:

<b>Value</b>	<b>Meaning</b>
PTR_COLOR_PRIMARY	Supports Primary Color (Usually Black)
PTR_COLOR_CUSTOM1	Supports 1 <sup>st</sup> Custom Color (Secondary Color, usually Red)
PTR_COLOR_CUSTOM2	Supports 2 <sup>nd</sup> Custom Color
PTR_COLOR_CUSTOM3	Supports 3 <sup>rd</sup> Custom Color
PTR_COLOR_CUSTOM4	Supports 4 <sup>th</sup> Custom Color
PTR_COLOR_CUSTOM5	Supports 5 <sup>th</sup> Custom Color
PTR_COLOR_CUSTOM6	Supports 6 <sup>th</sup> Custom Color
PTR_COLOR_CYAN	Supports Cyan Color for full color printing
PTR_COLOR_MAGENTA	Supports Magenta Color for full color printing
PTR_COLOR_YELLOW	Supports Yellow Color for full color printing
PTR_COLOR_FULL	Supports Full Color.

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

**Errors**      A JposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

**CapSlpDhigh Property R****Type**      **boolean****Remarks**      If true, then the slip can print double high characters.

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

**Errors**      A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapSlpDwide Property R

Type	<b>boolean</b>
Remarks	If true, then the slip can print double wide characters.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapSlpDwideDhigh Property R

Type	<b>boolean</b>
Remarks	If true, then the slip can print double high / double wide characters.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapSlpEmptySensor Property R

Type	<b>boolean</b>
Remarks	If true, then the slip has a “slip in” sensor.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapSlpFullslip Property R

Type	<b>boolean</b>
Remarks	If true, then the slip is a full slip station. It can print full-length forms. If false, then the slip is a “validation” type station. This usually limits the number of print lines, and disables access to the receipt and/or journal stations while the validation slip is being used.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapSlpItalic Property R

Type	<b>boolean</b>
Remarks	If true, then the slip can print italic characters.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapSlpLeft90 Property R

Type	<b>boolean</b>
Remarks	If true, then the slip can print in a rotated 90° left mode.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapSlpNearEndSensor Property R

Type	<b>boolean</b>
Remarks	If true, then the slip has a “slip near end” sensor.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapSlpPresent Property R

Type	<b>boolean</b>
Remarks	If true, then the slip print station is present.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapSlpRight90 Property R

Type	<b>boolean</b>
Remarks	If true, then the slip can print in a rotated 90° right mode.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapSlpRotate180 Property R

Type	<b>boolean</b>
Remarks	If true, then the slip can print in a rotated upside down mode.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapSlpUnderline Property R

Type	<b>boolean</b>
Remarks	If true, then the slip can underline characters.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapTransaction Property R

Type	<b>boolean</b>
Remarks	If true, then printer transactions are supported by each station.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

**CartridgeNotify Property R/W****Added in Release 1.5****Type**      **int****Remarks**      Contains the type of cartridge state notification selected by the application.The **CartridgeNotify** values are:

<b>Value</b>	<b>Meaning</b>
PTR_CN_DISABLED	The Control will not provide any cartridge state notifications to the application or set any cartridge related <i>ErrorCodeExtended</i> values. No cartridge state notification <b>StatusUpdateEvents</b> will be fired, and <b>JrnCartridgeState</b> , <b>RecCartridgeState</b> , and <b>SlpCartridgeState</b> may not be set.
PTR_CN_ENABLED	The Control will fire cartridge state notification <b>StatusUpdateEvents</b> and update <b>JrnCartridgeState</b> , <b>RecCartridgeState</b> and <b>SlpCartridgeState</b> , beginning when <b>DeviceEnabled</b> is set true. The level of functionality depends upon <b>CapJrnCartridgeSensor</b> , <b>CapRecCartridgeSensor</b> and <b>CapSlpCartridgeSensor</b> .

**CartridgeNotify** may only be set while the device is disabled, that is, while **DeviceEnabled** is false.This property is initialized to PTR\_CN\_DISABLED by the **open** method. This value provides compatibility with earlier releases.**Errors**      A JposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.Some possible values of the exception’s *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
JPOS_E_ILLEGAL	One of the following errors occurred: The device is already enabled. <b>CapJrnCartridgeSensor</b> , <b>CapRecCartridgeSensor</b> , and <b>CapSlpCartridgeSensor</b> = “0”.

**See Also**      **CapJrnCartridgeSensor** Property, **CapRecCartridgeSensor** Property,  
**CapSlpCartridgeSensor** Property, **JrnCartridgeState** Property,  
**RecCartridgeState** Property, **SlpCartridgeState** Property.



**CharacterSet Property R/W**

<b>Type</b>	<b>int</b>												
<b>Remarks</b>	<p>Holds the character set for printing characters. It has one of the following values:</p> <table border="1"> <thead> <tr> <th><b>Value</b></th> <th><b>Meaning</b></th> </tr> </thead> <tbody> <tr> <td>Range 101 - 199</td> <td>Device-specific character sets that do not match a code page or the ASCII or ANSI character sets.</td> </tr> <tr> <td>Range 400 - 990</td> <td>Code page; matches one of the standard values.</td> </tr> <tr> <td>PTR_CS_UNICODE</td> <td>The character set supports UNICODE. The value of this constant is 997.</td> </tr> <tr> <td>PTR_CS_ASCII</td> <td>The ASCII character set, supporting the ASCII characters 0x20 through 0x7F. The value of this constant is 998.</td> </tr> <tr> <td>PTR_CS_ANSI</td> <td>The ANSI character set. The value of this constant is 999.</td> </tr> </tbody> </table> <p>This property is initialized when the device is first enabled following the <b>open</b> method.</p>	<b>Value</b>	<b>Meaning</b>	Range 101 - 199	Device-specific character sets that do not match a code page or the ASCII or ANSI character sets.	Range 400 - 990	Code page; matches one of the standard values.	PTR_CS_UNICODE	The character set supports UNICODE. The value of this constant is 997.	PTR_CS_ASCII	The ASCII character set, supporting the ASCII characters 0x20 through 0x7F. The value of this constant is 998.	PTR_CS_ANSI	The ANSI character set. The value of this constant is 999.
<b>Value</b>	<b>Meaning</b>												
Range 101 - 199	Device-specific character sets that do not match a code page or the ASCII or ANSI character sets.												
Range 400 - 990	Code page; matches one of the standard values.												
PTR_CS_UNICODE	The character set supports UNICODE. The value of this constant is 997.												
PTR_CS_ASCII	The ASCII character set, supporting the ASCII characters 0x20 through 0x7F. The value of this constant is 998.												
PTR_CS_ANSI	The ANSI character set. The value of this constant is 999.												
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.												
<b>See Also</b>	<b>CharacterSetList</b> Property.												

**CharacterSetList Property R**

<b>Type</b>	<b>String</b>
<b>Remarks</b>	<p>Holds the character set numbers. It consists of ASCII numeric set numbers separated by commas.</p> <p>For example, if the string is “101,850,999”, then the device supports a device-specific character set, code page 850, and the ANSI character set.</p> <p>This property is initialized by the <b>open</b> method.</p>
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
<b>See Also</b>	<b>CharacterSet</b> Property

**CoverOpen Property R**

<b>Type</b>	<b>boolean</b>
<b>Remarks</b>	<p>If true, then the printer's cover is open.</p> <p>If <b>CapCoverSensor</b> is false, then the printer does not have a cover open sensor, and this property always returns false.</p> <p>This property is initialized and kept current while the device is enabled.</p>
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see "Exceptions" on page 15.

**ErrorLevel Property R**

<b>Type</b>	<b>int</b>								
<b>Remarks</b>	<p>Holds the severity of the error condition. It has one of the following values:</p> <table> <thead> <tr> <th><b>Value</b></th> <th><b>Meaning</b></th> </tr> </thead> <tbody> <tr> <td>PTR_EL_NONE</td> <td>No error condition is present.</td> </tr> <tr> <td>PTR_EL_RECOVERABLE</td> <td>A recoverable error has occurred. (Example: Out of paper.)</td> </tr> <tr> <td>PTR_EL_FATAL</td> <td>A non-recoverable error has occurred. (Example: Internal printer failure.)</td> </tr> </tbody> </table> <p>This property is set just before delivering an <b>ErrorEvent</b>. When the error is cleared, then the property is changed to PTR_EL_NONE.</p>	<b>Value</b>	<b>Meaning</b>	PTR_EL_NONE	No error condition is present.	PTR_EL_RECOVERABLE	A recoverable error has occurred. (Example: Out of paper.)	PTR_EL_FATAL	A non-recoverable error has occurred. (Example: Internal printer failure.)
<b>Value</b>	<b>Meaning</b>								
PTR_EL_NONE	No error condition is present.								
PTR_EL_RECOVERABLE	A recoverable error has occurred. (Example: Out of paper.)								
PTR_EL_FATAL	A non-recoverable error has occurred. (Example: Internal printer failure.)								
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see "Exceptions" on page 15.								

## ErrorStation Property R

<b>Type</b>	<b>int</b>
<b>Remarks</b>	<p>Holds the station or stations that were printing when an error was detected.</p> <p>This property will be set to one of the following values: PTR_S_JOURNAL PTR_S_RECEIPT PTR_S_SLIP PTR_S_JOURNAL_RECEIPT PTR_S_JOURNAL_SLIP PTR_S_RECEIPT_SLIP PTR_TWO_RECEIPT_JOURNAL PTR_TWO_SLIP_JOURNAL PTR_TWO_SLIP_RECEIPT</p> <p>This property is only valid if the ErrorLevel is not equal to PTR_EL_NONE. It is set just before delivering an <b>ErrorEvent</b>.</p>
<b>Errors</b>	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>

## ErrorString Property R

<b>Type</b>	<b>String</b>
<b>Remarks</b>	<p>Holds a vendor-supplied description of the current error.</p> <p>This property is set just before delivering an <b>ErrorEvent</b>. If no description is available, the property is set to an empty string. When the error is cleared, then the property is changed to an empty string.</p>
<b>Errors</b>	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>

## FlagWhenIdle Property R/W

<b>Type</b>	<b>boolean</b>
<b>Remarks</b>	<p>If true, a <b>StatusUpdateEvent</b> will be enqueued when the device is in the idle state.</p> <p>This property is automatically reset to false when the status event is delivered.</p> <p>The main use of idle status event that is controlled by this property is to give the application control when all outstanding asynchronous outputs have been processed. The event will be enqueued if the outputs were completed successfully or if they were cleared by the <b>clearOutput</b> method or by an <b>ErrorEvent</b> handler.</p> <p>If the <b>State</b> is already set to JPOS_S_IDLE when this property is set to true, then a <b>StatusUpdateEvent</b> is enqueued immediately. The application can therefore depend upon the event, with no race condition between the starting of its last asynchronous output and the setting of this flag.</p> <p>This property is initialized to false by the <b>open</b> method.</p>
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

## FontTypefaceList Property R

<b>Type</b>	<b>String</b>
<b>Remarks</b>	<p>Holds the fonts and/or typefaces that are supported by the printer. The string consists of font or typeface names separated by commas. The application selects a font or typeface for a printer station by using the font typeface selection escape sequence (ESC #fT). The “#” character is replaced by the number of the font or typeface within the list: 1, 2, and so on.</p> <p>In Japan, this property will frequently include the fonts “Mincho” and “Gothic.” Other fonts or typefaces may be commonly supported in other countries.</p> <p>An empty string indicates that only the default typeface is supported.</p> <p>This property is initialized by the <b>open</b> method.</p>
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
<b>See Also</b>	“Data Characters and Escape Sequences”

**JrnCartridgeState Property R****Added in Release 1.5****Type**      **int****Remarks**      This property contains the status of the currently selected Journal cartridge (ink, ribbon or toner).

It contains one of the following values:

<b>Value</b>	<b>Meaning</b>
PTR_CART_UNKNOWN	Cannot determine the cartridge state, for one of the following reasons: <b>CapJrnCartridgeSensor</b> = "0". Device does not support cartridge state reporting. <b>CartridgeNotify</b> = PTR_CN_DISABLED. Cartridge state notifications are disabled. <b>DeviceEnabled</b> = FALSE. Cartridge state monitoring does not occur until the device is enabled.
PTR_CART_REMOVED	The cartridge selected by <b>JrnCurrentCartridge</b> has been removed.
PTR_CART_EMPTY	The cartridge selected by <b>JrnCurrentCartridge</b> is empty.
PTR_CART_CLEANING	The head selected by <b>JrnCurrentCartridge</b> is being cleaned.
PTR_CART_NEAREND	The cartridge selected by <b>JrnCurrentCartridge</b> is near end.
PTR_CART_OK	The cartridge selected by <b>JrnCurrentCartridge</b> is in normal condition.

Note that the above mentioned values are arranged according to their priority level.

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

**Errors**      A JposException may be thrown when this property is accessed. For further information, see "Errors" on page 15.**See Also**      **JrnCurrentCartridge** Property, **CapJrnCartridgeSensor** Property, **CartridgeNotify** Property.

**JrnCurrentCartridge Property R/W**     *Added in Release 1.5*

<b>Type</b>	<b>int</b>				
<b>Remarks</b>	<p>This property specifies the currently selected Journal cartridge.</p> <p>This property is initialized when the device is first enabled following the <b>open</b> method call.</p> <p>This value is guaranteed to be one of the color cartridges specified by the <b>CapJrnColor</b> property. (PTR_COLOR_FULL can not be set.)</p> <p>Setting <b>JrnCurrentCartridge</b> may also update <b>JrnCartridgeState</b>.</p>				
<b>Errors</b>	<p>A JposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.</p> <p>Some possible values of the exception’s <i>ErrorCode</i> property are:</p> <table border="1"> <thead> <tr> <th><b>Value</b></th> <th><b>Meaning</b></th> </tr> </thead> <tbody> <tr> <td>JPOS_E_ILLEGAL</td> <td>An invalid property value was specified.</td> </tr> </tbody> </table>	<b>Value</b>	<b>Meaning</b>	JPOS_E_ILLEGAL	An invalid property value was specified.
<b>Value</b>	<b>Meaning</b>				
JPOS_E_ILLEGAL	An invalid property value was specified.				
<b>See Also</b>	<b>JrnCartridgeState</b> Property.				

**JrnEmpty Property R**

<b>Type</b>	<b>boolean</b>
<b>Remarks</b>	<p>If true, the journal is out of paper. If false, journal paper is present.</p> <p>If <b>CapJrnEmptySensor</b> is false, then the value of this property is always false.</p> <p>This property is initialized and kept current while the device is enabled.</p>
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
<b>See Also</b>	<b>JrnNearEnd</b> Property

### JrnLetterQuality Property R/W

Type	<b>boolean</b>
Remarks	<p>If true, prints in high quality mode. If false, prints in high speed mode.</p> <p>This property advises the Device Service that either high quality or high speed printing is desired. For example, printers with bi-directional print capability may be placed in unidirectional mode for high quality, so that column alignment is more precise.</p> <p>Setting this property may also update <b>JrnLineWidth</b>, <b>JrnLineHeight</b>, and <b>JrnLineSpacing</b> if <b>MapMode</b> is PTR_MM_DOTS. (See the footnote at <b>MapMode</b>.)</p> <p>This property is initialized to false when the device is first enabled following the <b>open</b> method.</p>
Errors	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>

### JrnLineChars Property R/W

Type	<b>int</b>
Remarks	<p>Holds the number of characters that may be printed on a journal line.</p> <p>If changed to a line character width that can be supported, then the width is set to the specified value. If the exact width cannot be supported, then subsequent lines will be printed with a character size that most closely supports the specified characters per line. (For example, if set to 36 and the printer can print either 30 or 40 characters per line, then the Device Service should select the 40 characters per line size and print only up to 36 characters per line.)</p> <p>If the character width cannot be supported, then an exception is thrown. (For example, if set to 42 and the printer can print either 30 or 40 characters per line, then the Device Service cannot support the request.)</p> <p>Setting this property may also update <b>JrnLineWidth</b>, <b>JrnLineHeight</b>, and <b>JrnLineSpacing</b>, since the character pitch or font may be changed.</p> <p>This property is initialized to the printer’s default line character width when the device is first enabled following the <b>open</b> method.</p>
Errors	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>
See Also	<b>JrnLineCharsList</b> Property

## JrnLineCharsList Property R

<b>Type</b>	<b>String</b>
<b>Remarks</b>	<p>Holds the line character widths supported by the journal station. The string consists of ASCII numeric set numbers separated by commas.</p> <p>For example, if the string is “32,36,40”, then the station supports line widths of 32, 36, and 40 characters.</p> <p>This property is initialized by the <b>open</b> method.</p>
<b>Errors</b>	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>
<b>See Also</b>	<b>JrnLineChars</b> Property

## JrnLineHeight Property R/W

<b>Type</b>	<b>int</b>
<b>Remarks</b>	<p>Holds the journal print line height. Expressed in the unit of measure given by <b>MapMode</b>.</p> <p>If changed to a height that can be supported with the current character width, then the line height is set to this value. If the exact height cannot be supported, then the height is set to the closest supported value.</p> <p>When <b>JrnLineChars</b> is changed, this property is updated to the default line height for the selected width.</p> <p>This property is initialized to the printer’s default line height when the device is first enabled following the <b>open</b> method.</p>
<b>Errors</b>	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>



## JrnLineSpacing Property R/W

Type	int
Remarks	<p>Holds the spacing of each single-high print line, including both the printed line height plus the whitespace between each pair of lines. Depending upon the printer and the current line spacing, a multi-high print line might exceed this value. Line spacing is expressed in the unit of measure given by <b>MapMode</b>.</p> <p>If changed to a spacing that can be supported by the printer, then the line spacing is set to this value. If the spacing cannot be supported, then the spacing is set to the closest supported value.</p> <p>When <b>JrnLineChars</b> or <b>JrnLineHeight</b> is changed, this property is updated to the default line spacing for the selected width or height.</p> <p>This property is initialized to the printer's default line spacing when the device is first enabled following the <b>open</b> method.</p>
Errors	<p>A JposException may be thrown when this property is accessed. For further information, see "Exceptions" on page 15.</p>

## JrnLineWidth Property R

Type	int
Remarks	<p>Holds the width of a line of <b>JrnLineChars</b> characters. Expressed in the unit of measure given by <b>MapMode</b>.</p> <p>Setting <b>JrnLineChars</b> may also update this property.</p> <p>This property is initialized to the printer's default line width when the device is first enabled following the <b>open</b> method.</p>
Errors	<p>A JposException may be thrown when this property is accessed. For further information, see "Exceptions" on page 15.</p>

**JrnNearEnd Property R**

<b>Type</b>	<b>boolean</b>
<b>Remarks</b>	If true, the journal paper is low. If false, journal paper is not low.  If <b>CapJrnNearEndSensor</b> is false, then this property is always false.  This property is initialized and kept current while the device is enabled.
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
<b>See Also</b>	<b>JrnEmpty</b> Property

**MapMode Property R/W**

Type        **int**

Remarks    Holds the mapping mode of the printer. The mapping mode defines the unit of measure used for other properties, such as line heights and line spacings. It has one of the following values:

<b>Value</b>	<b>Meaning</b>
PTR_MM_DOTS	The printer's dot width. This width may be different for each printer station. <sup>1</sup>
PTR_MM_TWIPS	1/1440 of an inch.
PTR_MM_ENGLISH	0.001 inch.
PTR_MM_METRIC	0.01 millimeter.

Setting this property may also change **JrnLineHeight**, **JrnLineSpacing**, **JrnLineWidth**, **RecLineHeight**, **RecLineSpacing**, **RecLineWidth**, **SlpLineHeight**, **SlpLineSpacing**, and **SlpLineWidth**.

This property is initialized to PTR\_MM\_DOTS when the device is first enabled following the **open** method.

Errors        A JposException may be thrown when this property is accessed. For further information, see "Exceptions" on page 15.

---

<sup>1</sup>. From the JavaPOS POS Printer perspective, the exact definition of a "dot" is not significant. It is a Printer/device service unit used to express various metrics. For example, some printers define a "half-dot" that is used in high-density graphics printing, and perhaps in text printing. A POS Printer Service may handle this case in one of these ways:

- (a) Consistently define a "dot" as the printer's smallest physical size, that is, a half-dot.
- (b) If the device service changes bitmap graphics printing density based on the **XxxLetterQuality** setting, then alter the size of a dot to match the bitmap density (that is, a physical printer dot when false and a half-dot when true). Note that this choice should not be used if the printer's text metrics are based on half-dot sizes, since accurate values for the metrics may not then be possible.

## RecBarCodeRotationList Property R

**Type**      **String**

**Remarks**      Holds the directions in which a receipt barcode may be rotated. The string consists of rotation strings separated by commas. An empty string indicates that bar code printing is not supported. The legal rotation strings are:

<b>Value</b>	<b>Meaning</b>
0	Bar code may be printed in the normal orientation.
R90	Bar code may be rotated 90° to the right.
L90	Bar code may be rotated 90° to the left.
180	Bar code may be rotated 180° - upside down.

For example, if the string is “0,180”, then the printer can print normal bar codes and upside down bar codes.

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

**Errors**      A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

**See Also**      **RotateSpecial** Property, **printBarCode** Method

**RecCartridgeState Property R** *Added in Release 1.5***Type** int**Remarks** This property contains the status of the currently selected Receipt cartridge (ink, ribbon or toner).

It contains one of the following values:

<b>Value</b>	<b>Meaning</b>
PTR_CART_UNKNOWN	Cannot determine the cartridge state, for one of the following reasons: <b>CapRecCartridgeSensor</b> = "0". Device does not support cartridge state reporting. <b>CartridgeNotify</b> = PTR_CN_DISABLED. Cartridge state notifications are disabled. <b>DeviceEnabled</b> = FALSE. Cartridge state monitoring does not occur until the device is enabled.
PTR_CART_REMOVED	The cartridge selected by <b>RecCurrentCartridge</b> has been removed.
PTR_CART_EMPTY	The cartridge selected by <b>RecCurrentCartridge</b> is empty.
PTR_CART_CLEANING	The head selected by <b>RecCurrentCartridge</b> is being cleaned.
PTR_CART_NEAREND	The cartridge selected by <b>RecCurrentCartridge</b> is near end.
PTR_CART_OK	The cartridge selected by <b>RecCurrentCartridge</b> is in normal condition.

Note that the above mentioned values are arranged according to their priority level.

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

**Errors** A JposException may be thrown when this property is accessed. For further information, see "Errors" on page 15.**See Also** **RecCurrentCartridge** Property, **CapRecCartridgeSensor** Property, **CartridgeNotify** Property.

**RecCurrentCartridge Property R**      *Added in Release 1.5*

<b>Type</b>	<b>int</b>				
<b>Remarks</b>	<p>This property specifies the currently selected Receipt cartridge.</p> <p>This property is initialized when the device is first enabled following the <b>open</b> method call.</p> <p>This value is guaranteed to be one of the color cartridges specified by the <b>CapRecColor</b> property. (PTR_COLOR_FULL can not be set.)</p> <p>Setting <b>RecCurrentCartridge</b> may also update <b>RecCartridgeState</b>.</p>				
<b>Errors</b>	<p>A JposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.</p> <p>Some possible values of the exception’s <i>ErrorCode</i> property are:</p> <table> <thead> <tr> <th><b>Value</b></th> <th><b>Meaning</b></th> </tr> </thead> <tbody> <tr> <td>JPOS_E_ILLEGAL</td> <td>An invalid property value was specified.</td> </tr> </tbody> </table>	<b>Value</b>	<b>Meaning</b>	JPOS_E_ILLEGAL	An invalid property value was specified.
<b>Value</b>	<b>Meaning</b>				
JPOS_E_ILLEGAL	An invalid property value was specified.				
<b>See Also</b>	<b>RecCartridgeState</b> Property.				

**RecEmpty Property R**

<b>Type</b>	<b>boolean</b>
<b>Remarks</b>	<p>If true, the receipt is out of paper. If false, receipt paper is present.</p> <p>If <b>CapRecEmptySensor</b> is false, then this property is always false.</p> <p>This property is initialized and kept current while the device is enabled.</p>
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
<b>See Also</b>	<b>RecNearEnd</b> Property

## RecLetterQuality Property R/W

Type	<b>boolean</b>
Remarks	<p>If true, prints in high quality mode. If false, prints in high speed mode.</p> <p>This property advises the Device Service that either high quality or high speed printing is desired. For example:</p> <ul style="list-style-type: none"><li>• Printers with bi-directional print capability may be placed in unidirectional mode for high quality, so that column alignment is more precise.</li><li>• Bitmaps may be printed in a high-density graphics mode for high-quality, and in a low-density mode for high speed.</li></ul> <p>Setting this property may also update <b>RecLineWidth</b>, <b>RecLineHeight</b>, and <b>RecLineSpacing</b> if <b>MapMode</b> is PTR_MM_DOTS. (See the footnote at <b>MapMode</b>.)</p> <p>This property is initialized to false when the device is first enabled following the <b>open</b> method.</p>
Errors	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>

## RecLineChars Property R/W

Type	<b>int</b>
Remarks	<p>Holds the number of characters that may be printed on a receipt line.</p> <p>If changed to a line character width that can be supported, then the width is set to the specified value. If the exact width cannot be supported and the value is less than the maximum value allowed for the printer, then subsequent lines will be printed with a character size that most closely supports the specified characters per line. (For example, if set to 36 and the printer can print either 30 or 40 characters per line, then the Device Service should select the 40 characters per line size and print only up to 36 characters per line.)</p> <p>If the character width cannot be supported and the value is greater than the maximum value allowed for the printer, then an exception is thrown. (For example, if set to 42 and the printer can print either 30 or 40 characters per line, then the Device Service cannot support the request.)</p> <p>Setting this property may also update <b>RecLineWidth</b>, <b>RecLineHeight</b>, and <b>RecLineSpacing</b>, since the character pitch or font may be changed.</p> <p>This property is initialized to the printer’s default line character width when the device is first enabled following the <b>open</b> method.</p>
Errors	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>
See Also	<b>RecLineCharsList</b> Property

## RecLineCharsList Property R

Type	<b>String</b>
Remarks	<p>Holds the line character widths supported by the receipt station. The string consists of ASCII numeric set numbers, separated by commas.</p> <p>For example, if the string is “32,36,40”, then the station supports line widths of 32, 36, and 40 characters.</p> <p>This property is initialized by the <b>open</b> method.</p>
Errors	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>
See Also	<b>RecLineChars</b> Property

## RecLineHeight Property R/W

Type	<b>int</b>
Remarks	<p>Holds the receipt print line height, expressed in the unit of measure given by <b>MapMode</b>.</p> <p>If changed to a height that can be supported with the current character width, then the line height is set to this value. If the exact height cannot be supported, then the height is set to the closest supported value.</p> <p>When <b>RecLineChars</b> is changed, this property is updated to the default line height for the selected width.</p> <p>This property is initialized to the printer’s default line height when the device is first enabled following the <b>open</b> method.</p>
Errors	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>
See Also	<b>RecLineChars</b> Property



## RecLineSpacing Property R/W

<b>Type</b>	<b>int</b>
<b>Remarks</b>	<p>Holds the spacing of each single-high print line, including both the printed line height plus the whitespace between each pair of lines. Depending upon the printer and the current line spacing, a multi-high print line might exceed this value. Line spacing is expressed in the unit of measure given by <b>MapMode</b>.</p> <p>If changed to a spacing that can be supported by the printer, then the line spacing is set to this value. If the spacing cannot be supported, then the spacing is set to the closest supported value.</p> <p>When <b>RecLineChars</b> or <b>RecLineHeight</b> are changed, this property is updated to the default line spacing for the selected width or height.</p> <p>This property is initialized to the printer's default line spacing when the device is first enabled following the <b>open</b> method.</p>
<b>Errors</b>	<p>A JposException may be thrown when this property is accessed. For further information, see "Exceptions" on page 15.</p>

## RecLinesToPaperCut Property R

<b>Type</b>	<b>int</b>
<b>Remarks</b>	<p>Holds the number of lines that must be advanced before the receipt paper is cut.</p> <p>If <b>CapRecPapercut</b> is true, then this is the line count before reaching the paper cut mechanism. Otherwise, this is the line count before the manual tear-off bar.</p> <p>Changing the properties <b>RecLineChars</b>, <b>RecLineHeight</b>, and <b>RecLineSpacing</b> may cause this property to change.</p> <p>This property is initialized when the device is first enabled following the <b>open</b> method.</p>
<b>Errors</b>	<p>A JposException may be thrown when this property is accessed. For further information, see "Exceptions" on page 15.</p>

## RecLineWidth Property R

Type	<b>int</b>
Remarks	<p>Holds the width of a line of <b>RecLineChars</b> characters, expressed in the unit of measure given by <b>MapMode</b>.</p> <p>Setting <b>RecLineChars</b> may also update this property.</p> <p>This property is initialized to the printer's default line width when the device is first enabled following the <b>open</b> method.</p>
Errors	<p>A <b>JposException</b> may be thrown when this property is accessed. For further information, see "Exceptions" on page 15.</p>

## RecNearEnd Property R

Type	<b>boolean</b>
Remarks	<p>If true, the receipt paper is low. If false, receipt paper is not low.</p> <p>If <b>CapRecNearEndSensor</b> is false, then this property is always false.</p> <p>This property is initialized and kept current while the device is enabled.</p>
Errors	<p>A <b>JposException</b> may be thrown when this property is accessed. For further information, see "Exceptions" on page 15.</p>
See Also	<b>RecEmpty</b> Property

## RecSidewaysMaxChars Property R

Type	<b>int</b>
Remarks	<p>Holds the maximum number of characters that may be printed on each line in sideways mode.</p> <p>If <b>CapRecLeft90</b> and <b>CapRecRight90</b> are both false, then this property is zero.</p> <p>Changing the properties <b>RecLineHeight</b>, <b>RecLineSpacing</b>, and <b>RecLineChars</b> may cause this property to change.</p> <p>This property is initialized when the device is first enabled following the <b>open</b> method.</p>
Errors	<p>A <b>JposException</b> may be thrown when this property is accessed. For further information, see "Exceptions" on page 15.</p>
See Also	<b>RecSidewaysMaxLines</b> Property

**RecSidewaysMaxLines Property R**

<b>Type</b>	<b>int</b>
<b>Remarks</b>	<p>Holds the maximum number of lines that may be printed in sideways mode.</p> <p>If <b>CapRecLeft90</b> and <b>CapRecRight90</b> are both false, then this property is zero.</p> <p>Changing the properties <b>RecLineHeight</b>, <b>RecLineSpacing</b>, and <b>RecLineChars</b> may cause this property to change.</p> <p>This property is initialized when the device is first enabled following the <b>open</b> method.</p>
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
<b>See Also</b>	<b>RecSidewaysMaxChars</b> Property

**RotateSpecial Property R/W**

<b>Type</b>	<b>int</b>										
<b>Remarks</b>	<p>Holds the rotation orientation for bar codes. It has one of the following values:</p> <table border="1"> <thead> <tr> <th><b>Value</b></th> <th><b>Meaning</b></th> </tr> </thead> <tbody> <tr> <td>PTR_RP_NORMAL</td> <td>Print subsequent bar codes in normal orientation.</td> </tr> <tr> <td>PTR_RP_RIGHT90</td> <td>Rotate printing 90° to the right (clockwise)</td> </tr> <tr> <td>PTR_RP_LEFT90</td> <td>Rotate printing 90° to the left (counter-clockwise)</td> </tr> <tr> <td>PTR_RP_ROTATE180</td> <td>Rotate printing 180°,that is, print upside-down</td> </tr> </tbody> </table> <p>This property is initialized to PTR_RP_NORMAL by the <b>open</b> method.</p>	<b>Value</b>	<b>Meaning</b>	PTR_RP_NORMAL	Print subsequent bar codes in normal orientation.	PTR_RP_RIGHT90	Rotate printing 90° to the right (clockwise)	PTR_RP_LEFT90	Rotate printing 90° to the left (counter-clockwise)	PTR_RP_ROTATE180	Rotate printing 180°,that is, print upside-down
<b>Value</b>	<b>Meaning</b>										
PTR_RP_NORMAL	Print subsequent bar codes in normal orientation.										
PTR_RP_RIGHT90	Rotate printing 90° to the right (clockwise)										
PTR_RP_LEFT90	Rotate printing 90° to the left (counter-clockwise)										
PTR_RP_ROTATE180	Rotate printing 180°,that is, print upside-down										
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.										
<b>See Also</b>	<b>printBarCode</b> Method										

## SlpBarcodeRotationList Property R

**Type**      **String**

**Remarks**      Holds the directions in which a slip barcode may be rotated. The string consists of rotation strings separated by commas. An empty string indicates that bar code printing is not supported. The legal rotation strings are:

<b>Value</b>	<b>Meaning</b>
0	Bar code may be printed in the normal orientation.
R90	Bar code may be rotated 90° to the right.
L90	Bar code may be rotated 90° to the left.
180	Bar code may be rotated 180° - upside down.

For example, if the string is “0,180”, then the printer can print normal bar codes and upside down bar codes.

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

**Errors**      A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

**See Also**      **RotateSpecial** Property, **printBarcode** Method

**SlpCartridgeState Property R****Added in Release 1.5****Type**      **int****Remarks**      This property contains the status of the currently selected Slp cartridge (ink, ribbon or toner).

It contains one of the following values:

<b>Value</b>	<b>Meaning</b>
PTR_CART_UNKNOWN	Cannot determine the cartridge state, for one of the following reasons: <b>CapSlpCartridgeSensor</b> = "0". Device does not support cartridge state reporting. <b>CartridgeNotify</b> = PTR_CN_DISABLED. Cartridge state notifications are disabled. <b>DeviceEnabled</b> = FALSE. Cartridge state monitoring does not occur until the device is enabled.
PTR_CART_REMOVED	The cartridge selected by <b>SlpCurrentCartridge</b> has been removed.
PTR_CART_EMPTY	The cartridge selected by <b>SlpCurrentCartridge</b> is empty.
PTR_CART_CLEANING	The head selected by <b>SlpCurrentCartridge</b> is being cleaned.
PTR_CART_NEAREND	The cartridge selected by <b>SlpCurrentCartridge</b> is near end.
PTR_CART_OK	The cartridge selected by <b>SlpCurrentCartridge</b> is in normal condition.

Note that the above mentioned values are arranged according to their priority level.

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

**Errors**      A JposException may be thrown when this property is accessed. For further information, see "Errors" on page 15.**See Also**      **SlpCurrentCartridge** Property, **CapSlpCartridgeSensor** Property, **CartridgeNotify** Property.

**SlpCurrentCartridge Property R/W** *Added in Release 1.5*

<b>Type</b>	<b>int</b>				
<b>Remarks</b>	<p>This property specifies the currently selected slip cartridge.</p> <p>This property is initialized when the device is first enabled following the <b>open</b> method call.</p> <p>This value is guaranteed to be one of the color cartridges specified by the <b>CapSlpColor</b> property. (PTR_COLOR_FULL can not be set.)</p> <p>Setting <b>SlpCurrentCartridge</b> may also update <b>SlpCartridgeState</b>.</p>				
<b>Errors</b>	<p>A <i>JposException</i> may be thrown when this property is accessed. For further information, see “Errors” on page 15.</p> <p>Some possible values of the exception’s <i>ErrorCode</i> property are:</p> <table border="1"> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>JPOS_E_ILLEGAL</td> <td>An invalid property value was specified.</td> </tr> </tbody> </table>	Value	Meaning	JPOS_E_ILLEGAL	An invalid property value was specified.
Value	Meaning				
JPOS_E_ILLEGAL	An invalid property value was specified.				
<b>See Also</b>	<b>RecCartridgeState</b> Property.				

**SlpEmpty Property R**

<b>Type</b>	<b>boolean</b>
<b>Remarks</b>	<p>If true, a slip form is not present. If false, a slip form is present.</p> <p>If <b>CapSlpEmptySensor</b> is false, then this property is always false.</p> <p>This property is initialized and kept current while the device is enabled.</p> <hr/> <p><b>Note</b></p> <p>The “slip empty” sensor should be used primarily to determine whether a form has been inserted before printing, and can be monitored to determine whether a form is still in place. This sensor is usually placed one or more print lines above the slip print head.</p> <p>However, the “slip near end” sensor (when present) should be used to determine when nearing the end of the slip. This sensor is usually placed one or more print lines below the slip print head.</p>
<b>Errors</b>	A <i>JposException</i> may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
<b>See Also</b>	<b>SlpNearEnd</b> Property

**SlpLetterQuality Property R/W**

<b>Type</b>	<b>boolean</b>
<b>Remarks</b>	<p>If true, prints in high quality mode. If false, prints in high speed mode.</p> <p>This property advises that either high quality or high speed printing is desired.</p> <p>For example:</p> <ul style="list-style-type: none"><li>• Printers with bi-directional print capability may be placed in unidirectional mode for high quality, so that column alignment is more precise.</li><li>• Bitmaps may be printed in a high-density graphics mode for high-quality, and in a low-density mode for high speed.</li></ul> <p>Setting this property may also update <b>SlpLineWidth</b>, <b>SlpLineHeight</b>, and <b>SlpLineSpacing</b> if <b>MapMode</b> is PTR_MM_DOTS. (See the footnote at <b>MapMode</b>.)</p> <p>This property is initialized to false when the device is first enabled following the <b>open</b> method.</p>
<b>Errors</b>	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>

## SlpLineChars Property R/W

<b>Type</b>	<b>int</b>
<b>Remarks</b>	<p>Holds the number of characters that may be printed on a slip line.</p> <p>If changed to a line character width that can be supported, then the width is set to the specified value. If the exact width cannot be supported, then subsequent lines will be printed with a character size that most closely supports the specified characters per line. (The Device Service should print the requested characters in the column positions closest to the side of the slip table at which the slip is aligned. (For example, if the operator inserts the slip with the right edge against the table side and if <b>SlpLineChars</b> is set to 36 and the printer prints 60 characters per line, then the Device Service should add 24 spaces at the left margin and print the characters in columns 25 through 60.)</p> <p>If the character width cannot be supported, then an exception is thrown. (For example, if set to 65 and the printer can only print 60 characters per line, then the Device Service cannot support the request.)</p> <p>Setting this property may also update <b>SlpLineWidth</b>, <b>SlpLineHeight</b>, and <b>SlpLineSpacing</b>, since the character pitch or font may be changed.</p> <p>This property is initialized to the printer's default line character width when the device is first enabled following the <b>open</b> method.</p>
<b>Errors</b>	<p>A JposException may be thrown when this property is accessed. For further information, see "Exceptions" on page 15.</p>
<b>See Also</b>	<b>SlpLineCharsList</b> Property

## SlpLineCharsList Property R

<b>Type</b>	<b>String</b>
<b>Remarks</b>	<p>Holds the line character widths supported by the slip station. The string consists of ASCII numeric set numbers, separated by commas.</p> <p>For example, if the string is "32,36,40", then the station supports line widths of 32, 36, and 40 characters.</p> <p>This property is initialized by the <b>open</b> method.</p>
<b>Errors</b>	<p>A JposException may be thrown when this property is accessed. For further information, see "Exceptions" on page 15.</p>
<b>See Also</b>	<b>SlpLineChars</b> Property



## SlpLineHeight Property R/W

Type	<b>int</b>
Remarks	<p>Holds the slip print-line height, expressed in the unit of measure given by <b>MapMode</b>.</p> <p>If changed to a height that can be supported with the current character width, then the line height is set to this value. If the exact height cannot be supported, then the height is set to the closest supported value.</p> <p>When <b>SlpLineChars</b> is changed, this property is updated to the default line height for the selected width.</p> <p>This property is initialized to the printer's default line height when the device is first enabled following the <b>open</b> method.</p>
Errors	<p>A JposException may be thrown when this property is accessed. For further information, see "Exceptions" on page 15.</p>
See Also	<b>SlpLineChars</b> Property

## SlpLinesNearEndToEnd Property R

Type	<b>int</b>
Remarks	<p>Holds the number of lines that may be printed after the "slip near end" sensor is true but before the printer reaches the end of the slip.</p> <p>This property may be used to optimize the use of the slip, so that the maximum number of lines may be printed.</p> <p>Changing the <b>SlpLineHeight</b>, <b>SlpLineSpacing</b>, or <b>SlpLineChars</b> properties may cause this property to change.</p> <p>This property is initialized when the device is first enabled following the <b>open</b> method.</p>
Errors	<p>A JposException may be thrown when this property is accessed. For further information, see "Exceptions" on page 15.</p>
See Also	<b>SlpEmpty</b> Property, <b>SlpNearEnd</b> Property

## SlpLineSpacing Property R/W

Type	<b>int</b>
Remarks	<p>Holds the spacing of each single-high print line, including both the printed line height plus the whitespace between each pair of lines. Depending upon the printer and the current line spacing, a multi-high print line might exceed this value. Line spacing is expressed in the unit of measure given by <b>MapMode</b>.</p> <p>If changed to a spacing that can be supported by the printer, then the line spacing is set to this value. If the spacing cannot be supported, then the spacing is set to the closest supported value.</p> <p>When <b>SlpLineChars</b> or <b>SlpLineHeight</b> are changed, this property is updated to the default line spacing for the selected width or height.</p> <p>The value of this property is initialized to the printer's default line spacing when the device is first enabled following the <b>open</b> method.</p>
Errors	<p>A JposException may be thrown when this property is accessed. For further information, see "Exceptions" on page 15.</p>

## SlpLineWidth Property R

Type	<b>int</b>
Remarks	<p>Holds the width of a line of <b>SlpLineChars</b> characters, expressed in the unit of measure given by <b>MapMode</b>.</p> <p>Setting <b>SlpLineChars</b> may also update this property.</p> <p>This property is initialized to the printer's default line width when the device is first enabled following the <b>open</b> method.</p>
Errors	<p>A JposException may be thrown when this property is accessed. For further information, see "Exceptions" on page 15.</p>

## SlpMaxLines Property R

<b>Type</b>	<b>int</b>
<b>Remarks</b>	<p>Holds the maximum number of lines that can be printed on a form.</p> <p>When <b>CapSlpFullslip</b> is true, then this property will be zero, indicating an unlimited maximum slip length. When <b>CapSlpFullslip</b> is false, then this value will be non-zero.</p> <p>Changing the <b>SlpLineHeight</b>, <b>SlpLineSpacing</b>, or <b>SlpLineChars</b> properties may cause this property to change.</p> <p>This property is initialized when the device is first enabled following the <b>open</b> method.</p>
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

## SlpNearEnd Property R

<b>Type</b>	<b>boolean</b>
<b>Remarks</b>	<p>If true, the slip form is near its end. If false, the slip form is not near its end.</p> <p>The “near end” sensor is also sometimes called the “trailing edge” sensor, referring to the bottom edge of the slip.</p> <p>If <b>CapSlpNearEndSensor</b> is false, then this property is always false.</p> <p>This property is initialized and kept current while the device is enabled.</p> <hr/> <p><b>Note</b></p> <p>The “slip empty” sensor should be used primarily to determine whether a form has been inserted before printing, and can be monitored to determine whether a form is still in place. This sensor is usually placed one or more print lines above the slip print head.</p> <p>However, the “slip near end” sensor (when present) should be used to determine when nearing the end of the slip. This sensor is usually placed one or more print lines below the slip print head.</p> <hr/>
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
<b>See Also</b>	<b>SlpEmpty</b> Property, <b>SlpLinesNearEndToEnd</b> Property

**SlpPrintSide Property R** *Added in Release 1.5***Type** `int`**Remarks** This property holds the current side of the slip document on which printing will occur.

If the Slip is not selected, the value of the property is PTR\_PS\_UNKNOWN.

If the printer has both side print capability, **CapSlpBothSidesPrint** is true, then when a slip is inserted, the value stored here will be either PTR\_PS\_SIDE1 or PTR\_PS\_SIDE2. This property value may be changed when the **changePrintSide** method is executed.

If a printer does not have both side print capability, **CapSlpBothSidesPrint** is false, then when a slip is inserted, the property is always set to PTR\_PS\_SIDE1.

If a printer has both side print capability, the value of **SlpPrintSide** property is PTR\_PS\_SIDE1 after **beginInsertion/endInsertion** methods are executed. However, after **beginInsertion/endInsertion** methods for MICR processing are executed, the value of **SlpPrintSide** property is not limited to PTR\_PS\_SIDE1. In this case, **SlpPrintSide** property indicates the side of the validation printing.

It contains one of the following values:

<b>Value</b>	<b>Meaning</b>
PTR_PS_UNKNOWN	Slip is not inserted.
PTR_PS_SIDE1	Default Print side. (After slip paper insertion, printer can print this side immediately.)
PTR_PS_SIDE2	The other side of the document to print on. (Reverse side of default.)

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

**Errors** A JposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.**See Also** **CapSlpBothSidesPrint** Property, **changePrintSide** Method.

## SlpSidewaysMaxChars Property R

Type	int
Remarks	<p>Holds the maximum number of characters that may be printed on each line in sideways mode.</p> <p>If <b>CapSlpLeft90</b> and <b>CapSlpRight90</b> are both false, then this property is zero.</p> <p>Changing the properties <b>SlpLineHeight</b>, <b>SlpLineSpacing</b>, and <b>SlpLineChars</b> may cause this property to change.</p> <p>This property is initialized when the device is first enabled following the <b>open</b> method.</p>
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>SlpSidewaysMaxLines</b> Property

## SlpSidewaysMaxLines Property R

Type	int
Remarks	<p>Holds the maximum number of lines that may be printed in sideways mode.</p> <p>If <b>CapSlpLeft90</b> and <b>CapSlpRight90</b> are both false, then this property is zero.</p> <p>Changing the properties <b>SlpLineHeight</b>, <b>SlpLineSpacing</b>, and <b>SlpLineChars</b> may cause this property to change.</p> <p>This property is initialized when the device is first enabled following the <b>open</b> method.</p>
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>SlpSidewaysMaxChars</b> Property

## Methods

### beginInsertion Method

**Syntax**      **void beginInsertion (int *timeout*) throws JposException;**

Parameter	Description
-----------	-------------

<i>timeout</i>	The number of milliseconds before failing the method
----------------	--

If zero, the method initiates the begin insertion mode, then returns the appropriate status immediately. If JPOS\_FOREVER (-1), the method initiates the begin insertion mode, then waits as long as needed until either the form is inserted or an error occurs.

**Remarks**      Initiates slip processing.

When called, the slip station is made ready to receive a form by opening the form's handling "jaws" or activating a form insertion mode. This method is paired with the **endInsertion** method for controlling form insertion.

If the printer device cannot be placed into insertion mode, a JposException is thrown. Otherwise, form insertion is monitored until either:

- The form is successfully inserted.
- The form is not inserted before *timeout* milliseconds have elapsed, or an error is reported by the printer device. In this case, a JposException is thrown with an *ErrorCode* of JPOS\_E\_TIMEOUT or another value. The printer device remains in form insertion mode. This allows an application to perform some user interaction and reissue the **beginInsertion** method without altering the form handling mechanism.

**Errors**      A JposException may be thrown when this method is invoked. For further information, see "Exceptions" on page 15.

Some possible values of the exception's *ErrorCode* property are:

Value	Meaning
JPOS_E_ILLEGAL	The slip station does not exist (see the <b>CapSlpPresent</b> property) or an invalid <i>timeout</i> parameter was specified.
JPOS_E_TIMEOUT	The specified time has elapsed without the form being properly inserted.

**See Also**      **beginRemoval** Method, **endInsertion** Method, **endRemoval** Method

## beginRemoval Method

**Syntax**      **void beginRemoval (int *timeout*) throws JposException;**

Parameter	Description
-----------	-------------

<i>timeout</i>	The number of milliseconds before failing the method
----------------	--

If zero, the method initiates the begin removal mode, then returns the appropriate status immediately. If JPOS\_FOREVER (-1), the method initiates the begin removal mode, then waits as long as needed until either the form is removed or an error occurs.

**Remarks**      Initiates form removal processing.

When called, the printer is made ready to remove a form by opening the form handling “jaws” or activating a form ejection mode. This method is paired with the **endRemoval** method for controlling form removal.

If the printer device cannot be placed into removal or ejection mode, a JposException is thrown. Otherwise, form removal is monitored until either:

- The form is successfully removed.
- The form is not removed before *timeout* milliseconds have elapsed, or an error is reported by the printer device. In this case, the a JposException is thrown with an *ErrorCode* of JPOS\_E\_TIMEOUT or another value. The printer device remains in form removal mode. This allows an application to perform some user interaction and reissue the **beginRemoval** method without altering the form handling mechanism.

**Errors**      A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_ILLEGAL	The printer does not have a slip station (see the <b>CapSlpPresent</b> property) or an invalid <i>timeout</i> parameter was specified.
JPOS_E_TIMEOUT	The specified time has elapsed without the form being properly removed.

**See Also**      **beginInsertion** Method, **endInsertion** Method, **endRemoval** Method

**changePrintSide Method****Added in Release 1.5****Syntax**      **void changePrintSide (int *side*) throws JposException;**

The *side* parameter indicates the side on which to print. Valid values are:

<b>Value</b>	<b>Description</b>
PTR_PS_SIDE1	Indicates that the default print side of the document is selected. (Default print side is the side where printing will occur immediately after a document has been inserted. Therefore, PTR_PS_SIDE1 is selected after <b>beginInsertion/endInsertion</b> is executed.)
PTR_PS_SIDE2	Indicates that the opposite side of the document from the one that the printer defaults to is to be selected. (Reverse side of PTR_PS_SIDE1.)
PTR_PS_OPPOSITE	Indicates that the current printing side is switched and printing will now occur on the opposite side of the slip. (e.g., if <b>SlpPrintSide</b> was PTR_PS_SIDE1, it is to be changed to PTR_PS_SIDE2.)

**Remarks**      Selects the side of the document where printing is to occur.

This allows a print operation to occur on both sides of a document. This may be accomplished by mechanical paper handling of the document or by using multiple print heads that are positioned to print on each side of the document.

If a document is not inserted, an error is returned.

If *side* is not **SlpPrintSide** or *side* is PTR\_PS\_OPPOSITE, the side of the document is changed and the document is fed to TOF. If *side* is **SlpPrintSide**, nothing occurs and method returns.

Executing the method may cause the **SlpPrintSide** property to change.



**Errors** A JposException may be thrown when this method is invoked. For further information, see “Errors” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
JPOS_E_BUSY	Cannot be performed while output is in progress. (Can only apply if <b>AsyncMode</b> is false.)
JPOS_E_ILLEGAL	One of the following errors occurred: * The slip station does not exist (see the <b>CapSlpPresent</b> property) * the printer does not support both sides printing (see the <b>CapSlpBothSidesPrint</b> property) * an invalid <i>side</i> parameter was specified
JPOS_E_EXTENDED	<i>ErrorCodeExtended</i> = JPOS_EPTR_COVER_OPEN: The printer cover is open. (Can only apply if <b>AsyncMode</b> is false.) <i>ErrorCodeExtended</i> = JPOS_EPTR_SLP_EMPTY: The slip station was specified, but a form is not inserted. (Can only apply if <b>AsyncMode</b> is false.)

**See Also** **CapSlpBothSidesPrint** Property, **CapSlpPresent** Property, **SlpPrintSide** Property, **cutPaper** Method.

**cutPaper Method**

**Syntax**      **void cutPaper (int *percentage*) throws JposException;**

<b>Parameter</b>	<b>Description</b>
------------------	--------------------

<i>percentage</i>	The percentage of paper to cut.
-------------------	---------------------------------

The constant identifier PTR\_CP\_FULLLCUT or the value 100 causes a full paper cut. Other values request a partial cut percentage.

**Remarks**      Cuts the receipt paper.

This method is performed synchronously if **AsyncMode** is false, and asynchronously if **AsyncMode** is true.

Many printers with paper cut capability can perform both full and partial cuts. Some offer gradations of partial cuts, such as a perforated cut and an almost-full cut. Although the exact type of cut will vary by printer capabilities, the following general guidelines apply:

<b>Value</b>	<b>Meaning</b>
100	Full cut.
90	Leave only a small portion of paper for very easy final separation.
70	Perforate the paper for final separation that is somewhat more difficult and unlikely to occur by accidental handling.
50	Partial perforation of the paper.

The Device Service will select an appropriate type of cut based on the capabilities of its device and these general guidelines.

An escape sequence embedded in a **printNormal** or **printImmediate** method call may also be used to cause a paper cut.

**Errors** A `JposException` may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_BUSY	Cannot perform while output is in progress. (Can only apply if <b>AsyncMode</b> is false.)
JPOS_E_ILLEGAL	An invalid percentage was specified, the receipt station does not exist (see the <b>CapRecPresent</b> property), or the receipt printer does not have paper cutting ability (see the <b>CapRecPapercut</b> property).
JPOS_E_EXTENDED	<p><i>ErrorCodeExtended</i> = JPOS_EPTR_COVER_OPEN: The printer cover is open. (Can only apply if <b>AsyncMode</b> is false.)</p> <p><i>ErrorCodeExtended</i> = JPOS_EPTR_REC_EMPTY: The receipt station is out of paper. (Can only apply if <b>AsyncMode</b> is false.)</p>

**See Also** “Data Characters and Escape Sequences”

**endInsertion Method**

**Syntax**      **void endInsertion () throws JposException;**

**Remarks**      Ends form insertion processing.

When called, the printer is taken out of form insertion mode. If the slip device has forms “jaws,” they are closed by this method. If no form is present, a *JposException* is thrown with its *ErrorCodeExtended* property set to *JPOS\_EPTR\_SLP\_EMPTY*.

This method is paired with the **beginInsertion** method for controlling form insertion. The application may choose to call this method immediately after a successful **beginInsertion** if it wants to use the printer sensors to determine when a form is positioned within the slip printer. Alternatively, the application may prompt the user and wait for a key press before calling this method.

**Errors**          A *JposException* may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
<i>JPOS_E_ILLEGAL</i>	The printer is not in slip insertion mode.
<i>JPOS_E_EXTENDED</i>	<i>ErrorCodeExtended</i> = <i>JPOS_EPTR_COVER_OPEN</i> : The device was taken out of insertion mode while the printer cover was open.  <i>ErrorCodeExtended</i> = <i>JPOS_EPTR_SLP_EMPTY</i> : The device was taken out of insertion mode without a form being inserted.

**See Also**      **beginInsertion** Method, **beginRemoval** Method, **endRemoval** Method

## endRemoval Method

**Syntax**      **void endRemoval () throws JposException;**

**Remarks**      Ends form removal processing.

When called, the printer is taken out of form removal or ejection mode. If a form is present, a `JposException` is thrown with its *ErrorCodeExtended* property set to `JPOS_EPTR_SLP_FORM`.

This method is paired with the **beginRemoval** method for controlling form removal. The application may choose to call this method immediately after a successful **beginRemoval** if it wants to use the printer sensors to determine when the form has been removed. Alternatively, the application may prompt the user and wait for a key press before calling this method.

**Errors**      A `JposException` may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
<code>JPOS_E_ILLEGAL</code>	The printer is not in slip removal mode.
<code>JPOS_E_EXTENDED</code>	<i>ErrorCodeExtended</i> = <code>JPOS_EPTR_SLP_FORM</code> : The device was taken out of removal mode while a form was still present.

**See Also**      **beginInsertion** Method, **endInsertion** Method, **beginRemoval** Method

**markFeed Method*****Added in Release 1.5*****Syntax**      **void markFeed (int *type*) throws JposException;**

The *type* parameter indicates the type of mark sensed paper handling. Valid values are:

<b>Value</b>	<b>Description</b>
PTR_MF_TO_TAKEUP	Feed the Mark Sensed paper to the paper take-up position.
PTR_MF_TO_CUTTER	Feed the Mark Sensed paper to the auto cutter cutting position.
PTR_MF_TO_CURRENT_TOF	Feed the Mark Sensed paper to the present paper's top of form. (Reverse feed.)
PTR_MF_TO_NEXT_TOF	Feed the Mark Sensed paper to the next paper's top of form.

**Remarks**      This method is used to utilize the printer's mark sensor for receipt paper.

This method is performed synchronously if **AsyncMode** is false, and asynchronously if **AsyncMode** is true.

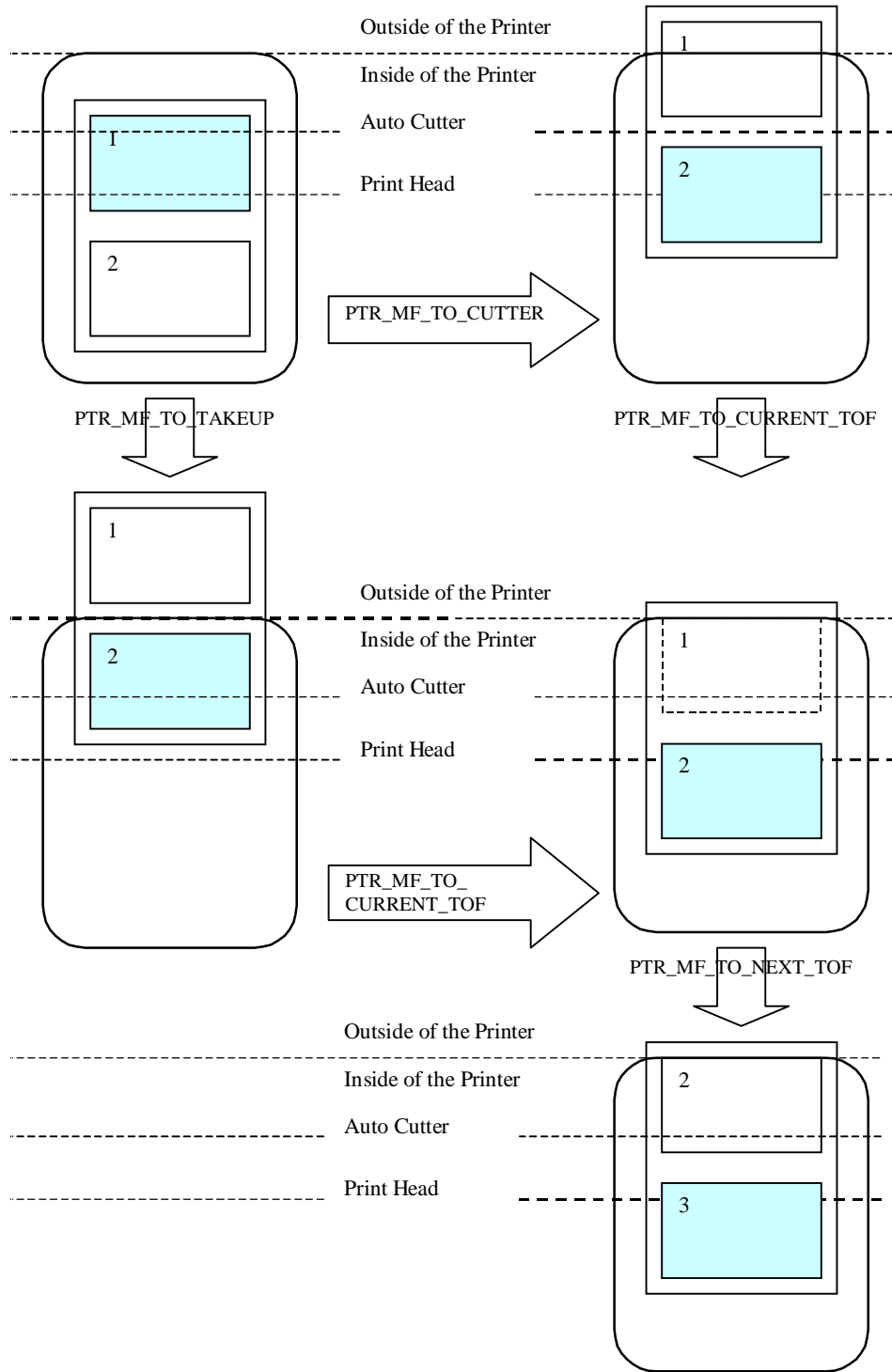
If *type* is PTR\_MF\_TO\_TAKEUP, the printer will feed the mark sensed paper so that the present form is moved so that it can be manually removed by the operator.

If *type* is PTR\_MF\_TO\_CUTTER, the printer will feed the mark sensed paper so that the present form is in position to be cut off by the auto cutter. This will usually be followed by a call to the **cutPaper** method.

If *type* is PTR\_MF\_TO\_CURRENT\_TOF, the printer will feed the mark sensed paper (backwards if necessary) so that the print head points to the top of the present form.

If *type* is PTR\_MF\_TO\_NEXT\_TOF, the printer will feed the mark sensed paper so that print head points to the top of the next form.

The following diagram provides a pictorial representation of the functions performed by this method.



**Errors** A `JposException` may be thrown when this method is invoked. For further information, see “Errors” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_BUSY	Cannot be performed while output is in progress. (Can only apply if <b>AsyncMode</b> is false.)
JPOS_E_ILLEGAL	The receipt print station does not support the given mark sensed paper handling function. (Refer to the <b>CapRecMarkFeed</b> property)
JPOS_E_EXTENDED	<i>ErrorCodeExtended</i> = JPOS_EPTR_COVER_OPEN: The printer cover is open. (Can only apply if <b>AsyncMode</b> is false.)  <i>ErrorCodeExtended</i> = JPOS_EPTR_REC_EMPTY: The receipt paper is empty. (Can only apply if <b>AsyncMode</b> is false.)

**See Also** **CapRecMarkFeed** Property.





The *symbology* parameter has one of the following values:

<b>Value</b>	<b>Meaning</b>
<i>One Dimensional Symbolologies</i>	
PTR_BCS_UPCA	UPC-A
PTR_BCS_UPCA_S	UPC-A with supplemental barcode
PTR_BCS_UPCE	UPC-E
PTR_BCS_UPCE_S	UPC-E with supplemental barcode
PTR_BCS_UPCD1	UPC-D1
PTR_BCS_UPCD2	UPC-D2
PTR_BCS_UPCD3	UPC-D3
PTR_BCS_UPCD4	UPC-D4
PTR_BCS_UPCD5	UPC-D5
PTR_BCS_EAN8	EAN 8 (= JAN 8)
PTR_BCS_JAN8	JAN 8 (= EAN 8)
PTR_BCS_EAN8_S	EAN 8 with supplemental barcode
PTR_BCS_EAN13	EAN 13 (= JAN 13)
PTR_BCS_JAN13	JAN 13 (= EAN 13)
PTR_BCS_EAN13_S	EAN 13 with supplemental barcode
PTR_BCS_EAN128	EAN-128
PTR_BCS_TF	Standard (or discrete) 2 of 5
PTR_BCS_ITF	Interleaved 2 of 5
PTR_BCS_Codabar	Codabar
PTR_BCS_Code39	Code 39
PTR_BCS_Code93	Code 93
PTR_BCS_Code128	Code 128
PTR_BCS_OCRA	OCR "A"
PTR_BCS_OCRB	OCR "B"
<i>Two Dimensional Symbolologies</i>	
PTR_BCS_PDF417	PDF 417
PTR_BCS_MAXICODE	MAXICODE

*Special Cases*

**PTR\_BCS\_OTHER** If a device service defines additional symbologies, they will be greater or equal to this value.

**Remarks** Prints a bar code on the specified printer station.

This method is performed synchronously if **AsyncMode** is false, and asynchronously if **AsyncMode** is true.

If **RotateSpecial** indicates that the bar code is to be rotated, then perform the rotation. The *height*, *width*, and *textPosition* parameters are applied to the bar code before the rotation. For example, if PTR\_BC\_TEXT\_BELOW is specified and the bar code is rotated left, then the text will appear on the paper to the right of the bar code.

**Errors** A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
JPOS_E_ILLEGAL	One of the following errors occurred: * <i>station</i> does not exist * <i>station</i> does not support bar code printing * <i>height</i> or <i>width</i> are zero or too big * <i>symbology</i> is not supported * <i>alignment</i> is invalid or too big * <i>textPosition</i> is invalid * The <b>RotateSpecial</b> rotation is not supported
JPOS_E_BUSY	Cannot perform while output is in progress. (Can only apply if <b>AsyncMode</b> is false.)
JPOS_E_EXTENDED	<i>ErrorCodeExtended</i> = JPOS_EPTR_COVER_OPEN: The printer cover is open.  <i>ErrorCodeExtended</i> = JPOS_EPTR_JRN_EMPTY: The journal station was specified but is out of paper.  <i>ErrorCodeExtended</i> = JPOS_EPTR_JRN_CARTRIDGE_REMOVED: A journal cartridge has been removed. (Can only apply if <b>AsyncMode</b> is false.)  <i>ErrorCodeExtended</i> = JPOS_EPTR_JRN_CARTRIDGE_EMPTY: A journal cartridge is empty. (Can only apply if <b>AsyncMode</b> is false.)  <i>ErrorCodeExtended</i> = JPOS_EPTR_JRN_HEAD_CLEANING:

A journal cartridge head is being cleaned.  
(Can only apply if **AsyncMode** is false.)

*ErrorCodeExtended* =

JPOS\_EPTR\_REC\_EMPTY:

The receipt station was specified but is out of paper.  
(Can only apply if **AsyncMode** is false.)

*ErrorCodeExtended* =

JPOS\_EPTR\_REC\_CARTRIDGE\_REMOVED:

A receipt cartridge has been removed.  
(Can only apply if **AsyncMode** is false.)

*ErrorCodeExtended* =

JPOS\_EPTR\_REC\_CARTRIDGE\_EMPTY:

A receipt cartridge is empty.  
(Can only apply if **AsyncMode** is false.)

*ErrorCodeExtended* =

JPOS\_EPTR\_REC\_HEAD\_CLEANING:

A receipt cartridge head is being cleaned.  
(Can only apply if **AsyncMode** is false.)

*ErrorCodeExtended* = JPOS\_EPTR\_SLP\_EMPTY:

The slip station was specified, but a form is not inserted.  
(Can only apply if **AsyncMode** is false.)

*ErrorCodeExtended* =

JPOS\_EPTR\_SLP\_CARTRIDGE\_REMOVED:

A slip cartridge has been removed.  
(Can only apply if **AsyncMode** is false.)

*ErrorCodeExtended* =

JPOS\_EPTR\_SLP\_CARTRIDGE\_EMPTY:

A slip cartridge is empty.  
(Can only apply if **AsyncMode** is false.)

*ErrorCodeExtended* =

JPOS\_EPTR\_SLP\_HEAD\_CLEANING:

A slip cartridge head is being cleaned.  
(Can only apply if **AsyncMode** is false.)

**printBitmap Method**

**Syntax**      **void printBitmap (int station, String fileName, int width, int alignment) throws JposException;**

<b>Parameter</b>	<b>Description</b>
<i>station</i>	The printer station to be used. May be either PTR_S_RECEIPT or PTR_S_SLIP.
<i>fileName</i>	File name or URL of bitmap file. Various file formats may be supported, such as bmp (uncompressed format), gif or jpeg files.
<i>width</i>	Printed width of the bitmap to be performed. See values below.
<i>alignment</i>	Placement of the bitmap. See values below.

The *width* parameter has one of the following values:

<b>Value</b>	<b>Meaning</b>
PTR_BM_ASIS	Print the bitmap with one bitmap pixel per printer dot.
<i>Other Values</i>	Bitmap width expressed in the unit of measure given by <b>MapMode</b> .

The *alignment* parameter has one of the following values:

<b>Value</b>	<b>Meaning</b>
PTR_BM_LEFT	Align with the left-most print column.
PTR_BM_CENTER	Align in the center of the station.
PTR_BM_RIGHT	Align with the right-most print column.
<i>Other Values</i>	Distance from the left-most print column to the start of the bitmap. Expressed in the unit of measure given by <b>MapMode</b> .

**Remarks**      Prints a bitmap on the specified printer station.

This method is performed synchronously if **AsyncMode** is false, and asynchronously if **AsyncMode** is true.

The *width* parameter controls transformation of the bitmap. If *width* is `PTR_BM_ASIS`, then no transformation is performed. The bitmap is printed with one bitmap pixel per printer dot. Advantages of this option are that it:

- Provides the highest performance bitmap printing.
- Works well for bitmaps tuned for a specific printer's aspect ratio between horizontal dots and vertical dots.

If *width* is non-zero, then the bitmap will be transformed by stretching or compressing the bitmap such that its width is the specified width and the aspect ratio is unchanged. Advantages of this option are:

- Sizes a bitmap to fit a variety of printers.
- Maintains the bitmap's aspect ratio.

Disadvantages are:

- Lowers performance than untransformed data.
- Some lines and images that are "smooth" in the original bitmap may show some "ratcheting."

## Errors

A `JposException` may be thrown when this method is invoked. For further information, see "Exceptions" on page 15.

Some possible values of the exception's *ErrorCode* property are:

Value	Meaning
<code>JPOS_E_BUSY</code>	Cannot perform while output is in progress. (Can only apply if <b>AsyncMode</b> is false.)
<code>JPOS_E_ILLEGAL</code>	One of the following errors occurred: * <i>station</i> does not exist * <i>station</i> does not support bitmap printing * <i>width</i> parameter is invalid or too big * <i>alignment</i> is invalid or too big
<code>JPOS_E_NOEXIST</code>	<i>fileName</i> was not found.
<code>JPOS_E_EXTENDED</code>	<i>ErrorCodeExtended</i> = <code>JPOS_EPTR_COVER_OPEN</code> : The printer cover is open.  <i>ErrorCodeExtended</i> = <code>JPOS_EPTR_JRN_EMPTY</code> : The journal station was specified but is out of paper.  <i>ErrorCodeExtended</i> = <code>JPOS_EPTR_JRN_CARTRIDGE_REMOVED</code> : A journal cartridge has been removed. (Can only apply if <b>AsyncMode</b> is false.)  <i>ErrorCodeExtended</i> = <code>JPOS_EPTR_JRN_CARTRIDGE_EMPTY</code> : A journal cartridge is empty.

(Can only apply if **AsyncMode** is false.)

*ErrorCodeExtended* =

JPOS\_EPTR\_JRN\_HEAD\_CLEANING:

A journal cartridge head is being cleaned.

(Can only apply if **AsyncMode** is false.)

*ErrorCodeExtended* =

JPOS\_EPTR\_REC\_EMPTY:

The receipt station was specified but is out of paper.

(Can only apply if **AsyncMode** is false.)

*ErrorCodeExtended* =

JPOS\_EPTR\_REC\_CARTRIDGE\_REMOVED:

A receipt cartridge has been removed.

(Can only apply if **AsyncMode** is false.)

*ErrorCodeExtended* =

JPOS\_EPTR\_REC\_CARTRIDGE\_EMPTY:

A receipt cartridge is empty.

(Can only apply if **AsyncMode** is false.)

*ErrorCodeExtended* =

JPOS\_EPTR\_REC\_HEAD\_CLEANING:

A receipt cartridge head is being cleaned.

(Can only apply if **AsyncMode** is false.)

*ErrorCodeExtended* = JPOS\_EPTR\_SLP\_EMPTY:

The slip station was specified, but a form is not inserted.

(Can only apply if **AsyncMode** is false.)

*ErrorCodeExtended* =

JPOS\_EPTR\_SLP\_CARTRIDGE\_REMOVED:

A slip cartridge has been removed.

(Can only apply if **AsyncMode** is false.)

*ErrorCodeExtended* =

JPOS\_EPTR\_SLP\_CARTRIDGE\_EMPTY:

A slip cartridge is empty.

(Can only apply if **AsyncMode** is false.)

*ErrorCodeExtended* =

JPOS\_EPTR\_SLP\_HEAD\_CLEANING:

A slip cartridge head is being cleaned.

(Can only apply if **AsyncMode** is false.)

**printImmediate Method**

**Syntax**      **void printImmediate (int *station*, String *data*) throws JposException;**

Parameter	Description
<i>station</i>	The printer station to be used. May be either PTR_S_JOURNAL, PTR_S_RECEIPT or PTR_S_SLIP.
<i>data</i>	The characters to be printed. May consist of printable characters, escape sequences, carriage returns (13 decimal), and newline / line feed (10 decimal).

**Remarks**      Prints *data* on the printer *station* immediately.

This method tries to print its data immediately – that is, as the very next printer operation. It may be called when asynchronous output is outstanding. This method is primarily intended for use in exception conditions when asynchronous output is outstanding, such as within an error event handler.

Special character values within *data* are:

Value	Meaning
Newline / Line Feed (10 decimal)	Print any data in the line buffer, and feed to the next print line. (A Carriage Return is not required in order to cause the line to be printed.)
Carriage Return (13 decimal)	If a Carriage Return immediately precedes a Line Feed, or if the line buffer is empty, then it is ignored.  Otherwise, the line buffer is printed and the printer does not feed to the next print line. On some printers, print without feed may be directly supported. On others, a print may always feed to the next line, in which case the device service will print the line buffer and perform a reverse line feed if supported. If the printer does not support either of these features, then Carriage Return acts like a Line Feed.  The <b>validateData</b> method may be used to determine whether a Carriage Return without Line Feed is possible, and whether a reverse line feed is required to support it.



**Errors** A `JposException` may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
JPOS_E_ILLEGAL	The specified <i>station</i> does not exist. (See the <b>CapJrnPresent</b> , <b>CapRecPresent</b> , and <b>CapSlpPresent</b> properties.)
JPOS_E_EXTENDED	<p><i>ErrorCodeExtended</i> = JPOS_EPTR_COVER_OPEN: The printer cover is open.</p> <p><i>ErrorCodeExtended</i> = JPOS_EPTR_JRN_EMPTY: The journal station was specified but is out of paper.</p> <p><i>ErrorCodeExtended</i> = JPOS_EPTR_JRN_CARTRIDGE_REMOVED: A journal cartridge has been removed. (Can only apply if <b>AsyncMode</b> is false.)</p> <p><i>ErrorCodeExtended</i> = JPOS_EPTR_JRN_CARTRIDGE_EMPTY: A journal cartridge is empty. (Can only apply if <b>AsyncMode</b> is false.)</p> <p><i>ErrorCodeExtended</i> = JPOS_EPTR_JRN_HEAD_CLEANING: A journal cartridge head is being cleaned. (Can only apply if <b>AsyncMode</b> is false.)</p> <p><i>ErrorCodeExtended</i> = JPOS_EPTR_REC_EMPTY: The receipt station was specified but is out of paper. (Can only apply if <b>AsyncMode</b> is false.)</p> <p><i>ErrorCodeExtended</i> = JPOS_EPTR_REC_CARTRIDGE_REMOVED: A receipt cartridge has been removed. (Can only apply if <b>AsyncMode</b> is false.)</p> <p><i>ErrorCodeExtended</i> = JPOS_EPTR_REC_CARTRIDGE_EMPTY: A receipt cartridge is empty. (Can only apply if <b>AsyncMode</b> is false.)</p> <p><i>ErrorCodeExtended</i> = JPOS_EPTR_REC_HEAD_CLEANING: A receipt cartridge head is being cleaned. (Can only apply if <b>AsyncMode</b> is false.)</p>

*ErrorCodeExtended* = JPOS\_EPTR\_SLP\_EMPTY:  
The slip station was specified, but a form is not inserted.  
(Can only apply if **AsyncMode** is false.)

*ErrorCodeExtended* =  
JPOS\_EPTR\_SLP\_CARTRIDGE\_REMOVED:  
A slip cartridge has been removed.  
(Can only apply if **AsyncMode** is false.)

*ErrorCodeExtended* =  
JPOS\_EPTR\_SLP\_CARTRIDGE\_EMPTY:  
A slip cartridge is empty.  
(Can only apply if **AsyncMode** is false.)

*ErrorCodeExtended* =  
JPOS\_EPTR\_SLP\_HEAD\_CLEANING:  
A slip cartridge head is being cleaned.  
(Can only apply if **AsyncMode** is false.)

**See Also**    **printNormal** Method, **printTwoNormal** Method

**printNormal Method**

**Syntax**      **void printNormal (int *station*, String *data*) throws JposException;**

Parameter	Description
<i>station</i>	The printer station to be used. May be either PTR_S_JOURNAL, PTR_S_RECEIPT or PTR_S_SLIP.
<i>data</i>	The characters to be printed. May consist of printable characters, escape sequences, carriage returns (13 decimal), and Newline / line feed (10 decimal).

**Remarks**      Prints *data* on the printer *station*.

This method is performed synchronously if **AsyncMode** is false, and asynchronously if **AsyncMode** is true.

Special character values within *data* are:

Value	Meaning
Newline / Line Feed (10 Decimal)	Print any data in the line buffer, and feed to the next print line. (A Carriage Return is not required in order to cause the line to be printed.)
Carriage Return (13 Decimal)	If a Carriage Return immediately precedes a Line Feed, or if the line buffer is empty, then it is ignored.  Otherwise, the line buffer is printed and the printer does not feed to the next print line. On some printers, print without feed may be directly supported. On others, a print may always feed to the next line, in which case the device service will print the line buffer and perform a reverse line feed if supported. If the printer does not support either of these features, then Carriage Return acts like a Line Feed.  The <b>validateData</b> method may be used to determine whether a Carriage Return without Line Feed is possible, and whether a reverse line feed is required to support it.

**Errors**      A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_ILLEGAL	The specified <i>station</i> does not exist. (See the

**CapJrnPresent, CapRecPresent, and CapSlpPresent** properties.)

JPOS_E_BUSY	Cannot perform while output is in progress. (Can only apply if <b>AsyncMode</b> is false.)
JJPOS_E_EXTENDED	<p><i>ErrorCodeExtended</i> = JPOS_EPTR_COVER_OPEN: The printer cover is open.</p> <p><i>ErrorCodeExtended</i> = JPOS_EPTR_JRN_EMPTY: The journal station was specified but is out of paper.</p> <p><i>ErrorCodeExtended</i> = JPOS_EPTR_JRN_CARTRIDGE_REMOVED: A journal cartridge has been removed. (Can only apply if <b>AsyncMode</b> is false.)</p> <p><i>ErrorCodeExtended</i> = JPOS_EPTR_JRN_CARTRIDGE_EMPTY: A journal cartridge is empty. (Can only apply if <b>AsyncMode</b> is false.)</p> <p><i>ErrorCodeExtended</i> = JPOS_EPTR_JRN_HEAD_CLEANING: A journal cartridge head is being cleaned. (Can only apply if <b>AsyncMode</b> is false.)</p> <p><i>ErrorCodeExtended</i> = JPOS_EPTR_REC_EMPTY: The receipt station was specified but is out of paper. (Can only apply if <b>AsyncMode</b> is false.)</p> <p><i>ErrorCodeExtended</i> = JPOS_EPTR_REC_CARTRIDGE_REMOVED: A receipt cartridge has been removed. (Can only apply if <b>AsyncMode</b> is false.)</p> <p><i>ErrorCodeExtended</i> = JPOS_EPTR_REC_CARTRIDGE_EMPTY: A receipt cartridge is empty. (Can only apply if <b>AsyncMode</b> is false.)</p> <p><i>ErrorCodeExtended</i> = JPOS_EPTR_REC_HEAD_CLEANING: A receipt cartridge head is being cleaned. (Can only apply if <b>AsyncMode</b> is false.)</p> <p><i>ErrorCodeExtended</i> = JPOS_EPTR_SLP_EMPTY: The slip station was specified, but a form is not inserted. (Can only apply if <b>AsyncMode</b> is false.)</p> <p><i>ErrorCodeExtended</i> = JPOS_EPTR_SLP_CARTRIDGE_REMOVED: A slip cartridge has been removed. (Can only apply if <b>AsyncMode</b> is false.)</p>

*ErrorCodeExtended* =  
JPOS\_EPTR\_SLP\_CARTRIDGE\_EMPTY:  
A slip cartridge is empty.  
(Can only apply if **AsyncMode** is false.)

*ErrorCodeExtended* =  
JPOS\_EPTR\_SLP\_HEAD\_CLEANING:  
A slip cartridge head is being cleaned.  
(Can only apply if **AsyncMode** is false.)

**See Also**    **printImmediate** Method, **printTwoNormal** Method



or internal print order.

Due to this inconsistency, the application should use the new constants if the Device Control and Device Service versions indicate Release 1.3 or later.

**Release 1.3 and later**

Device Service for Release 1.3 or later should support both sets of constants. The vendor should define and document the behavior of the obsolete constants.

The sequence of stations in the constants does not imply the physical printing sequence on the stations. The physical sequence depends on the printer and may be different based on the bi-directional printing multiple print heads and so on.

**Errors** A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
JPOS_E_ILLEGAL	The specified <i>stations</i> do not support concurrent printing. (See the <b>CapConcurrentJrnRec</b> , <b>CapConcurrentJrnSlp</b> , and <b>CapConcurrentRecSlp</b> properties.)
JPOS_E_BUSY	Cannot perform while output is in progress. (Can only apply if <b>AsyncMode</b> is false.)
JPOS_E_EXTENDED	<p><i>ErrorCodeExtended</i> = JPOS_EPTR_COVER_OPEN: The printer cover is open.</p> <p><i>ErrorCodeExtended</i> = JPOS_EPTR_JRN_EMPTY: The journal station was specified but is out of paper.</p> <p><i>ErrorCodeExtended</i> = JPOS_EPTR_JRN_CARTRIDGE_REMOVED: A journal cartridge has been removed. (Can only apply if <b>AsyncMode</b> is false.)</p> <p><i>ErrorCodeExtended</i> = JPOS_EPTR_JRN_CARTRIDGE_EMPTY: A journal cartridge is empty. (Can only apply if <b>AsyncMode</b> is false.)</p> <p><i>ErrorCodeExtended</i> = JPOS_EPTR_JRN_HEAD_CLEANING: A journal cartridge head is being cleaned. (Can only apply if <b>AsyncMode</b> is false.)</p> <p><i>ErrorCodeExtended</i> = JPOS_EPTR_REC_EMPTY: The receipt station was specified but is out of paper. (Can only apply if <b>AsyncMode</b> is false.)</p> <p><i>ErrorCodeExtended</i> =</p>

JPOS\_EPTR\_REC\_CARTRIDGE\_REMOVED:

A receipt cartridge has been removed.  
(Can only apply if **AsyncMode** is false.)

*ErrorCodeExtended* =

JPOS\_EPTR\_REC\_CARTRIDGE\_EMPTY:

A receipt cartridge is empty.  
(Can only apply if **AsyncMode** is false.)

*ErrorCodeExtended* =

JPOS\_EPTR\_REC\_HEAD\_CLEANING:

A receipt cartridge head is being cleaned.  
(Can only apply if **AsyncMode** is false.)

*ErrorCodeExtended* = JPOS\_EPTR\_SLP\_EMPTY:

The slip station was specified, but a form is not inserted.  
(Can only apply if **AsyncMode** is false.)

*ErrorCodeExtended* =

JPOS\_EPTR\_SLP\_CARTRIDGE\_REMOVED:

A slip cartridge has been removed.  
(Can only apply if **AsyncMode** is false.)

*ErrorCodeExtended* =

JPOS\_EPTR\_SLP\_CARTRIDGE\_EMPTY:

A slip cartridge is empty.  
(Can only apply if **AsyncMode** is false.)

*ErrorCodeExtended* =

JPOS\_EPTR\_SLP\_HEAD\_CLEANING:

A slip cartridge head is being cleaned.  
(Can only apply if **AsyncMode** is false.)

See Also **printNormal** Method



## rotatePrint Method

**Syntax**      **void rotatePrint (int station, int rotation) throws JposException;**

Parameter	Description
<i>station</i>	The printer station to be used. May be PTR_S_RECEIPT or PTR_S_SLIP.
<i>rotation</i>	Direction of rotation. See values below.
Value	Meaning
PTR_RP_RIGHT90	Rotate printing 90° to the right (clockwise)
PTR_RP_LEFT90	Rotate printing 90° to the left (counter-clockwise)
PTR_RP_ROTATE180	Rotate printing 180°, that is, print upside-down
PTR_RP_NORMAL	End rotated printing.

**Remarks**      Enters or exits rotated print mode.

This method is performed synchronously if **AsyncMode** is false, and asynchronously if **AsyncMode** is true.

If *rotation* is PTR\_RP\_ROTATE180, then upside-down print mode is entered. Subsequent calls to **printNormal** or **printImmediate** will print the data upside-down until **rotatePrint** is called with the *rotation* parameter set to PTR\_RP\_NORMAL.

Each print line is rotated by 180°. Lines are printed in the order that they are sent, with the start of each line justified at the right margin of the printer station. Only print methods **printNormal** and **printImmediate** may be used while in upside-down print mode.

If *rotation* is PTR\_RP\_RIGHT90 or PTR\_RP\_LEFT90, then sideways print mode is entered. Subsequent calls to **printNormal** will buffer the print data (either at the printer or the Device Service, depending on the printer capabilities) until **rotatePrint** is called with the *rotation* parameter set to PTR\_RP\_NORMAL. (In this case, **printNormal** only buffers the data – it does not initiate printing. Also, the value of the **AsyncMode** property does not affect its operation: No **OutputID** will be assigned to the request, nor will an **OutputCompleteEvent** be enqueued.) Each print line is rotated by 90°. If the lines are not all the same length, then they are justified at the start of each line. Only **printNormal** may be used while in sideways print mode.

If *rotation* is PTR\_RP\_NORMAL, then rotated print mode is exited. If sideways-rotated print mode was in effect and some data was buffered by calls to the **printNormal** method, then the buffered data is printed. The entire rotated block of lines are treated as one message.

Changing the rotation mode may also change the station's line height, line spacing, line width, and other metrics.

Calling the **clearOutput** method cancels rotated print mode. Any buffered sideways rotated print lines are also cleared.

**Errors** A `JposException` may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_ILLEGAL	The specified <i>station</i> does not exist (see the <b>CapJrnPresent</b> , <b>CapRecPresent</b> , and <b>CapSlpPresent</b> properties), or the <i>station</i> does not support the specified rotation (see the station’s rotation capability properties).
JPOS_E_BUSY	Cannot perform while output is in progress. (Can only apply if <b>AsyncMode</b> is false.)
JPOS_E_EXTENDED	<p><i>ErrorCodeExtended</i> = JPOS_EPTR_COVER_OPEN: The printer cover is open.</p> <p><i>ErrorCodeExtended</i> = JPOS_EPTR_JRN_EMPTY: The journal station was specified but is out of paper.</p> <p><i>ErrorCodeExtended</i> = JPOS_EPTR_JRN_CARTRIDGE_REMOVED: A journal cartridge has been removed. (Can only apply if <b>AsyncMode</b> is false.)</p> <p><i>ErrorCodeExtended</i> = JPOS_EPTR_JRN_CARTRIDGE_EMPTY: A journal cartridge is empty. (Can only apply if <b>AsyncMode</b> is false.)</p> <p><i>ErrorCodeExtended</i> = JPOS_EPTR_JRN_HEAD_CLEANING: A journal cartridge head is being cleaned. (Can only apply if <b>AsyncMode</b> is false.)</p> <p><i>ErrorCodeExtended</i> = JPOS_EPTR_REC_EMPTY: The receipt station was specified but is out of paper. (Can only apply if <b>AsyncMode</b> is false.)</p> <p><i>ErrorCodeExtended</i> = JPOS_EPTR_REC_CARTRIDGE_REMOVED: A receipt cartridge has been removed. (Can only apply if <b>AsyncMode</b> is false.)</p> <p><i>ErrorCodeExtended</i> = JPOS_EPTR_REC_CARTRIDGE_EMPTY: A receipt cartridge is empty. (Can only apply if <b>AsyncMode</b> is false.)</p> <p><i>ErrorCodeExtended</i> =</p>

JPOS\_EPTR\_REC\_HEAD\_CLEANING:

A receipt cartridge head is being cleaned.

(Can only apply if **AsyncMode** is false.)

*ErrorCodeExtended* = JPOS\_EPTR\_SLP\_EMPTY:

The slip station was specified, but a form is not inserted.

(Can only apply if **AsyncMode** is false.)

*ErrorCodeExtended* =

JPOS\_EPTR\_SLP\_CARTRIDGE\_REMOVED:

A slip cartridge has been removed.

(Can only apply if **AsyncMode** is false.)

*ErrorCodeExtended* =

JPOS\_EPTR\_SLP\_CARTRIDGE\_EMPTY:

A slip cartridge is empty.

(Can only apply if **AsyncMode** is false.)

*ErrorCodeExtended* =

JPOS\_EPTR\_SLP\_HEAD\_CLEANING:

A slip cartridge head is being cleaned.

(Can only apply if **AsyncMode** is false.)

**See Also**    “Data Characters and Escape Sequences”

**setBitmap Method**

**Syntax**      **void setBitmap (int *bitmapNumber*, int *station*, String *fileName*, int *width*, int *alignment*) throws JposException;**

Parameter	Description
<i>bitmapNumber</i>	The number to be assigned to this bitmap. Two bitmaps, numbered 1 and 2, may be set.
<i>station</i>	The printer station to be used. May be either PTR_S_RECEIPT or PTR_S_SLIP.
<i>fileName</i>	File name or URL of bitmap file. Various file formats may be supported, such as bmp, gif or jpeg files. The file must be in uncompressed format. If set to an empty string (“”), then the bitmap is unset.
<i>width</i>	Printed width of the bitmap to be performed. See <b>printBitmap</b> for values.
<i>alignment</i>	Placement of the bitmap. See <b>printBitmap</b> for values.

**Remarks**      Saves information about a bitmap for later printing.

The bitmap may then be printed by calling the **printNormal** or **printImmediate** method with the print bitmap escape sequence in the print data. The print bitmap escape sequence will typically be included in a string for printing top and bottom transaction headers.

A Device Service may choose to cache the bitmap for later use to provide better performance. Regardless, the bitmap file and parameters are validated for correctness by this method.

The application must ensure that the printer station metrics, such as character width, line height, and line spacing are set for the *station* before calling this method. The device service may perform transformations on the bitmap in preparation for later printing based upon the current values.

The application may set bitmaps numbered 1 and 2 for each of the two valid *stations*. If desired, the same bitmap *fileName* may be set to the same *bitmapNumber* for each station, so that the same print bitmap escape sequence may be used for either station.

**Errors**      A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_ILLEGAL	One of the following errors occurred: * <i>bitmapNumber</i> is invalid * <i>station</i> does not exist * <i>station</i> does not support bitmap printing

\* *width* is too big  
 \* *alignment* is invalid or too big

JPOS\_E\_NOEXIST *fileName* was not found.

JPOS\_E\_EXTENDED *ErrorCodeExtended* = JPOS\_EPTR\_TOOBIG:  
 The bitmap is either too wide to print without transformation, or it is too big to transform.

*ErrorCodeExtended* = JPOS\_EPTR\_BADFORMAT:  
 The specified file is either not a bitmap file, or it is in an unsupported format.

**See Also** “Data Characters and Escape Sequences”, **printBitmap** Method

## setLogo Method

**Syntax** `void setLogo (int location, String data) throws JposException;`

Parameter	Description
<i>location</i>	The logo to be set. May be PTR_L_TOP or PTR_L_BOTTOM.
<i>data</i>	The characters that produce the logo. May consist of printable characters, escape sequences, carriage returns (13 decimal), and Newline / line feed (10 decimal).

**Remarks** Saves a data string as the top or bottom logo.

A logo may then be printed by calling the **printNormal**, **printTwoNormal**, or **printImmediate** method with the print top logo or print bottom logo escape sequence in the print data.

**Errors** A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_ILLEGAL	An invalid <i>location</i> was specified.

**See Also** “Data Characters and Escape Sequences”

**transactionPrint Method**

**Syntax**      **void transactionPrint (int station, int control) throws JposException;**

Parameter	Description
<i>station</i>	The printer station to be used. May be PTR_S_JOURNAL, PTR_S_RECEIPT, or PTR_S_SLIP.
<i>control</i>	Transaction control. See values below:

Value	Meaning
PTR_TP_TRANSACTION	Begin a transaction.
PTR_TP_NORMAL	End a transaction by printing the buffered data.

**Remarks**      Enters or exits transaction mode.

If *control* is PTR\_TP\_TRANSACTION, then transaction mode is entered. Subsequent calls to **printNormal**, **cutPaper**, **rotatePrint**, **printBarCode**, **printBitmap**, and **markFeed** will buffer the print data (either at the printer or the Device Service, depending on the printer capabilities) until **transactionPrint** is called with the *control* parameter set to PTR\_TP\_NORMAL. (In this case, the print methods only validate the method parameters and buffer the data – they do not initiate printing. Also, the value of the **AsyncMode** property does not affect their operation: No **OutputID** will be assigned to the request, nor will an **OutputCompleteEvent** be queued.)

If *control* is PTR\_TP\_NORMAL, then transaction mode is exited. If some data was buffered by calls to the methods **printNormal**, **cutPaper**, **rotatePrint**, **printBarCode**, **printBitmap**, and **markFeed** then the buffered data is printed. The entire transaction is treated as one message. This method is performed synchronously if **AsyncMode** is false, and asynchronously if **AsyncMode** is true.

Calling the **clearOutput** method cancels transaction mode. Any buffered print lines are also cleared.

**Errors**      A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_ILLEGAL	The specified <i>station</i> does not exist (see the <b>CapJrnPresent</b> , <b>CapRecPresent</b> , and <b>CapSlpPresent</b> properties), or <b>CapTransaction</b> is false.
JPOS_E_BUSY	Cannot perform while output is in progress. (Can only apply if <b>AsyncMode</b> is false and <i>control</i> is PTR_TP_NORMAL.)

JPOS\_E\_EXTENDED *ErrorCodeExtended* = JPOS\_EPTR\_COVER\_OPEN:  
The printer cover is open.

*ErrorCodeExtended* = JPOS\_EPTR\_JRN\_EMPTY:  
The journal station was specified but is out of paper.

*ErrorCodeExtended* =  
JPOS\_EPTR\_JRN\_CARTRIDGE\_REMOVED:  
A journal cartridge has been removed.  
(Can only apply if **AsyncMode** is false.)

*ErrorCodeExtended* =  
JPOS\_EPTR\_JRN\_CARTRIDGE\_EMPTY:  
A journal cartridge is empty.  
(Can only apply if **AsyncMode** is false.)

*ErrorCodeExtended* =  
JPOS\_EPTR\_JRN\_HEAD\_CLEANING:  
A journal cartridge head is being cleaned.  
(Can only apply if **AsyncMode** is false.)

*ErrorCodeExtended* =  
JPOS\_EPTR\_REC\_EMPTY:  
The receipt station was specified but is out of paper.  
(Can only apply if **AsyncMode** is false.)

*ErrorCodeExtended* =  
JPOS\_EPTR\_REC\_CARTRIDGE\_REMOVED:  
A receipt cartridge has been removed.  
(Can only apply if **AsyncMode** is false.)

*ErrorCodeExtended* =  
JPOS\_EPTR\_REC\_CARTRIDGE\_EMPTY:  
A receipt cartridge is empty.  
(Can only apply if **AsyncMode** is false.)

*ErrorCodeExtended* =  
JPOS\_EPTR\_REC\_HEAD\_CLEANING:  
A receipt cartridge head is being cleaned.  
(Can only apply if **AsyncMode** is false.)

*ErrorCodeExtended* = JPOS\_EPTR\_SLP\_EMPTY:  
The slip station was specified, but a form is not inserted.  
(Can only apply if **AsyncMode** is false.)

*ErrorCodeExtended* =  
JPOS\_EPTR\_SLP\_CARTRIDGE\_REMOVED:  
A slip cartridge has been removed.  
(Can only apply if **AsyncMode** is false.)

*ErrorCodeExtended* =  
JPOS\_EPTR\_SLP\_CARTRIDGE\_EMPTY:  
A slip cartridge is empty.  
(Can only apply if **AsyncMode** is false.)

*ErrorCodeExtended* =  
 JPOS\_EPTR\_SLP\_HEAD\_CLEANING:  
 A slip cartridge head is being cleaned.  
 (Can only apply if **AsyncMode** is false.)

## validateData Method

**Syntax**      **void validateData (int *station*, String *data*) throws JposException;**

Parameter	Description
<i>station</i>	The printer station to be used. May be PTR_S_JOURNAL, PTR_S_RECEIPT, or PTR_S_SLIP.
<i>data</i>	The data to be validated. May include printable data and escape sequences.

**Remarks**      Determines whether a data sequence, possibly including one or more escape sequences, is valid for the specified station, before calling the **printImmediate**, **printNormal**, or **printTwoNormal** methods.

This method does not cause any printing, but is used to determine the capabilities of the station.

**Errors**      A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_ILLEGAL	Some of the data is not precisely supported by the printer station, but the Device Service can select valid alternatives.
JPOS_E_FAILURE	Some of the data is not supported. No alternatives can be selected.

Cases which cause *ErrorCode* of JPOS\_E\_ILLEGAL:

Escape Sequence	Condition
Paper cut	The percentage ‘#’ is not precisely supported: Device Service will select the closest supported value.
Feed and Paper cut	The percentage ‘#’ is not precisely supported: Device Service will select the closest supported value.
Feed, Paper cut, and Stamp	The percentage ‘#’ is not precisely supported: Device Service will select the closest supported value.



---

Feed units	The unit count '#' is not precisely supported: Device Service will select the closest supported value.
Feed reverse	The line count '#' is too large: Device Service will select the maximum supported value.
Underline	The thickness '#' is not precisely supported: Device Service will select the closest supported value.
Shading	The percentage '#' is not precisely supported: Device Service will select the closest supported value.
Scale horizontally	The scaling factor '#' is not supported: Device Service will select the closest supported value.
Scale vertically	The scaling factor '#' is not supported: Device Service will select the closest supported value.

**Data****Condition**

---

*data1***CR***data2***LF** (Where **CR** is a Carriage Return and **LF** is a Line Feed)  
In order to print data *data1* and remain on the same line, the Device Service will print with a line advance, then perform a reverse line feed. The data *data2* will then overprint *data1*.

Cases which will cause Error Code of JPOS\_E\_FAILURE:

Escape Sequence	Condition
(General)	The escape sequence format is not valid.
Paper cut	Not supported.
Feed and Paper cut	Not supported.
Feed, Paper cut, and Stamp	Not supported.
Fire stamp	Not supported.
Print bitmap	Bitmap printing is not supported, or the bitmap number '#' is out of range.
Feed reverse	Not supported.
Font typeface	The typeface '#' is not supported.
Bold	Not supported.
Underline	Not supported.
Italic	Not supported.
Reverse video	Not supported.
Shading	Not supported.
Single high & wide	Not supported.
Double wide	Not supported.
Double high	Not supported.
Double high & wide	Not supported.
Alternate color	Not supported
RGB color	Not supported
Data	Condition
<i>data1</i> <b>CR</b> <i>data2</i> <b>LF</b>	(Where <b>CR</b> is a Carriage Return and <b>LF</b> is a Line Feed) Not able to print data and remain on the same line. The data <i>data1</i> will print on one line, and the data <i>data2</i> will print on the next line.

**See Also** "Data Characters and Escape Sequences"

## Events

### DirectIOEvent

**Interface**    **jpos.events.DirectIOListener**

**Method**      **directIOOccurred (DirectIOEvent e);**

**Description**    Provides Device Service information directly to the application. This event provides a means for a vendor-specific POS Printer Device Service to provide events to the application that are not otherwise supported by the Device Control.

**Properties**      This event contains the following properties:

<b>Property</b>	<b>Type</b>	<b>Description</b>
<i>EventNumber</i>	<i>int</i>	Event number whose specific values are assigned by the Device Service.
<i>Data</i>	<i>int</i>	Additional numeric data. Specific values vary by the <i>EventNumber</i> and the Device Service. This property is settable.
<i>Object</i>	<i>Object</i>	Additional data whose usage varies by the <i>EventNumber</i> and Device Service. This property is settable.

**Remarks**      This event is to be used only for those types of vendor specific functions that are not otherwise described as part of the JavaPOS standard. Use of this event may restrict the application program from being used with other vendor's POS Printer devices which may not have any knowledge of the Device Service's need for this event.

**See Also**      "Events" on page 18, **directIO** Method

**ErrorEvent**

<b>Interface</b>	<b>jpos.events.ErrorListener</b>
<b>Method</b>	<b>errorOccurred (ErrorEvent e);</b>
<b>Description</b>	Notifies the application that a printer error has been detected and a suitable response by the application is necessary to process the error condition.
<b>Properties</b>	This event contains the following properties:

<b>Property</b>	<b>Type</b>	<b>Description</b>
<i>ErrorCode</i>	<i>int</i>	Error Code causing the error event. See list of <i>ErrorCodes</i> on page 16.
<i>ErrorCodeExtended</i>	<i>int</i>	Extended Error Code causing the error event. If <i>ErrorCode</i> is JPOS_E_EXTENDED, then see values below. Otherwise, it may contain a Service-specific value.
<i>ErrorLocus</i>	<i>int</i>	Location of the error, and is set to JPOS_EL_OUTPUT indicating the error occurred while processing asynchronous output.
<i>ErrorResponse</i>	<i>int</i>	Error response, whose default value may be overridden by the application (i.e., this property is settable). See values below.

If *ErrorCode* is JPOS\_E\_EXTENDED, then *ErrorCodeExtended* has one of the following values:

<b>Value</b>	<b>Meaning</b>
JPOS_EPTR_COVER_OPEN	The printer cover is open.
JPOS_EPTR_JRN_EMPTY	The journal station is out of paper.
JPOS_EPTR_REC_EMPTY	The receipt station is out of paper.
JPOS_EPTR_SLP_EMPTY	A form is not inserted in the slip station.
JPOS_EPTR_JRN_CARTRIDGE_REMOVED:	A journal cartridge has been removed.
JPOS_EPTR_JRN_CARTRIDGE_EMPTY:	A journal cartridge is empty.
JPOS_EPTR_JRN_HEAD_CLEANING:	A journal cartridge head is being cleaned.
JPOS_EPTR_REC_CARTRIDGE_REMOVED:	A receipt cartridge has been removed.

JPOS\_EPTR\_REC\_CARTRIDGE\_EMPTY:

A receipt cartridge is empty.

JPOS\_EPTR\_REC\_HEAD\_CLEANNING:

A receipt cartridge head is being cleaned.

JPOS\_EPTR\_SLP\_CARTRIDGE\_REMOVED:

A slip cartridge has been removed.

JPOS\_EPTR\_SLP\_CARTRIDGE\_EMPTY:

A slip cartridge is empty.

JPOS\_EPTR\_SLP\_HEAD\_CLEANNING:

A slip cartridge head is being cleaned.

The application's error event listener may change *ErrorResponse* to one of the following values:

Value	Meaning
JPOS_ER_RETRY	Retry the asynchronous output. The error state is exited. The default.
JPOS_ER_CLEAR	Clear the asynchronous output or buffered output data. The error state is exited.

**Remarks** Enqueued when an error is detected and the Control's State transitions into the error state. This event is not delivered until **DataEventEnabled** is true, so that proper application sequencing occurs.

**See Also** "Device Output Models" on page 25, "Device States" on page 30

## OutputCompleteEvent

**Interface** `jpos.events.OutputCompleteListener`

**Method** `outputCompleteOccurred (OutputCompleteEvent e);`

**Description** Notifies the application that the queued output request associated with the *OutputID* property has completed successfully.

**Properties** This event contains the following property:

Property	Type	Description
<i>OutputID</i>	<i>int</i>	The ID number of the asynchronous output request that is complete.

**Remarks** This event is enqueued after the request's data has been both sent and the Device Service has confirmation that it was processed by the device successfully.

**See Also** "Device Output Models" on page 25

## StatusUpdateEvent

**Interface** `jpos.events.StatusUpdateListener`

**Method** `statusUpdateOccurred (StatusUpdateEvent e);`

**Description** Notifies the application that a printer has had an operation status change.

**Properties** This event contains the following property:

Property	Type	Description
<i>Status</i>	<i>int</i>	Indicates the status change, and has one of the following values:
Value	Meaning	
PTR_SUE_COVER_OPEN	Printer cover is open.	
PTR_SUE_COVER_OK	Printer cover is closed.	
PTR_SUE_JRN_EMPTY	No journal paper.	
PTR_SUE_JRN_NEAREMPTY	Journal paper is low.	
PTR_SUE_JRN_PAPEROK	Journal paper is ready.	
PTR_SUE_REC_EMPTY	No receipt paper.	
PTR_SUE_REC_NEAREMPTY	Receipt paper is low.	
PTR_SUE_REC_PAPEROK	Receipt paper is ready.	
PTR_SUE_SLP_EMPTY	No slip form.	
PTR_SUE_SLP_NEAREMPTY	Almost at the bottom of the slip form.	
PTR_SUE_SLP_PAPEROK	Slip form is inserted.	
PTR_SUE_IDLE	All asynchronous output has finished, either successfully or because output has been cleared. The printer State is now <code>JPOS_S_IDLE</code> . The <b>FlagWhenIdle</b> property must be true for this event to be delivered, and the property is automatically reset to false just before the event is delivered.	

*Note that Release 1.3* added Power State Reporting with additional *Power reporting StatusUpdateEvent values*. See “StatusUpdateEvent” description on page 78.

### ***Release 1.5 and later – Cartridge State Reporting***

If **CartridgeNotify** = PTR\_CN\_ENABLED, **StatusUpdateEvents** with the following *status* parameter values may be fired.

<b>Value</b>	<b>Meaning</b>
PTR_SUE_JRN_CARTRIDGE_EMPTY	A journal cartridge needs to be replaced. Cartridge is empty or not present.
PTR_SUE_JRN_HEAD_CLEANING	A journal cartridge has begun cleaning.
PTR_SUE_JRN_CARTRIDGE_NEAREMPTY	A journal cartridge is near end.
PTR_SUE_JRN_CARTRIDGE_OK	All journal cartridges are ready. It gives no indication of the amount of media in the cartridge.
PTR_SUE_REC_CARTRIDGE_EMPTY	A receipt cartridge needs to be replaced. Cartridge is empty or not present.
PTR_SUE_REC_HEAD_CLEANING	A receipt cartridge has begun cleaning.
PTR_SUE_REC_CARTRIDGE_NEAREMPTY	A receipt cartridge is near end.
PTR_SUE_REC_CARTRIDGE_OK	All receipt cartridges are ready. It gives no indication of the amount of media in the cartridge.
PTR_SUE_SLP_CARTRIDGE_EMPTY	A slip cartridge needs to be replaced. Cartridge is empty or not present.
PTR_SUE_SLP_HEAD_CLEANING	A slip cartridge has begun cleaning.
PTR_SUE_SLP_CARTRIDGE_NEAREMPTY	A slip cartridge is near end.
PTR_SUE_SLP_CARTRIDGE_OK	All slip cartridges are ready. It gives no indication of the amount of media in the cartridge.

**Remarks** Enqueued when a significant status event has occurred.

**See Also** “Events” on page 18





# Remote Order Display

## Summary

### Properties

<i>Common</i>	<i>Ver</i>	<i>Type</i>	<i>Access</i>	<i>May Use After</i>
<b>AutoDisable</b>	1.3	boolean	R/W	<i>Not Supported</i>
<b>CapPowerReporting</b>	1.3	int	R	open
<b>CheckHealthText</b>	1.3	String	R	open
<b>Claimed</b>	1.3	boolean	R	open
<b>DataCount</b>	1.3	int	R	open
<b>DataEventEnabled</b>	1.3	boolean	R/W	open
<b>DeviceEnabled</b>	1.3	boolean	R/W	open & claim
<b>FreezeEvents</b>	1.3	boolean	R/W	open
<b>OutputID</b>	1.3	int	R	open
<b>PowerNotify</b>	1.3	int	R/W	open
<b>PowerState</b>	1.3	int	R	open
<b>State</b>	1.3	int	R	--
<b>DeviceControlDescription</b>	1.3	String	R	--
<b>DeviceControlVersion</b>	1.3	int	R	--
<b>DeviceServiceDescription</b>	1.3	String	R	open
<b>DeviceServiceVersion</b>	1.3	int	R	open
<b>PhysicalDeviceDescription</b>	1.3	String	R	open
<b>PhysicalDeviceName</b>	1.3	String	R	open

<i>Specific</i>	<i>Ver</i>	<i>Type</i>	<i>Access</i>	<i>May Use After</i>
<b>CapTransaction</b>	1.3	boolean	R	open
<b>AsyncMode</b>	1.3	boolean	R/W	open, claim, & enable
<b>EventType</b>	1.3	int	R/W	open
<b>SystemClocks</b>	1.3	int	R	open, claim & enable
<b>SystemVideoSaveBuffers</b>	1.3	int	R	open, claim, & enable
<b>Timeout</b>	1.3	int	R/W	open
<b>UnitsOnline</b>	1.3	int	R	open, claim, & enable
<b>CurrentUnitID</b>	1.3	int	R/W	open, claim, & enable
<b>CapSelectCharacterSet</b>	1.3	boolean	R	open, claim, & enable
<b>CapTone</b>	1.3	boolean	R	open, claim, & enable
<b>CapTouch</b>	1.3	boolean	R	open, claim, & enable
<b>AutoToneDuration</b>	1.3	int	R/W	open, claim, & enable
<b>AutoToneFrequency</b>	1.3	int	R/W	open, claim, & enable
<b>CharacterSet</b>	1.3	int	R	open, claim, & enable
<b>CharacterSetList</b>	1.3	String	R	open, claim, & enable
<b>Clocks</b>	1.3	int	R	open, claim, & enable
<b>VideoDataCount</b>	1.3	int	R	open, claim, & enable
<b>VideoMode</b>	1.3	int	R/W	open, claim, & enable
<b>VideoModesList</b>	1.3	String	R	open, claim, & enable
<b>VideoSaveBuffers</b>	1.3	int	R	open, claim, & enable
<b>ErrorUnits</b>	1.3	int	R	open
<b>ErrorString</b>	1.3	String	R	open
<b>EventUnitID</b>	1.3	int	R	open & claim
<b>EventUnits</b>	1.3	int	R	open & claim
<b>EventString</b>	1.3	String	R	open & claim

**Methods***Common*

	<i>Ver</i>	<i>May use after</i>
<b>open</b>	1.3	-
<b>close</b>	1.3	open
<b>claim</b>	1.3	open
<b>release</b>	1.3	open & claim
<b>checkHealth</b>	1.3	open, claim, & enable
<b>clearInput</b>	1.3	open & claim
<b>clearOutput</b>	1.3	open & claim
<b>directIO</b>	1.3	open

*Specific*

<b>controlClock</b>	1.3	open, claim, & enable
<b>controlCursor</b>	1.3	open, claim, & enable
<b>freeVideoRegion</b>	1.3	open, claim, & enable
<b>resetVideo</b>	1.3	open, claim, & enable
<b>selectCharacterSet</b>	1.3	open, claim, & enable
<b>setCursor</b>	1.3	open, claim, & enable
<b>clearVideo</b>	1.3	open, claim, & enable
<b>clearVideoRegion</b>	1.3	open, claim, & enable
<b>copyVideoRegion</b>	1.3	open, claim, & enable
<b>displayData</b>	1.3	open, claim, & enable
<b>drawBox</b>	1.3	open, claim, & enable
<b>restoreVideoRegion</b>	1.3	open, claim, & enable
<b>saveVideoRegion</b>	1.3	open, claim, & enable
<b>updateVideoRegionAttribute</b>	1.3	open, claim, & enable
<b>videoSound</b>	1.3	open, claim, & enable
<b>transactionDisplay</b>	1.3	open, claim, & enable

**Events**

<i>Name</i>	<i>Ver</i>	<i>May Occur After</i>
<b>DataEvent</b>	1.3	open, claim, & enable
<b>DirectIOEvent</b>	1.3	open & claim
<b>ErrorEvent</b>	1.3	open, claim, & enable
<b>OutputCompleteEvent</b>	1.3	open, claim, & enable
<b>StatusUpdateEvent</b>	1.3	open, claim, & enable

## General Information

The Remote Order Display Control's class name is "jpos.RemoteOrderDisplay". The device constants are contained in the class "jpos.RemoteOrderDisplayConst". See "Package Structure" on page 40.

*This device was added in JavaPOS Release 1.3.*

### Capabilities

The Remote Order Display has the following minimal set of capabilities:

- Supports color or monochrome text character displays.
- Supports 8 foreground colors (or gray scale on monochrome display) with the option of using the intensity attribute.
- Supports 8 background colors (or gray scale on monochrome display) with the option of using only a blinking attribute.
- The individual event types support disabling such that the application only receives a subset of data events if requested.
- Supports video region buffering.
- Supports cursor functions.
- Supports clock functions.
- Supports resetting a video unit to power on state.

The Remote Order Display may also have the following additional capabilities:

- Supports multiple video displays each with possibly different video modes.
- Supports touch video input for a touch screen display unit.
- Supports video enunciator output with frequency and duration.
- Supports tactile feedback via an automatic tone when a video display unit is touched (for touch screen only).
- Supports downloading alternate character sets to one or many video units.
- Supports transaction mode display output to one or many video units.

The following capability is not supported:

- Support for graphical displays, where the video display is addressable by individual pixels or dots. The addition of this support is under investigation for future revisions.

## Model

The general model of a Remote Order Display:

The Remote Order Display device class is a subsystem of video units. The initial targeted environment is food service, to display order preparation and fulfillment information. Remote Order Displays are often used in conjunction with Bump Bars.

The general model of a Remote Order Display bar is an output device but may also be an input device when, in some implementations, the device can report additional status or user input data back to the application program.

- The subsystem can support up to 32 video units.

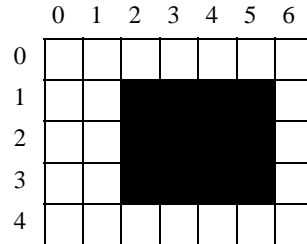
Typically, one application on one workstation (or POS Terminal) would manage and control the entire subsystem of Remote Order Displays. However, if applications on the same or other workstations (or POS Terminals) would need to access the subsystem, then one of the applications must act as a subsystem server and expose the necessary interfaces to other applications.

- All specific methods are broadcast methods. This means that the method can apply to one unit, a selection of units or all online units. The *units* parameter is an **int**, with each bit identifying an individual video unit. The Device Service will attempt to satisfy the method for all units indicated in the *units* parameter. If an error is received from one or more units, the **ErrorUnits** property is updated with the appropriate units in error. The **ErrorString** property is updated with a description of the error or errors received. The method will then throw a `JposException`. In the case where two or more units encounter different errors, the exception's `errorCode` will indicate the more severe error.
- The common methods **checkHealth**, **clearInput**, and **clearOutput** are not broadcast methods and use the unit ID indicated in the **CurrentUnitID** property. See the description of these common methods to understand how the current unit ID property is used.
- When the current unit ID property is set by the application, all the corresponding properties are updated to reflect the settings for that unit.

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

The **CurrentUnitID** uniquely represent a single video unit. The definitions range from `ROD_UID_1` to `ROD_UID_32`. These definitions are also used to create the bitwise parameter, *units*, used in the broadcast methods.

- The rows and columns are numbered beginning with (0,0) at the top-left corner of the video display. The dimensions are defined by the height and width parameters. The region depicted below would have the parameters *row = 1, column = 2, height = 3, and width = 4*.



All position parameters are expressed in text characters.

- The VGA-like *attribute* parameter, that is used in various methods, is an **int**. Bits 7-0 define the text attribute and bits 31-8 are reserved and must be 0, otherwise a JPOS\_E\_ILLEGAL. The following table defines bits 7-0:

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
Blinking	Background and Color			Intensity	Foreground Color		

If a foreground or background color is requested, but the Device Service does not support that color, it chooses the best fit from the colors supported.

The following constants may be used, with up to one constant selected from each category:

- Blinking: ROD\_ATTR\_BLINK
- Background Color: ROD\_ATTR\_BG\_color, where *color* is replaced by BLACK, BLUE, GREEN, CYAN, RED, MAGENTA, BROWN, or GRAY
- Intensity: ROD\_ATTR\_INTENSITY
- Foreground Color: ROD\_ATTR\_FG\_color, where *color* is replaced by BLACK, BLUE, GREEN, CYAN, RED, MAGENTA, BROWN, or GRAY

For touch video input, the Remote Order Display Control follows the general “Input Model” for event-driven input with some differences:

- When input is received a **DataEvent** is enqueued.
- This device does not support the **AutoDisable** property, so will not automatically disable itself when a **DataEvent** is enqueued.
- An enqueued **DataEvent** is delivered to the application when the **DataEventEnabled** property is true and other event delivery requirements are met. Just before delivering this event, data is copied into the properties, and further data events are disabled by setting the **DataEventEnabled** property to false. This causes subsequent input data to be enqueued while the application processes the current input and associated properties. When the application has finished the current input and is ready for more data, it reenables events by setting **DataEventEnabled** to true.
- An **ErrorEvent** is enqueued if an error occurs while gathering or processing input, and is delivered to the application when the **DataEventEnabled** property is true and other event delivery requirements are met.
- The **VideoDataCount** property may be read to obtain the number of video **DataEvents** for a specific unit ID enqueued. The **DataCount** property can be read to obtain the total number of data events enqueued.
- Input enqueued may be deleted by calling the **clearInput** method. See **clearInput** method description for more details.

For video and tone output, the Remote Order Display follows the general Output Model, with some enhancements:

- The following methods are always performed synchronously: **controlClock**, **controlCursor**, **selectCharacterSet**, **resetVideo**, and **setCursor**. These methods will fail if asynchronous output is outstanding. The following method is also always performed synchronously but without regard to outstanding asynchronous output: **freeVideoRegion**.
- The following methods are performed either synchronously or asynchronously, depending on the value of the **AsyncMode** property: **clearVideo**, **clearVideoRegion**, **copyVideoRegion**, **displayData**, **drawBox**, **restoreVideoRegion**, **saveVideoRegion**, **transactionDisplay**, **updateVideoRegionAttribute**, and **videoSound**. When **AsyncMode** is false, then these methods operate synchronously.

When **AsyncMode** is true, then these methods operate as follows:

- The request is buffered, the **OutputID** property is set to an identifier for this request, and returns as soon as possible. When the device completes the request successfully, then the **EventUnits** property is updated and an **OutputCompleteEvent** is enqueued. A property of this event contains the output ID of the completed request.

Asynchronous methods will not throw a **JposException** due to a display problem, such as communications failure. These errors will only be reported by an **ErrorEvent**. A **JposException** is thrown only if the display is not claimed and enabled, a parameter is invalid, or the request cannot be enqueued. The first two error cases are due to an application



error, while the last is a serious system resource exception.

- If an error occurs while performing an asynchronous request, an **ErrorEvent** is enqueued. The **EventUnits** property is set to the unit or units in error. The **EventString** property is also set.  
*Note: **ErrorEvent** updates **EventUnits** and **EventString**. If an error is reported by a synchronous broadcast method, then **ErrorUnits** and **ErrorString** are set instead.*

The event handler may call synchronous display methods (but not asynchronous methods), then can either retry the outstanding output or clear it.

- Asynchronous output is performed on a first-in first-out basis.
- All unit output buffered may be deleted by setting the **CurrentUnitID** property and calling the **clearOutput** method. **OutputCompleteEvents** will not be delivered for cleared output. This method also stops any output that may be in progress (when possible).

When **AsyncMode** is false, then these methods operate synchronously and the Device returns to the application after completion. When operating synchronously, a **JposException** is thrown if the method could not complete successfully.

- The Remote Order Display device may support transaction mode. A transaction is a sequence of display operations that are sent to a video unit as a single unit. Display operations which may be included in a transaction are **clearVideo**, **clearVideoRegion**, **copyVideoRegion**, **displayData**, **drawBox**, **restoreVideoRegion**, **saveVideoRegion**, and **updateVideoRegionAttribute**. During a transaction, the display operations are first validated. If valid, they are added to the transaction but not displayed yet. Once the application has added as many operations as required, then the transaction display method is called.

If the transaction is displayed synchronously, then any exception thrown indicates that an error occurred during the display. If the transaction is displayed asynchronously, then the asynchronous display rules listed above are followed. If an error occurs and the Error Event handler causes a retry, the entire transaction is retried.

## Device Sharing

The Remote Order Display is an exclusive-use device. Its device sharing rules are:

- The application must claim the device before enabling it.
- The application must claim and enable the device before accessing many Remote Order Display specific properties.
- The application must claim and enable the device before calling methods that manipulate the device.
- When a **claim** method is called again, settable device characteristics are restored to their condition at **release**. Examples of restored characteristics are character set, video mode, and tone frequency. Region memory buffers, clock and cursor settings are considered state characteristics and are not restored.
- See the “Summary” table for precise usage prerequisites.

## Properties

### AsyncMode Property R/W

Type	boolean
Remarks	<p>If true, then the <b>clearVideo</b>, <b>clearVideoRegion</b>, <b>copyVideoRegion</b>, <b>displayData</b>, <b>drawBox</b>, <b>restoreVideoRegion</b>, <b>saveVideoRegion</b>, <b>transactionDisplay</b>, <b>updateVideoRegionAttribute</b>, and <b>videoSound</b> methods will be performed asynchronously. If false, they will be performed synchronously.</p> <p>This property is initialized to false by the <b>open</b> method.</p>
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### AutoToneDuration Property R/W

Type	int				
Remarks	<p>Holds the duration (in milliseconds) of the automatic tone for the video unit indicated in the <b>CurrentUnitID</b> property.</p> <p>This property is initialized to the default value for each online video unit when the device is first enabled following the <b>open</b> method.</p>				
Errors	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p> <p>Some possible values of the exception’s <i>ErrorCode</i> property are:</p> <table border="1"> <thead> <tr> <th colspan="2">ValueMeaning</th> </tr> </thead> <tbody> <tr> <td>JPOS_E_ILLEGAL</td> <td>An illegal value was specified. The <b>ErrorString</b> property is updated.</td> </tr> </tbody> </table>	ValueMeaning		JPOS_E_ILLEGAL	An illegal value was specified. The <b>ErrorString</b> property is updated.
ValueMeaning					
JPOS_E_ILLEGAL	An illegal value was specified. The <b>ErrorString</b> property is updated.				
See Also	<b>CurrentUnitID</b> Property				

**AutoToneFrequency Property R/W**

<b>Type</b>	<b>boolean</b>				
<b>Remarks</b>	<p>Holds the frequency (in Hertz) of the automatic tone for the video unit indicated in the <b>CurrentUnitID</b> property.</p> <p>This property is initialized to the default value for each online video unit when the device is first enabled following the <b>open</b> method.</p>				
<b>Errors</b>	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p> <p>Some possible values of the exception’s <i>ErrorCode</i> property are:</p> <table border="1"> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>JPOS_E_ILLEGAL</td> <td>An illegal value was specified. The <b>ErrorString</b> property is updated.</td> </tr> </tbody> </table>	Value	Meaning	JPOS_E_ILLEGAL	An illegal value was specified. The <b>ErrorString</b> property is updated.
Value	Meaning				
JPOS_E_ILLEGAL	An illegal value was specified. The <b>ErrorString</b> property is updated.				
<b>See Also</b>	<b>CurrentUnitID</b> Property				

**CapSelectCharacterSet Property R**

<b>Type</b>	<b>boolean</b>
<b>Remarks</b>	<p>If true, the video unit indicated in the <b>CurrentUnitID</b> property may be loaded with an alternate, user supplied character set.</p> <p>This property is initialized for each video unit online when the device is first enabled following the <b>open</b> method.</p>
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
<b>See Also</b>	<b>CurrentUnitID</b> Property

**CapTone Property R**

<b>Type</b>	<b>boolean</b>
<b>Remarks</b>	<p>If true, the video unit indicated in the <b>CurrentUnitID</b> property supports an enunciator.</p> <p>This property is initialized for each video unit online when the device is first enabled following the <b>open</b> method.</p>
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
<b>See Also</b>	<b>CurrentUnitID</b> Property

## CapTouch Property R

<b>Type</b>	<b>boolean</b>
<b>Remarks</b>	<p>If true, the video unit indicated in the <b>CurrentUnitID</b> property supports the ROD_DE_TOUCH_UP, ROD_DE_TOUCH_DOWN, and ROD_DE_TOUCH_MOVE event types.</p> <p>This property is initialized for each video unit online when the device is first enabled following the <b>open</b> method.</p>
<b>Errors</b>	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>
<b>See Also</b>	<b>CurrentUnitID</b> Property, <b>DataEvent</b>

## CapTransaction Property R

<b>Type</b>	<b>boolean</b>
<b>Remarks</b>	<p>If true, then transactions are supported by each video unit.</p> <p>This property is initialized by the <b>open</b> method.</p>
<b>Errors</b>	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>

## CharacterSet Property R

**Type**      **int**

**Remarks**      Holds the character set for displaying characters for the video unit indicated by the **CurrentUnitID** property. When **CapSelectCharacterSet** is true, this property can be set with one of the following values:

Value	Meaning
Range 101 - 199	A device-specific character set that does not match a code page, nor the ASCII or ANSI character sets.
Range 400 - 990	Code page; matches one of the standard values.
ROD_CS_UNICODE	The character set supports UNICODE. The value of this constant is 997.
ROD_CS_ASCII	The ASCII character set, supporting the ASCII characters 0x20 through 0x7F. The value of this constant is 998.
ROD_CS_ANSI	The ANSI character set. The value of this constant is 999.

This property is initialized to the default video character set used by each video unit online when the device is first enabled following the **open** method.

This is updated during the **selectCharacterSet** method.

**Errors**      A `JposException` may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

**See Also**      **CurrentUnitID** Property, **CharacterSetList** Property, **CapSelectCharacterSet** Property, **selectCharacterSet** Method

## CharacterSetList Property R

**Type**      **String**

**Remarks**      Holds a string of character set numbers for the video unit indicated in the **CurrentUnitID** property.

If **CapSelectCharacterSet** is true, this property is initialized for each video unit online when the device is first enabled following the **open** method.

The character set number string consists of an ASCII numeric set of numbers, separated by commas.

For example, if the string is “101, 850, 999”, the video unit supports a device-specific character set, code page 850, and the ANSI character set.

**Errors**      A `JposException` may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

**See Also** **CurrentUnitID** Property, **CharacterSet** Property, **CapSelectCharacterSet** Property, **selectCharacterSet** Method

## Clocks Property R

**Type** **int**

**Remarks** Holds the number of clocks the video unit, indicated in the **CurrentUnitID** property, can support.

This property is initialized for each online video unit when the device is first enabled following the **open** method.

**Errors** A **JposException** may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

**See Also** **CurrentUnitID** Property

## CurrentUnitID Property R/W

**Type** **int**

**Remarks** Holds the current video unit ID. Up to 32 units are allowed on one Remote Order Display device. The unit ID definitions range from **ROD\_UID\_1** to **ROD\_UID\_32**.

The following properties and methods apply only to the selected video unit ID:

- Properties: **AutoToneDuration**, **AutoToneFrequency**, **CapSelectCharacterSet**, **CapTone**, **CapTouch**, **CharacterSet**, **CharacterSetList**, **Clocks**, **VideoDataCount**, **VideoMode**, **VideoModesList**, **VideoSaveBuffers**.

Setting **CurrentUnitID** will update these properties to the current values for the specified unit.

Methods: **checkHealth**, **clearInput**, **clearOutput**.

This property is initialized to **ROD\_UID\_1** when the device is first enabled following the **open** method.

**Errors** A **JposException** may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_ILLEGAL	An illegal unit id was specified. The <b>ErrorString</b> property is updated.

## DataCount Property (Common) R

<b>Type</b>	<b>int</b>
<b>Remarks</b>	<p>Holds the total number of <b>DataEvents</b> enqueued. All units online are included in this value. The number of enqueued events for a specific unit ID is stored in the <b>VideoDataCount</b> property.</p> <p>The application may read this property to determine whether additional input is enqueued from a device, but has not yet been delivered because of other application processing, freezing of events, or other causes.</p> <p>This property is initialized to zero by the <b>open</b> method.</p>
<b>Errors</b>	A <b>JposException</b> may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
<b>See Also</b>	“Device Input Model” on page 22, <b>VideoDataCount</b> Property, <b>DataEvent</b>

## ErrorString Property R

<b>Type</b>	<b>String</b>
<b>Remarks</b>	<p>Holds a description of the error which occurred to the unit(s) specified by the <b>ErrorUnits</b> property, when an error occurs for any method that acts on a bitwise set of video units.</p> <p>If an error occurs during processing of an asynchronous request, the <b>ErrorEvent</b> updates the property <b>EventString</b> instead.</p> <p>This property is initialized to an empty string by the <b>open</b> method.</p>
<b>Errors</b>	A <b>JposException</b> may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
<b>See Also</b>	<b>ErrorUnits</b> Property

## ErrorUnits Property R

<b>Type</b>	<b>int</b>
<b>Remarks</b>	<p>Holds a bitwise mask of the unit(s) that encountered an error, when an error occurs for any method that acts on a bitwise set of video units.</p> <p>If an error occurs during processing of an asynchronous request, the <b>ErrorEvent</b> updates the property <b>EventUnits</b> instead.</p> <p>This property is initialized to zero by the <b>open</b> method.</p>
<b>Errors</b>	A <b>JposException</b> may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
<b>See Also</b>	<b>ErrorString</b> Property



## EventString Property R

Type	String
Remarks	<p>Holds a description of the error which occurred to the unit(s) specified by the <b>EventUnits</b> property, when an <b>ErrorEvent</b> is delivered.</p> <p>This property is initialized to an empty string by the <b>open</b> method.</p>
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>EventUnits</b> Property, <b>ErrorEvent</b>

## EventType Property R/W

Type	int				
Remarks	<p>Holds a bitwise mask that is used to selectively indicate which event types are to be delivered by the <b>DataEvent</b>, for all video units online. See the <b>DataEvent</b> description for event type definitions.</p> <p>This property is initialized to all defined event types by the <b>open</b> method.</p>				
Errors	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p> <p>Some possible values of the exception’s <i>ErrorCode</i> property are:</p> <table border="1"> <thead> <tr> <th>Value</th> <th>Meaning</th> </tr> </thead> <tbody> <tr> <td>JPOS_E_ILLEGAL</td> <td>An illegal unit id was specified. The <b>ErrorString</b> property is updated.</td> </tr> </tbody> </table>	Value	Meaning	JPOS_E_ILLEGAL	An illegal unit id was specified. The <b>ErrorString</b> property is updated.
Value	Meaning				
JPOS_E_ILLEGAL	An illegal unit id was specified. The <b>ErrorString</b> property is updated.				
See Also	<b>DataEvent</b>				

## EventUnitID Property R

Type	int
Remarks	Holds the video unit ID of the last delivered <b>DataEvent</b> . The unit ID definitions range from BB_UID_1 to BB_UID_32.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>DataEvent</b>

## EventUnits Property R

Type	int
Remarks	Holds a bitwise mask of the unit(s) when an <b>OutputCompleteEvent</b> , output <b>ErrorEvent</b> , or <b>StatusUpdateEvent</b> is delivered.  This property is initialized to zero by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>OutputCompleteEvent</b> , <b>ErrorEvent</b> , <b>StatusUpdateEvent</b>

## SystemClocks Property R

Type	int
Remarks	Holds the total number of clocks the Remote Order Display device can support at one time.  This property is initialized when the device is first enabled following the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>Clocks</b> Property

## SystemVideoSaveBuffers Property R

Type	int
Remarks	Holds the total number of video save buffers the Remote Order Display device can support at one time.  This property is initialized when the device is first enabled following the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>VideoSaveBuffers</b> Property

## Timeout Property R/W

<b>Type</b>	<b>int</b>				
<b>Remarks</b>	<p>Holds the timeout value in milliseconds used by the Remote Order Display device to complete all output methods supported. If the device cannot successfully complete an output method within the timeout value, then the method throws a <code>JposException</code> if <b>AsyncMode</b> is false, or enqueues an <b>ErrorEvent</b> if <b>AsyncMode</b> is true.</p> <p>This property is initialized to a Device Service dependent default timeout following the <b>open</b> method.</p>				
<b>Errors</b>	<p>A <code>JposException</code> may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p> <p>Some possible values of the exception’s <i>ErrorCode</i> property are:</p> <table> <thead> <tr> <th><b>Value</b></th> <th><b>Meaning</b></th> </tr> </thead> <tbody> <tr> <td><code>JPOS_E_ILLEGAL</code></td> <td>An illegal unit id was specified. The <b>ErrorString</b> property is updated.</td> </tr> </tbody> </table>	<b>Value</b>	<b>Meaning</b>	<code>JPOS_E_ILLEGAL</code>	An illegal unit id was specified. The <b>ErrorString</b> property is updated.
<b>Value</b>	<b>Meaning</b>				
<code>JPOS_E_ILLEGAL</code>	An illegal unit id was specified. The <b>ErrorString</b> property is updated.				
<b>See Also</b>	<b>AsyncMode</b> Property				

## UnitsOnline Property R

<b>Type</b>	<b>int</b>
<b>Remarks</b>	<p>Holds a bitwise mask indicating the video units online. Bit 0 is <code>ROD_UID_1</code>. 32 video units are supported. See <i>Model</i> on page 620.</p> <p>This property is initialized when the device is first enabled following the <b>open</b> method. This property is updated as changes are detected, such as before a <b>StatusUpdateEvent</b> is enqueued and during the <b>checkHealth</b> method.</p>
<b>Errors</b>	A <code>JposException</code> may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
<b>See Also</b>	“Model” on page 620, <b>checkHealth</b> Method, <b>StatusUpdateEvent</b>

## VideoDataCount Property R

<b>Type</b>	<b>int</b>
<b>Remarks</b>	<p>Holds the number of <b>DataEvents</b> enqueued for the video unit indicated in the <b>CurrentUnitID</b> property.</p> <p>The application may read this property to determine whether additional input is enqueued a video unit, but has not yet been delivered because of other application processing, freeing of events, or other causes.</p> <p>This property is initialized to zero by the <b>open</b> method.</p>
<b>Errors</b>	A <b>JposException</b> may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
<b>See Also</b>	<b>CurrentUnitID</b> Property, <b>DataEvent</b>

## VideoMode Property R/W

<b>Type</b>	<b>int</b>
<b>Remarks</b>	<p>Holds the video <b>ModeId</b> selected for the video unit indicated by the <b>CurrentUnitID</b> property. The <b>ModeId</b> represents one of the selections in the <b>VideoModesList</b> property.</p> <p>This property is initialized to the Device Service dependent default video <b>ModeId</b> used by each video unit online when the device is first enabled following the <b>open</b> method.</p>
<b>Errors</b>	<p>A <b>JposException</b> may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p> <p>Some possible values of the exception’s <i>ErrorCode</i> property are:</p>
<b>ValueMeaning</b>	
JPOS_E_ILLEGAL	An illegal unit id was specified. The <b>ErrorString</b> property is updated.
JPOS_E_FAILURE	An error occurred while communicating with the video unit indicated in the <b>CurrentUnitID</b> property. The <b>ErrorString</b> property is updated.
<b>See Also</b>	<b>CurrentUnitID</b> Property, <b>VideoModesList</b> Property

## VideoModesList Property R

<b>Type</b>	<b>String</b>
<b>Remarks</b>	<p>Holds the video modes supported for the video unit indicated in the <b>CurrentUnitID</b> property. The video modes are listed in a comma delineated string with the following format:          &lt;ModeId&gt;:&lt;Height&gt;x&lt;Width&gt;x&lt;NumberOfColors&gt;&lt;M C&gt;.</p> <p>The ModeId values are determined by the Remote Order Display system. M = Monochrome (and gray scales) and C = Color.</p> <p>For example, if the string is “1:40x25x16C,2:80x25x16C”, then the video unit supports two video modes, ModeId 1 and ModeId 2. ModeId 1 has 40 rows, 25 columns, 16 colors, and is Color. ModeId 2 has 80 rows, 25 columns, 16 colors, and is Color.</p> <p>The ModeId is used to initialize the <b>VideoMode</b> property for each video unit online.</p> <p>This property is initialized to the video modes list supported by each video unit online when the device is first enabled following the <b>open</b> method.</p>
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
<b>See Also</b>	<b>CurrentUnitID</b> Property, <b>VideoMode</b> Property

## VideoSaveBuffers Property R

<b>Type</b>	<b>int</b>
<b>Remarks</b>	<p>Holds the number of save buffers for the video unit indicated in the <b>CurrentUnitID</b> property. This property should be consulted when using the <b>saveVideoRegion</b>, <b>restoreVideoRegion</b> and <b>freeVideoRegion</b> methods. When set to 0, this indicates that buffering for the selected unit is not supported. When this property is greater than 0, the Remote Order Display device can save at minimum one entire video screen for the selected video unit.</p> <p>This property is initialized for each video unit online when the device is first enabled following the <b>open</b> method.</p>
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
<b>See Also</b>	<b>CurrentUnitID</b> Property, <b>saveVideoRegion</b> Method, <b>restoreVideoRegion</b> Method, <b>freeVideoRegion</b> Method

## Methods

### checkHealth Method (Common)

**Syntax**      **void checkHealth (int level) throws JposException;**

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

Value	Meaning
JPOS_CH_INTERNAL	Perform a health check that does not physically change the device. The device is tested by internal tests to the extent possible.
JPOS_CH_EXTERNAL	Perform a more thorough test that may change the device. For example, a pattern may be displayed on the video.
JPOS_CH_INTERACTIVE	Perform an interactive test of the device. The Device Service will typically display a modal dialog box to present test options and results.

**Remarks**      When JPOS\_CH\_INTERNAL or JPOS\_CH\_EXTERNAL level is requested, the method checks the health of the unit indicated in the **CurrentUnitID** property. If the current unit ID property is zero, a JPOS\_EROD\_NOUNITS error is set. When the current unit ID property is set to a unit that is not currently online, the device will attempt to check the health of the video unit and report a communication error if necessary. The JPOS\_CH\_INTERACTIVE health check operation is up to the Device Service designer.

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

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

This method is always synchronous.

**Errors**      A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_EXTENDED	<b>ErrorCodeExtended</b> = JPOS_EROD_NOUNITS: The <b>CurrentUnitID</b> property is zero.
JPOS_E_FAILURE	An error occurred while communicating with the video unit indicated in <b>CurrentUnitID</b> property.

**See Also**      **CurrentUnitID** Property, **UnitsOnline** Property

**clearInput Method (Common)**

**Syntax**      **void clearInput () throws JposException;**

**Remarks**      Clears the device input that has been buffered for the unit indicated in the **CurrentUnitID** property. If the current unit ID property is zero, a JPOS\_EROD\_NOUNITS is set.

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

**Errors**          A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
JPOS_E_EXTENDED	<b>ErrorCodeExtended</b> = JPOS_EROD_NOUNITS: The <b>CurrentUnitID</b> property is zero.

**See Also**      **CurrentUnitID** Property, “Device Input Model” on page 22

**clearOutput Method (Common)**

**Syntax**      **void clearOutput () throws JposException;**

**Remarks**      Clears all outputs that have been buffered for the unit indicated in the **CurrentUnitID** property, including video and tone outputs. If the current unit ID property is zero, a JPOS\_EROD\_NOUNITS is set.

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

**Errors**          A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
JPOS_E_EXTENDED	<b>ErrorCodeExtended</b> = JPOS_EROD_NOUNITS: The <b>CurrentUnitID</b> property is set to zero.

**See Also**      **CurrentUnitID** Property, “Device Output Models” on page 25

## clearVideo Method

**Syntax**      **void clearVideo (int *units*, int *attribute*) throws JposException;**

<b>Parameter</b>	<b>Description</b>
<i>units</i>	Bitwise mask indicating which video unit(s) to operate on.
<i>attribute</i>	See Model on page 620 in the General Information section.

**Remarks**      Clears the entire display area for the video unit(s) indicated in the *units* parameter. The display area will be cleared using the attribute placed in the *attribute* parameter.

This method is performed synchronously if **AsyncMode** is false, and asynchronously if **AsyncMode** is true.

**Errors**          A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

**See Also**        **AsyncMode** Property, “Model” on page 620



**clearVideoRegion Method**

**Syntax**      **void clearVideoRegion (int units, int row, int column, int height, int width, int attribute) throws JposException;**

Parameter	Description
<i>units</i>	Bitwise mask indicating which video unit(s) to operate on.
<i>row</i>	The region's start row.
<i>column</i>	The region's start column.
<i>height</i>	The number of rows in the region.
<i>width</i>	The number of columns in the region.
<i>attribute</i>	See "Model" on page 620 in the General Information section.

**Remarks**      Clears the specified video region for the video unit(s) indicated in the *units* parameter. The display area will be cleared using the attribute placed in the *attribute* parameter.

This method is performed synchronously if **AsyncMode** is false, and asynchronously if **AsyncMode** is true.

**Errors**          A JposException may be thrown when this method is invoked. For further information, see "Exceptions" on page 15.

Some possible values of the exception's *ErrorCode* property are:

Value	Meaning
JPOS_E_FAILURE	An error occurred while communicating with one of the video units indicated in <i>units</i> . The <b>ErrorUnits</b> and <b>ErrorString</b> properties are updated. (Can only occur if <b>AsyncMode</b> is false.)

**See Also**      **AsyncMode** Property, **ErrorString** Property, **ErrorUnits** Property, "Model" on page 620

**controlClock Method**

**Syntax**      **void controlClock (int units, int function, int clockId, int hour, int min, int sec, int row, int column, int attribute, int mode) throws JposException;**

Parameter	Description
<i>units</i>	Bitwise mask indicating which video unit(s) to operate on.
<i>function</i>	The requested clock command. See values below.
<i>clockId</i>	Clock identification number. The valid values can be from 1 - <b>Clocks</b> . When the <i>function</i> parameter is ROD_CLK_PAUSE, ROD_CLK_RESUME, or ROD_CLK_STOP then <i>clockId</i> can be ROD_CLK_ALL to specify all clocks started on the specified video unit(s).
<i>hour</i>	The initial hours for the clock display.
<i>min</i>	The initial minutes for the clock display.
<i>sec</i>	The initial seconds for the clock display.
<i>row</i>	The clock's row.
<i>column</i>	The clock's start column.
<i>attribute</i>	See "Model" on page 620 in the General Information section.
<i>mode</i>	The type of clock to display. See values below.

The *function* parameter values are:

Value	Meaning
ROD_CLK_START	Starts a clock display assigned to the given <i>clockId</i> .
ROD_CLK_PAUSE	Temporarily stops a clock from updating the display until a ROD_CLK_RESUME requested.
ROD_CLK_RESUME	Resumes a clock that was previously paused, such that display updates continue.
ROD_CLK_STOP	Permanently stops the clock from updating the display and the <i>clockId</i> becomes free.
ROD_CLK_MOVE	Moves an instantiated clock to a new position.

The *mode* parameter values are:

Value	Meaning
ROD_CLK_SHORT	Displays a clock with “M:SS” format.
ROD_CLK_NORMAL	Displays a clock with “MM:SS” format.
ROD_CLK_12_int	Displays a 12 hour clock with “HH:MM:SS” format.
ROD_CLK_24_int	Displays a 24 hour clock with “HH:MM:SS” format.

**Remarks** Performs the clock command requested in the *function* parameter on the video unit(s) indicated in the *units* parameter. The clock will be displayed in the requested *mode* format at the location found in the *row* and *column* parameters.

The clock will start at the specified *hour*, *min*, and *sec*, time values and will be updated every second until a ROD\_CLK\_PAUSE or ROD\_CLK\_STOP is requested for this *clockId*.

When a ROD\_CLK\_PAUSE, ROD\_CLK\_RESUME, or ROD\_CLK\_STOP command is issued, the *hour*, *min*, *sec*, *row*, *column*, *attribute*, and *mode* parameters are ignored. During a ROD\_CLK\_PAUSE command, the clock display updates are suspended. During a ROD\_CLK\_RESUME command, the clock updates continue.

If a ROD\_CLK\_PAUSE, ROD\_CLK\_RESUME, ROD\_CLK\_STOP or ROD\_CLK\_MOVE command is requested on an uninitialized *clockId* for any of the video units indicated in the *units* parameter, a JPOS\_EROD\_BADCLK error is thrown. If a ROD\_CLK\_RESUME command is requested without doing a ROD\_CLK\_PAUSE, this has no effect and no exception is thrown.

When a ROD\_CLK\_MOVE command is issued, the clock is moved to the new location found in the *row* and *column* parameters. The *hour*, *min*, *sec*, *attribute* and *mode* parameters are ignored for this command function.

Generally a video unit can support the number of clocks indicated in the **Clocks** property. However, the ROD\_CLK\_START command will return JPOS\_EROD\_NOCLOCKS if it exceeds the number of **SystemClocks** even though the **Clocks** property may indicate the unit can support more clocks than allocated for that unit.

**Errors** A `JposException` may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s `ErrorCode` property are:

Value	Meaning
JPOS_E_EXTENDED	<p><b>ErrorCodeExtended</b> = JPOS_EROD_BADCLK: A ROD_CLK_PAUSE, ROD_CLK_RESUME, ROD_CLK_START, ROD_CLK_MOVE command was requested and the specified <code>clockId</code> has not been initialized by the ROD_CLK_START command.</p> <p><b>ErrorCodeExtended</b> = JPOS_EROD_NOCLOCKS: The ROD_CLK_START failed because the number of <b>SystemClocks</b> has been reached.</p> <p>The <b>ErrorUnits</b> and <b>ErrorString</b> properties are updated.</p>
JPOS_E_FAILURE	<p>An error occurred while communicating with one of the video units indicated in the <code>units</code> parameter. The <b>ErrorUnits</b> and <b>ErrorString</b> properties are updated.</p>
JPOS_E_BUSY	<p>When a ROD_CLK_START command is requested but the specified <code>clockId</code> is in use. The <b>ErrorUnits</b> and <b>ErrorString</b> properties are updated.</p>

**See Also** **Clocks** Property, **ErrorString** Property, **ErrorUnits** Property, “Model” on page 620.

**controlCursor Method**

**Syntax**      **void controlCursor (int *units*, int *function*) throws JposException;**

Parameter	Description
<i>units</i>	Bitwise mask indicating which video unit(s) to operate on.
<i>function</i>	The cursor command, indicating the type of cursor to display. See values below.

Value	Meaning
ROD_CRIS_LINE	enable a solid underscore line.
ROD_CRIS_LINE_BLINK	enable a blinking solid underscore cursor.
ROD_CRIS_BLOCK	enable a solid block cursor.
ROD_CRIS_BLOCK_BLINK	enable a blinking solid block cursor.
ROD_CRIS_OFF	Disable cursor.

**Remarks**      Enables or disables the cursor depending on the *function* parameter, for the video unit(s) indicated in the *units* parameter.

When the *function* is ROD\_CRIS\_OFF, the cursor is disabled, otherwise the cursor is enabled as the requested cursor type. If the video unit cannot support the requested cursor type, the Device Service will use the next closest cursor type.

The cursor attribute is taken from the current cursor location.

**Errors**      A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_FAILURE	An error occurred communicating with one of the video units indicated in <i>units</i> . The <b>ErrorUnits</b> and <b>ErrorString</b> properties are updated.

**See Also**      **ErrorString** Property, **ErrorUnits** Property





**drawBox Method**

**Syntax**      **void drawBox (int units, int row, int column, int height, int width, int attribute, int bordertype) throws JposException;**

Parameter	Description
<i>units</i>	Bitwise mask indicating which video unit(s) to operate on.
<i>row</i>	The box's start row.
<i>column</i>	The box's start column.
<i>height</i>	The number of rows in the box.
<i>width</i>	The number of columns in the box.
<i>attribute</i>	The video attribute. See "Model" on page 620 in the General Information section.
<i>bordertype</i>	The border type to be drawn. Can be any printable character or a defined border type. See values below.

Value	Meaning
ROD_BDR_SINGLE	A single line border.
ROD_BDR_DOUBLE	A double line border.
ROD_BDR_SOLID	A solid block border.

**Remarks**      Draws a box on the video units(s) indicated in the *units* parameter.

The Remote Order Display will attempt to draw a box with the border type specified. If the character set does not support the chosen border type, the Device Service will choose the best fit from the given character set.

This method is performed synchronously if **AsyncMode** is false, and asynchronously if **AsyncMode** is true.

**Errors**      A JposException may be thrown when this method is invoked. For further information, see "Exceptions" on page 15.

Some possible values of the exception's *ErrorCode* property are:

Value	Meaning
JPOS_E_FAILURE	An error occurred while communicating with one of the displays indicated in units. The <b>ErrorUnits</b> and <b>ErrorString</b> properties are updated.

**See Also**      **AsyncMode** Property, **ErrorString** Property, **ErrorUnits** Property, "Model" on page 620.



## freeVideoRegion Method

**Syntax**      **void freeVideoRegion (int units, int bufferId) throws JposException;**

Parameter	Description
<i>units</i>	Bitwise mask indicating which video unit(s) to operate on.
<i>bufferId</i>	Number identifying the video buffer to free. Valid values range from 1 to the <b>VideoSaveBuffers</b> property for a selected unit(s).

**Remarks**      Frees any buffer memory allocated for the video unit(s) indicated in the *units* parameter. The number of video buffers supported is stored in the **VideoSaveBuffers** property for each video unit online. If the *bufferId* was never used in a previous **saveVideoRegion** method, no action is taken.

**Errors**          A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_FAILURE	An error occurred communicating with one of the video units indicated in units. The <b>ErrorUnits</b> and <b>ErrorMessage</b> properties are updated.

**See Also**      **ErrorMessage** Property, **ErrorUnits** Property, **VideoSaveBuffers** Property, **saveVideoRegion** Method

## resetVideo Method

**Syntax**      **void resetVideo (int units) throws JposException;**

*units* is a bitwise mask indicating which video unit(s) to operate on.

**Remarks**      Sets the video unit(s) indicated in the *units* parameter to a power on state. All Device Service buffers and clocks associated with the unit(s) are released. All settable characteristics are set to default values.

**Errors**          A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_FAILURE	An error occurred while communicating with one of the video units indicated in units. The <b>ErrorUnits</b> and <b>ErrorMessage</b> properties are updated.

**See Also**      **ErrorMessage** Property, **ErrorUnits** Property

## restoreVideoRegion Method

**Syntax**      **void restoreVideoRegion (int units, int targetRow, int targetColumn, int bufferId) throws JposException;**

Parameter	Description
<i>units</i>	Bitwise mask indicating which video unit(s) to operate on.
<i>targetRow</i>	The start row of the target location.
<i>targetColumn</i>	The start column of the target location.
<i>bufferId</i>	Number identifying the source video buffer to use. Valid values range from 1 to the <b>VideoSaveBuffers</b> property for the selected unit(s).

**Remarks**      Restores a previously saved video region of the display area from the requested *bufferId* for the video unit(s) indicated in the *units* parameter. A region can be saved using the **saveVideoRegion** method. The number of video buffers supported is stored in the **VideoSaveBuffers** property for each video unit online. The target location is defined by the *targetRow* and *targetColumn* parameters. This method doesn't free the memory after restoring, therefore, this method can be used to copy a video region to multiple locations on the display. Use the **freeVideoRegion** method to free any memory allocated for a video buffer.

If the *bufferId* does not contain a previously saved video region for the *units* selected, a JPOS\_EROD\_NOREGION exception is thrown.

Video regions cannot be restored between video units. For example, the **saveVideoRegion** method is called with *units* = 0000 1000 and *bufferId* = 1. This will save a video region for the Unit Id 4, in to Buffer 1 for that unit. If this method is called with *units* = 0000 0100 and *bufferId* = 1 with the intention of restoring the previously saved buffer to Unit Id 3, then either a JposException with ErrorCode of JPOS\_EROD\_NOREGION would be thrown, or an unwanted region would be restored.

This method is performed synchronously if **AsyncMode** is false, and asynchronously if **AsyncMode** is true.

**Errors** A `JposException` may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s `ErrorCode` property are:

Value	Meaning
<code>JPOS_E_EXTENDED</code>	<b>ErrorCodeExtended</b> = <code>JPOS_EROD_NOREGION</code> : The <code>bufferId</code> does not contain a previously saved video region.
<code>JPOS_E_FAILURE</code>	An error occurred while communicating with one of the video units indicated in <code>units</code> . The <b>ErrorUnits</b> and <b>ErrorString</b> properties are updated. (Can only occur if <b>AsyncMode</b> is false.)

**See Also** **AsyncMode** Property, **ErrorString** Property, **ErrorUnits** Property, **VideoSaveBuffers** Property, **saveVideoRegion** Method

## saveVideoRegion Method

**Syntax** `void saveVideoRegion (int units, int row, int column, int height, int width, int bufferId) throws JposException;`

Parameter	Description
<i>units</i>	Bitwise mask indicating which video unit(s) to operate on.
<i>row</i>	The start row of the region to save.
<i>column</i>	The start column of the region to save.
<i>height</i>	The number of rows in the region to save.
<i>width</i>	The number of columns in the region to save.
<i>bufferId</i>	Number identifying the video buffer to use. Valid values range from 1 to the <b>VideoSaveBuffers</b> property for a selected unit(s).

**Remarks** Saves the specified video region of the display area to one of the provided video buffers for the video unit(s) indicated in the *units* parameter. The number of video buffers supported is stored in the **VideoSaveBuffers** property for each video unit online. However, a `JposException` will be thrown if the requested buffer exceeds the number of **SystemVideoSaveBuffers** even though the **VideoSaveBuffers** property may indicate the unit can support more save buffers than currently allocated for that unit.

If **VideoSaveBuffers** is greater than 0, the Device Service will be able to support at minimum one entire video screen. This does not guarantee that the Device Service can save an entire video screen in each supported buffer for a single unit. A `JposException` is thrown when all the buffer memory has been allocated for a specific unit.

The source area is defined by the *row*, *column*, *height*, and *width* parameters. The video region can be restored to the screen by calling the **restoreVideoRegion** method. If **saveVideoRegion** is called twice with the same *bufferId*, the previous video data is lost, and any allocated memory is returned to the system.

This method is performed synchronously if **AsyncMode** is false, and asynchronously if **AsyncMode** is true.

**Errors** A `JposException` may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_ILLEGAL	bufferId, row, column, height, or width, are out of range. The <b>ErrorUnits</b> and <b>ErrorString</b> properties are updated.
JPOS_E_EXTENDED	<b>ErrorCodeExtended</b> = JPOS_EROD_NOBUFFERS: Requested buffer exceeds the number of <b>SystemVideoSaveBuffers</b> .  <b>ErrorCodeExtended</b> = JPOS_EROD_NOROOM: All the buffer memory has been allocated for a specific unit. The <b>ErrorUnits</b> and <b>ErrorString</b> properties are updated.
JPOS_E_FAILURE	An error occurred while communicating with one of the video units indicated in units. The <b>ErrorUnits</b> and <b>ErrorString</b> properties are updated. (Can only occur if <b>AsyncMode</b> is false.)

**See Also** **AsyncMode** Property, **ErrorString** Property, **ErrorUnits** Property, **SystemVideoSaveBuffers** Property, **VideoSaveBuffers** Property, **restoreVideoRegion** Method

**selectCharacterSet Method**

**Syntax**      **void selectCharacterSet (int *units*, int *characterSet*) throws JposException;**

<b>Parameter</b>	<b>Description</b>
<i>units</i>	Bitwise mask indicating which video unit(s) to operate on.
<i>characterSet</i>	Contain the character set for displaying characters. Values are:
<b>Value</b>	<b>Meaning</b>
Range 101 - 199	A device-specific character set that does not match a code page, nor the ASCII or ANSI character sets.
Range 400 - 990	Code page; matches one of the standard values.
ROD_CS_UNICODE	The character set supports UNICODE. The value of this constant is 997.
ROD_CS_ASCII	The ASCII character set, supporting the ASCII characters between 20-hex and 7F-hex. The value of this constant is 998.
ROD_CS_ANSI	The ANSI character set. The value of this constant is 999. .

**Remarks**      Selects a compatible character set for the video unit(s) indicated in the *units* parameter.

The **CharacterSet** property is updated for each video unit id that is successfully selected a new character set.

**Errors**      A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
JPOS_E_FAILURE	An error occurred while communicating with one of the video units indicated in <i>units</i> . The <b>ErrorUnits</b> and <b>ErrorString</b> properties are updated.

**See Also**      **ErrorString** Property, **ErrorUnits** Property, **CapSelectCharacterSet** Property, **CharacterSet** Property

**setCursor Method**

**Syntax**      **void setCursor (int *units*, int *row*, int *column*) throws JposException;**

<b>Parameter</b>	<b>Description</b>
<i>units</i>	Bitwise mask indicating which video unit(s) to operate on.
<i>row</i>	Row to place the cursor on.
<i>column</i>	Column to place the cursor on.

**Remarks**      Updates the cursor position on the video unit(s) indicated in the *units* parameter.

**Errors**          A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
JPOS_E_FAILURE	An error occurred while communicating with one of the video units indicated in <i>units</i> . The <b>ErrorUnits</b> and <b>ErrorString</b> properties are updated.

**See Also**      **ErrorString** Property, **ErrorUnits** Property

**transactionDisplay Method**

**Syntax**      **void transactionDisplay (int *units*, int *function*) throws JposException;**

Parameter	Description
<i>units</i>	Bitwise mask indicating which video unit(s) to operate on.
<i>function</i>	Transaction control function. Valid values are:

Value	Meaning
ROD_TD_TRANSACTION	Begin a transaction.
ROD_TD_NORMAL	End a transaction by displaying the buffered data.

**Remarks**      Enters or exits transaction mode for the video unit(s) indicated in the *units* parameter.

If *function* is ROD\_TD\_TRANSACTION, then transaction mode is entered. Subsequent calls to **clearVideo**, **clearVideoRegion**, **copyVideoRegion**, **displayData**, **drawBox**, **restoreVideoRegion**, **saveVideoRegion**, and **updateVideoRegionAttribute** will buffer the display data (either at the video unit or the Device Service, depending on the display capabilities) until **transactionDisplay** is called with the *function* parameter set to ROD\_TD\_NORMAL. (In this case, the display methods only validate the method parameters and buffer the data – they do not initiate displaying. Also, the value of the **AsyncMode** property does not affect their operation: No **OutputID** will be assigned to the request, nor will an **OutputCompleteEvent** be enqueued.)

If *function* is ROD\_TD\_NORMAL, then transaction mode is exited. If some data was buffered by calls to the methods **clearVideo**, **clearVideoRegion**, **copyVideoRegion**, **displayData**, **drawBox**, **restoreVideoRegion**, **saveVideoRegion**, and **updateVideoRegionAttribute**, then the buffered data is displayed. The entire transaction is treated as one message. This method is performed synchronously if **AsyncMode** is false, and asynchronously if **AsyncMode** is true.

Calling the **clearOutput** method cancels transaction mode for the unit indicated in the **CurrentUnitID** property. Any buffered print lines are also cleared.

**Errors** A `JposException` may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s `ErrorCode` property are:

Value	Meaning
<code>JPOS_E_BUSY</code>	Cannot perform while output is in progress for one of the video units indicated in <code>units</code> . The <b>ErrorUnits</b> and <b>ErrorString</b> properties are updated. (Can only occur if <b>AsyncMode</b> is false and function is <code>ROD_TD_NORMAL</code> )
<code>JPOS_E_FAILURE</code>	An error occurred communicating with one of the video units indicated in <code>units</code> . The <b>ErrorUnits</b> and <b>ErrorString</b> properties are updated. (Can only occur if <b>AsyncMode</b> is false and function is <code>ROD_TD_NORMAL</code> )

### updateVideoRegionAttribute Method

**Syntax**     `void updateVideoRegionAttribute (int units, int function, int row, int column, int height, int width, int attribute) throws JposException;`

Parameter	Description
<i>units</i>	Bitwise mask indicating which video unit(s) to operate on.
<i>function</i>	The attribute command. See values below.
<i>row</i>	The region’s start row.
<i>column</i>	The region’s start column.
<i>height</i>	The number of rows in the region.
<i>width</i>	The number of columns in the region.
<i>attribute</i>	See Model on page 620 in the General Information section.

The *function* parameter values are:

Value	Meaning
<code>ROD_UA_SET</code>	Set the region with the new attribute.
<code>ROD_UA_INTENSITY_ON</code>	Turn on foreground intensity in the region.
<code>ROD_UA_INTENSITY_OFF</code>	Turn off foreground intensity in the region.



Value	Meaning
ROD_UA_REVERSE_ON	Reverse video the region.
ROD_UA_REVERSE_OFF	Remove reverse video from the region.
ROD_UA_BLINK_ON	Turn on blinking in the region.
ROD_UA_BLINK_OFF	Turn off blinking in the region.

**Remarks** Modifies the attribute on the video unit(s) indicated in the *units* parameter in the region defined by the *row*, *column*, *height*, and *width* parameters. When the *function* parameter is ROD\_UA\_SET, the region's attributes will be replaced with the new value in the *attribute* parameter; otherwise the *attribute* parameter is ignored and the region's attributes will be modified.

This method is performed synchronously if **AsyncMode** is false, and asynchronously if **AsyncMode** is true.

**Errors** A JposException may be thrown when this method is invoked. For further information, see "Exceptions" on page 15.

Some possible values of the exception's *ErrorCode* property are:

Value	Meaning
JPOS_E_FAILURE	An error occurred while communicating with one of the video units indicated in <i>units</i> . The <b>ErrorUnits</b> and <b>ErrorString</b> properties are updated. (Can only occur if <b>AsyncMode</b> is false.)

**See Also** **AsyncMode** Property, **ErrorString** Property, **ErrorUnits** Property, "Model" on page 620.

**videoSound Method**

**Syntax**      **void videoSound (int units, int frequency, int duration, int numberOfCycles, int interSoundWait) throws JposException;**

Parameter	Description
<i>units</i>	Bitwise mask indicating which video unit(s) to operate on.
<i>frequency</i>	Tone frequency in Hertz.
<i>duration</i>	Tone duration in milliseconds.
<i>numberOfCycles</i>	If JPOS_FOREVER, then start tone sounding and, repeat continuously. Else perform the specified number of cycles.
<i>interSoundWait</i>	When <i>numberOfCycles</i> is not one, then pause for <i>interSoundWait</i> milliseconds before repeating the tone cycle (before playing the tone again)

**Remarks**      Sounds the video enunciator for the video(s) indicated in the *units* parameter.

This method is performed synchronously if **AsyncMode** is false, and asynchronously if **AsyncMode** is true.

The duration of a video tone cycle is:

*Duration* parameter + *interSoundWait* parameter (except on the last tone cycle)

After the video has started an asynchronous sound, then the **clearOutput** method will stop the sound. (When an *interSoundWait* value of JPOS\_FOREVER was used to start the sound, then the application must use **clearOutput** to stop the continuous sounding of tones.)

If **CapTone** is false for the selected unit(s), a JposException is thrown.

**Errors**      A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_FAILURE	An error occurred while communicating with one of the video units indicated in the <i>units</i> parameter. The <b>ErrorUnits</b> and <b>ErrorString</b> properties are updated. (Can only occur if <b>AsyncMode</b> is false.)

**See Also**      **AsyncMode** Property, **ErrorString** Property, **ErrorUnits** Property, **CapTone** Property, **clearOutput** Method

## Events

### DataEvent

**Interface** `jpos.events.DataListener`

**Method** `dataOccurred (DataEvent e);`

**Description** Notifies the application when input data from a video touch unit is available.

**Properties** This event contains the following property:

Property	Type	Description
----------	------	-------------

<i>Status</i>	<i>int</i>	As described below
---------------	------------	--------------------

The *Status* parameter is divided into four bytes as indicated below:

High Word		Low Word(Event Type)
High Byte Row	Low Byte Column	ROD_DE_TOUCH_UP ROD_DE_TOUCH_DOWN ROD_DE_TOUCH_MOVE

The low word contains the Event type. The high word contains additional data depending on the Event type. When the Event type is ROD\_DE\_TOUCH\_UP, ROD\_DE\_TOUCH\_DOWN, or ROD\_DE\_TOUCH\_MOVE, the high word indicates where the touch occurred. The low byte contains the Column position and the high byte contains the Row position, with valid values ranging from 0-255.

**Remarks** This event can be filtered at the Remote Order Display device by setting the **EventType** property.

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

**See Also** “Device Input Model” on page 22, **EventUnitID** Property, **DataEventEnabled** Property, **FreezeEvents** Property

## ErrorEvent

<b>Interface</b>	<b>jpos.events.ErrorListener</b>
<b>Method</b>	<b>errorOccurred (ErrorEvent e);</b>
<b>Description</b>	Notifies the application that a Remote Order Display error has been detected and a suitable response by the application is necessary to process the error condition.
<b>Properties</b>	This event contains the following properties:

Property	Type	Description
<i>ErrorCode</i>	<i>int</i>	Result code causing the error event. See a list of <i>ErrorCodes</i> on page 16.
<i>ErrorCodeExtended</i>	<i>int</i>	Extended Error Code causing the error event. If <i>ErrorCode</i> is JPOS_E_EXTENDED, then see values below. Otherwise, it may contain a Service-specific value.
<i>ErrorLocus</i>	<i>int</i>	Location of the error. See values below.
<i>ErrorResponse</i>	<i>int</i>	Error response, whose default value may be overridden by the application. (i.e., this property is settable). See values below.

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

Value	Meaning
JPOS_EL_OUTPUT	Error occurred while processing asynchronous output.
JPOS_EL_INPUT	Error occurred while gathering or processing event-driven input. No input data is available.
JPOS_EL_INPUT_DATA	Error occurred while gathering or processing event-driven input, and some previously buffered data is available.

The contents of the *ErrorResponse* property are preset to a default value, based on the *ErrorLocus*. The application's error event listener may change *ErrorResponse* to one of the following values:

Value	Meaning
JPOS_ER_RETRY	Use only when locus is JPOS_EL_OUTPUT. Retry the asynchronous output. The error state is exited. Default when locus is JPOS_EL_OUTPUT.
JPOS_ER_CLEAR	Clear the buffered input data. The error state is exited. Default when locus is JPOS_EL_INPUT.
JPOS_ER_CONTINUEINPUT	Use only when locus is JPOS_EL_INPUT_DATA. Acknowledges the error and directs the Device to

continue processing. The Device remains in the error state, and will deliver additional **DataEvents** as directed by the **DataEventEnabled** property. When all input has been delivered and the **DataEventEnabled** property is again set to true, then another **ErrorEvent** is delivered with locus JPOS\_EL\_INPUT.  
Default when locus is JPOS\_EL\_INPUT\_DATA.

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

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

**See Also** “Device Output Models” on page 25, “Device States” on page 30, **DataEventEnabled** Property, **EventUnits** Property, **EventString** Property

## OutputCompleteEvent

**Interface** `jpos.events.OutputCompleteListener`

**Method** `outputCompleteOccurred (OutputCompleteEvent e);`

**Description** Notifies the application that the queued output request associated with the OutputID property has completed successfully.

**Properties** This event contains the following property:

Property	Type	Description
<i>OutputID</i>	<i>int</i>	The ID number of the asynchronous output request that is complete.

**Remarks** Enqueued when a previously started asynchronous output request completes successfully. The **EventUnits** property is updated before the event is delivered.

**See Also** **EventUnits** Property, “Device Output Models” on page 25

## StatusUpdateEvent

**Interface**    `jpos.events.StatusUpdateListener`

**Method**      `statusUpdateOccurred (StatusUpdateEvent e);`

**Description**    Notifies the application that there is a change in the power status of a video unit.

**Properties**     This event contains the following property:

Property	Type	Description
<i>Status</i>	<i>int</i>	Reports a change in the power state of a display.

*Note that Release 1.3* added Power State Reporting with additional *Power reporting StatusUpdateEvent values*. See “StatusUpdateEvent” description on page 78.

**Remarks**      Enqueued when the Remote Order Display detects a power state change.

Deviation from the standard **StatusUpdateEvent** (see page 78):

- Before delivering the event, the **EventUnits** property is set to the units for which the new power state applies.
- When the Remote Order Display is enabled, then a **StatusUpdateEvent** is enqueued to specify the bitmask of online units.
- While the Remote Order Display is enabled, a **StatusUpdateEvent** is enqueued when the power state of one or more units change. If more than one unit changes state at the same time, the Device Service may choose to either enqueue multiple events or to coalesce the information into a minimal number of events applying to **EventUnits**.

**See Also**      **EventUnits** Property

# Scale

## Summary

### Properties

<i>Common</i>	<i>Ver</i>	<i>Type</i>	<i>Access</i>	<i>May Use After</i>
<b>AutoDisable</b>	1.3	boolean	R/W	open
<b>CapPowerReporting</b>	1.3	int	R	open
<b>CheckHealthText</b>		String	R	open
<b>Claimed</b>		boolean	R	open
<b>DataCount</b>	1.3	int	R	open
<b>DataEventEnabled</b>	1.3	boolean	R/W	open
<b>DeviceEnabled</b>		boolean	R/W	open & claim
<b>FreezeEvents</b>		boolean	R/W	open
<b>OutputID</b>		int	R	<i>Not Supported</i>
<b>PowerNotify</b>	1.3	int	R/W	open
<b>PowerState</b>	1.3	int	R	open
<b>State</b>		int	R	--
<b>DeviceControlDescription</b>		String	R	--
<b>DeviceControlVersion</b>		int	R	--
<b>DeviceServiceDescription</b>		String	R	open
<b>DeviceServiceVersion</b>		int	R	open
<b>PhysicalDeviceDescription</b>		String	R	open
<b>PhysicalDeviceName</b>		String	R	open

---

<i>Specific</i>	<i>Ver</i>	<i>Type</i>	<i>Access</i>	<i>May Use After</i>
<b>CapDisplay</b>		boolean	R	open
<b>CapDisplayText</b>	1.3	boolean	R	open
<b>CapPriceCalculating</b>	1.3	boolean	R	open
<b>CapTareWeight</b>	1.3	boolean	R	open
<b>CapZeroScale</b>	1.3	boolean	R	open
<b>AsyncMode</b>	1.3	boolean	R/W	open
<b>MaxDisplayTextChars</b>	1.3	int	R	open
<b>MaximumWeight</b>		int	R	open
<b>SalesPrice</b>	1.3	long	R	open
<b>TareWeight</b>	1.3	int	R/W	open
<b>UnitPrice</b>	1.3	long	R/W	open
<b>WeightUnit</b>		int	R	open



**Methods**

<i>Common</i>	<i>Ver</i>	<i>May Use After</i>
<b>open</b>		--
<b>close</b>		open
<b>claim</b>		open
<b>release</b>		open & claim
<b>checkHealth</b>		open, claim, & enable
<b>clearInput</b>	1.3	open & claim
<b>clearOutput</b>		<i>Not Supported</i>
<b>directIO</b>		open & claim
 <i>Specific</i>		
<b>displayText</b>	1.3	open, claim, & enable
<b>readWeight</b>		open, claim, & enable
<b>zeroScale</b>	1.3	open, claim, & enable

**Events**

<i>Name</i>	<i>Ver</i>	<i>May Occur After</i>
<b>DataEvent</b>	1.3	open, claim, & enable
<b>DirectIOEvent</b>	1.3	open & claim
<b>ErrorEvent</b>	1.3	open, claim, & enable
<b>OutputCompleteEvent</b>		<i>Not Supported</i>
<b>StatusUpdateEvent</b>	1.3	open, claim, & enable

## General Information

The Scale Control's class name is "jpos.Scale".

The device constants are contained in the class "jpos.ScaleConst".

See "Package Structure" on page 40.

### Capabilities

The scale has the following capability:

- Provides item weight to the application. The measure of weight may be in grams, kilograms, ounces, or pounds, depending upon the scale device.

The scale may have the following additional capabilities:

- Includes an integrated display with the current weight, or with the current weight plus application-specified text.
- Performs price calculations (weight X unit price) and returns the sale price. (This feature is mostly used in Europe at this time.)
- Supports application setting of tare weight.
- Supports application zeroing of the scale.

### Model

The general model of a scale is:

- A scale returns the weight of an item placed on its weighing surface.
- The primary scale method is **readWeight**. By default, it is performed synchronously. It returns after reading data from the scale; the weight is returned in the **readWeight**'s *weightData* parameter. If an error occurs or if the timeout elapses, a *JposException* will be thrown.

- **JavaPOS Release 1.3 and later - Asynchronous Input**

If the **AsyncMode** property is true when **readWeight** is called, then the method is performed asynchronously. It initiates event driven input and returns immediately. The timeout parameter specifies the maximum time the application wants to wait for a settled weight. Additional points are:

- If an error occurs while initiating event driven input (such as the device is offline), then a **JposException** is thrown. Otherwise, **readWeight** returns immediately to the application, and scale processing continues asynchronously.
- If a settled weight is received, then a **DataEvent** is enqueued containing the weight data in the *Status* property.
- If a scale error occurs (including a timeout with no settled weight), then an **ErrorEvent** is enqueued. The application event handler may retry the weighing process by setting the event's *ErrorResponse* property to **JPOS\_ER\_RETRY**.
- Only one asynchronous call to **readWeight** can be in progress at a time. An attempt to nest asynchronous scale operations will result in a **JposException** being thrown.
- An asynchronous scale operation may be cancelled with the **clearInput** method.

For price-calculating scales, the application should set the **UnitPrice** property before calling **readWeight**. After a weight is read (and just before the **DataEvent** is delivered to the application, for asynchronous mode), the **SalesPrice** property is set to the calculated price of the item.

## Device Sharing

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

- After opening the device, properties are readable.
- The application must claim the device before enabling it.
- The application must claim and enable the device before calling methods that manipulate the device.
- See the “Summary” table for precise usage prerequisites.

## Properties

### AsyncMode Property R/W *Added in Release 1.3*

Type	<b>boolean</b>
Remarks	If true, then the readWeight method will be performed asynchronously. If false, the readWeight method will be performed synchronously.  This property is initialized to false by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	readWeight Method

### CapDisplay Property R

Type	<b>boolean</b>
Remarks	If true, the scale includes an integrated display that shows the current weight. If false, the application may need to show the current weight on another display.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>CapDisplayText</b> Property, <b>MaxDisplayTextChars</b> Property

### CapDisplayText Property R *Added in Release 1.3*

Type	<b>boolean</b>
Remarks	If true, the scale includes an integrated display that shows the current weight and can also show text that describes the item being weighed. If false, extra text cannot be shown on the display.  If true, then <b>CapDisplay</b> must also be true.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>CapDisplay</b> Property, <b>MaxDisplayTextChars</b> Property

**CapPriceCalculating Property R**     *Added in Release 1.3*

Type	<b>boolean</b>
Remarks	<p>If true, the scale can calculate prices. If false, the scale only returns a weight.</p> <p>For price calculating scales the calculation unit is in the scale rather than in the data-receiving terminal.</p> <p>This property is initialized by the <b>open</b> method.</p>
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>readWeight</b> Method, <b>WeightUnit</b> Property, <b>UnitPrice</b> Property, <b>SalesPrice</b> Property

**CapTareWeight Property R**     *Added in Release 1.3*

Type	<b>boolean</b>
Remarks	<p>If true, the scale includes setting a tare value. If false, the scale does not support tare values.</p> <p>This property is initialized by the <b>open</b> method.</p>
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>TareWeight</b> Property

**CapZeroScale Property R**     *Added in Release 1.3*

Type	<b>boolean</b>
Remarks	<p>If true, the application can set the scale weight to zero. If false, the scale does not support programmatic zeroing.</p> <p>This property is initialized by the <b>open</b> method.</p>
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>zeroScale</b> Method

**MaxDisplayTextChars Property R** *Added in Release 1.3*

<b>Type</b>	<b>int</b>
<b>Remarks</b>	<p>Holds the number of characters that may be displayed on an integrated display for the text which describes an article.</p> <p>If <b>CapDisplayText</b> is false, then the device does not support text displaying and this property is always zero.</p> <p>This property is initialized by the <b>open</b> method.</p>
<b>Errors</b>	A <code>JposException</code> may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
<b>See Also</b>	<b>CapDisplay</b> Property, <b>CapDisplayText</b> Property

**MaximumWeight Property R**

<b>Type</b>	<b>int</b>
<b>Remarks</b>	<p>Holds the maximum weight measurement possible from the scale. The measurement unit is available via the <b>WeightUnit</b> property.</p> <p>This property has an assumed decimal place located after the “thousands” digit position. For example, an actual value of 12345 represents 12.345, and an actual value of 5 represents 0.005.</p> <p>Changing the <b>WeightUnit</b> property will also cause this property to change.</p> <p>This property is initialized by the <b>open</b> method.</p>
<b>Errors</b>	A <code>JposException</code> may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
<b>See Also</b>	<b>WeightUnit</b> Property

**SalesPrice Property R** *Added in Release 1.3*

<b>Type</b>	<b>long</b>
<b>Remarks</b>	<p>Holds the sales price read from the scale for price calculating scales. For price calculating scales the scale calculates this value during the process of weighing by multiplying the <b>UnitPrice</b> property by the acquired weight. This property is a monetary value stored using an implied four decimal places. For example, an actual value of 12345 represents 1.2345.</p> <p>This property is set before the <b>readWeight</b> method returns (in synchronous mode) or the <b>DataEvent</b> is delivered (in asynchronous mode).</p> <p>If <b>CapPriceCalculating</b> is false, then the device is not a price calculating scale and <b>SalesPrice</b> is always zero.</p> <p>This property is initialized by the <b>open</b> method to zero.</p>
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
<b>See Also</b>	<b>readWeight</b> Method, <b>WeightUnit</b> Property, <b>CapPriceCalculating</b> Property, <b>UnitPrice</b> Property

**TareWeight Property R/W** *Added in Release 1.3*

<b>Type</b>	<b>int</b>				
<b>Remarks</b>	<p>Holds the tare weight of scale data. This property has an assumed decimal place located after the “thousands” digit position. For example, an actual value of 12345 represents 12.345, and an actual value of 5 represents 0.005. The measured unit is specified in the <b>WeightUnit</b> property. If <b>CapTareWeight</b> is false, then the device does not support setting of a tare value and this property is always zero.</p> <p>Tare weight is not included in the item weight returned by the <b>readWeight</b> method.</p> <p>This property is initialized by the <b>open</b> method to the scale’s default tare weight (usually zero).</p>				
<b>Errors</b>	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15..</p> <p>Some possible values of the exception’s <i>ErrorCode</i> property are:</p> <table border="1"> <thead> <tr> <th><b>Value</b></th> <th><b>Meaning</b></th> </tr> </thead> <tbody> <tr> <td>JPOS_E_ILLEGAL</td> <td><b>CapTareWeight</b> is false or an invalid tare value was specified.</td> </tr> </tbody> </table>	<b>Value</b>	<b>Meaning</b>	JPOS_E_ILLEGAL	<b>CapTareWeight</b> is false or an invalid tare value was specified.
<b>Value</b>	<b>Meaning</b>				
JPOS_E_ILLEGAL	<b>CapTareWeight</b> is false or an invalid tare value was specified.				
<b>See Also</b>	<b>readWeight</b> Method, <b>WeightUnit</b> Property, <b>CapTareWeight</b> Property				

**UnitPrice Property R/W****Added in Release 1.3**

<b>Type</b>	<b>long</b>				
<b>Remarks</b>	<p>Holds the unit price of the article to be weighed. For price calculating scales this property is to be set before calling the <b>readWeight</b> method. During weighing, the scale sets the <b>SalesPrice</b> property to the product of the item's weight and this property. This property is a monetary value stored using an implied four decimal places. For example, an actual value of 12345 represents 1.2345.</p> <p>If <b>CapPriceCalculating</b> is false, then setting of a unit price is not supported and this property is always zero.</p> <p>This property is initialized by the <b>open</b> method to zero.</p>				
<b>Errors</b>	<p>A <i>JposException</i> may be thrown when this property is accessed. For further information, see "Exceptions" on page 15.</p> <p>Some possible values of the exception's <i>ErrorCode</i> property are:</p> <table> <thead> <tr> <th><b>Value</b></th> <th><b>Meaning</b></th> </tr> </thead> <tbody> <tr> <td>JPOS_E_ILLEGAL</td> <td><b>CapPriceCalculating</b> is false or an invalid price was specified.</td> </tr> </tbody> </table>	<b>Value</b>	<b>Meaning</b>	JPOS_E_ILLEGAL	<b>CapPriceCalculating</b> is false or an invalid price was specified.
<b>Value</b>	<b>Meaning</b>				
JPOS_E_ILLEGAL	<b>CapPriceCalculating</b> is false or an invalid price was specified.				
<b>See Also</b>	<b>readWeight</b> Method, <b>WeightUnit</b> Property, <b>CapPriceCalculating</b> Property, <b>SalesPrice</b> Property				

**WeightUnit Property R**

<b>Type</b>	<b>int</b>										
<b>Remarks</b>	<p>Holds the unit of weight of scale data, and has one of the following values:</p> <table> <thead> <tr> <th><b>Value</b></th> <th><b>Meaning</b></th> </tr> </thead> <tbody> <tr> <td>SCAL_WU_GRAM</td> <td>Unit is a gram.</td> </tr> <tr> <td>SCAL_WU_KILOGRAM</td> <td>Unit is a kilogram (= 1000 grams).</td> </tr> <tr> <td>SCAL_WU_OUNCE</td> <td>Unit is an ounce.</td> </tr> <tr> <td>SCAL_WU_POUND</td> <td>Unit is a pound (= 16 ounces).</td> </tr> </tbody> </table> <p>This property is initialized to the scale's weight unit by the <b>open</b> method.</p>	<b>Value</b>	<b>Meaning</b>	SCAL_WU_GRAM	Unit is a gram.	SCAL_WU_KILOGRAM	Unit is a kilogram (= 1000 grams).	SCAL_WU_OUNCE	Unit is an ounce.	SCAL_WU_POUND	Unit is a pound (= 16 ounces).
<b>Value</b>	<b>Meaning</b>										
SCAL_WU_GRAM	Unit is a gram.										
SCAL_WU_KILOGRAM	Unit is a kilogram (= 1000 grams).										
SCAL_WU_OUNCE	Unit is an ounce.										
SCAL_WU_POUND	Unit is a pound (= 16 ounces).										
<b>Errors</b>	A <i>JposException</i> may be thrown when this property is accessed. For further information, see "Exceptions" on page 15.										



## Methods

### displayText Method

***Added in Release 1.3***

**Syntax**      **void displayText (String *data*) throws JposException;**

Parameter	Description
-----------	-------------

<i>data</i>	The string of characters to display.
-------------	--------------------------------------

**Remarks**      If **CapDisplayText** is true, updates the text shown on the integrated display. Calling this method with an empty string (“”) will clear the display.

**Errors**        A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
-------	---------

JPOS_E_ILLEGAL	An invalid text was specified -- the text contains more characters than <b>MaxDisplayTextChars</b> , or <b>CapDisplayText</b> is false.
----------------	---

**See Also**      **CapDisplay** Property, **CapDisplayText** Property, **MaxDisplayTextChars** Property

## readWeight Method

**Syntax**      **void readWeight (int[] *weightData*, int *timeout*) throws JposException;**

Parameter	Description
<i>weightData</i>	If <b>AsyncMode</b> is false, contains the returned value for the weight measured by the scale, else zero.
<i>timeout</i>	The number of milliseconds to wait for a settled weight before failing the method. If zero, the method attempts to read the scale weight, then returns the appropriate status immediately. If JPOS_FOREVER (-1), the method waits as long as needed until a weight is successfully read or an error occurs.

**Remarks**      Reads a weight from the scale.

The weight returned, *weightData*, has an assumed decimal place located after the “thousands” digit position. For example, an actual value of 12345 represents 12.345, and an actual value of 5 represents 0.005.

### ***Release 1.2***

The weighing process is performed synchronously and the method will return after finishing the weighing process. The weight is returned in the *weightData* parameter.

### ***Release 1.3 and later***

If **AsyncMode** is false, then **readWeight** operates synchronously, as with earlier releases.

If **AsyncMode** is true, the weighing process is performed asynchronously. The method will initiate a read, then return immediately. Once the weighing process is complete, a **DataEvent** is delivered with the item’s weight contained in the event’s *Status* property.

**Errors** A `JposException` may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_ILLEGAL	An invalid <i>timeout</i> parameter was specified.
JPOS_E_TIMEOUT	A stable non-zero weight was not available before <i>timeout</i> milliseconds elapsed (only if <b>AsyncMode</b> is false).
JPOS_E_EXTENDED	<i>ErrorCodeExtended</i> = JPOS_ESCAL_OVERWEIGHT: The weight was over <b>MaximumWeight</b> (can only be returned if <b>AsyncMode</b> is false).

**See Also** **UnitPrice** Property, **WeightUnit** Property, **CapPriceCalculating** Property, **SalesPrice** Property, **TareWeight** Property

## zeroScale Method

**Added in Release 1.3**

**Syntax** `void zeroScale () throws JposException;`

**Remarks** If **CapZeroScale** is true, sets the current scale weight to zero. It may be used for initial calibration, or to account for tare weight on the scale.

**Errors** A `JposException` may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_ILLEGAL	<b>CapZeroScale</b> is false.

**See Also** **CapZeroScale** Property

## Events

### DataEvent

**Added in Release 1.3**

**Interface** `jpos.events.DataListener`

**Method** `dataOccurred (DataEvent e);`

**Description** Notifies the application that an asynchronous **readWeight** has completed.

**Properties** This event contains the following property:

Property	Type	Description
<i>Status</i>	<i>int</i>	The weight of the item.

**Remarks** If the scale is a price calculating scale, the unit price is placed in the **UnitPrice** property and the calculated sales price is placed in the **SalesPrice** property before this event is delivered.

**See Also** “Events” on page 18

### DirectIOEvent

**Interface** `jpos.events.DirectIOListener`

**Method** `directIOOccurred (DirectIOEvent e);`

**Description** Provides Device Service information directly to the application. This event provides a means for a vendor-specific Scale Device Service to provide events to the application that are not otherwise supported by the Device Control.

**Properties** This event contains the following properties:

Property	Type	Description
<i>EventNumber</i>	<i>int</i>	Event number whose specific values are assigned by the Device Service.
<i>Data</i>	<i>int</i>	Additional numeric data. Specific values vary by the <i>EventNumber</i> and the Device Service. This property is settable.
<i>Object</i>	<i>Object</i>	Additional data whose usage varies by the <i>EventNumber</i> and Device Service. This property is settable.

**Remarks** This event is to be used only for those types of vendor specific functions that are not otherwise described as part of the JavaPOS standard. Use of this event may restrict the application program from being used with other vendor’s Scale devices which may not have any knowledge of the Device Service’s need for this event.

**See Also** “Events” on page 18

**ErrorEvent****Added in Release 1.3****Interface** `jpos.events.ErrorListener`**Method** `errorOccurred (ErrorEvent e);`**Description** Notifies the application that a scale device error has been detected and a suitable response by the application is necessary to process the error condition.**Properties** This event contains the following properties:

Property	Type	Description
<i>ErrorCode</i>	<i>int</i>	Error code causing the error event. See list of <i>ErrorCodes</i> on page 16.
<i>ErrorCodeExtended</i>	<i>int</i>	Extended error code causing the error event. It may contain a Service-specific value.
<i>ErrorLocus</i>	<i>int</i>	Location of the error. See values below.
<i>ErrorResponse</i>	<i>int</i>	Error response, whose default value may be overridden by the application (i.e., this property is settable). See values below.

The *ErrorLocus* property has one of the following values:

Value	Meaning
JPOS_EL_INPUT	Error occurred while gathering or processing event-driven input. No input data is available.
JPOS_EL_INPUT_DATA	Error occurred while gathering or processing event-driven input, and some previously buffered data is available.

The application's error event listener may change *ErrorResponse* to one of the following values:

Value	Meaning
JPOS_ER_RETRY	Retry the asynchronous input. The error state is exited.
JPOS_ER_CLEAR	Clear the buffered input data. The error state is exited. Default when locus is JPOS_EL_INPUT.
JPOS_ER_CONTINUEINPUT	Use only when locus is JPOS_EL_INPUT_DATA. Acknowledges the error and directs the Device to continue processing. The Device remains in the error state, and will deliver additional <b>DataEvents</b> as directed by the <b>DataEventEnabled</b> property. When all input has been delivered and <b>DataEventEnabled</b> is again set to true, then another <b>ErrorEvent</b> is delivered with locus JPOS_EL_INPUT. Default when locus is JPOS_EL_INPUT_DATA.

	JPOS_ER_RETRY	Retry the asynchronous input. The error state is exited.
<b>Remarks</b>	Enqueued when an error is detected while trying to read scale data. This event is not delivered until <b>DataEventEnabled</b> is true, so that proper application sequencing occurs.	
<b>See Also</b>	“Events” on page 18	

**StatusUpdateEvent*****Added in Release 1.3***

**Interface**    `jpos.events.StatusUpdateListener`

**Method**      `statusUpdateOccurred (StatusUpdateEvent e);`

**Description**    Notifies the application that a scale has had an operation status change.

**Properties**      This event contains the following property:

<b>Property</b>	<b>Type</b>	<b>Description</b>
<i>Status</i>	<i>int</i>	Indicates a status change.

*Note that Release 1.3* added Power State Reporting with additional *Power reporting* **StatusUpdateEvent** values. See “StatusUpdateEvent” description on page 78.

**Remarks**      Enqueued when a change in status of the device has occurred.

**See Also**      “Events” on page 18

# Scanner (Bar Code Reader)

## Summary

### Properties

<i>Common</i>	<i>Ver</i>	<i>Type</i>	<i>Access</i>	<i>May Use After</i>
<b>AutoDisable</b>		boolean	R/W	open
<b>CapPowerReporting</b>	1.3	int	R	open
<b>CheckHealthText</b>		String	R	open
<b>Claimed</b>		boolean	R	open
<b>DataCount</b>		int	R	open
<b>DataEventEnabled</b>		boolean	R/W	open
<b>DeviceEnabled</b>		boolean	R/W	open & claim
<b>FreezeEvents</b>		boolean	R/W	open
<b>OutputID</b>		int	R	<i>Not Supported</i>
<b>PowerNotify</b>	1.3	int	R/W	open
<b>PowerState</b>	1.3	int	R	open
<b>State</b>		int	R	--
<b>DeviceControlDescription</b>		String	R	--
<b>DeviceControlVersion</b>		int	R	--
<b>DeviceServiceDescription</b>		String	R	open
<b>DeviceServiceVersion</b>		int	R	open
<b>PhysicalDeviceDescription</b>		String	R	open
<b>PhysicalDeviceName</b>		String	R	open

<i>Specific</i>	<i>Ver</i>	<i>Type</i>	<i>Access</i>	<i>May Use After</i>
<b>DecodeData</b>		boolean	R/W	open
<b>ScanData</b>		byte[]	R	open
<b>ScanDataLabel</b>		byte[]	R	open
<b>ScanDataType</b>		int	R	open

**Methods**

<i>Common</i>	<i>Ver</i>	<i>May Use After</i>
<b>open</b>		--
<b>close</b>		open
<b>claim</b>		open
<b>release</b>		open & claim
<b>checkHealth</b>		open, claim, & enable
<b>clearInput</b>		open & claim
<b>clearOutput</b>		<i>Not Supported</i>
<b>directIO</b>		open

**Events**

<i>Name</i>	<i>Ver</i>	<i>May Occur After</i>
<b>DataEvent</b>		open, claim, & enable
<b>DirectIOEvent</b>	1.3	open & claim
<b>ErrorEvent</b>		open, claim, & enable
<b>OutputCompleteEvent</b>		<i>Not Supported</i>
<b>StatusUpdateEvent</b>	1.3	open, claim, & enable



## General Information

The Scanner Control's class name is "jpos.Scanner".  
The device constants are contained in the class "jpos.ScannerConst".  
See "Package Structure" on page 40.

### Capabilities

The Scanner Control has the following capability:

- Reads encoded data from a label.

### Model

The Scanner Control follows the JavaPOS model for input devices:

- When input is received by the Device Service, it enqueues a **DataEvent**.
- If the **AutoDisable** property is true, then the Device automatically disables itself when a **DataEvent** is enqueued.
- A queued **DataEvent** can be delivered to the application when the **DataEventEnabled** property is true. Just before delivering this event, data is copied into properties, and further data events are disabled by setting **DataEventEnabled** to false. This causes subsequent input data to be enqueued while the application processes the current input and associated properties. When the application has finished the current input and is ready for more data, it reenables events by setting **DataEventEnabled** to true.
- An **ErrorEvent** (or events) is enqueued if an error occurs while gathering or processing input, and is delivered to the application when **DataEventEnabled** is true.
- The **DataCount** property may be read to obtain the number of queued **DataEvents**.
- All queued input may be deleted by calling **clearInput**.

Scanned data is placed into the property **ScanData**. If the application sets the property **DecodeData** to true, then the data is decoded into the **ScanDataLabel** and **ScanDataType** properties.

### Device Sharing

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

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

## Properties

### DecodeData Property R/W

Type	<b>boolean</b>
Remarks	If true, then <b>ScanData</b> will be decoded into the properties <b>ScanDataLabel</b> and <b>ScanDataType</b> .  This property is initialized to false by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	“Device Input Model” on page 22

## ScanData Property R

**Type**            **byte[]**

**Remarks**       Holds the data read from the scanner.

Scan data is, in general, in the format as delivered from the scanner. Message header and trailer information are removed, however, since they do not contain useful information for an application and are likely to be scanner-specific.

Common header information is a prefix character (such as an STX character). Common trailer information is a terminator character (such as an ETX or CR character) and a block check character if one is generated by the scanner.

This property should include a symbology character if one is returned by the scanner (for example, an 'A' for UPC-A). It should also include check digits if they are present in the label and returned by the scanner. (Note that both symbology characters and check digits may or may not be present, depending upon the scanner configuration. The scanner will return them if present, but will not generate or calculate them if they are absent.)

Some merchandise may be marked with a supplemental barcode. This barcode is typically placed to the right of the main barcode, and consists of an additional two or five characters of information. If the scanner reads merchandise that contains both main and supplemental barcodes, the supplemental characters are appended to the main characters, and the result is delivered to the application as one label. (Note that a scanner may support configuration that enables or disables the reading of supplemental codes.)

Some merchandise may be marked with multiple labels, sometimes called multi-symbol labels or tiered labels. These barcodes are typically arranged vertically, and may be of the same or different symbology. If the scanner reads merchandise that contains multiple labels, each barcode is delivered to the application as a separate label. This is necessary due to the current lack of standardization of these barcode types. One is not able to determine all variations based upon the individual barcode data. Therefore, the application will need to determine when a multiple label barcode has been read based upon the data returned. (Note that a scanner may or may not support reading of multiple labels.)

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

**Errors**            A `JposException` may be thrown when this property is accessed. For further information, see "Exceptions" on page 15.

**See Also**        "Device Input Model" on page 22

**ScanDataLabel Property R**

<b>Type</b>	byte[]
<b>Remarks</b>	Holds the decoded bar code label.

When **DecodeData** is false, this property will have zero length. When **DecodeData** is true, then **ScanData** is decoded into this property as follows:

- Scanner-generated symbology characters are removed, if present.
- If the label type contains a readable check digit (such as with UPC-A and EAN-13), then it must be present in this property. If the scanner does not return the check digit to the Device Service, then it is to be calculated and included.
- For variable length bar codes, the length identification is removed, if present.

For example, the EAN-13 barcode which appears printed as “5 018374 827715” on a label may be received from the scanner and placed into **ScanData** as the following:

<b>Received from scanner</b>	<b>ScanData Comment</b>
------------------------------	-------------------------

---

5018374827715	5018374827715Complete barcode only
501837482771<CR>	501837482771Without check digit with carriage return
F5018374827715<CR>	F5018374827715With scanner-dependent symbology character and carriage return
<STX>F5018374827715<ETX>F5018374827715	With header, symbology character, and trailer

For each of these cases (and any other variations), this property must always be set to the string “5018374827715”, and **ScanDataType** must be set to SCAN\_SDT\_EAN13.

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

<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
<b>See Also</b>	“Device Input Model” on page 22

**ScanDataType Property R**

**Type**        **int**

**Remarks**    Holds the decoded bar code label type.

When **DecodeData** is false, this property is set to SCAN\_SDT\_UNKNOWN.  
 When **DecodeData** is true, the Device Service tries to determine the scan label type. The following label types are defined:

<b>Value</b>	<b>Label Type</b>
<i>One Dimensional Symbolologies</i>	
SCAN_SDT_UPCA	UPC-A
SCAN_SDT_UPCA_S	UPC-A with supplemental barcode
SCAN_SDT_UPCE	UPC-E
SCAN_SDT_UPCE_S	UPC-E with supplemental barcode
SCAN_SDT_UPCD1	UPC-D1
SCAN_SDT_UPCD2	UPC-D2
SCAN_SDT_UPCD3	UPC-D3
SCAN_SDT_UPCD4	UPC-D4
SCAN_SDT_UPCD5	UPC-D5
SCAN_SDT_EAN8	EAN 8 (= JAN 8)
SCAN_SDT_JAN8	JAN 8 (= EAN 8)
SCAN_SDT_EAN8_S	EAN 8 with supplemental barcode
SCAN_SDT_EAN13	EAN 13 (= JAN 13)
SCAN_SDT_JAN13	JAN 13 (= EAN 13)
SCAN_SDT_EAN13_S	EAN 13 with supplemental barcode
SCAN_SDT_EAN128	EAN-128
SCAN_SDT_TF	Standard (or discrete) 2 of 5
SCAN_SDT_ITF	Interleaved 2 of 5
SCAN_SDT_Codabar	Codabar
SCAN_SDT_Code39	Code 39
SCAN_SDT_Code93	Code 93
SCAN_SDT_Code128	Code 128
SCAN_SDT_OCRA	OCR "A"

Value	Label Type
SCAN_SDT_OCRB	OCR “B”

*Two Dimensional Symbologies*

SCAN_SDT_PDF417	PDF 417
SCAN_SDT_MAXICODE	MAXICODE

*Special Cases*

SCAN\_SDT\_OTHER If greater or equal to this type, then the Device Service has returned a non-JavaPOS defined symbology.

SCAN\_SDT\_UNKNOWN  
The Device Service cannot determine the barcode symbology. **ScanDataLabel** may not be properly formatted for the actual barcode type.

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

**Errors** A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

**See Also** “Device Input Model” on page 22

## Events

### DataEvent

**Interface** `jpos.events.DataListener`

**Method** `dataOccurred (DataEvent e);`

**Description** Notifies the application that input data from the Scanner (Bar Code Reader) is available.

**Properties** This event contains the following property:

Property	Type	Description
----------	------	-------------

<i>Status</i>	<i>int</i>	Always zero.
---------------	------------	--------------

**Remarks** The scanner data is placed in the **ScanData**, **ScanDataLabel** and **ScanDataType** properties prior to a **DataEvent** being sent to the application.

**See Also** “Events” on page 18

### DirectIOEvent

**Interface** `jpos.events.DirectIOListener`

**Method** `directIOOccurred (DirectIOEvent e);`

**Description** Provides Device Service information directly to the application. This event provides a means for a vendor-specific Scanner Device Service to provide events to the application that are not otherwise supported by the Device Control.

**Properties** This event contains the following properties:

Property	Type	Description
----------	------	-------------

<i>EventNumber</i>	<i>int</i>	Event number whose specific values are assigned by the Device Service.
--------------------	------------	--

<i>Data</i>	<i>int</i>	Additional numeric data. Specific values vary by the <i>EventNumber</i> and the Device Service. This property is settable.
-------------	------------	--

<i>Object</i>	<i>Object</i>	Additional data whose usage varies by the <i>EventNumber</i> and Device Service. This property is settable.
---------------	---------------	---

- Remarks** This event is to be used only for those types of vendor specific functions that are not otherwise described as part of the JavaPOS standard. Use of this event may restrict the application program from being used with other vendor's Scanner devices which may not have any knowledge of the Device Service's need for this event.
- See Also** "Events" on page 18, **directIO** Method

## ErrorEvent

- Interface** `jpos.events.ErrorListener`
- Method** `errorOccurred (ErrorEvent e);`
- Description** Notifies the application that a scanner device error has been detected and a suitable response by the application is necessary to process the error condition.
- Properties** This event contains the following properties:

Property	Type	Description
<i>ErrorCode</i>	<i>int</i>	Error code causing the error event. See list of <i>ErrorCodes</i> on page 16.
<i>ErrorCodeExtended</i>	<i>int</i>	Extended error code causing the error event. It may contain a Service-specific value.
<i>ErrorLocus</i>	<i>int</i>	Location of the error. See values below.
<i>ErrorResponse</i>	<i>int</i>	Error response, whose default value may be overridden by the application (i.e., this property is settable). See values below.

The *ErrorLocus* property has one of the following values:

Value	Meaning
JPOS_EL_INPUT	Error occurred while gathering or processing event-driven input. No input data is available.
JPOS_EL_INPUT_DATA	Error occurred while gathering or processing event-driven input, and some previously buffered data is available.



The application's error event listener may change *ErrorResponse* to one of the following values:

Value	Meaning
JPOS_ER_CLEAR	Clear the buffered input data. The error state is exited. Default when locus is JPOS_EL_INPUT.
JPOS_ER_CONTINUEINPUT	Use only when locus is JPOS_EL_INPUT_DATA. Acknowledges the error and directs the Device to continue processing. The Device remains in the error state, and will deliver additional <b>DataEvents</b> as directed by the <b>DataEventEnabled</b> property. When all input has been delivered and <b>DataEventEnabled</b> is again set to true, then another <b>ErrorEvent</b> is delivered with locus JPOS_EL_INPUT. Default when locus is JPOS_EL_INPUT_DATA.

**Remarks** Enqueued when an error is detected while trying to read scanner data. This event is not delivered until **DataEventEnabled** is true, so that proper application sequencing occurs.

**See Also** "Events" on page 18

## StatusUpdateEvent

**Interface** `jpos.events.StatusUpdateListener`

**Method** `statusUpdateOccurred (StatusUpdateEvent e);`

**Description** Notifies the application that there is a change in the power status of a Scanner device.

**Properties** This event contains the following property:

Property	Type	Description
<i>Status</i>	<i>int</i>	Reports a change in the power state of a Scanner device.

*Note that Release 1.3* added Power State Reporting with additional *Power reporting StatusUpdateEvent values*. See "StatusUpdateEvent" description on page 78.

**Remarks** Enqueued when the Scanner device detects a power state change.

**See Also** "Events" on page 18



# Signature Capture

## Summary

### Properties

<i>Common</i>	<i>Ver</i>	<i>Type</i>	<i>Access</i>	<i>May Use After</i>
<b>AutoDisable</b>		boolean	R/W	open
<b>CapPowerReporting</b>	1.3	int	R	open
<b>CheckHealthText</b>		String	R	open
<b>Claimed</b>		boolean	R	open
<b>DataCount</b>		int	R	open
<b>DataEventEnabled</b>		boolean	R/W	open
<b>DeviceEnabled</b>		boolean	R/W	open & claim
<b>FreezeEvents</b>		boolean	R/W	open
<b>OutputID</b>		int	R	<i>Not Supported</i>
<b>PowerNotify</b>	1.3	int	R/W	open
<b>PowerState</b>	1.3	int	R	open
<b>State</b>		int	R	--
<b>DeviceControlDescription</b>		String	R	--
<b>DeviceControlVersion</b>		int	R	--
<b>DeviceServiceDescription</b>		String	R	open
<b>DeviceServiceVersion</b>		int	R	open
<b>PhysicalDeviceDescription</b>		String	R	open
<b>PhysicalDeviceName</b>		String	R	open

<i>Specific</i>	<i>Ver</i>	<i>Type</i>	<i>Access</i>	<i>May Use After</i>
<b>CapDisplay</b>		boolean	R	open
<b>CapRealTimeData</b>		boolean	R	open
<b>CapUserTerminated</b>		boolean	R	open
<b>MaximumX</b>		int	R	open
<b>MaximumY</b>		int	R	open
<b>RawData</b>		byte[]	R	open, claim, & enable
<b>RealTimeDataEnabled</b>		boolean	R/W	open
<b>PointArray</b>		Point[]	R	open, claim, & enable

**Methods**

<i>Common</i>	<i>Ver</i>	<i>May Use After</i>
<b>open</b>		--
<b>close</b>		open
<b>claim</b>		open
<b>release</b>		open & claim
<b>checkHealth</b>		open, claim, & enable
<b>clearInput</b>		open & claim
<b>clearOutput</b>		<i>Not Supported</i>
<b>directIO</b>		open
 <i>Specific</i>		
<b>beginCapture</b>		open, claim, & enable
<b>endCapture</b>		open, claim, & enable

**Events**

<i>Name</i>	<i>Ver</i>	<i>May Occur After</i>
<b>DataEvent</b>		open, claim, & enable
<b>DirectIOEvent</b>	1.3	open & claim
<b>ErrorEvent</b>		open, claim, & enable
<b>OutputCompleteEvent</b>		<i>Not Supported</i>
<b>StatusUpdateEvent</b>	1.3	open, claim, & enable

## General Information

The Signature Capture Control's class name is "jpos.SignatureCapture".  
The device constants are contained in the class "jpos.SignatureCaptureConst".  
See "Package Structure" on page 40.

### Capabilities

The Signature Capture Control has the following capability:

- Obtains a signature captured by a signature capture device. The captured signature data is in the form of lines consisting of a series of points. Each point lies within the coordinate system defined by the resolution of the device, where (0, 0) is the upper-left point of the device, and (**MaximumX**, **MaximumY**) is the lower-right point. The signature line points are presented to the application by a **DataEvent** with a single array of line points

The Signature Capture Control may have the following additional capabilities:

- Provides a way for the user to terminate signature capture – that is, to tell the device that she or he has completed the signature.
- Displays form/data on the signature capture device.
- Returns the signature in "real time" as it is entered on the device. If this capability is true and has been enabled by application by setting the **RealTimeDataEnabled** property to true, then a series of **DataEvents** are enqueued, each with an array of one or more line points representing a partial signature.

### Model

The signature capture device usage model is:

- Open and claim the device.
- Enable the device and set the property **DataEventEnabled** to true.
- Begin capturing a signature by calling **beginCapture**. This method displays a form or data screen (if the device has a display) and enables the stylus.
- If the device is capable of supplying signature data in real time as the signature is entered (**CapRealTimeData** is true), and if **RealTimeDataEnabled** is true, the signature is presented to the application as a series of partial signature data events until the signature capture is terminated.
- If the device provides a way for the user to terminate the signature, then when the user terminates, a **DataEvent** is enqueued. Otherwise, the application must call **endCapture** to terminate the signature.
- Disable the device. If the device has a display, this also clears the display.

The Signature Capture Control follows the JavaPOS model for input devices:

- When input is received by the Device Service, it enqueues a **DataEvent**.
- If **AutoDisable** is true, then the Device automatically disables itself when a **DataEvent** is enqueued. However, note that setting **AutoDisable** probably is not very useful for the Signature Capture control. If **RealTimeDataEnabled** is true, then **AutoDisable** does not make sense. If **RealTimeDataEnabled** is false, then the pacing of signatures is controlled by the application via the **beginCapture** method. It is probably in the best interest of the application not to use the **AutoDisable** property for this device class.
- A queued **DataEvent** can be delivered to the application when the property **DataEventEnabled** is true. Just before delivering this event, data is copied into properties, and further data events are disabled by setting **DataEventEnabled** to false. This causes subsequent input data to be enqueued while the application processes the current input and associated properties. When the application has finished the current input and is ready for more data, it reenables events by setting **DataEventEnabled** to true.
- An **ErrorEvent** (or events) are enqueued if the an error occurs while gathering or processing input, and is delivered to the application when **DataEventEnabled** is true.
- The **DataCount** property may be read to obtain the number of queued **DataEvents**.
- All queued input may be deleted by calling **clearInput**.

Deviations from the JavaPOS model for input devices are:

- The capture of signature data begins when **beginCapture** is called.
- If signature capture is terminated by calling **endCapture**, then no **DataEvent** will be enqueued.

## Device Sharing

The Signature Capture is an exclusive-use device, as follows:

- The application must claim the device before enabling it.
- The application must claim and enable the device before calling methods that manipulate the device or before changing some writable properties.
- See the “Summary” table for precise usage prerequisites.

## Properties

### CapDisplay Property R

Type	<b>boolean</b>
Remarks	If true, the device is able to display a form or data entry screen.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapRealTimeData Property R

Type	<b>boolean</b>
Remarks	If true, the device is able to supply signature data as the signature is being captured (“real time”).  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapUserTerminated Property R

Type	<b>boolean</b>
Remarks	If true, the user is able to terminate signature capture by checking a completion box, pressing a completion button, or performing some other interaction with the device. If false, the application must end signature capture by calling the <b>endCapture</b> method.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

**DeviceEnabled Property R/W (Common)**

<b>Type</b>	<b>boolean</b>
<b>Remarks</b>	If true, the signature capture device is enabled.  If <b>CapDisplay</b> is true, then the display screen of the device is cleared.  This property is initialized to false by the <b>open</b> method.
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

**MaximumX Property R**

<b>Type</b>	<b>int</b>
<b>Remarks</b>	Holds the maximum horizontal coordinate of the signature capture device. It must be less than 65,536.  This property is initialized by the <b>open</b> method.
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

**MaximumY Property R**

<b>Type</b>	<b>int</b>
<b>Remarks</b>	Holds the maximum vertical coordinate of the signature capture device. It must be less than 65,536.  This property is initialized by the <b>open</b> method.
<b>Errors</b>	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.



## PointArray Property R

Type	<code>java.awt.Point[]</code>
Remarks	<p>Holds the signature captured from the device. It consists of an array of <math>(x, y)</math> coordinate points. Each point is represented by four characters: <math>x</math> (low 8 bits), <math>x</math> (high 8 bits), <math>y</math> (low 8 bits), <math>y</math> (high 8 bits).</p> <p>A special point value is <math>(0xFFFF, 0xFFFF)</math> which indicates the end of a line (that is, a pen lift). Almost all signatures are comprised of more than one line.</p> <p>If <b>RealTimeDataEnabled</b> is false, then this property contains the entire captured signature. If <b>RealTimeDataEnabled</b> is true, then this property contains at least one point of the signature. The actual number of points delivered at one time is implementation dependent. The points from multiple data events are logically concatenated to form the entire signature, such that the last point from a data event is followed immediately by the first point of the next data event.</p> <p>The point representation definition is the same regardless of whether the signature is presented as a single <b>PointArray</b>, or as a series of real time <b>PointArrays</b>.</p> <p>Reconstruction of the signature using the points is accomplished by beginning a line from the first point in the signature to the second point, then to the third, and so on. When an end-of-line point is encountered, the drawing of the line ends, and the next line is drawn beginning with the next point. An end-of-line point is assumed (but need not be present in <b>PointArray</b>) at the end of the signature.</p> <p>This property is set prior to a <b>DataEvent</b> being sent to the application or by the <b>endCapture</b> method.</p>
Errors	A <code>JposException</code> may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>RawData</b> Property

## RawData Property R

Type	<code>byte[]</code>
Remarks	<p>Holds the signature captured from the device in a device-specific format.</p> <p>This data is often in a compressed form to minimize signature storage requirements. Reconstruction of the signature from this data requires device-specific processing.</p> <p>This property is set prior to a <b>DataEvent</b> being sent to the application or by the <b>endCapture</b> method.</p>
Errors	A <code>JposException</code> may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.
See Also	<b>PointArray</b> Property

**RealTimeDataEnabled Property R/W**

<b>Type</b>	<b>boolean</b>				
<b>Remarks</b>	<p>If true and <b>CapRealTimeData</b> is true, a series of partial signature data events is enqueued as the signature is captured until signature capture is terminated. Otherwise, the captured signature is enqueued as a single <b>DataEvent</b> when signature capture is terminated.</p> <p>Setting <b>RealTimeDataEnabled</b> will not cause any change in system behavior until a subsequent <b>beginCapture</b> method is performed. This prevents confusion regarding what would happen if it were modified between a <b>beginCapture</b> - <b>endCapture</b> pairing.</p> <p>This property is initialized to false by the <b>open</b> method.</p>				
<b>Errors</b>	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p> <p>Some possible values of the exception’s <i>ErrorCode</i> property are:</p> <table> <thead> <tr> <th><b>Value</b></th> <th><b>Meaning</b></th> </tr> </thead> <tbody> <tr> <td>JPOS_E_ILLEGAL</td> <td>Cannot set to true because <b>CapRealTimeData</b> is false.</td> </tr> </tbody> </table>	<b>Value</b>	<b>Meaning</b>	JPOS_E_ILLEGAL	Cannot set to true because <b>CapRealTimeData</b> is false.
<b>Value</b>	<b>Meaning</b>				
JPOS_E_ILLEGAL	Cannot set to true because <b>CapRealTimeData</b> is false.				

## Methods

### beginCapture Method

**Syntax**      **void beginCapture (String *formName*) throws JposException;**

Parameter	Description
<i>formName</i>	The parameter contains the JSD subkey name for obtaining form or data screen information for display on the device screen.

**Remarks**      Starts capturing a signature.

If **CapDisplay** is true, then *formName* is used to find information about the form or data screen to be displayed. The JSD key

***/device/JavaPOS/SignatureCapture/DeviceName/FormName***

is accessed to get this information. DeviceName is the Device Service's Device Name key. The format and features of each signature capture device's form/data screen varies widely and is often built with proprietary tools. Therefore, this key's data and additional values and data under this key contain information that varies by Device Service. Typically, the JSD key's data is set to a form/data screen file name, and extra JSD values and data are set as needed to control its display.

After displaying the form or data screen, when applicable, the signature capture stylus is enabled.

**Errors**      A JposException may be thrown when this method is invoked. For further information, see "Exceptions" on page 15.

Some possible values of the exception's *ErrorCode* property are:

Value	Meaning
JPOS_E_NOEXIST	<i>formName</i> was not found.

**See Also**      **endCapture** Method

**endCapture Method**

**Syntax**      **void endCapture () throws JposException;**

**Remarks**      Stops (terminates) capturing a signature.

If **RealTimeDataEnabled** is false and a signature was captured, then it is placed in the properties **PointArray** and **RawData**. If no signature was captured, then **PointArray** and **RawData** are set to a length of zero.

If **RealTimeDataEnabled** is true and there are signature points remaining which have not been delivered to the application by a **DataEvent**, then the remaining signature is placed into the properties **PointArray** and **RawData**. If no signature was captured or all signature points have been delivered to the application, then **PointArray** and **RawData** are set to a length of zero.

**Errors**      A JposException may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

<b>Value</b>	<b>Meaning</b>
JPOS_E_ILLEGAL	Signature capture was not in progress.

**See Also**      **beginCapture** Method, **DataEvent**

## Events

### DataEvent

**Interface** `jpos.events.DataListener`

**Method** `dataOccurred (DataEvent e);`

**Description** Notifies the application that input data is available.

**Properties** This event contains the following property:

Property	Type	Description
<i>Status</i>	<i>int</i>	Non-zero if the user has entered a signature before terminating capture. Zero if the user terminated capture with no signature.

**Remarks** This event can only be enqueued if the user can terminate signature capture – that is, if **CapUserTerminated** is true or **RealTimeDataEnabled** is true.

The properties **PointArray** and **RawData** are set to appropriate values prior to a **DataEvent** being sent to the application.

**See Also** `endCapture` Method

### DirectIOEvent

**Interface** `jpos.events.DirectIOListener`

**Method** `directIOOccurred (DirectIOEvent e);`

**Description** Provides Device Service information directly to the application. This event provides a means for a vendor-specific Signature Capture Device Service to provide events to the application that are not otherwise supported by the Device Control.

**Properties** This event contains the following properties:

Property	Type	Description
<i>EventNumber</i>	<i>int</i>	Event number whose specific values are assigned by the Device Service.
<i>Data</i>	<i>int</i>	Additional numeric data. Specific values vary by the <i>EventNumber</i> and the Device Service. This property is settable.
<i>Object</i>	<i>Object</i>	Additional data whose usage varies by the <i>EventNumber</i> and Device Service. This property is settable.

- Remarks** This event is to be used only for those types of vendor specific functions that are not otherwise described as part of the JavaPOS standard. Use of this event may restrict the application program from being used with other vendor's Signature Capture devices which may not have any knowledge of the Device Service's need for this event.
- See Also** "Events" on page 18, **directIO** Method

## ErrorEvent

- Interface** `jpos.events.ErrorListener`
- Method** `errorOccurred (ErrorEvent e);`
- Description** Notifies the application that a Signature Capture device error has been detected and a suitable response by the application is necessary to process the error condition.
- Properties** This event contains the following properties:

Property	Type	Description
<i>ErrorCode</i>	<i>int</i>	Error Code causing the error event. See the list of <i>ErrorCodes</i> on page 16.
<i>ErrorCodeExtended</i>	<i>int</i>	Extended Error Code causing the error event. This may contain a Service-specific value.
<i>ErrorLocus</i>	<i>int</i>	Location of the error. See values below.
<i>ErrorResponse</i>	<i>int</i>	Error response, whose default value may be overridden by the application (i.e., this property is settable). See values below.

The *ErrorLocus* property has one of the following values:

Value	Meaning
JPOS_EL_INPUT	Error occurred while gathering or processing event-driven input. No input data is available.
JPOS_EL_INPUT_DATA	Error occurred while gathering or processing event-driven input, and some previously buffered data is available. (Very unlikely – see <b>Remarks</b> .)

The application's error event listener may change *ErrorResponse* to one of the following values:

Value	Meaning
JPOS_ER_CLEAR	Clear the buffered input data. The error state is exited. Default when locus is JPOS_EL_INPUT.
JPOS_ER_CONTINUEINPUT	Use only when locus is JPOS_EL_INPUT_DATA. Acknowledges the error and directs the Device to continue processing. The Device remains in the error state, and will deliver additional <b>DataEvents</b> as directed by the <b>DataEventEnabled</b> property. When all input has been delivered and <b>DataEventEnabled</b> is again set to true, then another <b>ErrorEvent</b> is enqueued with locus JPOS_EL_INPUT. Default when locus is JPOS_EL_INPUT_DATA.

**Remarks** Enqueued when an error is detected while trying to read signature capture data. This event is not delivered until **DataEventEnabled** is true, so that proper application sequencing occurs.

**See Also** “Device Input Model” on page 22, “Device States” on page 30

## StatusUpdateEvent

**Interface** `jpos.events.StatusUpdateListener`

**Method** `statusUpdateOccurred (StatusUpdateEvent e);`

**Description** Notifies the application that there is a change in the power status of a Signature Capture device.

**Properties** This event contains the following property:

Property	Type	Description
<i>Status</i>	<i>int</i>	Reports a change in the power state of a Signature Capture device.

*Note that Release 1.3* added Power State Reporting with additional *Power reporting StatusUpdateEvent values*. See “StatusUpdateEvent” description on page 78.

**Remarks** Enqueued when the Signature Capture device detects a power state change.

**See Also** “Events” on page 18





# Tone Indicator

## Summary

### Properties

<i>Common</i>	<i>Ver</i>	<i>Type</i>	<i>Access</i>	<i>May Use After</i>
<b>AutoDisable</b>		boolean	R/W	<i>Not Supported</i>
<b>CapPowerReporting</b>	1.3	int	R	open
<b>CheckHealthText</b>		String	R	open
<b>Claimed</b>		boolean	R	open
<b>DataCount</b>		int	R	<i>Not Supported</i>
<b>DataEventEnabled</b>		boolean	R/W	<i>Not Supported</i>
<b>DeviceEnabled</b>		boolean	R/W	open
<b>FreezeEvents</b>		boolean	R/W	open
<b>OutputID</b>		int	R	open
<b>PowerNotify</b>	1.3	int	R/W	open
<b>PowerState</b>	1.3	int	R	open
<b>State</b>		int	R	--
<b>DeviceControlDescription</b>		String	R	--
<b>DeviceControlVersion</b>		int	R	--
<b>DeviceServiceDescription</b>		String	R	open
<b>DeviceServiceVersion</b>		int	R	open
<b>PhysicalDeviceDescription</b>		String	R	open
<b>PhysicalDeviceName</b>		String	R	open

<i>Specific</i>	<i>Ver</i>	<i>Type</i>	<i>Access</i>	<i>May Use After</i>
<b>AsyncMode</b>		boolean	R/W	open
<b>CapPitch</b>		boolean	R	open
<b>CapVolume</b>		boolean	R	open
<b>Tone1Pitch</b>		int	R/W	open
<b>Tone1Volume</b>		int	R/W	open
<b>Tone1Duration</b>		int	R/W	open
<b>Tone2Pitch</b>		int	R/W	open
<b>Tone2Volume</b>		int	R/W	open
<b>Tone2Duration</b>		int	R/W	open
<b>InterToneWait</b>		int	R/W	open

**Methods**

<i>Common</i>	<i>Ver</i>	<i>May Use After</i>
<b>open</b>		--
<b>close</b>		open
<b>claim</b>		open
<b>release</b>		open & claim
<b>checkHealth</b>		open & enable
<b>clearInput</b>		<i>Not Supported</i>
<b>clearOutput</b>		open & enable
<b>directIO</b>		open
<i>Specific</i>		
<b>sound</b>		open & enable
<b>soundImmediate</b>		open & enable

*Note:* Also requires that no other application has claimed the tone indicator.

**Events**

<i>Name</i>	<i>Ver</i>	<i>May Occur After</i>
<b>DataEvent</b>		<i>Not Supported</i>
<b>DirectIOEvent</b>	1.3	open & enable
<b>ErrorEvent</b>		open & enable
<b>OutputCompleteEvent</b>		open & enable
<b>StatusUpdateEvent</b>	1.3	open & enable

## General Information

The Tone Indicator Control's class name is "jpos.ToneIndicator".  
The device constants are contained in the class "jpos.ToneIndicatorConst".  
See "Package Structure" on page 40.

### Capabilities

The Tone Indicator Control has the following capabilities:

- Sound a tone device, which may be the PC or NC system speaker or another hardware device. In many cases the PC or NC speaker will not be available or in a position that is inaudible to the operator.
- Sound a two-tone indicator, providing simple pitch and volume control.
- Provide a synchronous one-shot indicator, similar to an Operating System's Beep function.

### Model

The Tone Indicator device is for use when the POS hardware platform provides such capabilities external to the PC or NC standard speaker. Many POS systems have such devices, for example the ICL 92R keyboard, so that an indicator is always present at the point of sale.

This device supports a two-tone sound so that "siren" tones can be produced. The indicator is in general also started asynchronously so applications may perform other functions while waiting for the user to acknowledge the tone. There are also options to start the tone asynchronously with no count, so it runs forever, and be stopped when running.

When the indicator is started asynchronously then an **OutputCompleteEvent** is enqueued when all the tones have been played. This allows the application to know that the tone has stopped. For example when the cash drawer is opened the tone could be started, quietly for a given number of cycles. If the cash drawer is closed then the tone is stopped explicitly by the application, if not then the **OutputCompleteEvent** allows us to alter the prompt to the operator and possibly restart the tone a little louder.

The Tone Indicator follows the JavaPOS model for output devices. Asynchronous output is handled as follows:

- The Device buffers the request, sets **OutputID** to an identifier for this request, and returns as soon as possible. When the request completes successfully, an **OutputCompleteEvent** is enqueued. A parameter of this event contains the **OutputID** of the completed request.

The **Sound** method will not return an error status due to a hardware problem. These errors will only be reported by an **ErrorEvent**. An error status is returned only if the control is not claimed and enabled, a parameter is invalid, or the request cannot be enqueued. The first two error cases are due to an application error, while the last is a serious system resource exception.

- If an error occurs while performing an asynchronous request, an **ErrorEvent** is enqueued.
- Asynchronous output is performed on a first-in first-out basis.
- All output buffered may be deleted by calling **clearOutput**. **OutputCompleteEvents** will not be delivered for cleared output. This method also stops any output that may be in progress (when possible).

## Device Sharing

The Tone Indicator is a sharable device. Its device sharing rules are:

- After opening and enabling the device, the application may access all properties, methods, and Enqueued **StatusUpdateEvents**.
- If more than one application has opened and enabled the device, each of these applications may access its properties and methods. **StatusUpdateEvents** will be delivered to all applications that are using the device and have registered to receive the event.
- If one application claims the tone indicator, then only that application may call **sound** and **soundImmediate**. Use of this feature will effectively restrict the tone indicator to the main application if that application claims the device at startup.
- The application that initiates asynchronous sounds is the only one that receives the corresponding **OutputCompleteEvents** and **ErrorEvents**.
- If a scenario exists such that an application is playing a sound and a separate application legally claims the device and plays a sound, then the sound being played from the first application will be interrupted. If the first application is in the midst of a synchronous **sound** method, it will return **JPOS\_E\_CLAIMED** from the method call. If the application has issued an asynchronous **sound** method, then no consistent reporting mechanism is possible and the first sound is simply terminated.
- See the “Summary” table for precise usage prerequisites.

## Properties

### AsyncMode Property R/W

Type	<b>boolean</b>
Remarks	If true, the <b>sound</b> method will be performed asynchronously. If false, tones are generated synchronously.  This property is initialized to false by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapPitch Property R

Type	<b>boolean</b>
Remarks	If true, the hardware tone generator has the ability to vary the pitch of the tone.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

### CapVolume Property R

Type	<b>boolean</b>
Remarks	If true, the hardware tone generator has the ability to vary the volume of the tone.  This property is initialized by the <b>open</b> method.
Errors	A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.

**InterToneWait Property R/W**

<b>Type</b>	<b>int</b>				
<b>Remarks</b>	<p>Holds the number of milliseconds of silence between tone-1 and tone-2. If a gap is required after tone-2 but before a repeat of tone-1, then set the <b>sound</b> parameter <i>interSoundWait</i>.</p> <p>This property is initialized to zero by the <b>open</b> method.</p>				
<b>Errors</b>	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p> <p>Some possible values of the exception’s <i>ErrorCode</i> property are:</p> <table border="1"> <thead> <tr> <th><b>Value</b></th> <th><b>Meaning</b></th> </tr> </thead> <tbody> <tr> <td>JPOS_E_ILLEGAL</td> <td>A negative value was specified.</td> </tr> </tbody> </table>	<b>Value</b>	<b>Meaning</b>	JPOS_E_ILLEGAL	A negative value was specified.
<b>Value</b>	<b>Meaning</b>				
JPOS_E_ILLEGAL	A negative value was specified.				

**Tone1Duration Property R/W**

<b>Type</b>	<b>int</b>
<b>Remarks</b>	<p>Holds the duration of the first tone in milliseconds. A value of zero or less will cause this tone not to sound.</p> <p>This property is initialized to zero by the <b>open</b> method.</p>
<b>Errors</b>	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>

**Tone1Pitch Property R/W**

<b>Type</b>	<b>int</b>
<b>Remarks</b>	<p>Holds the pitch or frequency of the first tone in hertz. A value of zero or less will cause this tone not to sound.</p> <p>If the device does not support user-defined pitch (<b>CapPitch</b> is false), then any value greater than zero indicates that the tone indicator uses its default value.</p> <p>This property is initialized to zero by the <b>open</b> method.</p>
<b>Errors</b>	<p>A JposException may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>

**Tone1Volume Property R/W**

<b>Type</b>	<b>int</b>
<b>Remarks</b>	<p>Holds the volume of the first tone in percent of the device's capability, where 0 (or less) is silent and 100 (or more) is maximum.</p> <p>If the device does not support user-defined volume (<b>CapVolume</b> is false), then any value greater than zero indicates that the tone indicator uses its default value.</p> <p>This property is initialized to 100 by the <b>open</b> method.</p>
<b>Errors</b>	<p>A JposException may be thrown when this property is accessed. For further information, see "Exceptions" on page 15.</p>

**Tone2Duration Property R/W**

<b>Type</b>	<b>int</b>
<b>Remarks</b>	<p>Holds the duration of the second tone in milliseconds. A value of zero or less will cause this tone not to sound.</p> <p>This property is initialized to zero by the <b>open</b> method.</p>
<b>Errors</b>	<p>A JposException may be thrown when this property is accessed. For further information, see "Exceptions" on page 15.</p>

**Tone2Pitch Property R/W**

<b>Type</b>	<b>int</b>
<b>Remarks</b>	<p>Holds the pitch or frequency of the second tone in hertz. A value of zero or less will cause this tone not to sound.</p> <p>If the device does not support user-defined pitch (<b>CapPitch</b> is false), then any value greater than zero indicates that the tone indicator uses its default value.</p> <p>This property is initialized to zero by the <b>open</b> method.</p>
<b>Errors</b>	<p>A JposException may be thrown when this property is accessed. For further information, see "Exceptions" on page 15.</p>



**Tone2Volume Property R/W**

<b>Type</b>	<b>int</b>
<b>Remarks</b>	<p>Holds the volume of the second tone in percent of the device's capability, where 0 (or less) is silent and 100 (or more) is maximum.</p> <p>If the device does not support user-defined volume (<b>CapVolume</b> is false), then any value greater than zero indicates that the tone indicator uses its default value.</p> <p>This property is initialized to 100 by the <b>open</b> method.</p>
<b>Errors</b>	<p>A <code>JposException</code> may be thrown when this property is accessed. For further information, see “Exceptions” on page 15.</p>

## Methods

### sound Method

**Syntax**      **void sound (int *numberOfCycles*, int *interSoundWait*) throws JposException;**

Parameter	Description
<i>numberOfCycles</i>	The number of cycles to sound the indicator device. If JPOS_FOREVER, then start the indicator sounding, and repeat continuously.
<i>interSoundWait</i>	When <i>numberOfCycles</i> is not one, then pause for <i>interSoundWait</i> milliseconds before repeating the tone cycle (before playing tone-1 again).

**Remarks**      Sounds the indicator device, or start it sounding asynchronously.

This method is performed synchronously if **AsyncMode** is false, and asynchronously if **AsyncMode** is true.

The duration of an indicator cycle is:

**Tone1Duration** property +  
**InterToneWait** property +  
**Tone2Duration** property +  
*interSoundWait* parameter (except on the last tone cycle)

After the tone indicator has started an asynchronous sound, then the sound may be stopped by using one of the following methods. (When a *numberOfCycles* value of JPOS\_FOREVER was used to start the sound, then the application must use one of these to stop the continuous sounding of the tones.)

- **clearOutput**
- **soundImmediate**

**Errors** A `JposException` may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

Some possible values of the exception’s *ErrorCode* property are:

Value	Meaning
JPOS_E_CLAIMED	Indicates that another application has claimed the device and has taken over the tone device causing the sound from this method to be interrupted (can only be returned if <b>AsyncMode</b> is false.)
JPOS_E_ILLEGAL	One of the following errors occurred: <ul style="list-style-type: none"> <li>• <i>numberOfCycles</i> is neither a positive, non-zero value nor JPOS_FOREVER.</li> <li>• <i>numberOfCycles</i> is JPOS_FOREVER when <b>AsyncMode</b> is false.</li> <li>• A negative <i>interSoundWait</i> was specified</li> <li>• A negative <i>interToneWait</i> was specified</li> </ul>

### soundImmediate Method

**Syntax** `void soundImmediate () throws JposException;`

**Remarks** Sounds the hardware tone generator once, synchronously. Both tone-1 and tone-2 are sounded using **InterToneWait**.

If asynchronous output is outstanding, then it is terminated before playing the immediate sound (as if **clearOutput** were called). This method is primarily intended for use in exception conditions when asynchronous output is outstanding, such as within an error event handler.

**Errors** A `JposException` may be thrown when this method is invoked. For further information, see “Exceptions” on page 15.

## Events

### DirectIOEvent

**Interface** `jpos.events.DirectIOListener`

**Method** `directIOOccurred (DirectIOEvent e);`

**Description** Provides Device Service information directly to the application. This event provides a means for a vendor-specific Tone Indicator Device Service to provide events to the application that are not otherwise supported by the Device Control.

**Properties** This event contains the following properties:

Property	Type	Description
<i>EventNumber</i>	<i>int</i>	Event number whose specific values are assigned by the Device Service.
<i>Data</i>	<i>int</i>	Additional numeric data. Specific values vary by the <i>EventNumber</i> and the Device Service. This property is settable.
<i>Object</i>	<i>Object</i>	Additional data whose usage varies by the <i>EventNumber</i> and Device Service. This property is settable.

**Remarks** This event to be used only for those types of vendor specific functions that are not otherwise described as part of the JavaPOS standard. Use of this event may restrict the application program from being used with other vendor's Tone Indicator devices which may not have any knowledge of the Device Service's need for this event.

**See Also** "Events" on page 18, **directIO** Method

## ErrorEvent

<b>Interface</b>	<b>jpos.events.ErrorListener</b>
<b>Method</b>	<b>errorOccurred (ErrorEvent e);</b>
<b>Description</b>	Notifies the application that an error has been detected at the device and a suitable response is necessary to process the error condition.
<b>Properties</b>	This event contains the following properties:

Property	Type	Description
<i>ErrorCode</i>	<i>int</i>	Error Code causing the error event. See list of <i>ErrorCodes</i> on page 16.
<i>ErrorCodeExtended</i>	<i>int</i>	Extended Error Code causing the error event. These values are device category specific.
<i>ErrorLocus</i>	<i>int</i>	Location of the error. See values below.
<i>ErrorResponse</i>	<i>int</i>	Error response, whose default value may be overridden by the application (i.e., this property is settable). See values below.

The *ErrorLocus* property has one of the following values:

Value	Meaning
JPOS_EL_OUTPUT	Error occurred while processing asynchronous output.

The application's error event listener may change *ErrorResponse* to one of the following values:

Value	Meaning
JPOS_ER_RETRY	Retry the asynchronous output. The error state is exited. This is the default value.
JPOS_ER_CLEAR	Clear the asynchronous output data. The error state is exited.

**Remarks** This event is enqueued when an error is detected and the Device's **State** transitions into the error state. This event is not delivered until **DataEventEnabled** is true, so that proper application sequencing occurs.

**See Also** "Device Output Models" on page 25, "Device States" on page 30, "ErrorCode" on page 16

## OutputCompleteEvent

**Interface**    **jpos.events.OutputCompleteListener**

**Method**      **outputCompleteOccurred (OutputCompleteEvent e);**

**Description**    Notifies the application that the queued output request associated with the *OutputID* property has completed successfully.

**Properties**      This event contains the following property:

Property	Type	Description
<i>OutputID</i>	<i>int</i>	The ID number of the asynchronous output request that is complete.

**Remarks**      This event is enqueued after the request's data has been both sent and the Device Service has confirmation that it was processed by the device successfully.

**See Also**      "Device Output Models" on page 25

## StatusUpdateEvent

**Interface**    **jpos.events.StatusUpdateListener**

**Method**      **statusUpdateOccurred (StatusUpdateEvent e);**

**Description**    Notifies the application that there is a change in the power status of a Tone Indicator device.

**Properties**      This event contains the following property:

Property	Type	Description
<i>Status</i>	<i>int</i>	Reports a change in the power state of a Tone Indicator device.

*Note that Release 1.3* added Power State Reporting with additional *Power reporting StatusUpdateEvent values*. See "StatusUpdateEvent" description on page 78.

**Remarks**      Enqueued when the Tone Indicator device detects a power state change.

**See Also**      "Events" on page 18

# Change History

## Release 1.3

Release 1.3 adds additional device classes, a few additional APIs, and some corrections. Release 1.3 is a superset of Release 1.2.

<u>Section</u>	<u>Change</u>
General	Modify the use of the term event “firing.” Use “enqueue” and “deliver” appropriately to describe event firing.
Bump Bar	New device: Add information in several locations, plus Bump Bar chapter and interface files.
Fiscal Printer	New device: Add information in several locations, plus Fiscal Printer chapter and interface files.
PIN Pad	New device: Add information in several locations, plus PIN Pad chapter and interface files.
Remote Order Display	New device: Add information in several locations, plus Remote Order Display chapter and interface files.
Several places	Relax <b>ErrorEvent</b> “retry” response to allow its use with some input devices.
Introduction <b>Events</b>	Clarify effect of the top event being blocked.
Introduction <b>Input Model</b>	Add details concerning enqueueing and delivering <b>ErrorEvents</b> . Add description of asynchronous input.
Introduction <b>Device Power Reporting Model</b>	Add this section.
Common <b>CapPowerReporting, PowerNotify, PowerState</b> properties	Add these sections.

Common <b>ErrorCode</b> property	Generalize the meaning of JPOS_E_BUSY.
Common <b>StatusUpdateEvent</b>	Add power state reporting information. Change parameter name from <i>Data</i> to <i>Status</i> .
Every Device	Add power reporting properties to Summary section. Add <b>StatusUpdateEvent</b> support (if previously not reported). Add power reporting reference to existing <b>StatusUpdateEvent</b> descriptions.
MSR <b>DecodeData</b>	Add “raw format” description and column to track data table.
MSR <b>ExpirationDate</b>	Specify the format.
MSR <b>TrackxData</b>	Specify that data excludes the sentinels and LRC. Add that decoding occurs when <b>DecodeData</b> is true.
MSR <b>ErrorEvent</b>	Clarify that <b>DataCount</b> and <b>AutoDisable</b> are not relevant for MSR error events.
POSPrinter <b>XxxLineChars</b>	Add implementation recommendations.
POSPrinter <b>printTwoNormal</b>	Clarify the meaning of the <i>stations</i> parameter, including the addition of new constants.
Scale	Add the following features: <ul style="list-style-type: none"> <li>◆ Asynchronous input. Property <b>AsyncMode</b>. Method <b>clearInput</b>, updates to <b>readWeight</b>. Events <b>DataEvent</b> and <b>ErrorEvent</b>.</li> <li>◆ Display of text. Properties <b>CapDisplayText</b>, <b>MaxDisplayTextChars</b>. Method <b>displayText</b>.</li> <li>◆ Price calculation. Properties <b>CapPriceCalculating</b>, <b>SalesPrice</b>, <b>UnitPrice</b>.</li> <li>◆ Tare weight. Properties <b>CapTareWeight</b>, <b>TareWeight</b>.</li> <li>◆ Scale zeroing. Property <b>CapZeroScale</b>. Method <b>zeroScale</b>.</li> </ul>



Tone Indicator <b>Summary</b> and <b>General Information's Device Sharing</b>	Consistently specify that Tone Indicator is a sharable device.
JposConst.java interface files	Add <b>CapPowerReporting</b> , <b>PowerState</b> , and <b>PowerNotify</b> properties. Add <b>StatusUpdateEvent</b> power reporting values.
POSPrinterConst.java interface files	Add new <b>printTwoNormal</b> station constants.
Throughout	Correct some editing errors.

## Release 1.4

Release 1.4 added the additional peripheral device, Credit Authorization Terminal (CAT). This device, as specified, is currently only used in the Japanese POS markets.

Addition of this device required re-ordering the chapters and modifications to the Table of Contents. Other minor changes to the standard are as noted below.

Release 1.4 is a superset of Release 1.3.

<u>Section</u>	<u>Change</u>
General	Update the "Package Structure" on page40 to include CAT device; update the files to correct some erroneous references to OPOS.
Fiscal Printer	Add clarification to when the <b>ErrorStation</b> property is valid.
POS Printer	Add clarification to when the <b>ErrorStation</b> property is valid.
Appendix B	Add clarification to the "Events" section description.
Throughout	Correct interface name to <b>jpos.events.OutputCompleteListener</b> . Correct minor spelling errors.

## Release 1.5

Release 1.5 adds two additional peripheral devices: Pointcard Reader Writer and POSPower, incorporates additional clarifications to the standard, adds a few new additional APIs for some of the existing devices, and makes some corrections to insure consistency in the device descriptions. Release 1.5 is a superset of Release 1.4.

<u>Section</u>	<u>Change</u>
Throughout	Correct notation for Java Unicode to “\uxxxx”
General	<p>Add clarification to when the Device exits the <b>Error</b> state.</p> <p>Remove the JPS documentation from the standard. The JPS implementation has been replaced with the JCL mechanism for locating and maintaining the Java Device Services. Revised the tables and diagrams as necessary to reflect these changes.</p> <p>Update the Standard and the Package Structure to reflect the additional new devices added to this version.</p>
Common Properties, Methods, and Events	Modified General section to reflect JDK version dependencies.
Bump Bar	Add clarification that this Device can be both an input and an output device.
Cash Changer	Add the necessary properties ( <b>DataCount</b> , <b>DataEventEnabled</b> , <b>CapDeposit</b> , <b>CapDepositDataEvent</b> , <b>CapPauseDeposit</b> , <b>CapRepayDeposit</b> , <b>DepositAmount</b> , <b>DepositCashList</b> , <b>DepositCodeList</b> , <b>DepositCounts</b> , <b>DepositStatus</b> ), methods ( <b>beginDeposit</b> , <b>endDeposit</b> , <b>fixDeposit</b> , <b>pauseDeposit</b> ) and events ( <b>DataEvent</b> ) for this device to optionally be able to handle cash acceptance.

---

Cash Drawer	Added new property, <b>CapStatusMultiDrawerDetect</b> to improve status reporting in multiple cash drawer environments.
CAT Changer	Correct the properties section to reflect the correct data type for <b>TransactionType</b> (an integer) and <b>TransactionNumber</b> (a String); other minor corrections to fix typographical errors.
Coin Dispenser	No Changes
Fiscal Printer	Added Russia to list of countries in the <b>CountryCode</b> property.  Added note to clarify that Currency value is specified to be four decimal places.  Changed the properties <b>CountryCode</b> , <b>ErrorOutID</b> , <b>PrinterState</b> , <b>QuantityDecimalPlaces</b> , and <b>QuantityLength</b> to clarify when the parameters are Initialized.  Corrected <b>DuplicateReceipt</b> to show that it is a R/W Property.
Hard Totals	No Changes
Keylock	No Changes
Line Display	Clarify properties <b>CharacterSet</b> and <b>CharacterSetList</b> to indicate when they are initialized and to what values they may be set.
MICR	Added clarification to description of Model concerning the availability of parsed data.  Clarify number of digits for <b>BankNumber</b> as specified by ABA Standard, Thomson Financial Publishing Inc.
MSR	Added properties <b>CapTransmitSentinels</b> , <b>Track4Data</b> , and <b>TransmitSentinels</b> to enhance the features that may be available in a global MSR device.  Updated the status byte definitions for the <b>DataEvent</b> event.

---

Pin Pad	Added the <b>Track4Data</b> property. Clarify that <b>Track1Data</b> , <b>Track2Data</b> , <b>Track3Data</b> , and <b>Track4Data</b> is assumed to be decoded data if a successful read takes place.
Pointcard Reader Writer	New device classification added to the standard. This device is used primarily in Asian markets.
POS Keyboard	<b>CapKeyUp</b> property type corrected from Long to boolean
POS Power	New device classification added to the standard to allow for systems that have the capability to report and manage alternative mains power (UPS type devices).
POS Printer	Revise this device classification to include properties, methods, and events to add multi-color printing, both side printing for documents such as checks, and marked paper and sensing capability for special POS printer forms handling. This section had significant changes to the General Information section as well to help clarify standard to reduce the possibility of creating a Device Service that does not meet the intent of the standard.
ROD	Clarify model remarks to indicate that this device can be both an output device and an input device.  Clarify General Model description explaining how Applications can manage and control the Remote Order Displays.  Clarify to indicate that <b>ErrorUnits</b> and <b>ErrorString</b> are updated instead by <u>synchronous</u> broadcast method.  Clarify what value the <b>CurentUnitID</b> property is initialized.
Scale	Clarify the properties <b>SalesPrice</b> , <b>TareWeight</b> , and <b>UnitPrice</b> to indicate when the values are initialized and can be expected to remain stable and valid.
Scanner (Bar Code Reader)	No Changes
Signature Capture	Update Model to discuss <b>AutoDisable</b> implications; clarify when <b>RealTimeDataEnabled</b> takes effect; correct <b>DataEvent</b> to indicate when this event may be fired to include real-time data.

**Tone Indicator**

Clarify all the specific properties to indicate when the values are initialized and can be expected to remain stable and valid. Also clarify handling of the **Sound** method when another application claims the device and calls the **Sound** method.



## A p p e n d i x B

# OPOS and JavaPOS

The Java for Retail POS (JavaPOS) and OLE for Retail POS (OPOS) industry standard initiatives are intentionally similar in many respects.

Support for Java requires several differences from OPOS in architecture, but the JavaPOS committee agreed that the general model of OPOS device classes should be reused as much as possible.

In order to reuse as much of the OPOS device models as possible, the following sections detail the general mapping rules from OPOS to JavaPOS. A later section lists the deviations of JavaPOS APIs from OPOS.

## API Mapping Rules

In most cases, OPOS APIs may be translated in a mechanical fashion to equivalent JavaPOS APIs. The exceptions to this mapping are largely due to differences in some string parameters.

Areas of data mapping include data types, methods and properties, and events.

## Data Types

Data types are mapped from OPOS to JavaPOS as follows, with exceptions noted after the table:

**Table 1:**

OPOS Type	JavaPOS Type	Usage
BOOL	boolean	Boolean true or false.
BOOL *	boolean[1]	Modifiable boolean.
LONG	int	32-bit integer.
LONG *	int[1]	Modifiable 32-bit integer.
CURRENCY	long	64-bit integer. Used for currency values, with an assumed 4 decimal places.
CURRENCY *	long[1]	Modifiable 64-bit integer.
<i>The string types are usually represented with the following mapping:</i>		
BSTR	String	Text character string.
BSTR *	String[1]	Modifiable text character string.
<i>For some APIs, the string types are represented in one of the following:</i>		
	byte[]	Array of bytes. May be modified, but size of array cannot be changed. Often used when non-textual data is possible.
	Point[]	Array of points. Used by Signature Capture.
	Object	An object. This will usually be subclassed to provide a Device Service-specific parameter for <b>directIO</b> or <b>DirectIOEvent</b> .
<i>nls (LONG)</i>	<i>nls (String)</i>	Operating System National Language Data type.

## Property & Method Names

Property and method names are mapped from OPOS to JavaPOS as follows:

**Table 2:**

Type	OPOS Examples	JavaPOS Examples	Mapping Rule
Property Read	Claimed	getClaimed()	Prepend "get" to the property name to form the property accessor method.
	DeviceEnabled	getDeviceEnabled()	No parameters.
	OutputID	getOutputID()	Return value is the property.
Property Write	AutoDisable	setAutoDisable(...)	Prepend "set" to the property name to form the property mutator method.
	DeviceEnabled	setDeviceEnabled(...)	One parameter, which is of the property's type. No return value.
Method	Open	open	Change first letter to lower-case.
	CheckHealth	checkHealth	Other characters are unchanged.
	DirectIO	directIO	



## Events

JavaPOS events use the Java Development Kit 1.1 event delegation model, whereby the application registers for events, supplying a class instance that implements an interface extended from **EventListener**.

For each *Event* type which the Application wishes to receive, the Application must implement the corresponding **jpos.events.EventListener** interface and handle its event method. Events are delivered by the JavaPOS Device by calling this event method.

## Constants

Constants are mapped from OPOS to JavaPOS as follows:

- If the constant begins with “OPOS”, then change “OPOS” to “JPOS.”
- Otherwise, make no changes to the constant name.

All constant interface files are available in the package “jpos.” All constants are of type “static final int.”

## API Deviations

The following OPOS APIs do not follow the above mapping rules:

- **BinaryConversion** property  
Not needed by JavaPOS.  
This OPOS property was used to overcome a COM-specific issue with passing binary data in strings. JavaPOS uses more appropriate types for these cases, such as byte arrays.
- **ResultCode** and **ResultCodeExtended** properties  
Not needed by JavaPOS.  
These OPOS properties are used for reporting failures on method calls and property sets. In JavaPOS, these failures (plus property get failures) cause a **JposException**. This exception includes the properties **ErrorCode** and **ErrorCodeExtended**, with values that match the OPOS properties.
- **DirectIO** method and **DirectIOEvent**  
The BSTR\* parameter is mapped to Object.
- Cash Drawer **WaitForDrawerClosed** method  
The tone function of this method may not work on non-PCs, since it depends on the availability of a speaker.
- Hard Totals **Read** method  
The BSTR\* parameter is mapped to byte[], with its size set to the requested number of bytes.
- Hard Totals **Write** method  
The BSTR parameter is mapped to byte[].
- MSR **Track1Data**, **Track1DiscretionaryData**, **Track2Data**, **Track2DiscretionaryData**, **Track3Data** properties  
These BSTR properties are mapped to byte[].
- Pinpad **PromptLanguage** property  
This LONG property is mapped to String.
- Scanner **ScanData** and **ScanDataLabel** properties  
These BSTR properties are mapped to byte[].
- Signature Capture **PointArray** property  
This BSTR property is mapped to Point[].
- Signature Capture **RawData** property  
This BSTR property is mapped to byte[].
- Signature Capture **TotalPoints** property  
Not needed by JavaPOS.  
This property is equivalent to “**PointArray.length**”, so **TotalPoints** is redundant.

## Future Versions

The JavaPOS committee has proposed that future device category API extensions be developed by a joint subcommittee of OPOS and JavaPOS, UnifiedPOS.

Future versions of OPOS and JavaPOS will be synchronized by the National Retail Federation, ARTS directed UnifiedPOS committee. Language or environment specific bindings will be performed by the respective OPOS or JavaPOS committees using the UML based standard that the UnifiedPOS committee produces.

