

UnifiedPOS

UnifiedPOS Retail Peripheral Architecture

Version 1.5

September 24, 2000

International Standard

For Implementation of Point Of Service Peripherals

UnifiedPOS Technical Committee Members:

**Epson, Fujitsu-ICL Systems, IBM, JC Penney, Microsoft, NCR,
PCMS Datafit, Research Computer Services, Sears Roebuck and Co.,
Sun Microsystems, Wincor-Nixdorf GmbH.**

Information regarding the activities of the UnifiedPOS committee can be viewed at the following web site:

<http://www.nrf-arts.org>

UnifiedPOS

UnifiedPOS Retail Peripheral Architecture

Information in this document is subject to change without notice.

JavaPOS is a trademark of Sun Microsystems, Inc.

Windows is a trademark of Microsoft Corporation.

Epson is a trademark of Seiko Epson Corporation.

Table of Contents

INTRODUCTION AND ARCHITECTURE	
UNIFIEDPOS ARCHITECTURE FOR RETAIL	1
WHAT IS UNIFIEDPOS?	1
GOALS	2
DEPENDENCIES	2
UNIFIEDPOS RELATIONSHIP TO OPOS AND JAVAPOS	2
WHO SHOULD READ THIS DOCUMENT	3
ARCHITECTURAL OVERVIEW	4
ARCHITECTURAL COMPONENTS	4
USE OF UML	5
DATA TYPES	8
DEVICE BEHAVIOR MODELS	9
INTRODUCTION TO PROPERTIES, METHODS, AND EVENTS	9
DEVICE INITIALIZATION AND FINALIZATION	11
DEVICE SHARING MODEL	13
EVENTS	14
ERRORS	15
<i>Error Codes</i>	15
DEVICE INPUT MODEL	17
DEVICE OUTPUT MODELS	20
DEVICE POWER REPORTING MODEL	21
<i>Power State Diagram</i>	22
DEVICE STATES	25
<i>Device State Diagram</i>	26
VERSION HANDLING	27
CHAPTER 1	
COMMON PROPERTIES, METHODS, AND EVENTS	29
SUMMARY	29
GENERAL INFORMATION	31
PROPERTIES (UML ATTRIBUTES)	33
METHODS (UML OPERATIONS)	44
EVENTS (UML INTERFACES)	50
CHAPTER 2	
BUMP BAR	57
SUMMARY	57
GENERAL INFORMATION	61
<i>Bump Bar Class Diagram</i>	62
<i>Bump Bar State Diagram</i>	66
PROPERTIES (UML ATTRIBUTES)	67
METHODS (UML OPERATIONS)	73
EVENTS (UML INTERFACES)	78

CHAPTER 3	
CASH CHANGER	83
SUMMARY	83
GENERAL INFORMATION	87
<i>CashChanger Class Diagram</i>	88
<i>Cash Changer State Diagram</i>	93
PROPERTIES (UML ATTRIBUTES)	95
METHODS (UML OPERATIONS)	105
EVENTS (UML INTERFACES)	112
CHAPTER 4	
CASH DRAWER	115
SUMMARY	115
GENERAL INFORMATION	118
PROPERTIES (UML ATTRIBUTES)	119
METHODS (UML OPERATIONS)	121
EVENTS (UML INTERFACES)	122
CHAPTER 5	
CAT - CREDIT AUTHORIZATION TERMINAL	125
SUMMARY	125
GENERAL INFORMATION	129
<i>CAT Class Diagram</i>	131
<i>CAT State Diagram</i>	136
PROPERTIES (UML ATTRIBUTES)	137
METHODS (UML OPERATIONS)	154
EVENTS (UML INTERFACES)	162
CHAPTER 6	
COIN DISPENSER	167
GENERAL INFORMATION	167
CHAPTER 7	
FISCAL PRINTER	169
GENERAL INFORMATION	169
PROPERTIES (UML ATTRIBUTES)	170
CHAPTER 8	
HARD TOTALS	171
GENERAL INFORMATION	171
CHAPTER 9	
KEYLOCK	173
GENERAL INFORMATION	173
CHAPTER 10	
LINE DISPLAY	175
GENERAL INFORMATION	175
PROPERTIES (UML ATTRIBUTES)	176

CHAPTER 11	
MICR - MAGNETIC INK CHARACTER RECOGNITION READER	179
GENERAL INFORMATION	179
CHAPTER 12	
MSR - MAGNETIC STRIPE READER	181
SUMMARY	181
GENERAL INFORMATION	185
<i>MSR Class Diagram</i>	186
<i>MSR State Diagrams</i>	188
<i>MSR Usage Diagram</i>	190
PROPERTIES (UML ATTRIBUTES)	191
EVENTS (UML INTERFACES)	202
CHAPTER 13	
PIN PAD	207
SUMMARY	207
GENERAL INFORMATION	211
<i>PINPad Class Diagram</i>	212
<i>Feature Not Supported</i>	213
<i>PINPad State Diagram</i>	216
PROPERTIES (UML ATTRIBUTES)	217
METHODS (UML OPERATIONS)	228
EVENTS (UML INTERFACES)	232
CHAPTER 14	
POINT CARD READER WRITER	235
SUMMARY	235
GENERAL INFORMATION	240
<i>Point Card Reader Writer Class Diagram</i>	241
<i>Data Characters and Escape Sequences</i>	247
<i>Point Card Reader Writer State Diagram</i>	249
PROPERTIES (UML ATTRIBUTES)	250
METHODS (UML OPERATIONS)	269
EVENTS (UML INTERFACES)	277
CHAPTER 15	
POS KEYBOARD	281
GENERAL INFORMATION	281
CHAPTER 16	
POS POWER	283
SUMMARY	283
GENERAL INFORMATION	286
<i>POSPower Class Diagram</i>	288
<i>POSPower State Diagram</i>	289
<i>POSPower PowerState Diagram - part 1</i>	290
<i>POSPower PowerState Diagram - part 2</i>	291
<i>POSPower PowerState Diagram - part 3</i>	292
<i>POSPower State chart Diagram for fan and temperature</i>	293

PROPERTIES (UML ATTRIBUTES)	294
METHODS (UML OPERATIONS)	298
EVENTS (UML INTERFACES)	299
CHAPTER 17	
POS PRINTER	301
SUMMARY	301
GENERAL INFORMATION	307
<i>POS Printer Class Diagram</i>	308
<i>Model</i>	310
<i>POS Printer State Diagram</i>	314
<i>“Both sides printing” sequence Diagram</i>	315
<i>Data Characters and Escape Sequences</i>	316
<i>POS Printer State Diagrams (Low Level)</i>	319
PROPERTIES (UML ATTRIBUTES)	324
METHODS (UML OPERATIONS)	367
EVENTS (UML INTERFACES)	397
CHAPTER 18	
REMOTE ORDER DISPLAY	403
GENERAL INFORMATION	403
PROPERTIES (UML ATTRIBUTES)	404
CHAPTER 19	
SCALE	405
GENERAL INFORMATION	405
CHAPTER 20	
SCANNER (BAR CODE READER)	407
GENERAL INFORMATION	407
CHAPTER 21	
SIGNATURE CAPTURE	409
GENERAL INFORMATION	409
CHAPTER 22	
TONE INDICATOR	411
GENERAL INFORMATION	411
APPENDIX A	
CHANGE HISTORY	A-1
RELEASE VERSION 1.4	A-1
RELEASE VERSION 1.5	A-1
APPENDIX B	
ADDITIONAL SOFTWARE REFERENCES	B-1
UML REFERENCES	B-1
APPENDIX C	
ADDITIONAL HARDWARE REFERENCES	C-1
USB PLUSPOWER CONNECTOR	C-1

INTRODUCTION AND ARCHITECTURE

UnifiedPOS Architecture for Retail

What Is UnifiedPOS?

UnifiedPOS is the acronym for **Unified Point of Service**. It is an architectural specification for application interfaces to point-of-service devices that are used in the retail environment. This standard is both operating system independent and language neutral and defines:

- An architecture for application interface to retail devices.
- A set of retail device behaviors sufficient to support a range of POS solutions.

The UnifiedPOS standard will include:

- The UnifiedPOS Retail Peripheral Architecture overview.
- Text descriptions of the interface to the functions of the device.
- UML terminology and diagrams for each new device category, to describe:
 - Relationships between classes/interfaces and objects in the system.
- Basis for creating C++, Java, IDL or other OO technology to implement the UML design.

The UnifiedPOS standard will **not** include:

- Specific language API specifications.
- Complete software components. Hardware providers or third-party providers develop and distribute these components.
- Certification mechanism; this must be handled by individual language standard committees (such as the OPOS and JavaPOS committees).

Goals

The goals of UnifiedPOS are to provide:

- Common device architecture that is international and extends across vendors, platforms, and retail format.
- Standards for application to device interfaces in an operating system independent and language neutral manner.
- Reduced implementation costs for vendors to support multiple (for example, Windows/COM and Java) platforms because they share the same architecture. This should produce speed to market for innovation.
- An environment avoiding competition between standards while encouraging competition among implementations.

Dependencies

Success of the goals of UnifiedPOS depends upon platform specific standard committees (such as JavaPOS and OLE for Retail POS (OPOS) technical committees) to advance the architecture into platform specific documentation, API definitions and implementations.

The specific technical implementations require:

- Platform specific Programmer's Guide.
- Source files, including:
 - Definition files. Various interface and class files described in the standard.
 - Example files. These will include a set of sample Control classes, to illustrate the interface presented to an application.

UnifiedPOS Relationship to OPOS and JavaPOS

The UnifiedPOS specification will formalize and document the underlying retail device architecture, currently shared by both the JavaPOS and OPOS standards, in an operating system independent and language neutral manner. The first release of the UnifiedPOS Specification was Version 1.4.

Both the JavaPOS v1.4 and OPOS v1.4 standards are established as conformant platform mappings of the UnifiedPOS specification. JavaPOS will be recognized as the only UnifiedPOS conformant, operating system neutral, Java language mapping. OPOS will be recognized as the only UnifiedPOS conformant language neutral COM mapping. Future UnifiedPOS mappings to platforms other than Java and COM will not be excluded however.

This acceptance of the existing standards is based on their close conformance to a common design model. Historically, the OPOS standards provided 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 involved 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.

Therefore, retail application developers and Service writers can continue to write their code in conformance with one or both of the JavaPOS or OPOS specifications, with assurance that such development meets the UnifiedPOS standard.

Who Should Read This Document

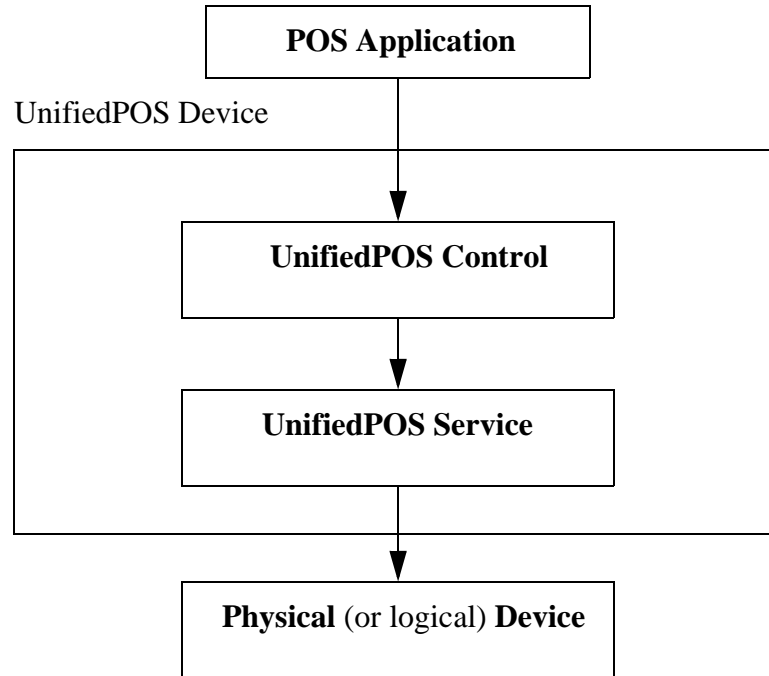
The UnifiedPOS Architecture is targeted to the standard committees that will provide the language specific mapping and Programmer's Guides. However, the application developer who will use POS devices, the system developer who will write POS device code, and the suppliers of POS devices for retail may be interested in the device characteristics as portrayed in this document.

This guide assumes that the standard committee member is familiar with the following:

- General characteristics of POS peripheral devices.
- UnifiedPOS terminology and architecture.
- UML for reading the design.

Architectural Overview

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



Architectural Components

The **POS Application** (or **Application**) is an Application that uses one or more UnifiedPOS devices.

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

Each UnifiedPOS Device is a combination of these components:

- **Control** for a device category. The Control class provides the interface between the Application and the device category. It contains no graphical component and is therefore invisible at runtime.

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

- **Service**, which is a component called by the Control through the **Service Interface**. The Service is used by the Control to implement UnifiedPOS-prescribed functionality for a Physical Device. It can also call special event methods provided by the Control to deliver events to the Application.

A set of 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 platform specific APIs which conform to the UnifiedPOS standard. 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 UnifiedPOS, an application treats each of these device categories as if it were an independent Physical Device. The UnifiedPOS Device standard developer 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**.

Use of UML

The UnifiedPOS standard includes the use of UML terminology and diagrams to define device categories. Following is a brief description of the extensions to UML to make it better fit the UnifiedPOS architecture (this extension is expected and allowed by the UML, see Booch98 reference in the “UML References” on page B-1).

Should any discrepancies exist between the UML diagrams and the specification text, then the text takes precedence.

Table of extensions to UML for UnifiedPOS.

Name	Applies to UML Symbol	Meaning
<<capability>>	Class attribute	stereotype which flags the attribute as a UnifiedPOS capability
<<prop>>	Class attribute	stereotype which flags the attribute as a UnifiedPOS property
<<event>>	Class	stereotype to indicate that the class/ interface will be mapped to a UnifiedPOS event which in turn is mapped to a JavaPOS event class or a COM event for OPOS
exclusive-use	Class	constraint that indicates this device service or service object follows the exclusive-use behavior defined in the UnifiedPOS documentation in section “Exclusive-Use Devices” on page 13.
sharable	Class	constraint that indicates this device service or service object follows the sharable behavior defined in the UnifiedPOS documentation in section “Sharable Devices” on page 13.
read-only read-write	Class attribute	constraint that indicates the mutability of the attribute. For example, in JavaPOS, read-only attributes translate to having a getter method for the attribute and read-write attributes have getter and setter methods for attributes.
access after <open> <open-claim> <open-enable> <open-claim-enable>	Class attribute	constraint that indicates this attribute is accessible when the service is in the state indicated. For example {access after opened-claim-enable} indicates that the attribute is accessible when the service has been opened, claimed and enabled in the order indicated.
raises-exception	Class operation	constraint that indicates this method can throw an exception if the implementation language supports exception; otherwise, some mechanism is used to notify the application that an invalid condition occurred. A value is returned to indicate the error.
use after <open> <open-claim> <open-enable> <open-claim-enable>	Class operation	constraint that indicates this operation is accessible when the service is in the state indicated. For example {use after open-claim-enable} indicates that the method is accessible when the service has been opened, claimed and enabled in the order indicated.

Package Diagram

UnifiedPOS uses Static Structure Diagrams to define common interfaces.



Note: This package diagram is included to give some logical structure to the interfaces in the UnifiedPOS interfaces UML diagrams. Some implementations may have a corresponding equivalence for the packages and some may not. Also, note that the name 'upos' may be replaced by an implementation specific prefix (eg. JavaPOS uses Java packages and maps the prefix 'upos' to 'jpos').

Data Types

UnifiedPOS uses textual references to data types which will be defined for specific language usage:

UnifiedPOS	JavaPOS	OPOS	UML	UnifiedPOS text Usage
<i>boolean</i>	boolean	BOOL	in <i>boolean</i>	Boolean true or false.
<i>boolean by reference</i>	boolean[1]	BOOL*	inout <i>boolean</i>	Modifiable boolean.
<i>binary</i>	byte[]	BSTR	in <i>binary</i>	Array of bytes. Binary byte array, may not be modified.
<i>binary by reference</i>	byte[]	BSTR*	inout <i>binary</i>	Array of bytes. May be modified, but size of array cannot be changed. Binary byte array by reference.
<i>int32</i>	int	LONG	in <i>int32</i>	32-bit integer.
<i>int32 by reference</i>	int[1]	LONG*	inout <i>int32</i>	Modifiable 32-bit integer.
<i>currency</i>	long	CURRENCY or CY	in <i>currency</i>	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”. See footnote ^a
<i>currency by reference</i>	long[1]	CURRENCY* or CY*	inout <i>currency</i>	64-bit integer by reference.
<i>string</i>	String	BSTR	in <i>string</i>	Text character string.
<i>string by reference</i>	String[1]	BSTR*	inout <i>string</i>	String by reference. Modifiable text character string.
<i>array of points</i>	Point[]	BSTR	inout <i>point[]</i>	Array of points. Used by Signature Capture.
<i>object</i>	Object	BSTR*	inout <i>object</i>	An object. This will usually be subclassed to provide a Service-specific parameter.
<i>nls</i>	String	LONG	in <i>nls</i>	Operating System National Language data type.

a. Six decimal place precision is required for all computation in conversion between currencies but is not required for the representation of the solution.

For Java:

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.

Device Behavior Models

Introduction to Properties, Methods, and Events

An application accesses a POS Device via platform specific APIs.

The three elements of UnifiedPOS standard for 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.
- **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.
- **Events.** A Device implementation may call back into the application via events. The application must specifically register for each event type that it needs to receive.

Properties (UML Attributes)

Note: For each interface a UML listing of the properties and methods of the interface will be included in a table. The properties are indicated as attributes. The generic UML naming pattern for attributes is the following:

visibility Name: type-expression = default-value { property-string }

where:

visibility in this document is always public for application visible interfaces but is not explicitly shown.

Name is the name of the attribute

type-expression is the type of the attribute, which is one of UnifiedPOS types defined in section "Data Types" on page 8.

*default-value*¹ the default value of the attributes in UML, (optional)

property-string property value to apply to the element. For attributes, we define two such strings: read-only and read-write, which indicates the mutability of the attribute.

An example of a property attribute is as follows:

DeviceEnabled: *boolean* { read-write }

¹Not used by UnifiedPOS standard

Methods (UML Operations)

The generic UML pattern for methods is the following:

visibility name (parameter-list): return-type-expr { property string }

where:

parameter - list is a comma separated list of formal parameters using the following generic UML naming pattern:

kind name: type-expression (= default-value)²

where:

kind is either: 'in', 'out', or 'inout' with the default set to 'in' if absent

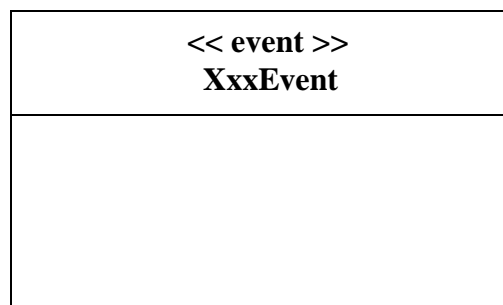
property-string is a property string to apply to the element. For methods an additional property string called 'raises-exception' is defined which means that this method can throw the exception if the implementation language supports exception; otherwise, some mechanism is used to notify the application that an invalid condition occurred.

An example of a method operation is as follows:

open (logicalDeviceName: string): void { raises-exception }

Events (UML Interfaces)

Events are being modeled as UML classes which will possibly contain attributes stereotyped with the event stereotype. The generic UML pattern for events is a UML box with the stereotype <<event>> (class diagram) with the event name and a list of the properties. This representation is different from Properties and Methods.



where:

XxxEvent stands for the UnifiedPOS event name and the second compartment of the box would contain a list of attributes for the event.

².*default-value* is not used by the UnifiedPOS standard

Device Initialization and Finalization

Initialization

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

- Obtain a reference to a Control,
- Prepare Control for the events that the application needs to receive, if necessary.

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

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

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

More than one instance of a 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 Control instances do not interfere with the use of the Device. An application can **release** the Device to share it with another 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 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 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 on the device.

Before exiting, an application should close all open Devices to free device resources in a timely manner.

Summary

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

Obtain a Control reference.

Prepare for events if necessary.

Call the **open** method to instantiate a Service and link it to the 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 13 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 Service from the Control.

Release events receipt if necessary

Remove the reference to the Control

Device Sharing Model

Devices fall into two sharing categories:

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

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

Exclusive-Use Devices

The most common device type is called an **exclusive-use device**. An example is the POS printer. Due to physical or operational characteristics, an exclusive-use device can only be used by one 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 failure condition to occur.

An application may in effect share an exclusive-use device by calling the 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 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 Control instances to call its methods and access its properties. Also, it may deliver its events to multiple Controls. A sharable device may still limit access to some methods or properties to the Control that has claimed it, or it may deliver some events only to the Control that has claimed it.

Events

UnifiedPOS architecture uses events to inform the application of various activities or changes with the 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. Devices may be able to report device power state. See "Device Power Reporting Model" on page 21.	Status change notification
DirectIOEvent	This event may be defined by a Service provider for purposes not covered by the specification.	Always, for Service-specific use

The Service must enqueue these events on an internally created and managed queue. All 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 17.) Events are delivered by an internally created and managed Service thread. The Service causes event delivery by calling an event firing callback method in the Control, which then delivers the event to the application.

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 17.)

Rules for event queue management are:

- The 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 data and output error events.

Errors

UnifiedPOS architecture deals with two kinds of errors as discussed in “Methods (UML Operations)” on page 10 and explanation of exceptions:

- Errors that are “invalid or bad invocations” which are recognized by the Service validation of the request. Method invocations and property accesses may be valid or invalid. If the action is invalid, an invalid condition is set and the application is notified in a fashion appropriate to the platform. For specific implementations, OPOS would produce a **ResultCode** other than OPOS_SUCCESS and JavaPOS would produce an exception.
- Errors that are caused by errant device behavior and produce error events.

Error Codes

This section lists the general meanings of the error code property when an invalid condition occurs. 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. In UML each error code is:

E_XXX : int32 { frozen }

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

Value	Meaning
E_CLOSED	An attempt was made to access a closed Device.
E_CLAIMED	An attempt was made to access a Physical Device that is claimed by another 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.
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 Control instance, then the status E_CLAIMED is returned instead.
E_NOSERVICE	The Control cannot communicate with the Service, normally because of a setup or configuration error.
E_DISABLED	Cannot perform this operation while the Device is disabled.

E_ILLEGAL	An attempt was made to perform an illegal or unsupported operation with the Device, or an invalid parameter value was used.
E_NOHARDWARE	The Physical Device is not connected to the system or is not powered on.
E_OFFLINE	The Physical Device is off-line.
E_NOEXIST	The file name (or other specified value) does not exist.
E_EXISTS	The file name (or other specified value) already exists.
E_FAILURE	The Device cannot perform the requested procedure, even though the Physical Device is connected to the system, powered on, and on-line.
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.
E_BUSY	The current Service state does not allow this request. For example, if asynchronous output is in progress, certain methods may not be allowed.
E_EXTENDED	A device category-specific error condition occurred. The error condition code is held in an extended error code.

When more than one result code is valid, the most descriptive code should be selected. For example, the closed, claimed, not claimed, and disabled errors must follow this order of error reporting precedence, from higher to lower:

E_CLOSED	The device must be opened.
E_CLAIMED	The device is opened but not claimed. Another application has the device claimed, so it cannot be claimed at this time.
E_NOTCLAIMED	The device is opened but not claimed. No other application has the device claimed, so it can and must be claimed.
E_DISABLED	The device is opened and claimed (if this is an exclusive-use device), but not enabled.

Extended Error Code

The extended error code is set as follows:

- When the error code is E_EXTENDED, the extended error code 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 the error code is any other value, the extended error code **may** be set by the Service to any Service-specific value. These values are only meaningful if an application adds Service-specific code to handle them.

Device Input Model

The standard UnifiedPOS 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.

If the **AutoDisable** property is true when data is received, then the 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 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 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 Device encounters an error while gathering or processing event-driven input, then the Device:

- Changes its **State** to S_ERROR.
- Enqueues an **ErrorEvent** with locus 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 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
EL_INPUT_DATA	<p>Only delivered if the error occurred when one or more DataEvents 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 DataEvent, 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 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>
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 ErrorEvent with the locus EL_INPUT_DATA was delivered first, and then this error event is delivered after all DataEvents have been delivered.</p> <p>Note: This EL_INPUT event is not delivered if: an EL_INPUT_DATA event was delivered and the application event handler responded with an ER_CLEAR error response.</p>

The application can cause the *ErrorResponse* property to be set one of the following:

ErrorResponse	Description
ER_CLEAR	Clear the buffered DataEvents and ErrorEvents and exit the error state, changing State to S_IDLE. This is the default response for locus EL_INPUT.
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 DataEventEnabled property. When all input has been delivered and the DataEventEnabled property is again set to true, another ErrorEvent is delivered with locus EL_INPUT. This is the default response when the locus is EL_INPUT_DATA, and is legal only with this locus.
ER_RETRY	This response directs the Device to retry the input. The error state is exited, and State is changed to S_IDLE. This response may only be selected when the device chapter specifically allows it and when the locus is EL_INPUT. An example is the scale.

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

- The application returns from the EL_INPUT **ErrorEvent**.
- The application calls the **clearInput** method.
- The application returns from the EL_INPUT_DATA **ErrorEvent** with *ErrorResponse* set to ER_CLEAR.

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 Device.

Calling the **clearInput** method deletes all input enqueued by a 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 UnifiedPOS 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 Device does not return until the output is completed; this means the physical device has performed the intended operation. For example the printer has successfully transferred all the output data as ink on the paper.

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 Device validates the method parameters and produces an error condition immediately if necessary. If the validation is successful, the 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 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 **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.

Device Power Reporting Model

Applications frequently need to know the power state of the devices they use.

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

UnifiedPOS architecture segments device power into three states:

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

In addition, one combination state is defined:

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

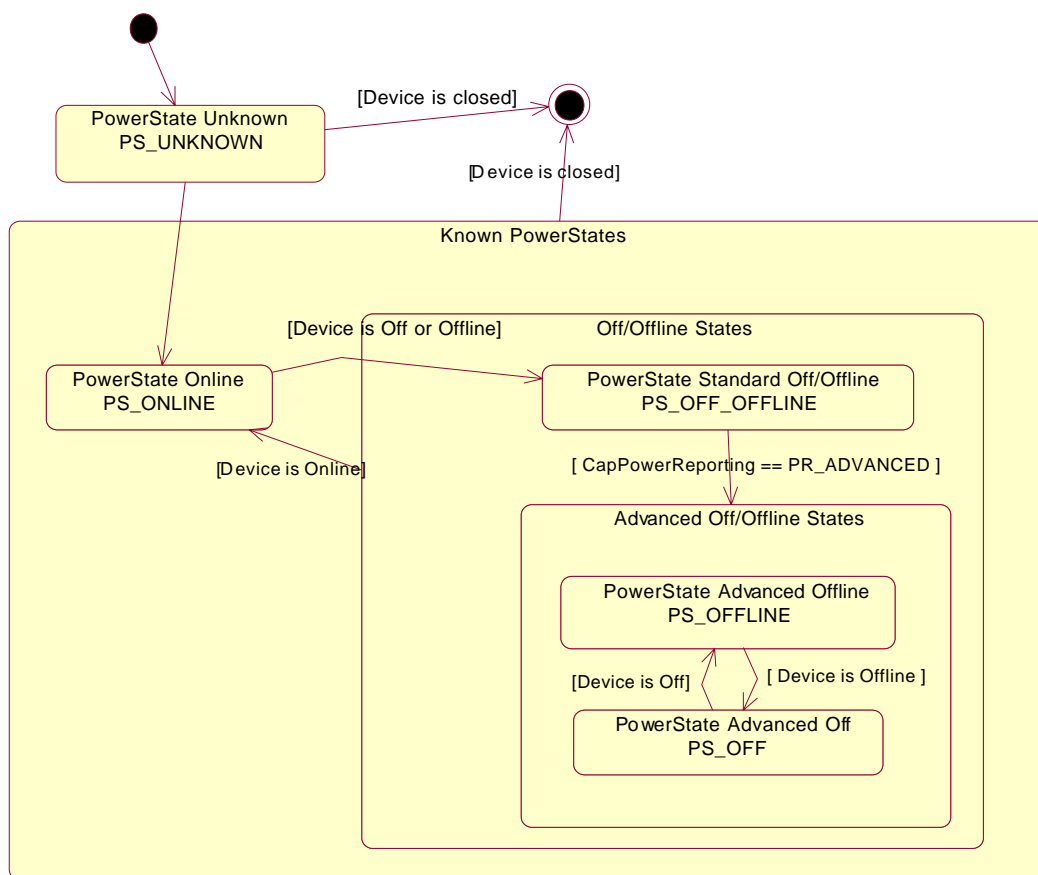
Power reporting only occurs while the device is open, claimed (if the device is exclusive-use), and enabled.

Note - Enabled/Disabled vs. Power States

These states are different and usually independent. UnifiedPOS 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, UnifiedPOS only reports the state while the device is enabled. (This restriction is necessary because a Service typically can only communicate with the device while enabled.)

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

Power State Diagram



Power Properties

The UnifiedPOS device power reporting model adds the following common elements across all device classes.

- **CapPowerReporting** property. Identifies the reporting capabilities of the device. The UML pattern for the property is:

PR_XXX : int32 { frozen }

This property may be one of:

- **PR_NONE**. The Service cannot determine the state of the device. Therefore, no power reporting is possible.
- **PR_STANDARD**. The Service can determine and report two of the power states - OFF_OFFLINE (that is, off or offline) and ONLINE.
- **PR_ADVANCED**. The Service can determine and report all three power states - ONLINE, OFFLINE, and OFF.
- **PowerState** property. Maintained by the Service at the current power condition, if it can be determined. The UML pattern for the property is:

PS_XXX : int32 { frozen }

This property may be one of:

- **PS_UNKNOWN**
- **PS_ONLINE**
- **PS_OFF**
- **PS_OFFLINE**
- **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 changed while the device is disabled (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. The UML pattern for the property is:

PN_XXX : int32 { frozen }

This property may be one of:

- **PN_DISABLED**
- **PN_ENABLED**

Power Reporting Requirements for DeviceEnabled

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

CapPowerReporting is not PR_NONE, and
PowerNotify is 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 PS_ONLINE.

A **StatusUpdateEvent** is enqueued with its *Status* property set to SUE_POWER_ONLINE.
 - If the Physical Device's power state is OFF, OFFLINE, or OFF_OFFLINE, then the Service may choose to fail the enable by notifying the application with error code E_NOHARDWARE or E_OFFLINE.

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

PowerState is set to PS_OFF, PS_OFFLINE, or PS_OFF_OFFLINE.

A **StatusUpdateEvent** is enqueued with its *Status* property set to SUE_POWER_OFF, SUE_POWER_OFFLINE, or SUE_POWER_OFF_OFFLINE.
- When the Device changes from **DeviceEnabled** true to false, UnifiedPOS assumes that the Device is no longer monitoring the power state and sets the value of **PowerState** to PS_UNKNOWN

Device States

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

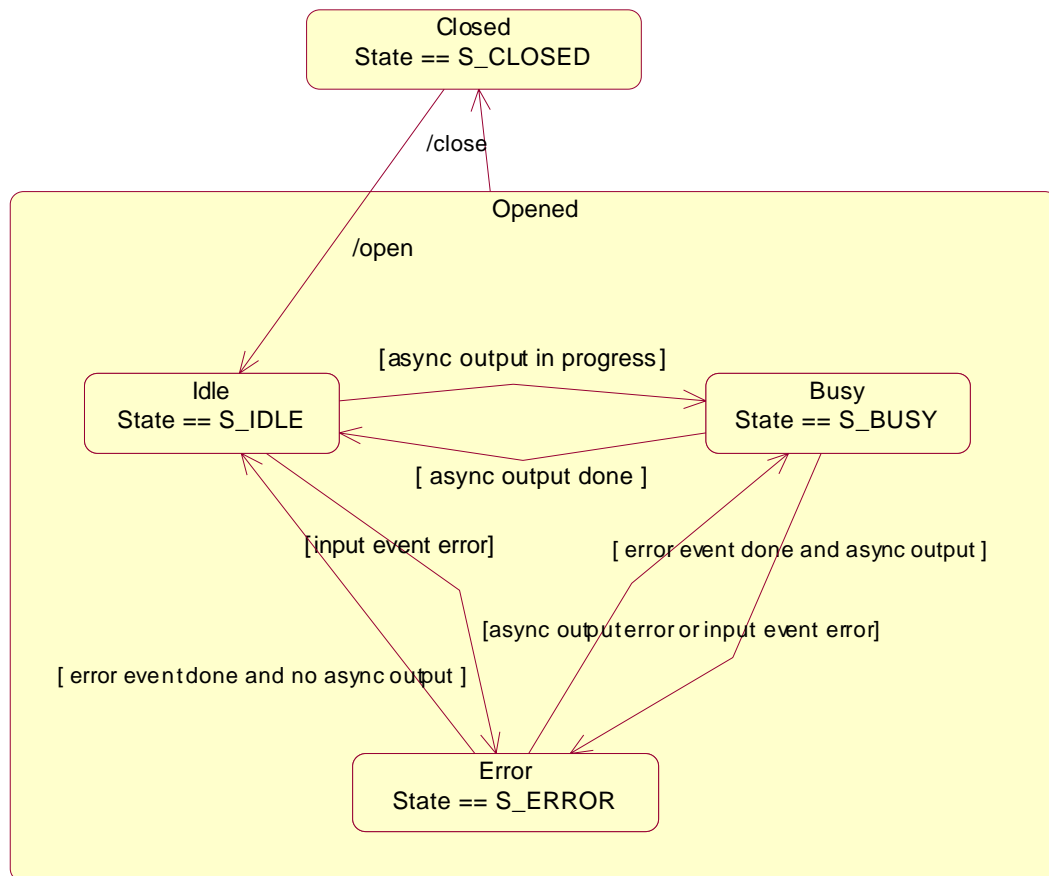
S_CLOSED
S_IDLE
S_BUSY
S_ERROR

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

- **State** is initially S_CLOSED.
- **State** is changed to S_IDLE when the **open** method is successfully called.
- **State** is set to S_BUSY when the Service is processing output. The **State** is restored to S_IDLE when the output has completed.
- The **State** is changed to 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 Service changes the **State** property to S_ERROR, it notifies the application of this error. 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.

Device State Diagram



Version Handling

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

- **Control requirements.** A Control for a device category must operate with any 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, then 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, the application will be notified of an error condition (E_NO_SERVICE).
- **Service requirements.** A Service for a device category must operate with any 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 Control and Service are at the proper major version and at or greater than the proper minor version. The versions are reported by the properties **DeviceControlVersion** (see page 36) and **DeviceServiceVersion** (see page 38).

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 (UML attributes)

<i>Name</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>	<i>Usage Notes</i>
AutoDisable:	<i>boolean</i>	{ read-write }		1
CapPowerReporting:	<i>int32</i>	{ read-only }		
CheckHealthText:	<i>string</i>	{ read-only }		
Claimed:	<i>boolean</i>	{ read-only }		
DataCount:	<i>int32</i>	{ read-only }		1
DataEventEnabled:	<i>boolean</i>	{ read-write }		1
DeviceEnabled:	<i>boolean</i>	{ read-write }		
FreezeEvents:	<i>boolean</i>	{ read-write }		
OutputID:	<i>int32</i>	{ read-only }		2
PowerNotify:	<i>int32</i>	{ read-write }		
PowerState:	<i>int32</i>	{ read-only }		
State:	<i>int32</i>	{ read-only }		
DeviceControlDescription:	<i>string</i>	{ read-only }		
DeviceControlVersion:	<i>int32</i>	{ read-only }		
DeviceServiceDescription:	<i>string</i>	{ read-only }		
DeviceServiceVersion:	<i>int32</i>	{ read-only }		
PhysicalDeviceDescription:	<i>string</i>	{ read-only }		
PhysicalDeviceName:	<i>string</i>	{ read-only }		

Methods (UML operations)*Name*

open (logicalDeviceName: *string*):
void { raises-exception }

close ():
void { raises-exception }

claim^a (timeout: *int32*):
void { raises-exception }

release^a ():
void { raises-exception }

checkHealth (level: *int32*):
void { raises-exception }

clearInput ():
void { raises-exception }

clearOutput ():
void { raises-exception }

directIO (command: *int32*, inout data: *int32*, inout obj: *object*):
void { raises-exception }

a. **Note:** In the OPOS environment starting with Release 1.5 the Claim and Release methods are also defined as ClaimDevice and ReleaseDevice due to Release being a reserved method used by Microsoft's Component Object Model (COM).

Events (UML interfaces)

<i>Name</i>	<i>Type</i>	<i>Mutability</i>	<i>Usage Notes</i>
upos::events::DataEvent			1
Status:	<i>int32</i>	{ read-only }	
upos::events::DirectIOEvent			
EventNumber:	<i>int32</i>	{ read-only }	
Data:	<i>int32</i>	{ read-write }	
Obj:	<i>object</i>	{ read-write }	
upos::events::ErrorEvent			
ErrorCode:	<i>int32</i>	{ read-only }	
ErrorCodeExtended:	<i>int32</i>	{ read-only }	
ErrorLocus:	<i>int32</i>	{ read-only }	
ErrorResponse:	<i>int32</i>	{ read-write }	
upos::events::OutputCompleteEvent			2
OutputID:	<i>int32</i>	{ read-only }	
upos::events::StatusUpdateEvent			
Status:	<i>int32</i>	{ read-only }	

Usage Notes:

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

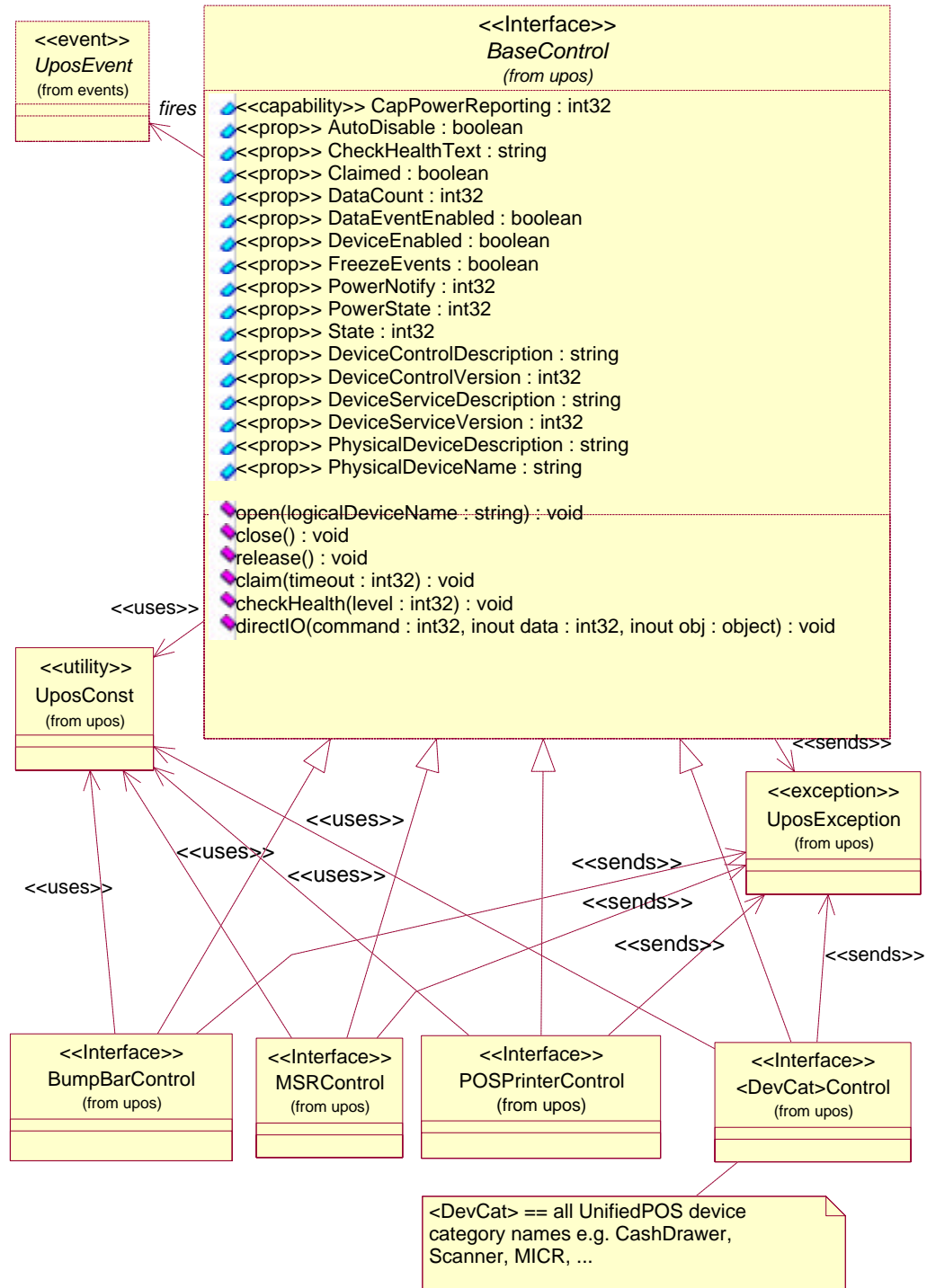
General Information

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 Control’s class.

A good understanding of the features of the UnifiedPOS architecture model is required. Please see “Device Behavior Models” on page 9 for additional information.

The following diagram shows the relationships between the Common classes.



Properties (UML attributes)

AutoDisable Property

Syntax	AutoDisable: <i>boolean</i> { read-write }
Remarks	<p>If true, the UnifiedPOS Service will set DeviceEnabled to false after it receives and enqueues data as a DataEvent. Before any additional input can be received, the application must set DeviceEnabled to true.</p> <p>If false, the UnifiedPOS 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 open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	“Device Input Model” on page 17.

CapPowerReporting Property

Syntax	CapPowerReporting: <i>int32</i> { read-only }								
Remarks	<p>Identifies the reporting capabilities of the Device. It has one of the following values:</p> <table> <tr> <th>Value</th><th>Meaning</th></tr> <tr> <td>PR_NONE</td><td>The UnifiedPOS Service cannot determine the state of the device. Therefore, no power reporting is possible.</td></tr> <tr> <td>PR_STANDARD</td><td>The UnifiedPOS Service can determine and report two of the power states - OFF_OFFLINE (that is, off or offline) and ONLINE.</td></tr> <tr> <td>PR_ADVANCED</td><td>The UnifiedPOS Service can determine and report all three power states - OFF, OFFLINE, and ONLINE.</td></tr> </table> <p>This property is initialized by the open method.</p>	Value	Meaning	PR_NONE	The UnifiedPOS Service cannot determine the state of the device. Therefore, no power reporting is possible.	PR_STANDARD	The UnifiedPOS Service can determine and report two of the power states - OFF_OFFLINE (that is, off or offline) and ONLINE.	PR_ADVANCED	The UnifiedPOS Service can determine and report all three power states - OFF, OFFLINE, and ONLINE.
Value	Meaning								
PR_NONE	The UnifiedPOS Service cannot determine the state of the device. Therefore, no power reporting is possible.								
PR_STANDARD	The UnifiedPOS Service can determine and report two of the power states - OFF_OFFLINE (that is, off or offline) and ONLINE.								
PR_ADVANCED	The UnifiedPOS Service can determine and report all three power states - OFF, OFFLINE, and ONLINE.								
Errors	None.								
See Also	“Device Power Reporting Model” on page 21, PowerState Property, PowerNotify Property.								

CheckHealthText Property

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

Claimed Property

Syntax	Claimed: <i>boolean</i> { read-only }
Remarks	<p>If true, the device is claimed for exclusive access. If false, the device is released for sharing with other applications.</p> <p>Many devices must be claimed before the Control will allow access to many of its methods and properties, and before it will deliver events to the application.</p> <p>This property is initialized to false by the open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	“Device Initialization and Finalization” on page 11, “Device Sharing Model” on page 13, claim Method, release Method.

DataCount Property

Syntax	DataCount: <i>int32</i> { read-only }
Remarks	<p>Holds the number of enqueued DataEvents.</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 open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	“Device Input Model” on page 17, DataEvent .

DataEventEnabled Property

Syntax	DataEventEnabled: <i>boolean</i> { read-write }
Remarks	<p>If true, a DataEvent will be delivered as soon as input data is enqueued. If changed to true and some input data is already queued, then a DataEvent is delivered immediately. (Note that other conditions may delay “immediate” delivery: if FreezeEvents is true or another event is already being processed at the application, the DataEvent will remain queued at the UnifiedPOS 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 ErrorEvent is not delivered while this property is false.</p> <p>This property is initialized to false by the open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	“Events” on page 14, DataEvent .

DeviceControlDescription Property

Syntax	DeviceControlDescription: <i>string</i> { read-only }
Remarks	<p>Holds an identifier for the UnifiedPOS Control and the company that produced it.</p> <p>A sample returned string is:</p> <pre>“POS Printer UnifiedPOS Compatible Control, (C) 1998 Epson”</pre> <p>This property is always readable.</p>
Errors	None.
See Also	DeviceControlVersion Property.

DeviceControlVersion Property

Syntax **DeviceControlVersion:** *int32* { **read-only** }

Remarks Holds the UnifiedPOS Control version number.

Three version levels are specified, as follows:

Version Level	Description
Major	The “millions” place. A change to the UnifiedPOS 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 UnifiedPOS 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 UnifiedPOS Control developer. Updated when corrections are made to the UnifiedPOS 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 UnifiedPOS Control.

This property is always readable.

Errors None.

See Also “Version Handling” on page 27, **DeviceControlDescription** Property.

DeviceEnabled Property

Syntax	DeviceEnabled: <i>boolean</i> { read-write }
Remarks	<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>The Device's power state may be reported while DeviceEnabled is true; See "Device Power Reporting Model" on page 21 for details.</p> <p>This property is initialized to false by the open method. Note that an exclusive use device must be claimed before the device may be enabled.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see "Errors" on page 15.
See Also	"Device Initialization and Finalization" on page 11.

DeviceServiceDescription Property

Syntax	DeviceServiceDescription: <i>string</i> { read-only }
Remarks	<p>Holds an identifier for the UnifiedPOS Service and the company that produced it.</p> <p>A sample returned string is:</p> <pre>"TM-U950 Printer UnifiedPOS Compatible Service Driver, (C) 1998 Epson"</pre> <p>This property is initialized by the open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see "Errors" on page 15.

DeviceServiceVersion Property

Syntax **DeviceServiceVersion:** *int32* { **read-only** }

Remarks Holds the UnifiedPOS Service version number.

Three version levels are specified, as follows:

Version Level	Description
Major	The “millions” place. A change to the UnifiedPOS 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 UnifiedPOS 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 UnifiedPOS Service developer. Updated when corrections are made to the UnifiedPOS 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 UnifiedPOS Service.

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

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

See Also “Version Handling” on page 27, **DeviceServiceDescription** Property.

FreezeEvents Property

Syntax	FreezeEvents: <i>boolean</i> { read-write }
Remarks	<p>If true, the UnifiedPOS Control will not deliver events. 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 open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

OutputID Property

Syntax	OutputID: <i>int32</i> { read-only }
Remarks	<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 OutputCompleteEvent will be enqueued with this output ID as a parameter.</p> <p>The output ID numbers are assigned by the UnifiedPOS Service and are guaranteed to be unique among the set of outstanding asynchronous outputs. No other facts about the ID should be assumed.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	“Device Output Models” on page 20, OutputCompleteEvent .

PowerNotify Property

Syntax **PowerNotify:** *int32* { read-write }

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

Value	Meaning
PN_DISABLED	The UnifiedPOS Service will not provide any power notifications to the application. No power notification StatusUpdateEvents will be fired, and PowerState may not be set.
PN_ENABLED	The UnifiedPOS Service will fire power notification StatusUpdateEvents and update PowerState , beginning when DeviceEnabled is set to true. The level of functionality depends upon CapPowerReporting .

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

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

Errors A **UposException** 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:

Value	Meaning
E_ILLEGAL	One of the following occurred: The device is already enabled. PowerNotify = PN_ENABLED but CapPowerReporting = PR_NONE.

See Also “Device Power Reporting Model” on page 21, **CapPowerReporting** Property, **PowerState** Property.

PowerState Property

Syntax **PowerState:** *int32* { **read-only** }

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

Value	Meaning
PS_UNKNOWN	Cannot determine the device's power state for one of the following reasons: CapPowerReporting = PR_NONE; the device does not support power reporting. PowerNotify = PN_DISABLED; power notifications are disabled. DeviceEnabled = false; Power state monitoring does not occur until the device is enabled.
PS_ONLINE	The device is powered on and ready for use. Can be returned if CapPowerReporting = PR_STANDARD or PR_ADVANCED.
PS_OFF	The device is powered off or detached from the POS terminal. Can only be returned if CapPowerReporting = PR_ADVANCED.
PS_OFFLINE	The device is powered on but is either not ready or not able to respond to requests. Can only be returned if CapPowerReporting = PR_ADVANCED.
PS_OFF_OFFLINE	The device is either off or offline. Can only be returned if CapPowerReporting = PR_STANDARD.

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

Errors None.

See Also “Device Power Reporting Model” on page 21, **CapPowerReporting** Property, **PowerNotify** Property.

PhysicalDeviceDescription Property

Syntax	PhysicalDeviceDescription: <i>string</i> { read-only }
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 open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see "Errors" on page 15.
See Also	PhysicalDeviceName Property.

PhysicalDeviceName Property

Syntax	PhysicalDeviceName: <i>string</i> { read-only }
Remarks	<p>Holds a short name identifying the physical device. This is a short version of PhysicalDeviceDescription 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 open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see "Errors" on page 15.
See Also	PhysicalDeviceDescription Property.

State Property

Syntax **State:** *int32* { **read-only** }

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

Value	Meaning
S_CLOSED	The Device is closed.
S_IDLE	The Device is in a good state and is not busy.
S_BUSY	The Device is in a good state and is busy performing output.
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 25.

Methods (UML operations)

checkHealth Method

Syntax **checkHealth (level: *int32*):**
 void { raises-exception }

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

Value	Meaning
CH_INTERNAL	Perform a health check that does not physically change the device. The device is tested by internal tests to the extent possible.
CH_EXTERNAL	Perform a more thorough test that may change the device. For example, a pattern may be printed on the printer.
CH_INTERACTIVE	Perform an interactive test of the device. The supporting UnifiedPOS 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 UposException 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
E_ILLEGAL	The specified health check level is not supported by the UnifiedPOS Service.

See Also **CheckHealthText** Property.

claim Method

Syntax	claim (timeout: <i>int32</i>): void { raises-exception }						
	<p>The <i>timeout</i> 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 FOREVER (-1), the method waits as long as needed until exclusive access is satisfied.</p>						
Remarks	<p>Requests exclusive access to the device. Many devices require an application to claim them before they can be used.</p> <p>When successful, the Claimed property is changed to true.</p>						
Errors	<p>A UposException may be thrown when this method is invoked. For further information, “Errors” on page 15.</p> <p>Some possible values of the exception’s <i>ErrorCode</i> property are:</p> <table> <tr> <th>Value</th><th>Meaning</th></tr> <tr> <td>E_ILLEGAL</td><td>This device cannot be claimed for exclusive access, or an invalid timeout parameter was specified.</td></tr> <tr> <td>E_TIMEOUT</td><td>Another application has exclusive access to the device, and did not relinquish control before timeout milliseconds expired.</td></tr> </table>	Value	Meaning	E_ILLEGAL	This device cannot be claimed for exclusive access, or an invalid timeout parameter was specified.	E_TIMEOUT	Another application has exclusive access to the device, and did not relinquish control before timeout milliseconds expired.
Value	Meaning						
E_ILLEGAL	This device cannot be claimed for exclusive access, or an invalid timeout parameter was specified.						
E_TIMEOUT	Another application has exclusive access to the device, and did not relinquish control before timeout milliseconds expired.						
See Also	“Device Sharing Model” on page 13, release Method.						

clearInput Method

Syntax	clearInput (): void { raises-exception }
Remarks	<p>Clears all device input that has been buffered.</p> <p>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.</p>
Errors	A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 15.
See Also	“Device Input Model” on page 17.

clearOutput Method

Syntax	clearOutput (): void { raises-exception }
Remarks	<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 FreezeEvents to be set to false – are also cleared.</p>
Errors	A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 15.
See Also	“Device Output Models” on page 20.

close Method

Syntax	close (): void { raises-exception }
Remarks	<p>Releases the device and its resources.</p> <p>If the DeviceEnabled property is true, then the device is disabled.</p> <p>If the Claimed property is true, then exclusive access to the device is released.</p>
Errors	A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 15.
See Also	“Device Initialization and Finalization” on page 11, open Method.

directIO Method

Syntax **directIO (command: *int32*, inout data: *int32*, inout obj: *object*):**
 void { raises-exception }

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

Remarks Communicates directly with the UnifiedPOS Service.

This method provides a means for a UnifiedPOS Service to provide functionality to the application that is not otherwise supported by the standard UnifiedPOS Control for its device category. Depending upon the UnifiedPOS 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 UposException may be thrown when this method is invoked. For further information, see "Errors" on page 15.

See Also **DirectIOEvent.**

open Method

Syntax **open (logicalDeviceName: *string*):**
 void { raises-exception }

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 UnifiedPOS Control should be used. The *logicalDeviceName* must exist in the operating system's reference locator system (such as Java's System Database or Window's Registry) for this device category so that its relationship to the physical device can be determined. Entries in the reference locator system are created by a setup or configuration utility.

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

Errors A UposException 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
E_ILLEGAL	The UnifiedPOS Control is already open.
E_NOEXIST	The specified <i>logicalDeviceName</i> was not found.
E_NOSERVICE	Could not establish a connection to the corresponding UnifiedPOS Service.

See Also "Device Initialization and Finalization" on page 11, "Version Handling" on page 27, **close** Method.

release Method

Syntax **release ():**
 void { raises-exception }

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 UposException 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:

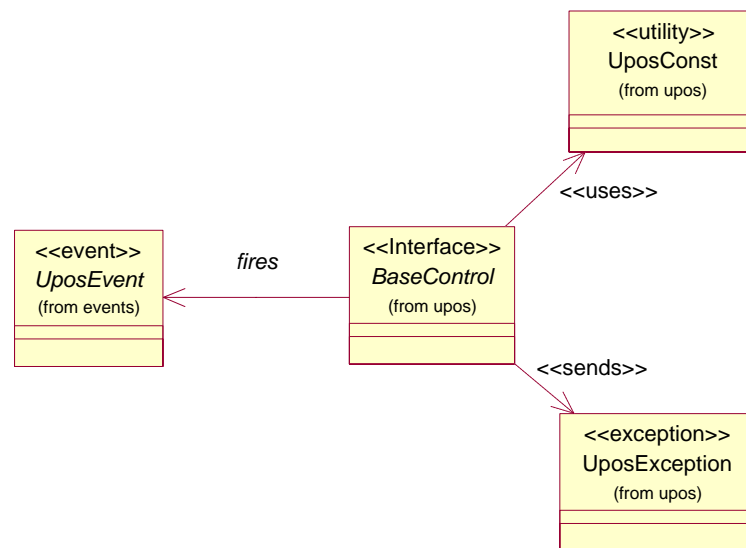
Value	Meaning
E_ILLEGAL	The application does not have exclusive access to the device.

See Also “Device Sharing Model” on page 13, **claim** Method.

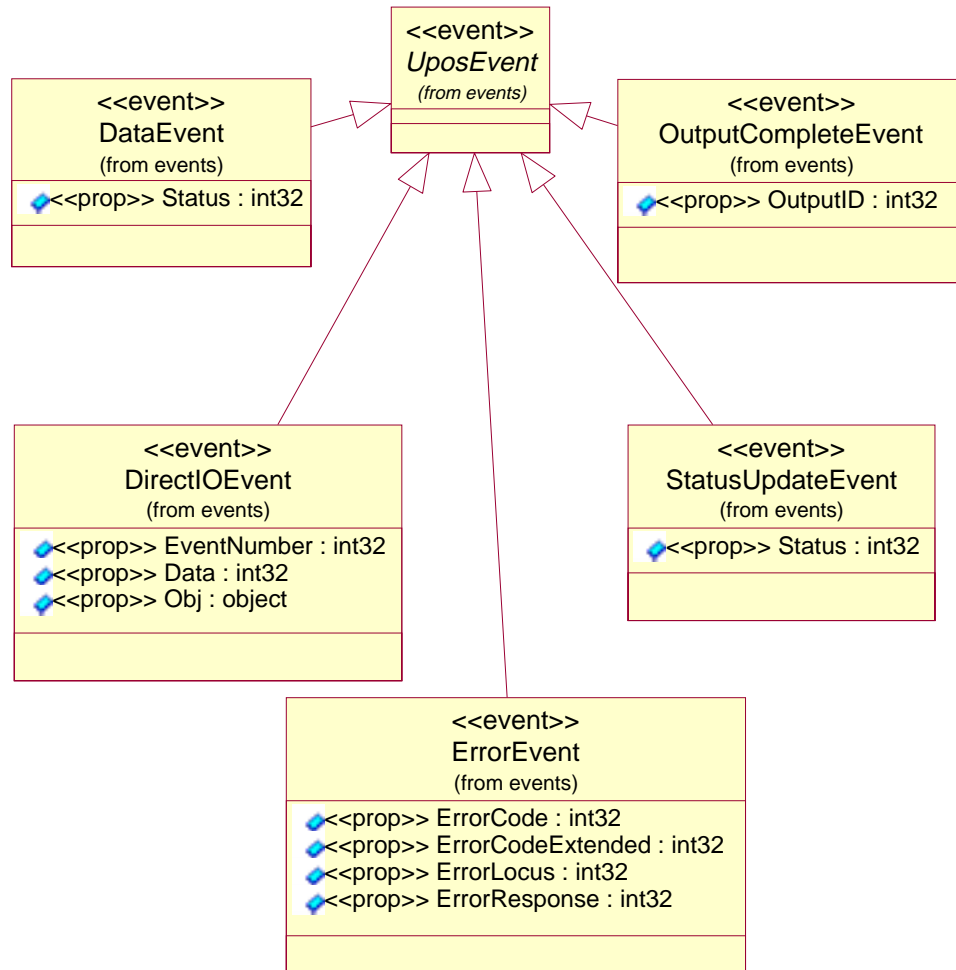
Events (UML interfaces)

The UnifiedPOS standard utilizes a common UML base control structure to derive a specific implementation case. The UML event base control model and interfaces are shown below for the events.

upos::BaseControl



upos::events interfaces



DataEvent

<<event>> **upos::events::DataEvent**
 Status: *int32* { read-only }

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

Attribute This event contains the following attribute:

Attribute	Type	Description
<i>Status</i>	<i>int32</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 14, “Device Input Model” on page 17, **DataEventEnabled** Property, **FreezeEvents** Property.

DirectIOEvent

```
<<event>>      upos::events::DirectIOEvent
                  EventNumber: int32 { read-only }
                  Data: int32 { read-write }
                  Obj: object { read-write }
```

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

Attributes This event contains the following attributes:

Attribute	Type	Description
<i>EventNumber</i>	<i>int32</i>	Event number whose specific values are assigned by the UnifiedPOS Service.
<i>Data</i>	<i>int32</i>	Additional numeric data. Specific values vary by the <i>EventNumber</i> and the UnifiedPOS Service. This attribute is settable.
<i>Obj</i>	<i>object</i>	Additional data whose usage varies by the <i>EventNumber</i> and the UnifiedPOS Service. This attribute 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 UnifiedPOS 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 UnifiedPOS Service's need for this event.

See Also "Events" on page 14, **directIO** Method.

ErrorEvent

```
<<event>>   upos::events::ErrorEvent
              ErrorCode: int32 { read-only }
              ErrorCodeExtended: int32 { read-only }
              ErrorLocus: int32 { read-only }
              ErrorResponse: int32 { read-write }
```

Description Notifies the application that an error has been detected and a suitable response is necessary to process the error condition.

Attributes This event contains the following attributes:

Attribute	Type	Description
<i>ErrorCode</i>	<i>int32</i>	Error Code causing the error event. See the list of <i>ErrorCodes</i> under “Error Codes” on page 15.
<i>ErrorCodeExtended</i>	<i>int32</i>	Extended Error Code causing the error event. These values are device category specific.
<i>ErrorLocus</i>	<i>int32</i>	Location of the error. See values below.
<i>ErrorResponse</i>	<i>int32</i>	Error response, whose default value may be overridden by the application (i.e., this attribute is settable). See values below.

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

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

The application’s error event handler can set the *ErrorResponse* attribute to one of the following values:

Value	Meaning
E_ER_RETRY	Retry the asynchronous output. The error state is exited. May be valid only when locus is E_EL_INPUT. Default when locus is E_EL_OUTPUT.
E_ER_CLEAR	Clear the asynchronous output or buffered input data. The error state is exited. Default when locus is E_EL_INPUT.

E_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 E_EL_INPUT.

Use only when locus is E_EL_INPUT_DATA. Default when locus is E_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 17, "Device States" on page 25.

OutputCompleteEvent

<<event>> **upos::events::OutputCompleteEvent**
OutputID: int32 { read-only }

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

Attribute This event contains the following attribute:

Attribute	Type	Description
<i>OutputID</i>	<i>int32</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 UnifiedPOS Service has confirmation that it was processed by the device successfully.

See Also "Device Output Models" on page 20.

CHAPTER 2

Bump Bar

This Chapter defines the Bump Bar device category.

Summary

Properties (UML attributes)

<i>Common</i>	<i>Type</i>	<i>Mutability</i>	<i>Version^a</i>	<i>May Use After</i>
AutoDisable:	<i>boolean</i>	{ read-write }	1.3	Not Supported
CapPowerReporting:	<i>int32</i>	{ read-only }	1.3	open
CheckHealthText:	<i>string</i>	{ read-only }	1.3	open
Claimed:	<i>boolean</i>	{ read-only }	1.3	open
DataCount:	<i>int32</i>	{ read-only }	1.3	open
DataEventEnabled:	<i>boolean</i>	{ read-write }	1.3	open
DeviceEnabled:	<i>boolean</i>	{ read-write }	1.3	open & claim
FreezeEvents:	<i>boolean</i>	{ read-write }	1.3	open
OutputID:	<i>int32</i>	{ read-only }	1.3	open
PowerNotify:	<i>int32</i>	{ read-write }	1.3	open
PowerState:	<i>int32</i>	{ read-only }	1.3	open
State:	<i>int32</i>	{ read-only }	1.3	--
DeviceControlDescription:	<i>string</i>	{ read-only }	1.3	--
DeviceControlVersion:	<i>int32</i>	{ read-only }	1.3	--
DeviceServiceDescription:	<i>string</i>	{ read-only }	1.3	open
DeviceServiceVersion:	<i>int32</i>	{ read-only }	1.3	open
PhysicalDeviceDescription:	<i>string</i>	{ read-only }	1.3	open
PhysicalDeviceName:	<i>string</i>	{ read-only }	1.3	open

- a. The version representation is an example. It provides the mechanism for recognizing when a change occurs to a property, method or event. This BumpBar definition was introduced in an existing standard and was not changed for the UnifiedPOS version 1.4 or 1.5.

Properties (Continued)

<i>Specific</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>	<i>May Use After</i>
AsyncMode:	<i>boolean</i>	{ read-write }	1.3	open, claim, & enable
Timeout:	<i>int32</i>	{ read-write }	1.3	open
UnitsOnline:	<i>int32</i>	{ read-only }	1.3	open, claim, & enable
CurrentUnitID:	<i>int32</i>	{ read-write }	1.3	open, claim, & enable
CapTone:	<i>boolean</i>	{ read-only }	1.3	open, claim, & enable
AutoToneDuration:	<i>int32</i>	{ read-write }	1.3	open, claim, & enable
AutoToneFrequency:	<i>int32</i>	{ read-write }	1.3	open, claim, & enable
BumpBarDataCount:	<i>int32</i>	{ read-only }	1.3	open, claim, & enable
Keys:	<i>int32</i>	{ read-only }	1.3	open, claim, & enable
ErrorUnits:	<i>int32</i>	{ read-only }	1.3	open
ErrorString:	<i>string</i>	{ read-only }	1.3	open
EventUnitID:	<i>int32</i>	{ read-only }	1.3	open & claim
EventUnits:	<i>int32</i>	{ read-only }	1.3	open & claim
EventString:	<i>string</i>	{ read-only }	1.3	open & claim

Methods (UML operations)**Common*****Name***

open (logicalDeviceName: *string*):
void { raises exception }

close ():
void { raises exception, use after open }

claim (timeout: *int32*) :
void { raises exception, use after open }

release ():
void { raises exception, use after open, claim }

checkHealth (level: *int32*):
void { raises exception, use after open, claim, enable }

clearInput ():
void { raises exception, use after open, claim }

clearOutput ():
void { raises exception, use after open, claim }

directIO (command: *int32*, inout data: *int32*, inout obj: *object*):
void { raises exception, use after open }

Specific***Name***

**bumpBarSound (units: *int32*, frequency: *int32*, duration: *int32*,
numberOfCycles: *int32*, interSoundWait: *int32*):**
void { raises exception, use after open, claim, enable }

setKeyTranslation (units: *int32*, scanCodes: *int32*, logicalKey: *int32*):
void { raises exception, use after open, claim, enable }

Events (UML interfaces)

<i>Name</i>	<i>Type</i>	<i>Mutability</i>
upos::events::DataEvent		
Status:	<i>int32</i>	{ read-only }
upos::events::DirectIOEvent		
EventNumber:	<i>int32</i>	{ read-only }
Data:	<i>int32</i>	{ read-write }
Obj:	<i>object</i>	{ read-write }
upos::events::ErrorEvent		
ErrorCode:	<i>int32</i>	{ read-only }
ErrorCodeExtended:	<i>int32</i>	{ read-only }
ErrorLocus:	<i>int32</i>	{ read-only }
ErrorResponse	<i>int32</i>	{ read-write }
upos::events::OutputCompleteEvent		
OutputID:	<i>int32</i>	{ read-only }
upos::events::StatusUpdateEvent		
Status:	<i>int32</i>	{ read-only }

General Information

The Bump Bar programmatic name is “BumpBar”.

Capabilities

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

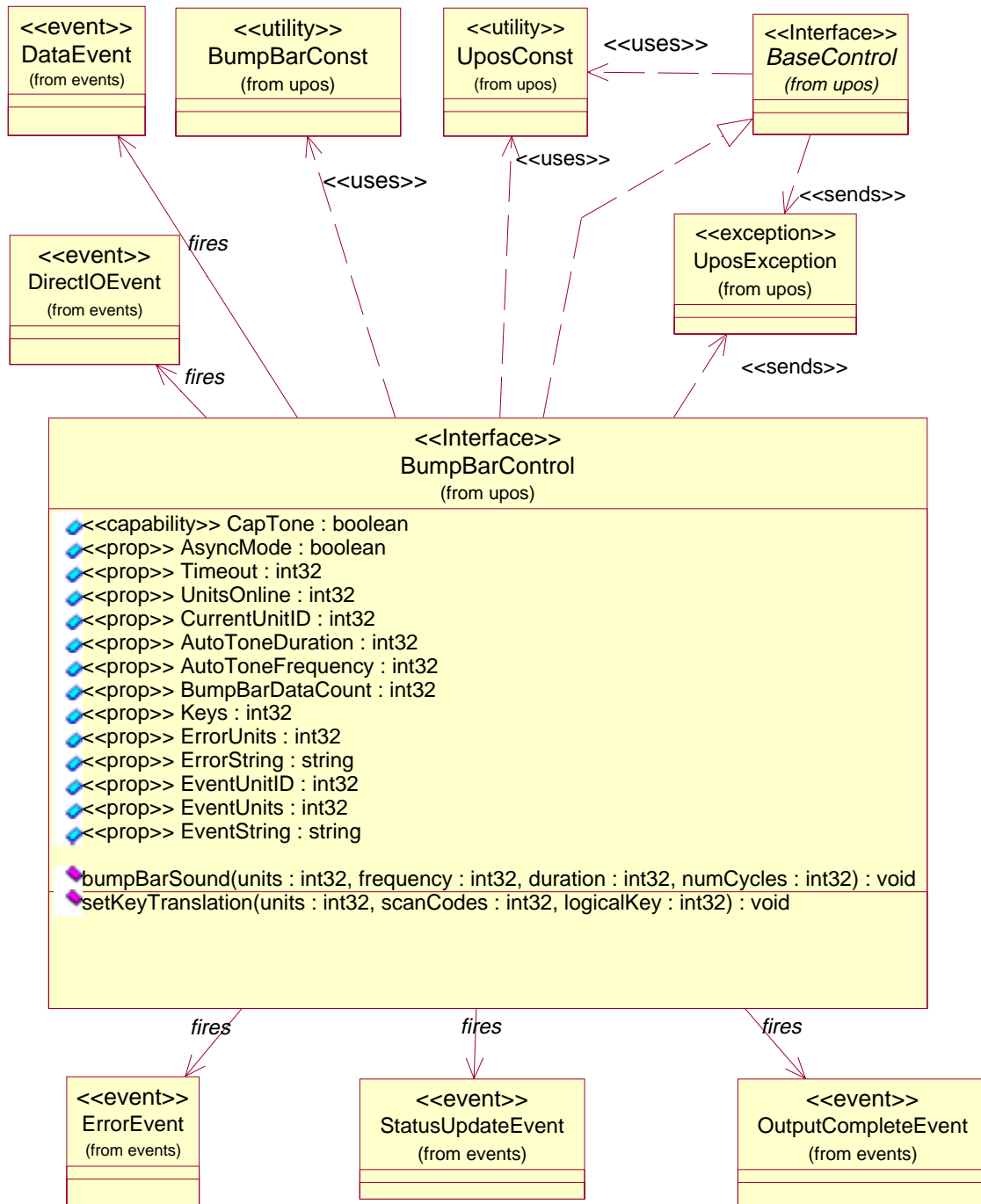
- Supports broadcast methods that can communicate with one, a range, or all bump bar units online.
- Supports bump bar input (keys 0-255).

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.

Bump Bar Class Diagram

The following diagram shows the relationships between the Bump Bar classes.



Model

The general model of a bump bar is:

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

The subsystem can support up to 32 bump bar units.

One application on one 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 *int32*, with each bit identifying an individual bump bar unit. (One or more of the constants BB_UID_1 through BB_UID_32 are bitwise ORed to form the bitmask.) The Service 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 **ErrorMessage** property is updated with a description of the error or errors received. The method will then notify the application of the error condition. In the case where two or more units encounter different errors, the Service should determine the most severe error to report.
- 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 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, the application is notified of an error 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.
 - 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.

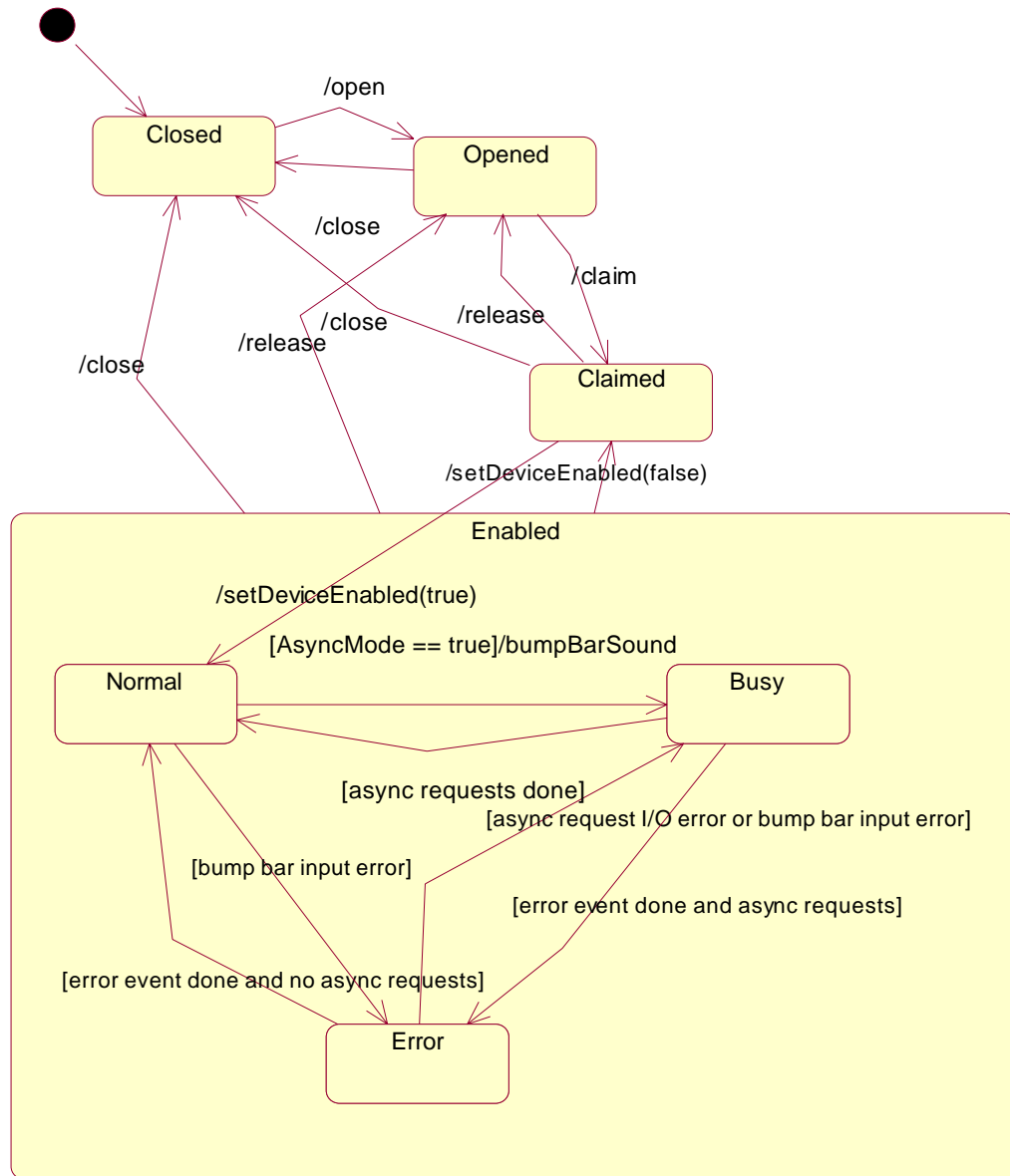
- Asynchronous output is performed on a first-in first-out basis.
- All output buffered may be deleted by setting the **CurrentUnitID** property and calling the **clearOutput** method. An **OutputCompleteEvent** will not be enqueued for cleared output. This method also stops any output that may be in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

Bump Bar State Diagram



Properties (UML attributes)

AsyncMode Property

Syntax	AsyncMode: <i>boolean</i> { read-write, access after open-claim-enable }
Remarks	<p>If true, then the bumpBarSound method will be performed asynchronously. If false, tones are generated synchronously.</p> <p>This property is initialized to false by the open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	bumpBarSound Method, “Device Output Models” on page 20.

AutoToneDuration Property

Syntax	AutoToneDuration: <i>int32</i> { read-write, access after open-claim-enable }
Remarks	<p>Holds the duration (in milliseconds) of the automatic tone for the bump bar unit specified by the CurrentUnitID property.</p> <p>This property is initialized to the default value for each online bump bar unit when the device is first enabled following the open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	CurrentUnitID Property.

AutoToneFrequency Property

Syntax	AutoToneFrequency: <i>int32</i> { read-write, access after open-claim-enable }
Remarks	<p>Holds the frequency (in Hertz) of the automatic tone for the bump bar unit specified by the CurrentUnitID property.</p> <p>This property is initialized to the default value for each online bump bar unit when the device is first enabled following the open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	CurrentUnitID Property.

BumpBarDataCount Property

Syntax	BumpBarDataCount: <i>int32</i> { read-only, access after open-claim-enable }
Remarks	<p>Holds the number of DataEvents enqueued for the bump bar unit specified by the CurrentUnitID 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 open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	CurrentUnitID Property, DataEvent .

CapTone Property

Syntax	CapTone: <i>boolean</i> { read-only, access after open-claim-enable }
Remarks	<p>If true, the bump bar unit specified by the CurrentUnitID property supports an enunciator.</p> <p>This property is initialized when the device is first enabled following the open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	CurrentUnitID Property.

CurrentUnitID Property

Syntax	CurrentUnitID: <i>int32</i> { read-write, access after open-claim-enable }
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">• Properties: AutoToneDuration, AutoToneFrequency, BumpBarDataCount, CapTone, and Keys.• Methods: checkHealth, clearInput, clearOutput. <p>This property is initialized to BB_UID_1 when the device is first enabled following the open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

DataCount Property

Syntax	DataCount: <i>int32</i> { read-only, access after open }
Remarks	<p>Holds the total number of DataEvents enqueued. All units online are included in this value. The number of enqueued events for a specific unit ID is stored in the BumpBarDataCount 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 open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	BumpBarDataCount Property, DataEvent Event, “Device Input Model” on page 17.

ErrorString Property

Syntax	ErrorString: <i>string</i> { read-only, access after open }
Remarks	<p>Holds a description of the error which occurred on the unit(s) specified by the ErrorUnits 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 ErrorEvent updates the property EventString instead.</p> <p>This property is initialized to an empty string by the open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	ErrorUnits Property.

ErrorUnits Property

Syntax	ErrorUnits: <i>int32</i> { read-only, access after open }
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 ErrorEvent updates the property EventUnits instead.</p> <p>This property is initialized to zero by the open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	ErrorString Property.

EventString Property

Syntax	EventString: <i>string</i> { read-only, access after open-claim }
Remarks	<p>Holds a description of the error which occurred to the unit(s) specified by the EventUnits property, when an ErrorEvent is delivered.</p> <p>This property is initialized to an empty string by the open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	EventUnits Property, ErrorEvent .

EventUnitID Property

Syntax	EventUnitID: <i>int32</i> { read-only, access after open-claim }
Remarks	Holds the bump bar unit ID causing a DataEvent . This property is set just before a DataEvent is delivered. The unit ID definitions range from BB_UID_1 to BB_UID_32.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	DataEvent .

EventUnits Property

Syntax	EventUnits: <i>int32</i> { read-only, access after open-claim }
Remarks	Holds a bitwise mask of the unit(s) when an OutputCompleteEvent , ErrorEvent , or StatusUpdateEvent is delivered. This property is initialized to zero by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	OutputCompleteEvent , ErrorEvent , StatusUpdateEvent .

Keys Property

Syntax	Keys: <i>int32</i> { read-only, access after open-claim-enable }
Remarks	Holds the number of keys on the bump bar unit specified by the CurrentUnitID property. This property is initialized when the device is first enabled following the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	CurrentUnitID Property.

Timeout Property

Syntax	Timeout: <i>int32</i> { read-write, access after open }
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 notifies the application of the error.</p> <p>This property is initialized to a Service dependent timeout following the open method.</p>
Errors	A <code>UposException</code> may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	AsyncMode Property, ErrorString Property, bumpBarSound Method.

UnitsOnline Property

Syntax	UnitsOnline: <i>int32</i> { read-only, access after open-claim-enable }
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 open method. This property is updated as changes are detected, such as before a StatusUpdateEvent is enqueued and during the checkHealth method.</p>
Errors	A <code>UposException</code> may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	checkHealth Method, StatusUpdateEvent .

Methods (UML operations)

bumpBarSound Method

Syntax **bumpBarSound** (**units**: *int32*, **frequency**: *int32*, **duration**: *int32*,
numberOfCycles: *int32*, **interSoundWait**: *int32*):
void { raises-exception, use after open-claim-enable }

Parameter	Description
<i>units</i>	Bitwise mask indicating which bump bar unit(s) to operate on.
<i>frequency</i>	Tone frequency in Hertz.
<i>duration</i>	Tone duration in milliseconds.
<i>numberOfCycles</i>	If FOREVER, then start bump bar sounding and, repeat continuously. Else perform the specified number of cycles.
<i>interSoundWait</i>	When numberOfCycles is not one, then pause for interSoundWait milliseconds before repeating the tone cycle (before playing the tone again)

Remarks Sounds the bump bar enunciator for the bump bar(s) specified by the *units* parameter.

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

The duration of a tone cycle is:

duration parameter + *interSoundWait* parameter (except on the last tone cycle)

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

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

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

checkHealth Method (Common)

Syntax **checkHealth (level: *int32*):**
 void { raises-exception, use after open-claim-enable }

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

Value	Meaning
CH_INTERNAL	Perform a health check that does not physically change the device. The device is tested by internal tests to the extent possible.
CH_EXTERNAL	Perform a more thorough test that may change the device.
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 CH_INTERNAL or 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 CH_INTERACTIVE health check operation is up to the 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 UpoException 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
E_FAILURE	An error occurred while communicating with the bump bar unit specified by the CurrentUnitID property.

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

clearInput Method (Common)

Syntax	clearInput (): void { raises-exception, use after open-claim }
Remarks	<p>Clears the device input that has been buffered for the unit specified by the CurrentUnitID property.</p> <p>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.</p>
Errors	A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 15.
See Also	CurrentUnitID Property, “Device Input Model” on page 17.

clearOutput Method (Common)

Syntax	clearOutput (): void { raises-exception, use after open-claim }
Remarks	<p>Clears the tone outputs that have been buffered for the unit specified by the CurrentUnitID property.</p> <p>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.</p>
Errors	A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 15.
See Also	CurrentUnitID Property, “Device Output Models” on page 20.

setKeyTranslation Method

Syntax **setKeyTranslation (units: *int32*, scanCode: *int32*, logicalKey: *int32*):**
 void { raises-exception, use after open-claim-enable }

Parameter	Description
<i>units</i>	Bitwise mask indicating which bump bar unit(s) to set key translation for.
<i>scanCode</i>	The bump bar generated key scan code. Valid values 0-255.
<i>logicalKey</i>	The translated logical key value. Valid values 0-255.

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

Errors A UpoException 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
E_ILLEGAL	One of the following errors occurred: <i>scanCode</i> or <i>logicalKey</i> are out of range. <i>units</i> is zero or a non-existent unit was specified. The ErrorUnits and ErrorString properties are updated prior to notifying the application of the error.

See Also **ErrorUnits** Property, **ErrorString** Property, **DataEvent**.

Events (UML interfaces)

DataEvent

<< event >> **upos::events::DataEvent**
Status: *int32* {read-only }

Description Notifies the application when status from the bump bar is available.

Attributes This event contains the following attribute:

Attributes	Type	Description
------------	------	-------------

<i>Status</i>	<i>int32</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	Low Byte	
Unused. Always zero.	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 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 17, **EventUnitID** Property, **DataEventEnabled** Property, **FreezeEvents** Property.

DirectIOEvent**<< event >> upos::events::DirectIOEvent****EventNumber:** *int32* { read-only }**Data:** *int32* { read-write }**Obj:** *object* { read-write }

Description Provides Service information directly to the application. This event provides a means for a vendor-specific Bump Bar 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>int32</i>	Event number whose specific values are assigned by the Device Service.
<i>Data</i>	<i>int32</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 Bump Bar devices which may not have any knowledge of the Device Service's need for this event.

See Also "Events" on page 14, **directIO** Method.

ErrorEvent

```
<< event >> upos::events::ErrorEvent
    ErrorCode: int32 { read-only }
    ErrorCodeExtended: int32 { read-only }
    ErrorLocus: int32 { read-only }
    ErrorResponse: int32 { read-write }
```

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.

Attributes This event contains the following properties:

Attributes	Type	Description
<i>ErrorCode</i>	<i>int32</i>	Result code causing the error event. See a list of Error Codes on page 15.
<i>ErrorCodeExtended</i>	<i>int32</i>	Extended Error code causing the error event. If <i>ErrorCode</i> is E_EXTENDED, then see values below. Otherwise, it may contain a Service-specific value.
<i>ErrorLocus</i>	<i>int32</i>	Location of the error. See values below.
<i>ErrorResponse</i>	<i>int32</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
EL_OUTPUT	Error occurred while processing asynchronous output.
EL_INPUT	Error occurred while gathering or processing event-driven input. No input data is available.
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
ER_RETRY	Use only when locus is EL_OUTPUT. Retry the asynchronous output. The error state is exited. Default when locus is EL_OUTPUT.
ER_CLEAR	Clear the buffered input data. The error state is exited. Default when locus is EL_INPUT.
ER_CONTINUEINPUT	Use only when locus is 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 EL_INPUT. Default when locus is 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 20, “Device States” on page 25, **DataEventEnabled** Property, **EventUnits** Property, **EventString** Property.

OutputCompleteEvent

<< event >> **upos::events::OutputCompleteEvent**
OutputID: int32 { read-only }

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

Attributes This event contains the following attribute:

Attributes	Type	Description
<i>OutputID</i>	<i>int32</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 20.

StatusUpdateEvent

<< event >> **upos::events::StatusUpdateEvent**
Status: int32 { read-only }

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

Attributes This event contains the following attribute:

Attributes	Type	Description
<i>Status</i>	<i>int32</i>	Reports a change in the power state of a bump bar unit. <i>Note that Release 1.3</i> added Power State Reporting with additional <i>Power reporting</i> StatusUpdateEvent values. See “StatusUpdateEvent” description on page 56.

Remarks Enqueued when the bump bar device detects a power state change.
 Deviation from the standard **StatusUpdateEvent** (See “StatusUpdateEvent” description on page 56)

- 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 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

This Chapter defines the Cash Changer device category.

Summary

Properties (UML attributes)

<i>Common</i>	<i>Type</i>	<i>Mutability</i>	<i>Version^a</i>	<i>May Use After</i>
AutoDisable:	<i>boolean</i>	{read-write}	1.2	Not Supported
CapPowerReporting:	<i>int32</i>	{read-only}	1.3	open
CheckHealthText:	<i>string</i>	{read-only}	1.2	open
Claimed:	<i>boolean</i>	{read-only}	1.2	open
DataCount:	<i>int32</i>	{read-only}	1.5	open
DataEventEnabled:	<i>boolean</i>	{read-write}	1.5	open
DeviceEnabled:	<i>boolean</i>	{read-write}	1.2	open & claim
FreezeEvents:	<i>boolean</i>	{read-write}	1.2	open
OutputID:	<i>int32</i>	{read-only}	1.2	Not Supported
PowerNotify:	<i>int32</i>	{read-write}	1.3	open
PowerState:	<i>int32</i>	{read-only}	1.3	open
State:	<i>int32</i>	{read-only}	1.2	--
DeviceControlDescription:	<i>string</i>	{read-only}	1.2	--
DeviceControlVersion:	<i>int32</i>	{read-only}	1.2	--
DeviceServiceDescription:	<i>string</i>	{read-only}	1.2	open
DeviceServiceVersion:	<i>int32</i>	{read-only}	1.2	open
PhysicalDeviceDescription:	<i>string</i>	{read-only}	1.2	open
PhysicalDeviceName:	<i>string</i>	{read-only}	1.2	open

- a. The version representation provides the mechanism for recognizing when a change occurs to a property, method or event. This CashChanger definition was introduced in an existing standard and was not changed for the UnifiedPOS version 1.4.

Properties (Continued)

<i>Specific (continued)</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>	<i>May Use After</i>
CapDeposit:	<i>boolean</i>	{ read-only }	1.5	open
CapDepositDataEvent:	<i>boolean</i>	{ read-only }	1.5	open
CapDiscrepancy:	<i>boolean</i>	{ read-only }	1.2	open
CapEmptySensor:	<i>boolean</i>	{ read-only }	1.2	open
CapFullSensor:	<i>boolean</i>	{ read-only }	1.2	open
CapNearEmptySensor:	<i>boolean</i>	{ read-only }	1.2	open
CapNearFullSensor:	<i>boolean</i>	{ read-only }	1.2	open
CapPauseDeposit:	<i>boolean</i>	{ read-only }	1.5	open
CapRepayDeposit:	<i>boolean</i>	{ read-only }	1.5	open
AsyncMode:	<i>boolean</i>	{ read-write }	1.2	open
AsyncResultCode:	<i>int32</i>	{ read-only }	1.2	open, claim, & enable
AsyncResultCodeExtended:	<i>int32</i>	{ read-only }	1.2	open, claim, & enable
CurrencyCashList:	<i>string</i>	{ read-only }	1.2	open
CurrencyCode:	<i>string</i>	{ read-write }	1.2	open
CurrencyCodeList:	<i>string</i>	{ read-only }	1.2	open
CurrentExit:	<i>int32</i>	{ read-write }	1.2	open
DepositAmount:	<i>int32</i>	{ read-only }	1.5	open
DepositCashList:	<i>string</i>	{ read-only }	1.5	open
DepositCodeList:	<i>string</i>	{ read-only }	1.5	open
DepositCounts:	<i>string</i>	{ read-only }	1.5	open
DepositStatus:	<i>int32</i>	{ read-only }	1.5	open, claim, & enable
DeviceExits:	<i>int32</i>	{ read-only }	1.2	open
DeviceStatus:	<i>int32</i>	{ read-only }	1.2	open, claim, & enable
ExitCashList:	<i>string</i>	{ read-only }	1.2	open
FullStatus:	<i>int32</i>	{ read-only }	1.2	open, claim, & enable

Methods (UML operations)

Common

Name

open (logicalDeviceName: *string*):
void { raises exception }

close ():
void { raises exception, use after open }

claim (timeout: *int32*):
void { raises exception, use after open }

release ():
void { raises exception, use after open, claim }

checkHealth (level: *int32*):
void { raises exception, use after open, claim, enable }

clearInput ():
void { raises exception, use after open, claim }

clearOutput (): *Not supported*
void { }

directIO (command: *int32*, inout data: *int32*, inout obj: *object*):
void { raises exception, use after open }

Specific

beginDeposit ():
void { raises exception, use after open, claim, enable }

dispenseCash (cashCounts: *string*):
void { raises exception, use after open, claim, enable }

dispenseChange (amount: *int32*):
void { raises exception, use after open, claim, enable }

endDeposit (success: *int32*):
void { raises exception, use after open, claim, enable }

fixDeposit ():
void { raises exception, use after open, claim, enable }

pauseDeposit (control: *int32*):
void { raises exception, use after open, claim, enable }

readCashCounts (inout cashCounts: *string*, inout discrepancy: *boolean*):
void { raises exception, use after open, claim, enable }

Events (UML interfaces)

<i>Name</i>	<i>Type</i>	<i>Mutability</i>
upos::events::DataEvent		
Status:	<i>int32</i>	{ read-only }
upos::events::DirectIOEvent		
EventNumber:	<i>int32</i>	{ read-only }
Data:	<i>int32</i>	{ read-write }
Obj:	<i>object</i>	{ read-write }
upos::events::StatusUpdateEvent		
Status:	<i>int32</i>	{ read-only }

General Information

The Cash Changer programmatic name is “CashChanger”.

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:

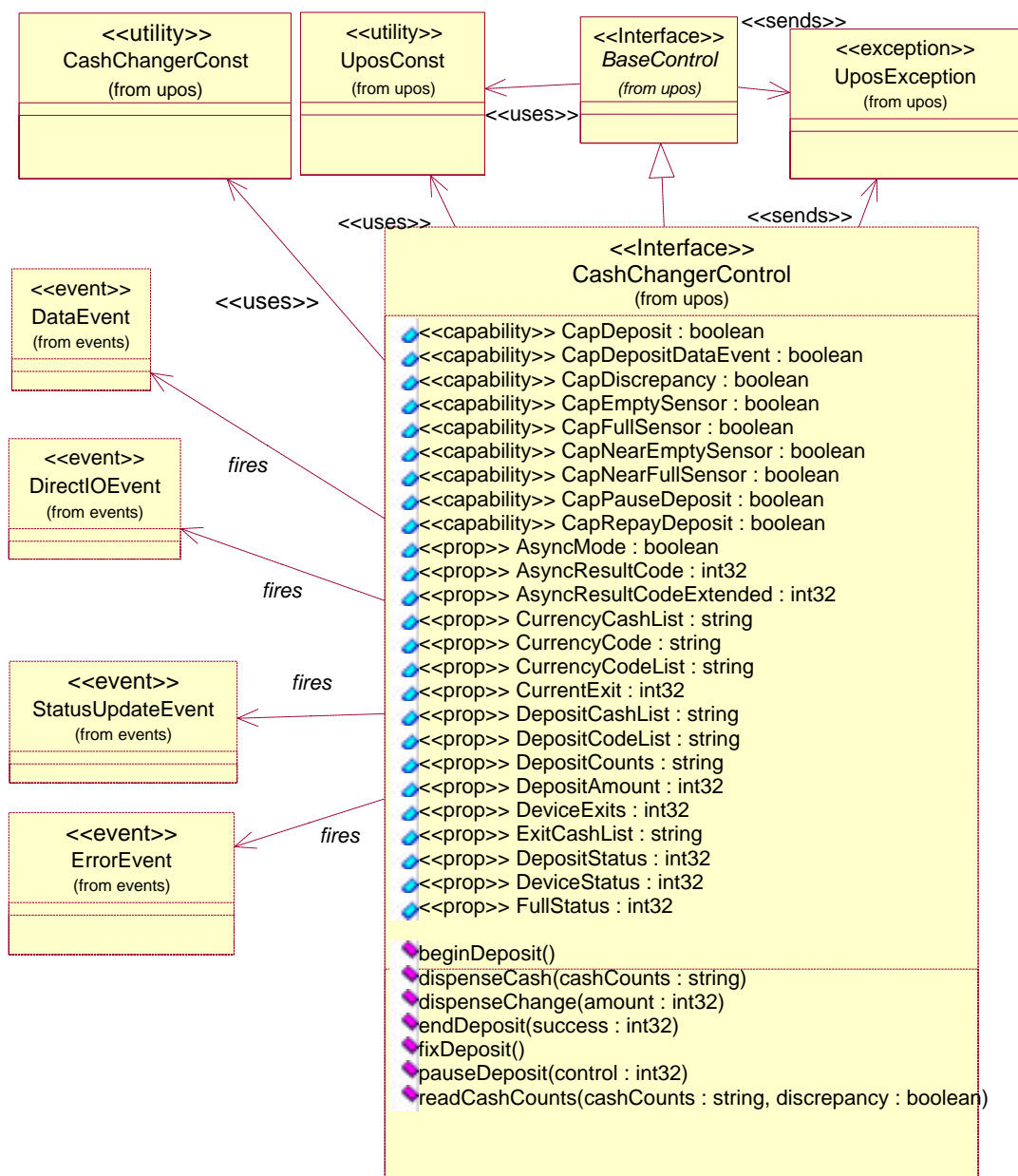
- Reporting the fullness levels of the Cash Changer’s cash units. Conditions which may be indicated include empty, near empty, full, and near full states.
- Reporting of a possible (or probable) cash count discrepancy in the data reported by the **readCashCounts** method.

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.

CashChanger Class Diagram

The following diagram shows the relationships between the CashChanger classes.



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 **UpoxException** when an error occurs during synchronous dispensing.

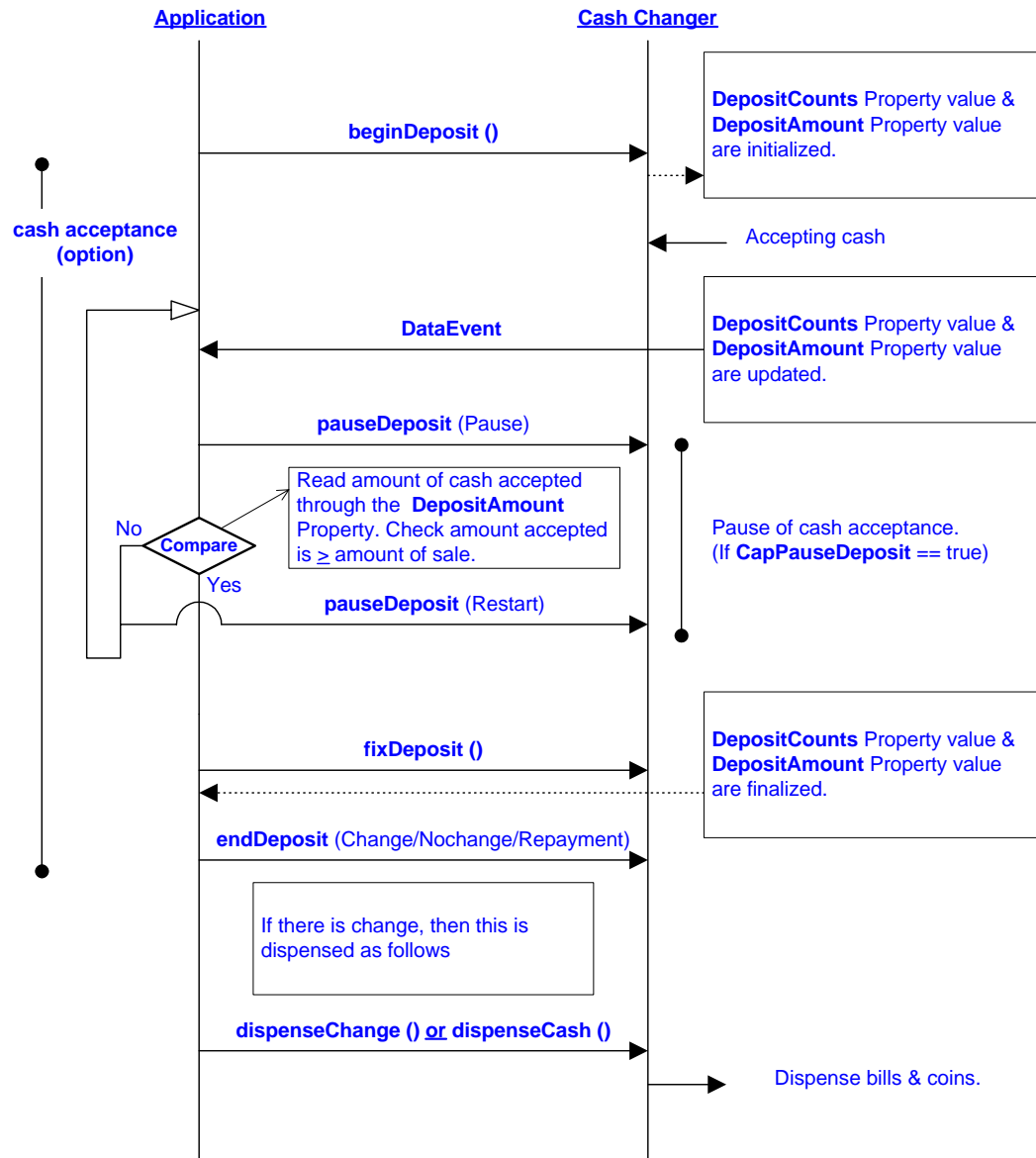
Nesting of asynchronous Cash Changer operations is illegal; only one asynchronous method can be processed at a time.

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

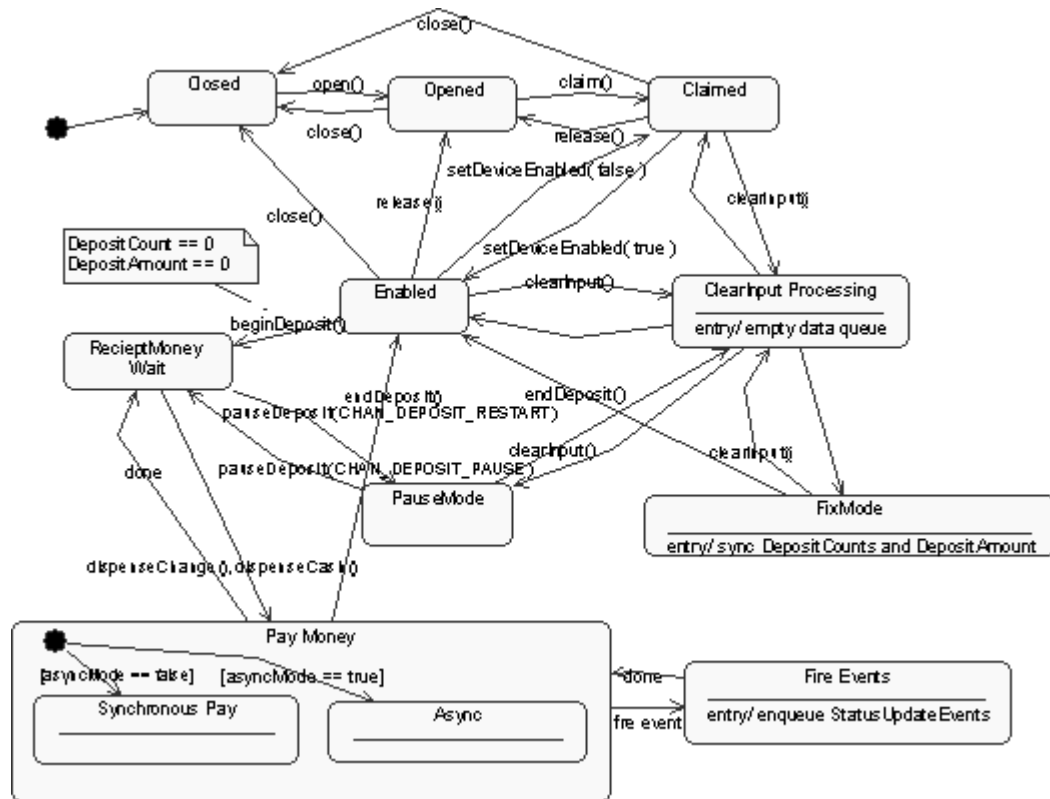
- May support more than one currency. The **CurrencyCode** property may be set to the currency, selecting from a currency in the list **CurrencyCodeList**. **CurrencyCashList**, **ExitCashList**, **dispenseCash**, **dispenseChange** and **readCashCounts** all act upon the current currency only.
- Sets 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.



Cash Changer State 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 events.
- See the “Summary” table for precise usage prerequisites.

Properties (UML attributes)

AsyncMode Property

Syntax	AsyncMode: <i>boolean</i> { read-write, access after open }
Remarks	If true, the dispenseCash and dispenseChange methods will be performed asynchronously. If false, these methods will be performed synchronously. This property is initialized to false by the Open method.
Errors	A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 15.
See Also	AsyncResultCode Property, AsyncResultCodeExtended Property, dispenseChange Method, dispenseCash Method.

AsyncResultCode Property

Syntax	AsyncResultCode: <i>int32</i> { read-only, access after open-claim-enable }
Remarks	Holds the completion status of the last asynchronous dispense request (i.e., when dispenseCash or dispenseChange was called with AsyncMode true). This property is set before a StatusUpdateEvent event is delivered with a <i>Status</i> value of CHAN_STATUS_ASYNC .
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	AsyncMode Property, dispenseCash Method, dispenseChange Method.

AsyncResultCodeExtended Property

Syntax	AsyncResultCodeExtended: <i>int32</i> { read-only, access after open-claim-enable }
Remarks	Holds the completion status of the last asynchronous dispense request (i.e., when dispenseCash or dispenseChange was called with AsyncMode true). This property is set before a StatusUpdateEvent event is delivered with a <i>Status</i> value of CHAN_STATUS_ASYNC .
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	AsyncMode Property, dispenseCash Method, dispenseChange Method.

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

Syntax	CapDeposit: <i>boolean</i> { read-only, access after open }
Remarks	If true, the Cash Changer supports cash acceptance. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	beginDeposit Method, endDeposit Method, fixDeposit Method, pauseDeposit Method.

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

Syntax	CapDepositDataEvent: <i>boolean</i> { read-only, access after open }
Remarks	If true, the Cash Changer can report a cash acceptance event. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	beginDeposit Method, endDeposit Method, fixDeposit Method, pauseDeposit Method.

CapDiscrepancy Property

Syntax	CapDiscrepancy: <i>boolean</i> { read-only, access after open }
Remarks	If true, the readCashCounts method can report effective <i>discrepancy</i> values. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	readCashCounts Method.

CapEmptySensor Property

Syntax	CapEmptySensor: <i>boolean</i> { read-only, access after open }
Remarks	If true, the Cash Changer can report the condition that some cash slots are empty. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	DeviceStatus Property, StatusUpdateEvent .

CapFullSensor Property

Syntax	CapFullSensor: <i>boolean</i> { read-only, access after open }
Remarks	If true, the Cash Changer can report the condition that some cash slots are full. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	FullStatus Property, StatusUpdateEvent .

CapNearEmptySensor Property

Syntax	CapNearEmptySensor: <i>boolean</i> { read-only, access after open }
Remarks	If true, the Cash Changer can report the condition that some cash slots are nearly empty. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	DeviceStatus Property, StatusUpdateEvent .

CapNearFullSensor Property

Syntax	CapNearFullSensor: <i>boolean</i> { read-only, access after open }
Remarks	If true, the Cash Changer can report the condition that some cash slots are nearly full. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	FullStatus Property, StatusUpdateEvent .

CapPauseDeposit Property *Added in Release 1.5*

Syntax	CapPauseDeposit: <i>boolean</i> { read-only, access after open }
Remarks	If true, the Cash Changer has the capability to suspend cash acceptance processing temporarily. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	pauseDeposit Method.

CapRepayDeposit Property *Added in Release 1.5*

Syntax	CapRepayDeposit: <i>boolean</i> { read-only, access after open }
Remarks	If true, the Cash Changer has the capability to return money that was deposited. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	endDeposit Method.

CurrencyCashList Property

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

CurrencyCode Property

Syntax	CurrencyCode: <i>string</i> { read-write, access after open }				
Remarks	Contains the active currency code to be used by Cash Changer operations. This property is initialized to an appropriate value by the open method. This value is guaranteed to be one of the set of currencies specified by the CurrencyCodeList property.				
Errors	<p>A UpoException 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><tr><th>Value</th><th>Meaning</th></tr><tr><td>E_ILLEGAL</td><td>A value was specified that is not within CurrencyCodeList.</td></tr></table>	Value	Meaning	E_ILLEGAL	A value was specified that is not within CurrencyCodeList .
Value	Meaning				
E_ILLEGAL	A value was specified that is not within CurrencyCodeList .				
See Also	CurrencyCodeList Property.				

CurrencyCodeList Property

Syntax	CurrencyCodeList: <i>string</i> { read-only, access after open }
Remarks	<p>Holds 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 open method.</p>
Errors	A UpoException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	CurrencyCode Property.

CurrentExit Property

Syntax **CurrentExit:** *int32* { read-write, access after open }

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

Below are examples of typical property value sets in Japan. **CurrencyCode** is “JPY” and **CurrencyCodeList** is “JPY”.

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

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

Errors A *UposException* 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
E_ILLEGAL	An invalid CurrentExit value was specified.

See Also **CurrencyCashList** Property, **DeviceExits** Property, **ExitCashList** Property.

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

Syntax	DepositAmount: <i>int32</i> { read-only, access after open }
Remarks	<p>The total amount of deposited cash.</p> <p>For example, if the currency is Japanese yen and DepositAmount is set to 18057, after the call to the beginDeposit method, there would be 18,057 yen in the Cash Changer.</p> <p>This property is initialized by the open method.</p>
Errors	A UpoException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	CurrencyCode Property.

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

Syntax	DepositCashList: <i>string</i> { read-only, access after open }
Remarks	<p>Holds the cash units supported in the Cash Changer for the currency represented by the CurrencyCode 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 DepositCashList values in Japan.</p> <ul style="list-style-type: none"> • “1,5,10,50,100,500” --- 1, 5, 10, 50, 100, 500 yen coin. • “1,5,10,50,100,500;1000,5000,10000” --- 1, 5, 10, 50, 100, 500 yen coin and 1000, 5000, 10000 yen bill. • “;1000,5000,10000” --- 1000, 5000, 10000 yen bill. <p>This property is initialized by the open method, and is updated when CurrencyCode is set.</p>
Errors	A UpoException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	CurrencyCode Property.

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

Syntax	DepositCodeList: <i>string</i> { read-only, access after open }
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 open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	CurrencyCode Property.

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

Syntax	DepositCounts: <i>string</i> { read-only, access after open }
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 dispenseCash method. Cash units inside the string are the same as the DepositCashList 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 DepositCounts 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 beginDeposit 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 open method</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	CurrencyCode Property.

DepositStatus Property***Added in Release 1.5***

Syntax	DepositStatus: <i>int32</i> { read-only, access after open-claim-enable }
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 UpoException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
---------------	--

DeviceExits Property

Syntax	DeviceExits: <i>int32</i> { read-only, access after open }
Remarks	The number of exits for dispensing cash. This property is initialized by the open method.
Errors	A UpoException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	CurrentExit Property.

DeviceStatus Property

Syntax	DeviceStatus: <i>int32</i> { read-only, access after open-claim-enable }										
Remarks	Holds the current status of the Cash Changer. It may be one of the following: <table> <tr> <th>Value</th><th>Meaning</th></tr> <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> </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: fault, empty, and near empty.</p>	Value	Meaning	CHAN_STATUS_OK	The current condition of the Cash Changer is satisfactory.	CHAN_STATUS_EMPTY	Some cash slots are empty.	CHAN_STATUS_NEAREMPTY	Some cash slots are nearly empty.	CHAN_STATUS_JAM	A mechanical fault has occurred.
Value	Meaning										
CHAN_STATUS_OK	The current condition of the Cash Changer is satisfactory.										
CHAN_STATUS_EMPTY	Some cash slots are empty.										
CHAN_STATUS_NEAREMPTY	Some cash slots are nearly empty.										
CHAN_STATUS_JAM	A mechanical fault has occurred.										
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.										

ExitCashList Property

Syntax	ExitCashList: <i>string</i> { read-only, access after open }
Remarks	Holds the cash units which may be dispensed to the exit which is denoted by CurrentExit property. The supported cash units are either the same as CurrencyCashList , or a subset of it. The string format is identical to that of CurrencyCashList . This property is initialized by the open method, and is updated when CurrencyCode or CurrentExit is set.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	CurrencyCode Property, CurrencyCashList Property, CurrentExit Property.

FullStatus Property

Syntax	FullStatus: <i>int32</i> { read-only, access after open }								
Remarks	Holds the current full status of the cash slots. It may be one of the following: <table> <tr> <th>Value</th><th>Meaning</th></tr> <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> </table> <p>This property is initialized and kept current while the device is enabled.</p>	Value	Meaning	CHAN_STATUS_OK	All cash slots are neither nearly full nor full.	CHAN_STATUS_FULL	Some cash slots are full.	CHAN_STATUS_NEARFULL	Some cash slots are nearly full.
Value	Meaning								
CHAN_STATUS_OK	All cash slots are neither nearly full nor full.								
CHAN_STATUS_FULL	Some cash slots are full.								
CHAN_STATUS_NEARFULL	Some cash slots are nearly full.								
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.								

Methods (UML operations)

beginDeposit Method

Added in Release 1.5

Syntax	beginDeposit (): void { raises-exception, use after open-claim-enable }				
Remarks	<p>Cash acceptance is started.</p> <p>The following property values are initialized by the call to this method:</p> <ul style="list-style-type: none"> • The value of each cash unit of the DepositCounts property is set to zero. • The DepositAmount property is set to zero. <p>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.</p>				
Errors	<p>A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 15.</p> <p>Some possible values of the exception’s <i>ErrorCode</i> property are:</p> <table> <tr> <th>Value</th><th>Meaning</th></tr> <tr> <td>E_ILLEGAL</td><td>Either the Cash Changer does not support cash acceptance, or the call sequence is not correct.</td></tr> </table>	Value	Meaning	E_ILLEGAL	Either the Cash Changer does not support cash acceptance, or the call sequence is not correct.
Value	Meaning				
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

Syntax **dispenseCash (cashCounts: *string*):**
 void { raises-exception, use after open-claim-enable }

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

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

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

Some *cashCounts* examples, using Japanese yen as the currency, are shown below.

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

Errors A *UpoException* 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
E_BUSY	Cash cannot be dispensed because an asynchronous method is in progress.
E_ILLEGAL	One of the following errors occurred: <ul style="list-style-type: none"> • The <i>cashCounts</i> parameter value was illegal for the current exit. • Cash could not be dispensed because cash acceptance was in progress.
E_EXTENDED	<i>ErrorCodeExtended</i> = ECHAN_OVERDISPENSE: The specified cash cannot be dispensed because of a cash shortage.

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

dispenseChange Method

Syntax **dispenseChange (amount: *int32*):**
 void { raises-exception, use after open-claim-enable }

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 UposException 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
E_BUSY	The specified change cannot be dispensed because an asynchronous method is in progress.
E_ILLEGAL	One of the following errors occurred: <ul style="list-style-type: none">• A negative or zero <i>amount</i> was specified.• The <i>amount</i> could not be dispensed based on the values specified in ExitCashList for the current exit.• Change could not be dispensed because cash acceptance was in progress.
E_EXTENDED	<i>ErrorCodeExtended</i> = ECHAN_OVERDISPENSE : The specified change can not be dispensed because of a cash shortage.

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

endDeposit Method***Added in Release 1.5***

Syntax **endDeposit (success: *int32*):**
 void { raises-exception, use after open-claim-enable }

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

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

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

Syntax	fixDeposit (): void { raises-exception, use after open-claim-enable }				
Remarks	When this method is called, all property values are updated to reflect the current values in the Cash Changer.				
Errors	<p>A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 15.</p> <p>Some possible values of the exception’s <i>ErrorCode</i> property are:</p> <table><tr><th>Value</th><th>Meaning</th></tr><tr><td>E_ILLEGAL</td><td>One of the following errors occurred:<ul style="list-style-type: none">• Cash acceptance is not supported.• The call sequence is invalid. beginDeposit must be called before calling this method.</td></tr></table>	Value	Meaning	E_ILLEGAL	One of the following errors occurred: <ul style="list-style-type: none">• Cash acceptance is not supported.• The call sequence is invalid. beginDeposit must be called before calling this method.
Value	Meaning				
E_ILLEGAL	One of the following errors occurred: <ul style="list-style-type: none">• Cash acceptance is not supported.• The call sequence is invalid. beginDeposit must be called before calling this method.				
See Also	DepositCounts Property, DepositAmount Property, beginDeposit Method, endDeposit Method, pauseDeposit Method.				

pauseDeposit Method***Added in Release 1.5***

Syntax **pauseDeposit (control: *int32*):**
 void { raises-exception, use after open-claim-enable }

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

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

readCashCounts Method

Syntax **readCashCounts (inout cashCounts: *string*, inout discrepancy: *boolean*):**
 void { raises-exception, use after open-claim-enable }

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

Remarks The format of the string *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 returned in *cashCounts* 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. 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 `UposException` 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
E_BUSY	Cash units and counts can not be read because an asynchronous method is in process.

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

Events (UML interfaces)

DataEvent

Added in Release 1.5

```
<< event >> upos::events::DataEvent
    Status: int32 { read-only }
```

Description Notifies the application when the Cash Changer has a status change.

Attributes This event contains the following attribute:

Attributes	Type	Description
<i>Status</i>	<i>int32</i>	The <i>Status</i> parameter contains zero.

DirectIOEvent

```
<< event >> upos::events::DirectIOEvent
    EventNumber: int32 { read-only }
    Data: int32 { read-write }
    Obj: object { read-write }
```

Description Provides Service information directly to the application. This event provides a means for a vendor-specific Cash Changer 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>int32</i>	Event number whose specific values are assigned by the Device Service.
<i>Data</i>	<i>int32</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 Cash Changer devices which may not have any knowledge of the Device Service's need for this event.

See Also "Events" on page 14, **directIO** Method.

StatusUpdateEvent

<< event >> **upos::events::StatusUpdateEvent**
Status: int32 { read-only }

Description Notifies the application that there is a change in the power status of the Cash Changer device.

Attributes This event contains the following attribute:

Attributes	Type	Description
<i>Status</i>	<i>int32</i>	Indicates a change in the status of the unit. See values below.

Note that Release 1.3 added Power State Reporting with additional *Power reporting StatusUpdateEvent values*. See “StatusUpdateEvent” description on page 56.

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

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

Remarks Fired when the Cash Changer detects a status change.

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

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

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

See Also **AsyncResultCode** Property, **AsyncResultCodeExtended** Property, “Events” on page 14.

CHAPTER 4

Cash Drawer

This Chapter defines the Cash Drawer device category.

Summary

Properties (UML attributes)

<i>Common</i>	<i>Type</i>	<i>Mutability</i>	<i>Version^a</i>	<i>May Use After</i>
AutoDisable:	<i>boolean</i>	{ read-write }	1.4	Not Supported
CapPowerReporting:	<i>int32</i>	{ read-only }	1.4	open
CheckHealthText:	<i>string</i>	{ read-only }	1.4	open
Claimed:	<i>boolean</i>	{ read-only }	1.4	open
DataCount:	<i>int32</i>	{ read-only }	1.4	Not Supported
DataEventEnabled:	<i>boolean</i>	{ read-write }	1.4	Not Supported
DeviceEnabled:	<i>boolean</i>	{ read-write }	1.4	open
FreezeEvents:	<i>boolean</i>	{ read-write }	1.4	open
OutputID:	<i>int32</i>	{ read-only }	1.4	Not Supported
PowerNotify:	<i>int32</i>	{ read-write }	1.4	open
PowerState:	<i>int32</i>	{ read-only }	1.4	open
State:	<i>int32</i>	{ read-only }	1.4	--
DeviceControlDescription:	<i>string</i>	{ read-only }	1.4	--
DeviceControlVersion:	<i>int32</i>	{ read-only }	1.4	--
DeviceServiceDescription:	<i>string</i>	{ read-only }	1.4	open
DeviceServiceVersion:	<i>int32</i>	{ read-only }	1.4	open
PhysicalDeviceDescription:	<i>string</i>	{ read-only }	1.4	open
PhysicalDeviceName:	<i>string</i>	{ read-only }	1.4	open

- a. The version representation provides the mechanism for recognizing when a change occurs to a property, method or event. This POS Printer definition was introduced in an existing standard and was not changed for the UnifiedPOS version 1.4.

Properties (Continued)

<i>Specific</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>	<i>May Use After</i>
CapStatus:	<i>boolean</i>	{ read-only }	1.4	open
CapStatusMultiDrawerDetect:	<i>boolean</i>	{ read-only }	1.5	open
DrawerOpened:	<i>boolean</i>	{ read-only }	1.4	open & enable

Methods (UML operations)**Common***Name***open (logicalDeviceName: *string*):**

void { raises exception }

close ():

void { raises exception, use after open }

claim (timeout: *int32*):

void { raises exception, use after open }

release ():

void { raises exception, use after open, claim }

checkHealth (level: *int32*):

void { raises exception, use after open, enable }

*Note***clearInput ():**

void { }

*Not supported***clearOutput ():**

void { }

*Not supported***directIO (command: *int32*, inout data: *int32*, inout obj: *object*):**

void { raises exception, use after open }

Specific**openDrawer (timeout: *int32*):**

void { raises exception, use after open, enable }

*Note***waitForDrawerClose (timeout: *int32*):**

void { raises exception, use after open, enable }

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

Events (UML interfaces)

<i>Name</i>	<i>Type</i>	<i>Mutability</i>
upos::events::DirectIOEvent		
EventNumber:	<i>int32</i>	{ read-only }
Data:	<i>int32</i>	{ read-write }
Obj:	<i>object</i>	{ read-write }
upos::events::StatusUpdateEvent		
Status:	<i>int32</i>	{ read-only }

General Information

The Cash Drawer programmatic name is “CashDrawer”.

Capabilities

The Cash Drawer Control has the following capability:

- Supports a command to “open” the cash drawer.

The cash drawer may have the following additional capability:

- Drawer status reporting 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 (UML attributes)

CapStatus Property

Syntax	CapStatus: <i>boolean</i> { read-only, access after open }
Remarks	<p>If true, the drawer can report status. If false, the drawer is not able to determine whether the cash drawer is open or closed.</p> <p>This property is initialized by the open method.</p>
Errors	A UpoException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapStatusMultiDrawerDetect Property

Added in Release 1.5

Syntax	CapStatusMultiDrawerDetect: <i>boolean</i> { read-only, access after open }
Remarks	<p>If true, the status unique to each drawer in a multiple cash drawer configuration¹ can be reported.</p> <p>If false, the following possibilities exist:</p> <p>DrawerOpened: value of false indicates that there are no drawers open.</p> <p>DrawerOpened: 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.</p> <p>Note: 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.</p> <p>This property is only meaningful if CapStatus is true.</p> <p>This property is initialized by the open method.</p>
Errors	A UpoException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	CapStatus Property, DrawerOpened Property.

¹. 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

Syntax	DrawerOpened: <i>boolean</i> { read-only, access after open }
Remarks	<p>If true, the drawer is open. If false, the drawer is closed.</p> <p>If the capability CapStatus is false, then the device does not support status reporting, and this property is always false.</p> <p>Note: If the capability CapStatusMultiDrawerDetect is false, then a DrawerOpened 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 CapStatusMultiDrawerDetect for further clarification.</p> <p>This property is initialized and kept current while the device is enabled.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	CapStatus Property, CapStatusMultiDrawerDetect Property.

Methods (UML operations)

openDrawer Method

Syntax	openDrawer (): void { raises-exception, use after open-claim-enable }
Remarks	Opens the drawer.
Errors	A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 15.

waitForDrawerClose Method

Syntax	waitForDrawerClose (beepTimeout: <i>int32</i>, beepFrequency: <i>int32</i>, beepDuration: <i>int32</i>, beepDelay: <i>int32</i>): void { raises-exception, use after open-claim-enable }										
	<table> <tr> <th>Parameter</th><th>Description</th></tr> <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> </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.										
Remarks	<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 UposException 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 CapStatusMultiDrawerDetect 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 CapStatus is false, then the device does not support status reporting, and this method will return immediately.</p>										
Errors	A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 15.										
See Also	CapStatus Property, CapStatusMultiDrawerDetect Property.										

Events (UML interfaces)

DirectIOEvent

<< event >> **upos::events::DirectIOEvent**

EventNumber: *int32* { read-only }

Data: *int32* { read-write }

Obj: *object* { read-write }

Description Provides Service information directly to the application. This event provides a means for a vendor-specific Cash Drawer 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>int32</i>	Event number whose specific values are assigned by the Service.
<i>Data</i>	<i>int32</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.

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 Cash Drawer devices which may not have any knowledge of the Service's need for this event.

See Also "Events" on page 14, **directIO** Method.

StatusUpdateEvent

<< event >> **upos::events::StatusUpdateEvent**
Status: *int32* { read-only }

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

Attributes This event contains the following attribute:

Attributes	Type	Description
<i>Status</i>	<i>int32</i>	The status reported from the Cash Drawer.

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

Value	Meaning
CASH_SUE_DRAWERCLOSED	The drawer is closed.
CASH_SUE_DRAWEROPEN	The drawer is open.

Note that Release 1.3 added Power State Reporting with additional *Power reporting StatusUpdateEvent values*. See “StatusUpdateEvent” on page 56.

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

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 14, **CapStatus** Property, **CapStatusMultiDrawerDetect** Property.

CAT - Credit Authorization Terminal

This Chapter defines the Credit Authorization Terminal device category.

Summary

Properties (UML attributes)

<i>Common</i>	<i>Type</i>	<i>Mutability</i>	<i>Version^a</i>	<i>May Use After</i>
AutoDisable:	<i>boolean</i>	{ read-write }	1.4	Not Supported
CapPowerReporting:	<i>int32</i>	{ read-only }	1.4	open
CheckHealthText:	<i>string</i>	{ read-only }	1.4	open
Claimed:	<i>boolean</i>	{ read-only }	1.4	open
DataCount:	<i>int32</i>	{ read-only }	1.4	Not Supported
DataEventEnabled:	<i>boolean</i>	{ read-write }	1.4	Not Supported
DeviceEnabled:	<i>boolean</i>	{ read-write }	1.4	open & claim
FreezeEvents:	<i>boolean</i>	{ read-write }	1.4	open
OutputID:	<i>int32</i>	{ read-only }	1.4	open
PowerNotify:	<i>int32</i>	{ read-write }	1.4	open
PowerState:	<i>int32</i>	{ read-only }	1.4	open
State:	<i>int32</i>	{ read-only }	1.4	--
DeviceControlDescription:	<i>string</i>	{ read-only }	1.4	--
DeviceControlVersion:	<i>int32</i>	{ read-only }	1.4	--
DeviceServiceDescription:	<i>string</i>	{ read-only }	1.4	open
DeviceServiceVersion:	<i>int32</i>	{ read-only }	1.4	open
PhysicalDeviceDescription:	<i>string</i>	{ read-only }	1.4	open
PhysicalDeviceName:	<i>string</i>	{ read-only }	1.4	open

- a. The version representation provides the mechanism for recognizing when a change occurs to a property, method or event. The CAT device was introduced in an existing standard and was added for the UnifiedPOS version 1.5.

Properties (Continued)

<i>Specific</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>	<i>May Use After</i>
AccountNumber:	<i>string</i>	{ read-only }	1.4	open
AdditionalSecurityInformation:	<i>string</i>	{ read-write }	1.4	open
ApprovalCode:	<i>string</i>	{ read-only }	1.4	open
AsyncMode:	<i>boolean</i>	{ read-write }	1.4	open
CapAdditionalSecurityInformation:	<i>boolean</i>	{ read-only }	1.4	open
CapAuthorizeCompletion:	<i>boolean</i>	{ read-only }	1.4	open
CapAuthorizePreSales:	<i>boolean</i>	{ read-only }	1.4	open
CapAuthorizeRefund:	<i>boolean</i>	{ read-only }	1.4	open
CapAuthorizeVoid:	<i>boolean</i>	{ read-only }	1.4	open
CapAuthorizeVoidPreSales:	<i>boolean</i>	{ read-only }	1.4	open
CapCenterResultCode:	<i>boolean</i>	{ read-only }	1.4	open
CapCheckCard:	<i>boolean</i>	{ read-only }	1.4	open
CapDailyLog:	<i>int32</i>	{ read-only }	1.4	open
CapInstallments:	<i>boolean</i>	{ read-only }	1.4	open
CapPaymentDetail:	<i>boolean</i>	{ read-only }	1.4	open
CapTaxOthers:	<i>boolean</i>	{ read-only }	1.4	open
CapTransactionNumber:	<i>boolean</i>	{ read-only }	1.4	open
CapTrainingMode:	<i>boolean</i>	{ read-only }	1.4	open
CardCompanyID:	<i>string</i>	{ read-only }	1.4	open
CenterResultCode:	<i>string</i>	{ read-only }	1.4	open
DailyLog:	<i>string</i>	{ read-only }	1.4	open
PaymentCondition:	<i>int32</i>	{ read-only }	1.4	open
PaymentDetail:	<i>string</i>	{ read-only }	1.4	open
PaymentMedia:	<i>int32</i>	{ read-write }	1.5	open
SequenceNumber:	<i>int32</i>	{ read-only }	1.4	open
SlipNumber:	<i>string</i>	{ read-only }	1.4	open
TrainingMode:	<i>boolean</i>	{ read-write }	1.4	open
TransactionNumber:	<i>string</i>	{ read-only }	1.4	open
TransactionType:	<i>int32</i>	{ read-only }	1.4	open

Methods (UML operations)

Common

Name

open (logicalDeviceName: *string*):
void { raises exception }

close ():
void { raises exception, use after open }

claim (timeout: *int32*):
void { raises exception, use after open }

release ():
void { raises exception, use after open, claim }

checkHealth (level: *int32*):
void { raises exception, use after open, claim, enable }

clearInput ():
void { } *Not supported*

clearOutput ():
void { raises exception, use after open, claim }

directIO (command: *int32*, inout data: *int32*, inout obj: *object*):
void { raises exception, use after open }

Specific

Name

accessDailyLog (sequenceNumber: *int32*, type: *int32*, timeout: *int32*):
void { raises exception, use after open, claim, enable }

authorizeCompletion (sequenceNumber: *int32*, amount: *currency*,
taxOthers: *currency*, timeout: *int32*):
void { raises exception, use after open, claim, enable }

authorizePreSales (sequenceNumber: *int32*, amount: *currency*, taxOthers:
currency, timeout: *int32*):
void { raises exception, use after open, claim, enable }

authorizeRefund (sequenceNumber: *int32*, amount: *currency*, taxOthers:
currency, timeout: *int32*):
void { raises exception, use after open, claim, enable }

authorizeSales (sequenceNumber: *int32*, amount: *currency*, taxOthers:
currency, timeout: *int32*):
void { raises exception, use after open, claim, enable }

authorizeVoid (sequenceNumber: *int32*, amount: *currency*, taxOthers:
currency, timeout: *int32*):
void { raises exception, use after open, claim, enable }

```

authorizeVoidPreSales ( sequenceNumber: int32, amount: currency,  

    taxOthers: currency, timeout: int32 ):  

    void { raises exception, use after open, claim, enable }

checkCard ( sequenceNumber: int32, timeout: int32 ):  

    void { raises exception, use after open, claim, enable }

```

Events (UML interfaces)

<i>Name</i>	<i>Type</i>	<i>Mutability</i>
upos::events::DirectIOEvent		
EventNumber:	<i>int32</i>	{ read-only }
Data:	<i>int32</i>	{ read-write }
Obj:	<i>object</i>	{ read-write }
upos::events::ErrorEvent		
ErrorCode:	<i>int32</i>	{ read-only }
ErrorCodeExtended:	<i>int32</i>	{ read-only }
ErrorLocus:	<i>int32</i>	{ read-only }
ErrorResponse	<i>int32</i>	{ read-write }
upos::events::OutputCompleteEvent		
OutputID:	<i>int32</i>	{ read-only }
upos::events::StatusUpdateEvent		
Status:	<i>int32</i>	{ read-only }

General Information

The CAT programmatic name is “CAT”.

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.

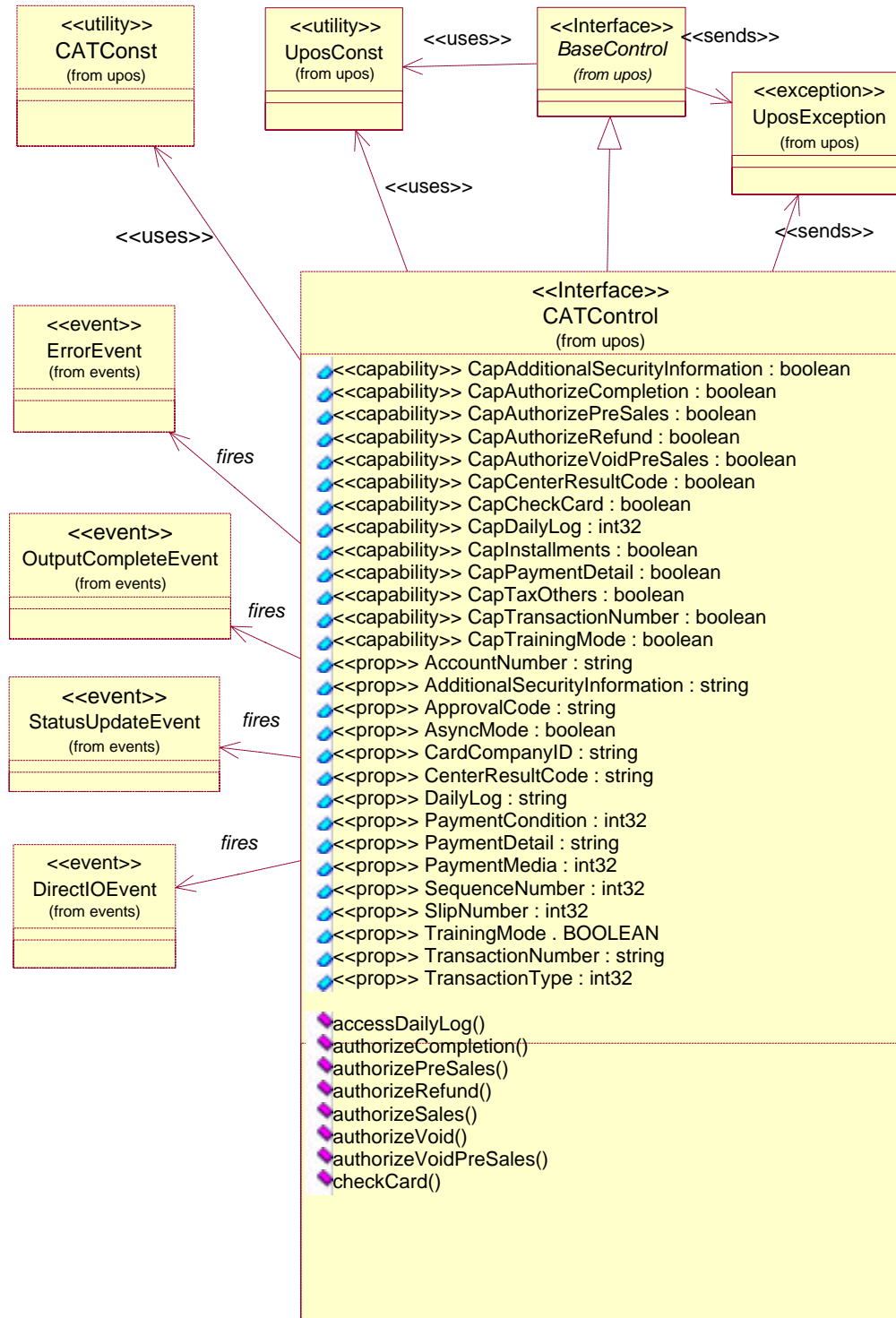
- **Card Check**
The transaction to perform a negative card file validation of the card presented by the customer. Typically negative card files contain card numbers that are known to fail approval. Therefore the Card Check operation removes then need for communication to the approval agency in some instances.
- **Daily log**
The daily log of card transactions that have been approved by the card companies.
- **Payment condition**
Condition of payment such as lump-sum payment, payment by bonus, payment in installments, revolving payment, and the combination of those payments. 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 hardware implementation. Therefore, the hardware implementation of a CAT device may be as follows:
 - **Separate type (POS interlock)**
The dedicated CAT device is externally connected to the POS (for instance, via an RS-232 connection).
 - **Built-in type**
The hardware structure is the same as the separate type but is installed within the POS housing.
- The CAT device receives each authorization request containing a purchase amount and tax from CAT control.
- The CAT device generally requests the user to swipe a magnetic card when it receives an authorization request from CAT control.
- Once a magnetic card is swiped at the CAT device, the device sends the purchase amount and tax to the approval agency using the communications device.
- The CAT device returns the result from the approval agency to the CAT control. The returned data will be stored in the authorization properties by the CAT control for access by applications.

CAT Class Diagram



Model

The general models for the CAT control are shown below:

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

Function	Method name	Corresponding Cap property
Purchase	authorizeSales	None
Cancel Purchase	authorizeVoid	CapAuthorizeVoid
Refund Purchase	authorizeRefund	CapAuthorizeRefund
Authorization Completion	authorizeCompletion	CapAuthorizeCompletion
Pre-Authorization	authorizePreSales	CapAuthorizePreSales
Cancel Pre-Authorization	authorizeVoidPreSales	CapAuthorizeVoidPreSales

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

Function	Method name	Corresponding Cap property
Card Check	checkCard	CapCheckCard
Daily log	accessDailyLog	CapDailyLog

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

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

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

Description	Property Name	Corresponding Cap Property
Daily log	DailyLog	CapDailyLog

- Sequence numbers are used to validate that the properties set at completion of a method are indeed associated with the completed method. An incoming **SequenceNumber** argument for each method is compared with the resulting **SequenceNumber** property after the operation associated with the method has completed. If the numbers do not match, or if an application fails to identify the number, there is no guarantee that the values of the properties listed in the two tables correspond to the completed method.
- The **AsyncMode** property determines if methods are run synchronously or asynchronously.
- When **AsyncMode** is false, methods will be executed synchronously and their corresponding properties will contain data when the method returns.
- When **AsyncMode** is true, methods will return immediately to the application. When the operation associated with the method completes, each corresponding property will be updated by the CAT control prior to an **OutputCompleteEvent**. When **AsyncMode** is true, methods cannot be issued immediately after issuing a prior method; only one outstanding asynchronous method is allowed at a time. However, **clearOutput** is an exception because its purpose is to cancel an outstanding asynchronous method.

The methods supported and their corresponding properties vary depending on the CAT control implementation. Applications should verify that particular **Cap** properties are supported before utilizing the capability dependent methods and properties.

- Results of synchronous calls to methods and writable properties will be stored in **ErrorCode**. Results of asynchronous processing will be indicated by an **OutputCompleteEvent** or returned in the *ErrorCode* argument of an **ErrorEvent**. If **ErrorCode** or the *ErrorCode* argument is E_EXTENDED, detailed device specific information may be stored to **ErrorCodeExtended** in synchronous mode and stored to **ErrorEvent** argument *ErrorCodeExtended* in asynchronous mode. The result code from the approval agency will be stored in **CenterResultCode** in either mode.
- Training mode occurs continually when **TrainingMode** is true. To discontinue training mode, set **TrainingMode** to false.
- An outstanding asynchronous method can be canceled via the **clearOutput** method.
- The Daily log can be collected by the **accessDailyLog** method. Collection will be run either synchronously or asynchronously according to the value of **AsyncMode**.

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

Synchronous Mode:

- **open**
- **claim**
- **setDeviceEnabled** (true)
- Definition of the argument *SequenceNumber*
- Set **PaymentMedia** **Added in Version 1.5**
- **authorizeSales()**
- Check *UpoxException* 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 *UpoxException* 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**
- **close**

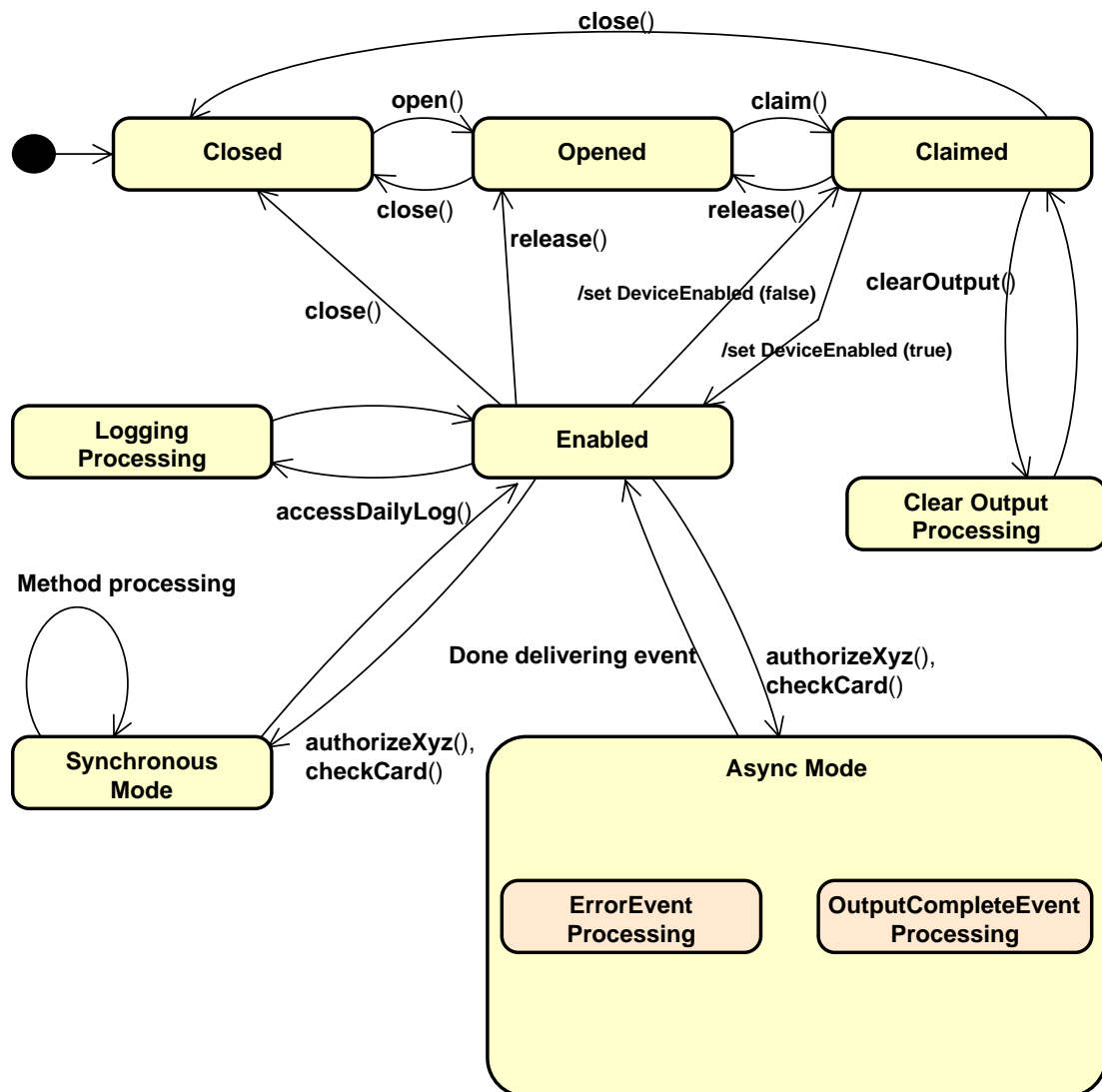
Device Sharing

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.

CAT State Diagram

The following diagram depicts the CAT states.



Properties (UML attributes)

AccountNumber Property

Syntax	AccountNumber: <i>string</i> { read-only , access after open }
Remarks	This property is initialized to NULL by the open method and is updated when an authorization operation successfully completes.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

AdditionalSecurityInformation Property

Syntax	AdditionalSecurityInformation: <i>string</i> { read-write , access after open }
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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	CapAdditionalSecurityInformation Property.

ApprovalCode Property

Syntax	ApprovalCode: <i>string</i> { read-only , access after open }
Remarks	This property is initialized to NULL by the open method and is updated when an authorization operation successfully completes.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

AsyncMode Property

Syntax	AsyncMode: <i>boolean</i> { read-write , access after open }
Remarks	If true, the authorization methods will run asynchronously. If false, the authorization methods will run synchronously. This property is initialized to false by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	Authorization Methods.

CapAdditionalSecurityInformation Property

Syntax	CapAdditionalSecurityInformation: <i>boolean</i> { read-only, access after open }
Remarks	If true, the AdditionalSecurityInformation property may be utilized; otherwise it is false. This property is initialized by open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	AdditionalSecurityInformation Property.

CapAuthorizeCompletion Property

Syntax	CapAuthorizeCompletion: <i>boolean</i> { read-only, access after open }
Remarks	If true, the authorizeCompletion method has been implemented; otherwise it is false. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	authorizeCompletion Method.

CapAuthorizePreSales Property

Syntax	CapAuthorizePreSales: <i>boolean</i> { read-only, access after open }
Remarks	If true, the authorizePreSales method has been implemented; otherwise it is false. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	authorizePreSales Method.

CapAuthorizeRefund Property

Syntax	CapAuthorizeRefund: <i>boolean</i> { read-only, access after open }
Remarks	If true, the authorizeRefund method has been implemented; otherwise it is false. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	authorizeRefund Method.

CapAuthorizeVoid Property

Syntax	CapAuthorizeVoid: <i>boolean</i> { read-only, access after open }
Remarks	If true, the authorizeVoid method has been implemented; otherwise it is false. This property is initialized by the open method.
Errors	A UpoException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	authorizeVoid Method.

CapAuthorizeVoidPreSales Property

Syntax	CapAuthorizeVoidPreSales: <i>boolean</i> { read-only, access after open }
Remarks	If true, the authorizeVoidPreSales method has been implemented; otherwise it is false. This property is initialized by the open method.
Errors	A UpoException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	authorizeVoidPreSales Method.

CapCenterResultCode Property

Syntax	CapCenterResultCode: <i>boolean</i> { read-only, access after open }
Remarks	If true, the CenterResultCode property has been implemented; otherwise it is false. This property is initialized by the open method.
Errors	A UpoException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	CenterResultCode Property.

CapCheckCard Property

Syntax	CapCheckCard: <i>boolean</i> { read-only, access after open }
Remarks	If true, the checkCard method has been implemented; otherwise it is false. This property is initialized by the open method.
Errors	A UpoException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	checkCard Method.

CapDailyLog Property

Syntax **CapDailyLog:** *int32* { **read-only**, **access after open** }

Remarks Shows the daily log ability of the device.

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

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

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

See Also **DailyLog** Property, **accessDailyLog** Method.

CapInstallments Property

Syntax **CapInstallments:** *boolean* { **read-only**, **access after open** }

Remarks If true, the item “Installments” which is stored in the **DailyLog** property as the result of **accessDailyLog** will be provided; otherwise it is false.

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

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

See Also **DailyLog** Property.

CapPaymentDetail Property

Syntax **CapPaymentDetail:** *boolean* { **read-only**, **access after open** }

Remarks If true, the **PaymentDetail** property has been implemented; otherwise it is false.

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

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

See Also **PaymentDetail** Property.

CapTaxOthers Property

Syntax	CapTaxOthers: <i>boolean</i> { read-only, access after open }
Remarks	<p>If true, the item “TaxOthers” which is stored in the DailyLog property as the result of access DailyLog will be provided; otherwise it is false.</p> <p>Note that this property is not related to the “TaxOthers” argument used with the authorization methods.</p> <p>This property is initialized by the open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	DailyLog Property.

CapTransactionNumber Property

Syntax	CapTransactionNumber: <i>boolean</i> { read-only, access after open }
Remarks	<p>If true, the TransactionNumber property has been implemented; otherwise it is false.</p> <p>This property is initialized by the open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	TransactionNumber Property.

CapTrainingMode Property

Syntax	CapTrainingMode: <i>boolean</i> { read-only, access after open }
Remarks	<p>If true, the TrainingMode property has been implemented; otherwise it is false.</p> <p>This property is initialized by the open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	TrainingMode Property.

CardCompanyID Property

Syntax	CardCompanyID: <i>string</i> { read-only, access after open }
Remarks	<p>This property is updated when an authorization operation successfully completes. It shows credit card company ID.</p> <p>The length of the ID string varies depending upon the CAT device.</p> <p>This property is initialized to NULL by the open method</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CenterResultCode Property

- Syntax** **CenterResultCode:** *string* { read-only, access after open }
- Remarks** Contains the code from the approval agency. Check the approval agency for the actual codes to be stored.
- This property is initialized to NULL by the **open** method and is updated when an authorization operation successfully completes
- Errors** A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

DailyLog Property

- Syntax** **DailyLog:** *string* { read-only, access after open }
- Remarks** Stores the result of the **accessDailyLog** method. The data is delimited by CR(13)+LF(10) for each transaction and is stored in ASCII code. The detailed data of each transaction is comma separated [i.e. delimited by “,” (44)].

The details of one transaction are shown as follows:

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

Notes from the previous table:

1) Format

Item	Format
Transaction date	YYYYMMDDHHMMSS
Purchase date	MMDD

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

2) Additional data

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

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

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

4) Amount

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

5) Purchase date

The date manually entered for the purchase transaction after approval.

Example An example of daily log content is shown below.

Item	Description	Meaning
Card company ID	102	JCB
Transaction type	CAT_TRANSACTION_SALES	Purchase
Transaction date	19980116134530	1/16/199813:45:30
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 `UpoException` may be thrown when this property is accessed. For further information, see “Errors” on page 15.

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

PaymentCondition Property

Syntax **PaymentCondition: *int32* { read-only, access after open }**

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

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

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

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

See Also **PaymentDetail** Property.

PaymentDetail Property

Syntax **PaymentDetail: *string* { read-only, access after open }**

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

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

*Maximum 6 installments

The payment types and names vary depending on the CAT device. The following are the payment types and terms available for CAT devices. Note that there are some differences between UnifiedPOS 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
			Unified OPOS Term	Card Company Terms				
Lump-sum	(None)	10	Lump-sum	Lump-sum	Lump-sum	Lump-sum	Lump-sum	Lump-sum
Bonus	(None)	21	Bonus 1	Bonus 1	Bonus 1	Bonus 1	Bonus 1	Bonus 1
	Number of bonus payments	22	Bonus 2	Bonus 2	Bonus 2	Bonus 2	Bonus 2	Bonus 2
	Bonus month(s)	23	Bonus 3	Bonus 3	Does not exist.	Does not exist.	Bonus 3	Bonus 3
	Number of bonus payments	24	Bonus 4	Bonus 4	Bonus 3	Bonus 3	Bonus 4 (Up to two entries for bonus month)	Bonus 4
	Bonus month (1)							
	Bonus month (2)							
	Bonus month (3)							
	Bonus month (4)							
	Bonus month (5)							
	Bonus month (6)							

	Number of bonus payments	25	Bonus 5	Bonus 5	Does not exist.	Does not exist.	Does not exist.	Bonus 5
	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)							
Installment	Payment start month	61	Installment 1	Installment 1	Installment 1	Installment 1	Installment 1	Installment 1
	Number of payments							

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

Payment start month	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
Number of payments							
Number of bonus payments							
Bonus month (1)							
Bonus month (2)							
Bonus month (3)							
Bonus month (4)							
Bonus month (5)							
Bonus month (6)							

	Payment start month	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
	Number of payments							
	Number of bonus payments							
	Bonus month (1)							
	Bonus amount(1)							
	Bonus month (2)							
	Bonus amount(2)							
	Bonus month (3)							
	Bonus amount(3)							
	Bonus month (4)							
	Bonus amount(4)							
	Bonus month (5)							
	Bonus amount(5)							
	Bonus month (6)							
	Bonus amount(6)							
Revolving	(None)	80	Revolving	Revolving	Revolving	Revolving	Revolving	Revolving
Debit	(None)	110	Debit	(Support depends on the actual device)	(Support depends on the actual device)	(Support depends on the actual device)	(Support depends on the actual device)	(Support depends on the actual device)

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

See Also **CapPaymentDetail** Property.

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

Syntax	PaymentMedia: <i>int32</i> { read-write, access after open }								
Remarks	<p>Holds the payment media type that the approval method should approve.</p> <p>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.</p> <table> <tr> <th>Value</th><th>Meaning</th></tr> <tr> <td>CAT_MEDIA_UNSPECIFIED</td><td>None specified.</td></tr> <tr> <td>CAT_MEDIA_CREDIT</td><td>Credit card.</td></tr> <tr> <td>CAT_MEDIA_DEBIT</td><td>Debit card.</td></tr> </table> <p>This property is initialized to CAT_MEDIA_UNSPECIFIED by the open method.</p>	Value	Meaning	CAT_MEDIA_UNSPECIFIED	None specified.	CAT_MEDIA_CREDIT	Credit card.	CAT_MEDIA_DEBIT	Debit card.
Value	Meaning								
CAT_MEDIA_UNSPECIFIED	None specified.								
CAT_MEDIA_CREDIT	Credit card.								
CAT_MEDIA_DEBIT	Debit card.								
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.								

SequenceNumber Property

Syntax	SequenceNumber: <i>int32</i> { read-only, access after open }
Remarks	<p>Stores a “sequence number” as the result of each method call. This number needs to be checked by an application to see if it matches with the argument <i>sequenceNumber</i> of the originating method.</p> <p>If the “sequence number” returned from the CAT device is not numeric, the CAT control set this property to zero.</p> <p>This property is initialized to zero by the open method and is updated when an authorization operation successfully completes.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

SlipNumber Property

Syntax	SlipNumber: <i>int32</i> { read-only, access after open }
Remarks	<p>Stores a “slip number” as the result of each authorization operation.</p> <p>This property is initialized to NULL by the open method and is updated when an authorization operation successfully completes.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

TrainingMode Property

Syntax	TrainingMode: <i>boolean</i> { read-write, access after open }				
Remarks	<p>If true, each operation will be run in training mode; otherwise each operation will be run in normal mode.</p> <p>TrainingMode needs to be explicitly set to false by an application to exit from training mode, because it will not automatically be set to false after the completion of an operation.</p> <p>This property will be initialized to false by the open method.</p>				
Errors	<p>A UposException 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> <tr> <th>Value</th><th>Meaning</th></tr> <tr> <td>E_ILLEGAL</td><td>CapTrainingMode is false.</td></tr> </table>	Value	Meaning	E_ILLEGAL	CapTrainingMode is false.
Value	Meaning				
E_ILLEGAL	CapTrainingMode is false.				

TransactionNumber Property

Syntax	TransactionNumber: <i>string</i> { read-only, access after open }
Remarks	<p>Stores a “transaction number” as the result of each authorization operation.</p> <p>This property is initialized to NULL by the open method and is updated when an authorization operation successfully completes.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

TransactionType Property

Syntax	TransactionType: <i>int32</i> { read-only, access after open }														
Remarks	<p>Stores a “transaction type” as the result of each authorization operation.</p> <p>This property is initialized to zero by the open method and is updated when an authorization operation successfully completes.</p> <p>This property will be set to one of the following values.</p> <table> <tr> <th>Value</th><th>Meaning</th></tr> <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_VOIDPRESALES</td><td>Cancel pre-authorization approval</td></tr> </table>	Value	Meaning	CAT_TRANSACTION_SALES	Sales	CAT_TRANSACTION_VOID	Cancellation	CAT_TRANSACTION_REFUND	Refund purchase	CAT_TRANSACTION_COMPLETION	Purchase after approval	CAT_TRANSACTION_PRESALES	Pre-authorization	CAT_TRANSACTION_VOIDPRESALES	Cancel pre-authorization approval
Value	Meaning														
CAT_TRANSACTION_SALES	Sales														
CAT_TRANSACTION_VOID	Cancellation														
CAT_TRANSACTION_REFUND	Refund purchase														
CAT_TRANSACTION_COMPLETION	Purchase after approval														
CAT_TRANSACTION_PRESALES	Pre-authorization														
CAT_TRANSACTION_VOIDPRESALES	Cancel pre-authorization approval														
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.														

Methods (UML operations)

accessDailyLog Method

Syntax `accessDailyLog (sequenceNumber: int32, type: int32, timeout: int32):
void { raises-exception, use after open-claim-enable }`

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. 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 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 UposException 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
E_ILLEGAL	Invalid or unsupported <i>type</i> or <i>timeout</i> parameter was specified, or CapDailyLog is false.
E_TIMEOUT	No response was received from CAT during the specified <i>timeout</i> time in milliseconds.
E_EXTENDED	The detail code has been stored in <i>ErrorCodeExtended</i> .
E_BUSY	The CAT device cannot accept any commands now.

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

authorizePreSales Method

Syntax **authorizePreSales** (*sequenceNumber*: *int32*, *amount*: *currency*, *taxOthers*: *currency*, *timeout*: *int32*):
 void { raises-exception, use after open-claim-enable }

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. FOREVER (-1), 0 and positive values can be specified.

Remarks Makes a pre-authorization.

Pre-authorization for *amount* and *taxOthers* is made as the approval specified by *sequenceNumber*.

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

Errors A UposException 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
E_ILLEGAL	Invalid <i>timeout</i> parameter was specified, or CapAuthorizePreSales is false.
E_TIMEOUT	No response was received from CAT during the specified <i>timeout</i> time in milliseconds.
E_EXTENDED	The detail code has been stored in <i>ErrorCodeExtended</i> .
E_BUSY	The CAT device cannot accept any commands now.

See Also **CapAuthorizePreSales** Property.

authorizeRefund Method

Syntax **authorizeRefund** (*sequenceNumber*: *int32*, *amount*: *currency*, *taxOthers*: *currency*, *timeout*: *int32*):
 void { raises-exception, use after open-claim-enable }

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. FOREVER (-1), 0 and positive values can be specified.

Remarks Refund purchase approval is intended.

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

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

Errors A UposException 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
E_ILLEGAL	Invalid <i>timeout</i> parameter was specified, or CapAuthorizeRefund is false.
E_TIMEOUT	No response was received from CAT during the specified <i>timeout</i> time in milliseconds.
E_EXTENDED	The detail code has been stored in <i>ErrorCodeExtended</i> .
E_BUSY	The CAT device cannot accept any commands now.

See Also **CapAuthorizeRefund** Property.

authorizeSales Method

Syntax **authorizeSales** (*sequenceNumber*: *int32*, *amount*: *currency*, *taxOthers*: *currency*, *timeout*: *int32*):
 void { raises-exception, use after open-claim-enable }

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. FOREVER (-1), 0 and positive values can be specified.

Remarks Normal purchase approval is intended.

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

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

Errors A UposException 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
E_ILLEGAL	Invalid <i>timeout</i> parameter was specified.
E_TIMEOUT	No response was received from CAT during the specified <i>timeout</i> time in milliseconds.
E_EXTENDED	The detail code has been stored in <i>ErrorCodeExtended</i> .
E_BUSY	The CAT device cannot accept any commands now.

authorizeVoid Method

Syntax **authorizeVoid** (*sequenceNumber*: *int32*, *amount*: *currency*, *taxOthers*: *currency*, *timeout*: *int32*):
 void { **raises-exception**, **use after open-claim-enable** }

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. FOREVER (-1), 0 and positive values can be specified.

Remarks Purchase cancellation approval is intended.

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

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

Errors A *UposException* 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
E_ILLEGAL	Invalid <i>timeout</i> parameter was specified, or CapAuthorizeVoid is false.
E_TIMEOUT	No response was received from CAT during the specified <i>timeout</i> time in milliseconds.
E_EXTENDED	The detail code has been stored in <i>ErrorCodeExtended</i> .
E_BUSY	The CAT device cannot accept any commands now.

See Also **CapAuthorizeVoid** Property.

authorizeVoidPreSales Method

Syntax	authorizeVoidPreSales (sequenceNumber: <i>int32</i>, amount: <i>currency</i>, taxOthers: <i>currency</i>, timeout: <i>int32</i>): void { raises-exception, use after open-claim-enable }
--------	--

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. FOREVER (-1), 0 and positive values can be specified.

Remarks	<p>Pre-authorization cancellation approval is intended.</p> <p>Pre-authorization cancellation approval for <i>amount</i> and <i>taxOthers</i> is intended as the approval specified by <i>sequenceNumber</i>.</p> <p>When <i>timeout</i> is FOREVER (-1), timeout never occurs and the device waits until it receives response from the CAT.</p> <p>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.</p>
---------	---

Errors	A <code>UposException</code> 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
E_ILLEGAL	Invalid <i>timeout</i> parameter was specified, or CapAuthorizeVoidPreSales is false.
E_TIMEOUT	No response was received from CAT during the specified <i>timeout</i> time in milliseconds.
E_EXTENDED	The detail code has been stored in <i>ErrorCodeExtended</i> .
E_BUSY	The CAT device cannot accept any commands now.

See Also **CapAuthorizeVoidPreSales** Property.

checkCard Method

Syntax **checkCard (sequenceNumber: *int32*, timeout: *int32*):**
 void { raises-exception, use after open-claim-enable }

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. FOREVER (-1), 0 and positive values can be specified.

Remarks Card Check is intended.

Card Check will be made as specified by SequenceNumber.

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

Errors A UposException 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
E_ILLEGAL	Invalid <i>timeout</i> parameter was specified, or CapCheckCard is false.
E_TIMEOUT	No response was received from CAT during the specified <i>timeout</i> time in milliseconds.
E_EXTENDED	The detail code has been stored in <i>ErrorCodeExtended</i> .
E_BUSY	The CAT device cannot accept any commands now.

See Also **CapCheckCard** Property.

Events (UML interfaces)

DirectIOEvent

<<event>> **upos::events::DirectIOEvent**
 EventNumber: *int32* { read-only }
 Data: *int32* { read-write }
 Obj: *object* { read-write }

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

Attributes This event contains the following attributes:

Attribute	Type	Description
<i>EventNumber</i>	<i>int32</i>	Event number whose specific values are assigned by the Service.
<i>Data</i>	<i>int32</i>	Additional numeric data. Specific values vary by the <i>EventNumber</i> and the Service. This attribute is settable.
<i>Obj</i>	<i>object</i>	Additional data whose usage varies by the <i>EventNumber</i> and the Service. This attribute 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 CAT devices which may not have any knowledge of the Service's need for this event.

See Also "Events" on page 14, **directIO** Method

ErrorEvent

```
<< event >> upos::events::ErrorEvent
    ErrorCode: int32 { read-only }
    ErrorCodeExtended: int32 { read-only }
    ErrorLocus: int32 { read-only }
    ErrorResponse: int32 { read-only }
```

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

Attributes This event contains the following properties:

Attributes	Type	Description
<i>ErrorCode</i>	<i>int32</i>	The code which caused the error event. Remarks <i>ErrorCode</i> for the value below for the value.
<i>ErrorCodeExtended</i>	<i>int32</i>	The extended code which caused the error event. Remarks the value below for the value.
<i>ErrorLocus</i>	<i>int32</i>	EL_OUTPUT is specified. An error occurred during asynchronous action.
<i>ErrorResponse</i>	<i>int32</i>	Pointer to the error event response. See values below.
If <i>ErrorCode</i> is E_EXTENDED, <i>ErrorCodeExtended</i> will be set to one of the following values:		

Value	Meaning
ECAT_CENTERERROR	An error was returned from the approval agency. The detail error code is defined in CenterResultCode .
ECAT_COMMANDERROR	The command sent to CAT is wrong. This error is never returned so long as CAT control is working correctly.
ECAT_RESET	CAT was stopped during processing by CAT reset key (stop key) and so on.
ECAT_COMMUNICATIONERROR	Communication error has occurred between the approval agency and CAT.
ECAT_DAILYLOGOVERFLOW	Daily log was too big to be stored. Keeping daily log has been stopped and the value of DailyLog property is uncertain.

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

Value	Meaning
ER_RETRY	Retries the asynchronous processing. The error state is exited.
ER_CLEAR	Clear the asynchronous processing. The error state is exited.

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

See Also “Device Output Models” on page 20, Device States on page 25.

OutputCompleteEvent

<<event>> **upos::events::OutputCompleteEvent**
OutputID: int32 { read-only }

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

Attribute This event contains the following attribute:

Attribute	Type	Description
<i>OutputID</i>	<i>int32</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 Service has confirmation that it was processed by the device successfully.

See Also “Device Output Models” on page 20.

StatusUpdateEvent

<< event >> **upos::events::StatusUpdateEvent**
Status: *int32* { read-only }

Description Notifies the application that there is a change in the power status of the CAT device.

Attributes This event contains the following attribute:

Attributes	Type	Description
<i>Status</i>	<i>int32</i>	Indicates a change in the power status of the unit.

Note that Release 1.3 added Power State Reporting with additional *Power reporting* **StatusUpdateEvent** values. See “StatusUpdateEvent” description on page 56.

Remarks Enqueued when the CAT device detects a power state change.

See Also “Events” on page 14.

CHAPTER 6

Coin Dispenser

This Chapter defines the Coin Dispenser device category.

General Information

The Coin Dispenser programmatic name is “CoinDispenser”.

This chapter is grandfathered in based on the OPOS and JavaPOS Version 1.4 specifications.

CHAPTER 7

Fiscal Printer

This Chapter defines the Fiscal Printer device category.

General Information

The Fiscal Printer programmatic name is “FiscalPrinter”.

This chapter is grandfathered in based on the OPOS and JavaPOS Version 1.4 specifications.

This device had minor revisions for Version 1.5, and only the changes are included in this specification.

Properties (UML attributes)

CountryCode Property

Updated in Release 1.5

Syntax **CountryCode:** *int32* { **read-only**, **access after open** }

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

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

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

Errors A UposException may be thrown when this property is accessed. For further information, see "Errors" on page 15.

CHAPTER 8

Hard Totals

This Chapter defines the Hard Totals device category.

General Information

The Hard Totals programmatic name is “HardTotals”.

This chapter is grandfathered in based on the OPOS and JavaPOS Version 1.4 specifications.

CHAPTER 9

Keylock

This Chapter defines the Keylock device category.

General Information

The Keylock programmatic name is “Keylock”.

This chapter is grandfathered in based on the OPOS and JavaPOS Version 1.4 specifications.

CHAPTER 10

Line Display

This Chapter defines the Line Display device category.

General Information

The Line Display programmatic name is “LineDisplay”.

This chapter is grandfathered in based on the OPOS and JavaPOS Version 1.4 specifications.

This device had minor revisions for Version 1.5, and only the changes are included in this specification.

Properties (UML attributes)

CapCharacterSet Property

Updated in Release 1.5

Syntax	CapCharacterSet: <i>int32</i> { read-only, access after open }												
Remarks	<p>Holds the default character set capability. It has one of the following values:</p> <table> <tr> <th>Value</th><th>Meaning</th></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> </table> <p>The default character set may contain a superset of these ranges. The initial CharacterSet property may be examined for additional information.</p> <p>This property is initialized by the open method.</p>	Value	Meaning	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.
Value	Meaning												
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.												
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.												
See Also	CharacterSet Property.												

CharacterSet Property***Updated in Release 1.5*****Syntax** **CharacterSet: *int32* { read-write, access after open-claim-enable }****Remarks** Holds the character set for displaying characters. It has one of the following values:

Value	Meaning
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.
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.

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

Errors A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.**See Also** **CharacterSetList** Property.

CHAPTER 11

MICR - Magnetic Ink Character Recognition Reader

This Chapter defines the MICR - Magnetic Ink Character Recognition Reader device category.

General Information

The MICR - Magnetic Ink Character Recognition Reader programmatic name is “MICR”.

This chapter is grandfathered in based on the OPOS and JavaPOS Version 1.4 specifications.

MSR - Magnetic Stripe Reader

This Chapter defines the Magnetic Stripe Reader device category.

Summary

Properties (UML attributes)

<i>Common</i>	<i>Type</i>	<i>Mutability</i>	<i>Version^a</i>	<i>May Use After</i>
AutoDisable:	<i>boolean</i>	{ read-write }	1.3	open
CapPowerReporting:	<i>int32</i>	{ read-only }	1.3	open
CheckHealthText:	<i>string</i>	{ read-only }	1.3	open
Claimed:	<i>boolean</i>	{ read-only }	1.3	open
DataCount:	<i>int32</i>	{ read-only }	1.3	open
DataEventEnabled:	<i>boolean</i>	{ read-write }	1.3	open
DeviceEnabled:	<i>boolean</i>	{ read-write }	1.3	open & claim
FreezeEvents:	<i>boolean</i>	{ read-write }	1.3	open
OutputID:	<i>int32</i>	{ read-only }	1.3	Not Supported
PowerNotify:	<i>int32</i>	{ read-write }	1.3	open
PowerState:	<i>int32</i>	{ read-only }	1.3	open
State:	<i>int32</i>	{ read-only }	1.3	--
DeviceControlDescription:	<i>string</i>	{ read-only }	1.3	--
DeviceControlVersion:	<i>int32</i>	{ read-only }	1.3	--
DeviceServiceDescription:	<i>string</i>	{ read-only }	1.3	open
DeviceServiceVersion:	<i>int32</i>	{ read-only }	1.3	open
PhysicalDeviceDescription:	<i>string</i>	{ read-only }	1.3	open
PhysicalDeviceName:	<i>string</i>	{ read-only }	1.3	open

- a. The version representation provides the mechanism for recognizing when a change occurs to a property, method or event. This MSR definition was introduced in an existing standard and was not changed for the UnifiedPOS version 1.4.

Properties (Continued)

<i>Specific</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>	<i>May Use After</i>
CapISO:	<i>boolean</i>	{ read-only }	1.3	open
CapJISOne:	<i>boolean</i>	{ read-only }	1.3	open
CapJISTwo:	<i>boolean</i>	{ read-only }	1.3	open
CapTransmitSentinels:	<i>boolean</i>	{ read-only }	1.5	open
AccountNumber:	<i>string</i>	{ read-only }	1.3	open
DecodeData:	<i>boolean</i>	{ read-write }	1.3	open
ErrorReportingType:	<i>int32</i>	{ read-write }	1.3	open
ExpirationDate:	<i>string</i>	{ read-only }	1.3	open
FirstName:	<i>string</i>	{ read-only }	1.3	open
MiddleInitial:	<i>string</i>	{ read-only }	1.3	open
ParseDecodeData:	<i>boolean</i>	{ read-write }	1.3	open
ServiceCode:	<i>string</i>	{ read-only }	1.3	open
Suffix:	<i>string</i>	{ read-only }	1.3	open
Surname:	<i>string</i>	{ read-only }	1.3	open
Title:	<i>string</i>	{ read-only }	1.3	open
Track1Data:	<i>binary</i>	{ read-only }	1.3	open
Track1DiscretionaryData:	<i>binary</i>	{ read-only }	1.3	open
Track2Data:	<i>binary</i>	{ read-only }	1.3	open
Track2DiscretionaryData:	<i>binary</i>	{ read-only }	1.3	open
Track3Data:	<i>binary</i>	{ read-only }	1.3	open
Track4Data:	<i>binary</i>	{ read-only }	1.5	open
TracksToRead:	<i>int32</i>	{ read-write }	1.3	open
TransmitSentinels:	<i>boolean</i>	{ read-write }	1.5	open

Methods (UML operations)

Common

Name

open (logicalDeviceName: *string*):
 void { raises exception }

close ():
 void { raises exception, use after open }

claim (timeout: *int32*):
 void { raises exception, use after open }

release ():
 void { raises exception, use after open, claim }

checkHealth (level: *int32*):
 void { raises exception, use after open, claim, enable }

clearInput ():
 void { raises exception, use after open, claim }

clearOutput (): **Not supported**
 void { }

directIO (command: *int32*, inout data: *int32*, inout obj: *object*):
 void { raises exception, use after open, claim }

Specific

None

Events (UML interfaces)

<i>Name</i>	<i>Type</i>	<i>Mutability</i>
upos::events::DataEvent		
Status:	<i>int32</i>	{ read-only }
upos::events::DirectIOEvent		
EventNumber:	<i>int32</i>	{ read-only }
Data:	<i>int32</i>	{ read-write }
Obj:	<i>object</i>	{ read-write }
upos::events::ErrorEvent		
ErrorCode:	<i>int32</i>	{ read-only }
ErrorCodeExtended:	<i>int32</i>	{ read-only }
ErrorLocus:	<i>int32</i>	{ read-only }
ErrorResponse:	<i>int32</i>	{ read-write }
upos::events::StatusUpdateEvent		
Status:	<i>int32</i>	{ read-only }

General Information

The Magnetic Stripe Reader programmatic name is “MSR”.

Capabilities

The MSR device class supports attachment of a card reader to provide input to the application from a card inserted (swiped) through the reader. The targeted environment is electronic funds data such as an account number, customer name, etc. from a magnetically encoded credit and/or debit card.

There are no specific methods for this device category.

The MSR Control has the following minimal set of capabilities:

- Reads encoded data from a magnetic stripe. Data is obtainable from any combination of ISO or JIS-I tracks 1, 2, 3, and JIS-II.
- 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:

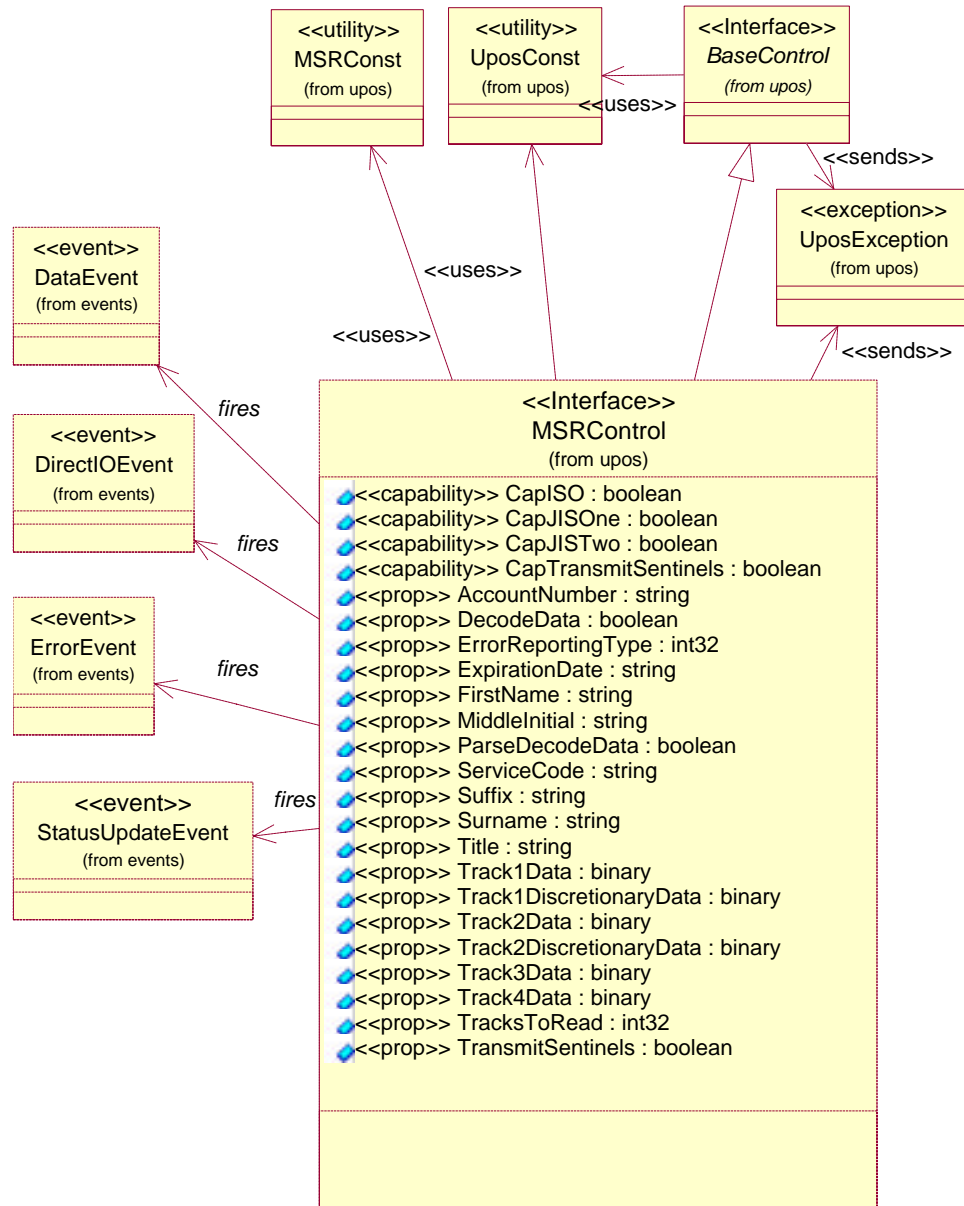
- Support for specific card types: ISO, JIS Type I and/or JIS Type II. Note: for the purpose of this standard, the following convention is assumed:
 - Track 1 is ISO or JIS-I Track 1
 - Track 2 is ISO or JIS-I Track 2
 - Track 3 is ISO or JIS-I Track 3
 - Track 4 is JIS-II data
 - Determination of the 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.

MSR Class Diagram

The following diagram shows the relationships between the MSR classes.



Device Behavior Model

The general device behavior model of the MSR is:

- Four unique writable properties control MSR data handling:
 - The **TracksToRead** property controls which combination of the 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 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.

Input – MSR

The MSR follows the general “Device Input Model” for event-driven input:

- When input is received from the card reader generated by the card swipe, a **DataEvent** is enqueued.
- If the **AutoDisable** property is true, the device will 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 re-enables 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 **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 the **clearInput** method description for more details.

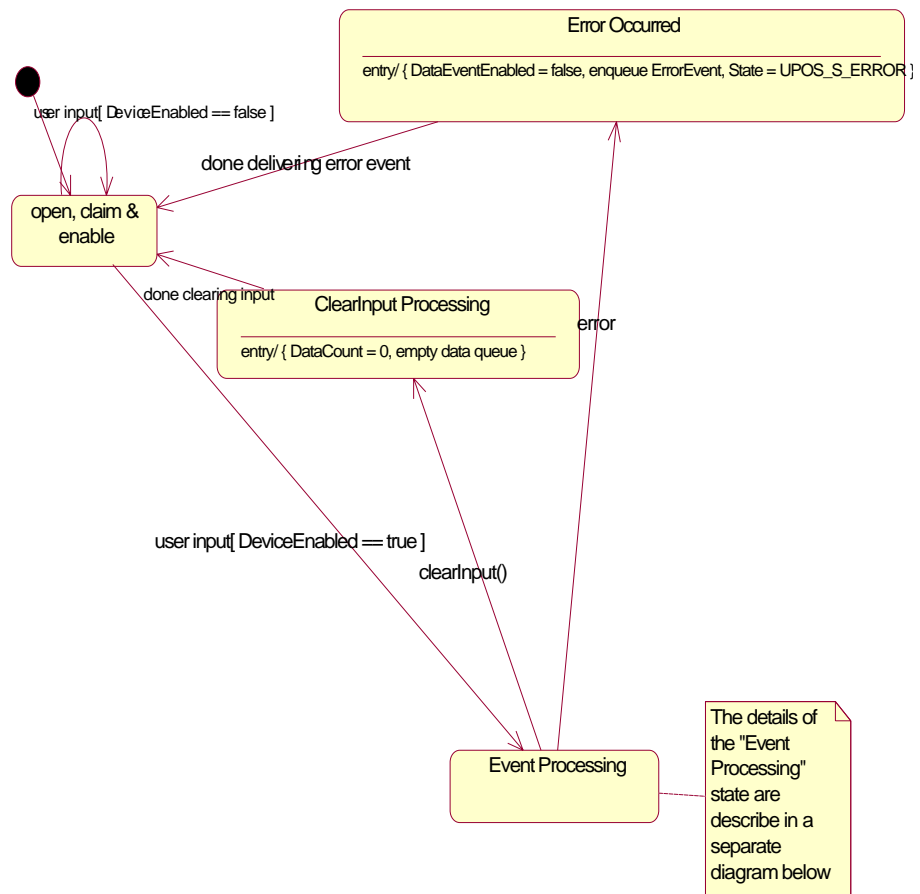
Device Sharing

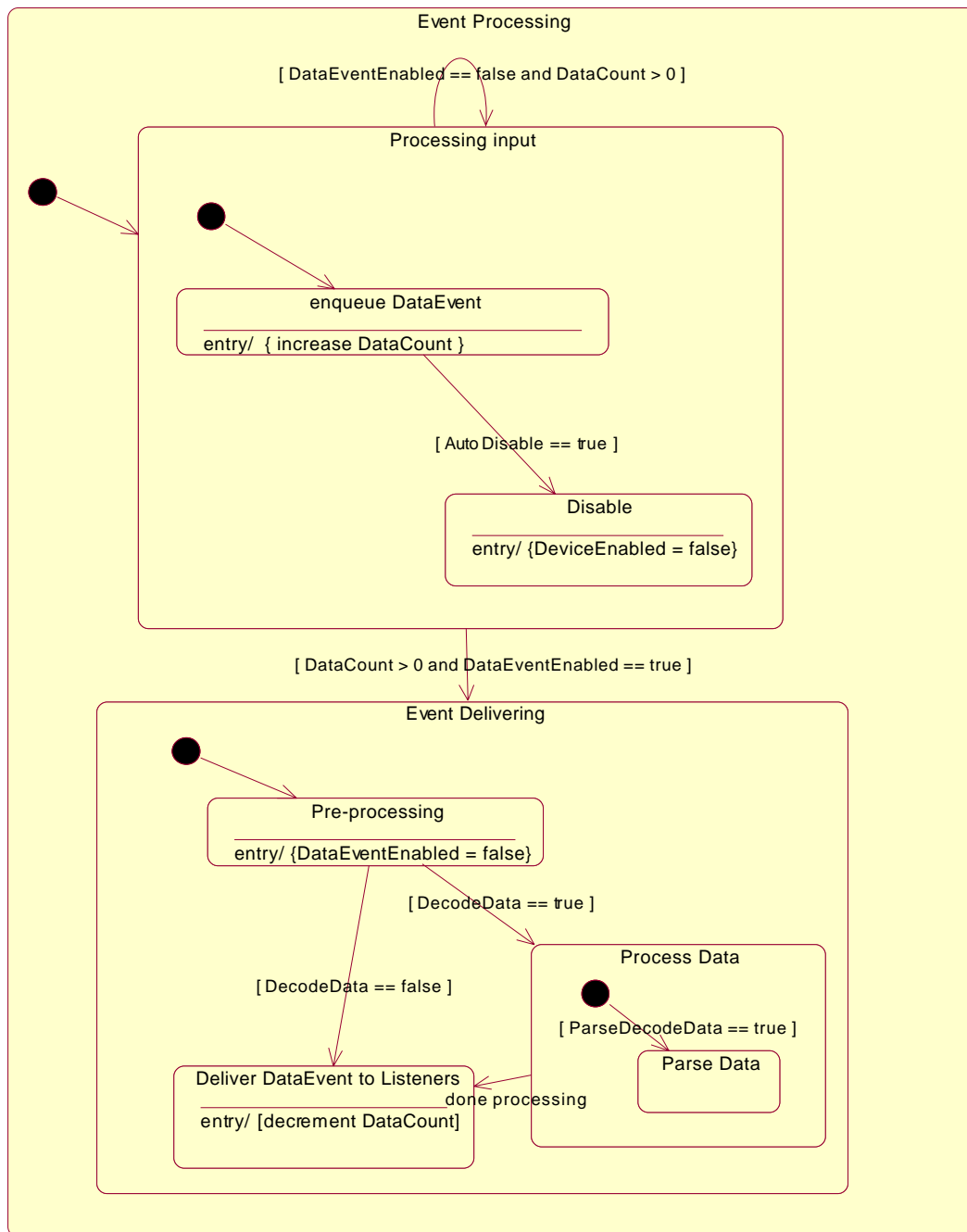
The MSR is an exclusive-use device, as follows:

- 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.

MSR State Diagrams

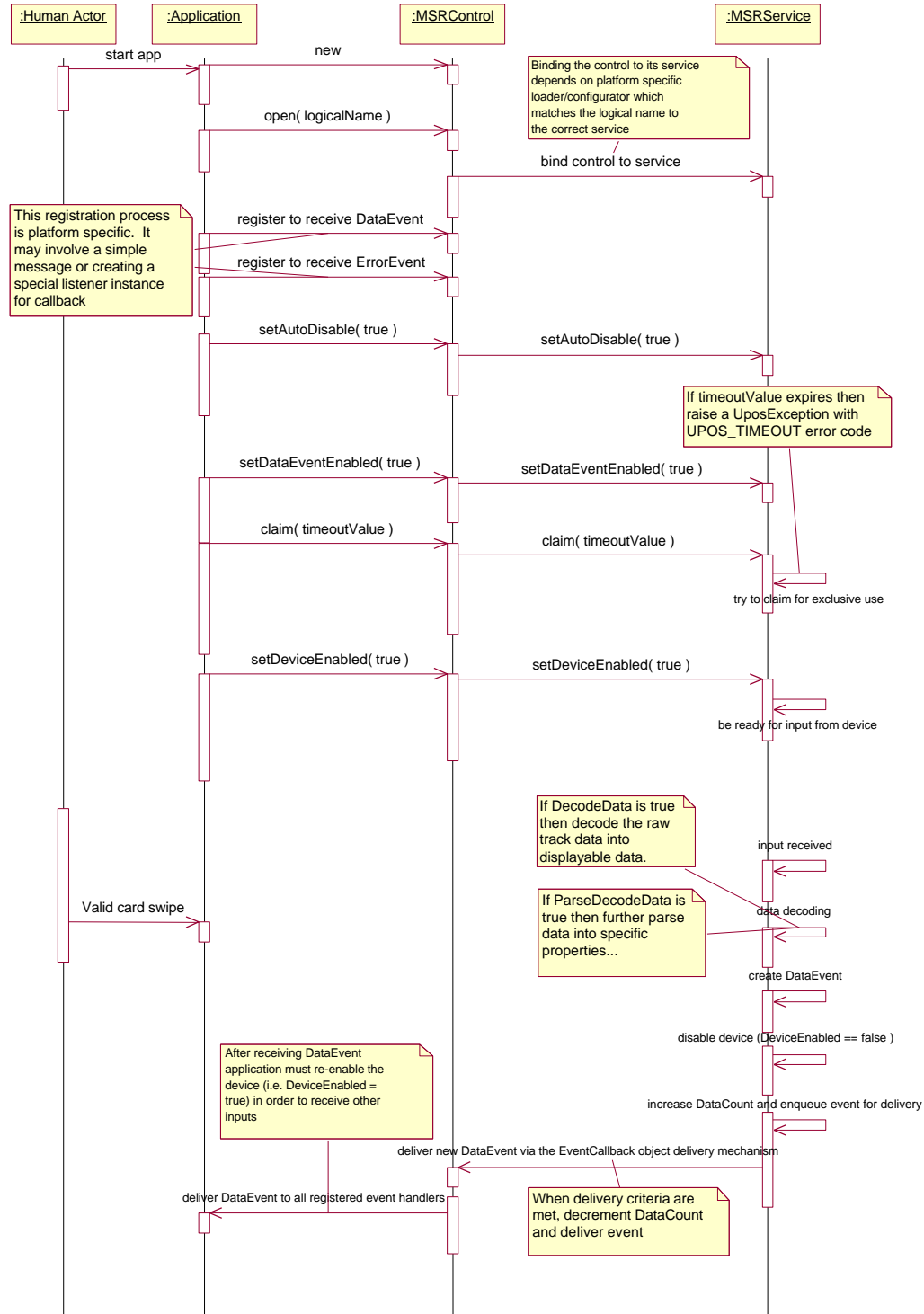
The following state diagrams depict the MSR Control device model.





MSR Usage Diagram

The following diagram is a representation of the typical usage of an MSR device.



Properties (UML attributes)

AccountNumber Property

Syntax	AccountNumber: <i>string</i> { read-only , access after open }
Remarks	<p>Holds the account number obtained from the most recently swiped card.</p> <p>This property is initialized to the empty string if:</p> <ul style="list-style-type: none"> • The field was not included in the track data obtained, or, • The track data format was not one of those listed in the ParseDecodeData property description, or, • ParseDecodeData is false.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	ParseDecodeData Property.

CapISO Property

Syntax	CapISO: <i>boolean</i> { read-only , access after open }
Remarks	<p>If true, the MSR device supports ISO cards.</p> <p>This property is initialized by the open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapJISOne Property

Syntax	CapJISOne: <i>boolean</i> { read-only , access after open }
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 CapJISOne is true, then it is implied that CapISO is also true.</p> <p>This property is initialized by the open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapJISTwo Property

Syntax	CapJISTwo: <i>boolean</i> { read-only , access after open }
Remarks	<p>If true, the MSR device supports JIS type-II cards.</p> <p>This property is initialized by the open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapTransmitSentinels Property *Added in Release 1.5*

- Syntax** **CapTransmitSentinels:** *boolean* { read-only, access after open }
- Remarks** 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 **open** method.
- Errors** A UpoException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
- See Also** **TransmitSentinels** Property.

DecodeData Property

- Syntax** **DecodeData:** *boolean* { read-write, access after open }
- Remarks** If false, the **Track1Data**, **Track2Data**, **Track3Data**, and **Track4Data** properties contain the original encoded bit sequences, known as “raw data format.”

If true, each byte of track data contained within the **Track1Data**, **Track2Data**, **Track3Data**, and **Track4Data**, 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	Raw Data Format	Raw Bytes	Decoded Values
ISO	Track1Data	6-Bit	0x00 - 0x3F	0x20 through 0x5F
	Track2Data	4-Bit	0x00 - 0x0F	0x30 through 0x3F
	Track3Data	4-Bit	0x00 - 0x0F	0x30 through 0x3F
JIS-I	Track1Data	6-Bit	0x00 - 0x3F	0x20 through 0x5F
	Track1Data	7-Bit	0x00 - 0x7F	Data Unaltered
	Track2Data	4-Bit	0x00 - 0x0F	0x20 through 0x3F
	Track3Data	4-Bit	0x00 - 0x0F	0x20 through 0x3F
	Track3Data	7-Bit	0x00 - 0x7F	Data Unaltered
JIS-II	Track4Data	7-Bit	0x00 - 0x7F	Data Unaltered

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

Setting this property to false automatically sets **ParseDecodeData** to false.

- Errors** A UpoException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
- See Also** **ParseDecodeData** Property.

ErrorReportingType Property

Syntax	ErrorReportingType: <i>int32</i> { read-write, access after open }						
Remarks	<p>Holds the type of errors to report via ErrorEvents. This property has one of the following values:</p> <table> <tr> <th>Value</th><th>Meaning</th></tr> <tr> <td>MSR_ERT_CARD</td><td>Report errors at a card level.</td></tr> <tr> <td>MSF_ERT_TRACK</td><td>Report errors at the track level</td></tr> </table> <p>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:</p> <ul style="list-style-type: none"> • 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. <p>This property is initialized to MSR_ERT_CARD by the open method.</p>	Value	Meaning	MSR_ERT_CARD	Report errors at a card level.	MSF_ERT_TRACK	Report errors at the track level
Value	Meaning						
MSR_ERT_CARD	Report errors at a card level.						
MSF_ERT_TRACK	Report errors at the track level						
Errors	A UposException may be thrown when this property is accessed. For further information, see "Errors" on page 15.						
See Also	ErrorEvent						

ExpirationDate Property

Syntax	ExpirationDate: <i>string</i> { read-only, access after open }
Remarks	<p>Holds the expiration date obtained from the most recently swiped card.</p> <p>This property is initialized to the empty string if:</p> <ul style="list-style-type: none">• The field was not included in the track data obtained, or,• The track data format was not one of those listed in the ParseDecodeData property description, or,• ParseDecodeData is false.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	ParseDecodeData Property.

FirstName Property

Syntax	FirstName: <i>string</i> { read-only, access after open }
Remarks	<p>Holds the first name obtained from the most recently swiped card.</p> <p>This property is initialized to an empty string if:</p> <ul style="list-style-type: none">• The field was not included in the track data obtained, or,• The track data format was not one of those listed in the ParseDecodeData property description, or,• ParseDecodeData is false.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	ParseDecodeData Property.

MiddleInitial Property

Syntax	MiddleInitial: <i>string</i> { read-only, access after open }
Remarks	<p>Holds the middle initial obtained from the most recently swiped card. This property is initialized to the empty string if:</p> <ul style="list-style-type: none">• The field was not included in the track data obtained, or,• The track data format was not one of those listed in the ParseDecodeData property description, or,• ParseDecodeData is false.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	ParseDecodeData Property.

ParseDecodeData Property

Syntax	ParseDecodeData: <i>boolean</i> { read-write, access after open }
Remarks	<p>When true, the decoded data contained within the Track1Data and Track2Data properties is further separated into fields for access via various other properties. Track3Data is not parsed because its data content is of an open format defined by the card issuer. JIS-I Track 1 Format C and ISO Track 1 Format C data are not parsed for similar reasons. Track4Data 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"> • JIS-I / ISO Track 1 Format A • JIS-I / ISO Track 1 Format B • JIS-I / ISO Track 1 VISA Format (a defacto standard) • JIS-I / ISO Track 2 Format <p>This property is initialized to true by the open method.</p> <p>Setting this property to true automatically sets DecodeData to true.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	DecodeData Property, Surname Property, Suffix Property, AccountNumber Property, FirstName Property, MiddleInitial Property, Title Property, ExpirationDate Property, ServiceCode Property, Track1DiscretionaryData Property, Track2DiscretionaryData Property.

ServiceCode Property

Syntax	ServiceCode: <i>string</i> { read-only, access after open }
Remarks	<p>Holds the service code obtained from the most recently swiped card.</p> <p>This property is initialized to the empty string if:</p> <ul style="list-style-type: none"> • The field was not included in the track data obtained, or, • The track data format was not one of those listed in the ParseDecodeData property description, or, • ParseDecodeData is false.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	ParseDecodeData Property.

Suffix Property

Syntax	Suffix: <i>string</i> { read-only, access after open }
Remarks	<p>Holds the suffix obtained from the most recently swiped card.</p> <p>This property is initialized to the empty string if:</p> <ul style="list-style-type: none">• The field was not included in the track data obtained, or,• The track data format was not one of those listed in the ParseDecodeData property description, or,• ParseDecodeData is false.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	ParseDecodeData Property.

Surname Property

Syntax	Surname: <i>string</i> { read-only, access after open }
Remarks	<p>Holds the surname obtained from the most recently swiped card.</p> <p>This property is initialized to the empty string if:</p> <ul style="list-style-type: none">• The field was not included in the track data obtained, or,• The track data format was not one of those listed in the ParseDecodeData property description, or,• ParseDecodeData is false.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	ParseDecodeData Property.

Title Property

Syntax	Title: <i>string</i> { read-only, access after open }
Remarks	<p>Holds the title obtained from the most recently swiped card.</p> <p>This property is initialized to the empty string if:</p> <ul style="list-style-type: none">• The field was not included in the track data obtained, or,• The track data format was not one of those listed in the ParseDecodeData property description, or,• ParseDecodeData is false.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	ParseDecodeData Property.

Track1Data Property

Syntax	Track1Data: <i>binary</i> { read-only , access after open }
Remarks	<p>Holds the track 1 data obtained from the most recently swiped card.</p> <p>If TransmitSentinels is false, this property contains track data between but not including the start and end sentinels. If TransmitSentinels is true, then the start and end sentinels are included.</p> <p>If DecodeData 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 ParseDecodeData property is set.</p> <p>A zero length array indicates that the track was not accessible.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	TracksToRead Property, TransmitSentinels Property, ParseDecodeData Property.

Track1DiscretionaryData Property

Syntax	Track1DiscretionaryData: <i>binary</i> { read-only , access after open }
Remarks	<p>Holds the track 1 discretionary data obtained from the most recently swiped card.</p> <p>The array will be zero length if:</p> <ul style="list-style-type: none">• The field was not included in the track data obtained, or,• The track data format was not one of those listed in the ParseDecodeData property description, or,• ParseDecodeData is false. <p>The amount of data contained in this property varies widely depending upon the format of the track 1 data.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	ParseDecodeData Property.

Track2Data Property

Syntax	Track2Data: <i>binary</i> { read-only , access after open }
Remarks	<p>Holds the track 2 data obtained from the most recently swiped card.</p> <p>If TransmitSentinels is false, this property contains track data between but not including the start and end sentinels. If TransmitSentinels is true, then the start and end sentinels are included.</p> <p>If DecodeData 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 ParseDecodeData property is set.</p> <p>A zero length array indicates that the track was not accessible.</p>
Errors	A <code>UposException</code> may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	TracksToRead Property, TransmitSentinels Property, ParseDecodeData Property.

Track2DiscretionaryData Property

Syntax	Track2DiscretionaryData: <i>binary</i> { read-only , access after open }
Remarks	<p>Holds the track 2 discretionary data obtained from the most recently swiped card.</p> <p>The array will be zero length if:</p> <ul style="list-style-type: none">• The field was not included in the track data obtained, or,• The track data format was not one of those listed in the ParseDecodeData property description, or,• ParseDecodeData is false. <p>The amount of data contained in this property varies widely depending upon the format of the track 2 data.</p>
Errors	A <code>UposException</code> may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	ParseDecodeData Property.

Track3Data Property

Syntax	Track3Data: <i>binary</i> { read-only, access after open }
Remarks	<p>Holds the track 3 data obtained from the most recently swiped card.</p> <p>If TransmitSentinels is false, this property contains track data between but not including the start and end sentinels. If TransmitSentinels is true, then the start and end sentinels are included.</p> <p>If DecodeData 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 ParseDecodeData property is set.</p> <p>A zero length array indicates that the track was not accessible.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	TracksToRead Property, TransmitSentinels Property, ParseDecodeData Property.

Track4Data Property

Added in Release 1.5

Syntax	Track4Data: <i>binary</i> { read-only, access after open }
Remarks	<p>Holds the track 4 data (JIS-II) obtained from the most recently swiped card.</p> <p>If TransmitSentinels is false, this property contains track data between but not including the start and end sentinels. If TransmitSentinels is true, then the start and end sentinels are included.</p> <p>If DecodeData 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 Track<i>n</i>Data property. However, it should be noted that to ensure application portability, Track4Data should be used to access JIS-II data.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	Track1Data Property, Track2Data Property, Track3Data Property, TransmitSentinels Property.

TracksToRead Property***Updated in Release 1.5*****Syntax** **TracksToRead: *int32* { read-write, access after open }****Remarks** Holds the track data that the application wishes to have placed into **Track1Data**, **Track2Data**, **Track3Data**, and **Track4Data** properties following a card swipe. This property has one of the following values:

Value	Meaning
MSR_TR_1	Obtain track 1.
MSR_TR_2	Obtain track 2.
MSR_TR_3	Obtain track 3.
MSR_TR_1_2	Obtain tracks 1 and 2.
MSR_TR_1_3	Obtain tracks 1 and 3.
MSR_TR_2_3	Obtain tracks 2 and 3.
MSR_TR_1_2_3	Obtain tracks 1, 2, and 3.
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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

TransmitSentinels Property *Added in Release 1.5*

Syntax	TransmitSentinels: <i>boolean</i> { read-write, access after open }				
Remarks	<p>If true, the Track1Data, Track2Data, Track3Data, and Track4Data 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 open method.</p>				
Errors	<p>A UpoException 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><tr><th>Value</th><th>Meaning</th></tr><tr><td>E_ILLEGAL</td><td>The CapTransmitSentinels property is false.</td></tr></table>	Value	Meaning	E_ILLEGAL	The CapTransmitSentinels property is false.
Value	Meaning				
E_ILLEGAL	The CapTransmitSentinels property is false.				
See Also	CapTransmitSentinels Property, Track1Data Property, Track2Data Property, Track3Data Property, Track4Data Property.				

Events (UML interfaces)

DataEvent

```
<< event >> upos::events::DataEvent
    Status: int32 { read-only }
```

Description Notifies the application when input data from the MSR device is available.

Attributes This event contains the following attribute:

Attributes	Type	Description
------------	------	-------------

<i>Status</i>	<i>int32</i>	See below.
---------------	--------------	------------

The *Status* property is divided into four bytes representing information on up to four tracks of data. 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

```
<< event >> upos::events::DirectIOEvent
    EventNumber: int32 { read-only }
    Data: int32 { read-write }
    Obj: object { read-write }
```

Description Provides Service information directly to the application. This event provides a means for a vendor-specific MSR 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>int32</i>	Event number whose specific values are assigned by the Device Service.
<i>Data</i>	<i>int32</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 MSR devices which may not have any knowledge of the Device Service's need for this event.

See Also "Events" on page 14, **directIO** Method.

ErrorEvent

```
<< event >> upos::events::ErrorEvent
    ErrorCode: int32 { read-only }
    ErrorCodeExtended: int32 { read-only }
    ErrorLocus: int32 { read-only }
    ErrorResponse: int32 { read-write }
```

Description Notifies the application that an error has been detected at the MSR device and a suitable response by the application is necessary to process the error condition.

Attributes This event contains the following properties:

Attributes	Type	Description
<i>ErrorCode</i>	<i>int32</i>	Result code causing the error event. See a list of Error Codes on page 15.
<i>ErrorCodeExtended</i>	<i>int32</i>	Extended Error code causing the error event. If <i>ErrorCode</i> is E_EXTENDED, then see values below. Otherwise, it may contain a Service-specific value.
<i>ErrorLocus</i>	<i>int32</i>	Location of the error. See values below.
<i>ErrorResponse</i>	<i>int32</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_TRACK, and *ErrorCode* is E_EXTENDED, then *ErrorCodeExtended* contains Track-level status, broken down as follows:

High Word		Low Word	
High Byte	Low Byte	High Byte	Low Byte
Track 4	Track 3	Track 2	Track 1

Where each of the track status bytes has one of the following values:

Value	Meaning
SUCCESS	No error occurred.
EMSR_START	Start sentinel error.
EMSR_END	End sentinel error.
EMSR_PARITY	Parity error.
EMSR_LRC	LRC error.
E_FAILURE	Other or general error.

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

Value	Meaning
EL_INPUT	Error occurred while gathering or processing event-driven input. No input data is available.
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
ER_CLEAR	Clear the buffered input data. The error state is exited. Default when locus is EL_INPUT.
ER_CONTINUEINPUT	Use only when locus is 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 EL_INPUT. Default when locus is EL_INPUT_DATA.

Remarks Enqueued when an error is detected while trying to read MSR data. This error event is not delivered until the **DataEventEnabled** property is true, so that proper application sequencing occurs.

If the **ErrorReportingType** property is MSR_ERT_CARD, then the track that caused the fault cannot be determined. The track data properties are not changed.

If the **ErrorReportingType** property is MSR_ERT_TRACK then the *ErrorCode* and the *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 **TrackxData** 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 Models” on page 9 and **ErrorReportingType** Property.

StatusUpdateEvent

<< event >> **upos::events::StatusUpdateEvent**
Status: *int32* { read-only }

Description Notifies the application that there is a change in the power status of the MSR device.

Attributes This event contains the following attribute:

Attributes	Type	Description
<i>Status</i>	<i>int32</i>	Indicates a change in the power status of the unit.

Note that Release 1.3 added Power State Reporting with additional *Power reporting* **StatusUpdateEvent** values. See “StatusUpdateEvent” description on page 56.

Remarks Enqueued when the magnetic stripe reader device detects a power state change.

See Also “Events” on page 14.

PIN Pad

This Chapter defines the PIN Pad device category.

Summary

Properties (UML attributes)

<i>Common</i>	<i>Type</i>	<i>Mutability</i>	<i>Version^a</i>	<i>May Use After</i>
AutoDisable:	<i>boolean</i>	{ read-write }	1.3	Not Supported
CapPowerReporting:	<i>int32</i>	{ read-only }	1.3	open
CheckHealthText:	<i>string</i>	{ read-only }	1.3	open
Claimed:	<i>boolean</i>	{ read-only }	1.3	open
DataCount:	<i>int32</i>	{ read-only }	1.3	open
DataEventEnabled:	<i>boolean</i>	{ read-write }	1.3	open
DeviceEnabled:	<i>boolean</i>	{ read-write }	1.3	open & claim
FreezeEvents:	<i>boolean</i>	{ read-write }	1.3	open
OutputID:	<i>int32</i>	{ read-only }	1.3	Not Supported
PowerNotify:	<i>int32</i>	{ read-write }	1.3	open
PowerState:	<i>int32</i>	{ read-only }	1.3	open
State:	<i>int32</i>	{ read-only }	1.3	--
DeviceControlDescription:	<i>string</i>	{ read-only }	1.3	--
DeviceControlVersion:	<i>int32</i>	{ read-only }	1.3	--
DeviceServiceDescription:	<i>string</i>	{ read-only }	1.3	open
DeviceServiceVersion:	<i>int32</i>	{ read-only }	1.3	open
PhysicalDeviceDescription:	<i>string</i>	{ read-only }	1.3	open
PhysicalDeviceName:	<i>string</i>	{ read-only }	1.3	open

- a. The version representation provides the mechanism for recognizing when a change occurs to a property, method or event. This PIN Pad definition was introduced in an existing standard and was not changed for the UnifiedPOS version 1.4.

Properties (Continued)

<i>Specific</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>	<i>May Use After</i>
CapDisplay:	<i>int32</i>	{ read-only }	1.3	open
CapKeyboard:	<i>boolean</i>	{ read-only }	1.3	open
CapLanguage:	<i>int32</i>	{ read-only }	1.3	open
CapMACCalculation:	<i>boolean</i>	{ read-only }	1.3	open
CapTone:	<i>boolean</i>	{ read-only }	1.3	open
AccountNumber:	<i>string</i>	{ read-write }	1.3	open
AdditionalSecurityInformation:	<i>string</i>	{ read-only }	1.3	open
Amount:	<i>int32</i>	{ read-write }	1.3	open
AvailableLanguagesList:	<i>string</i>	{ read-only }	1.3	open
AvailablePromptsList:	<i>string</i>	{ read-only }	1.3	open
EncryptedPIN:	<i>string</i>	{ read-only }	1.3	open
MaximumPINLength:	<i>int32</i>	{ read-write }	1.3	open
MerchantID:	<i>string</i>	{ read-write }	1.3	open
MinimumPINLength:	<i>int32</i>	{ read-write }	1.3	open
PINEntryEnabled:	<i>boolean</i>	{ read-only }	1.3	open
Prompt:	<i>int32</i>	{ read-write }	1.3	open
PromptLanguage:	<i>nls</i>	{ read-write }	1.3	open
TerminalID:	<i>string</i>	{ read-write }	1.3	open
Track1Data:	<i>binary</i>	{ read-write }	1.3	open
Track2Data:	<i>binary</i>	{ read-write }	1.3	open
Track3Data:	<i>binary</i>	{ read-write }	1.3	open
Track4Data:	<i>binary</i>	{ read-write }	1.5	open
TransactionType:	<i>string</i>	{ read-write }	1.3	open

Methods (UML operations)

Common

Name

open (logicalDeviceName: *string*):
void { raises exception }

close ():
void { raises exception, use after open }

claim (timeout: *int32*):
void { raises exception, use after open }

release ():
void { raises exception, use after open, claim }

checkHealth (level: *int32*):
void { raises exception, use after open, claim, enable }

clearInput ():
void { raises exception, use after open, claim }

clearOutput (): *Not supported*
void { }

directIO (command: *int32*, inout data: *int32*, inout obj: *object*):
void { raises exception, use after open }

Specific

beginEFTTransaction (PINPadSystem: *string*, transactionHost: *int32*):
void { raises exception, use after open, claim, enable }

computeMAC (inMsg: *string*, outMsg: *object*):
void { raises exception, use after beginEFTTransaction }

enablePINEntry():
void { raises exception, use after beginEFTTransaction }

endEFTTransaction (completionCode: *int32*):
void { raises exception, use after beginEFTTransaction }

updateKey (keyNum: *int32*, key: *string*):
void { raises exception, use after beginEFTTransaction }

verifyMAC (message: *string*):
void { raises exception, use after beginEFTTransaction }

Events (UML interfaces)

<i>Name</i>	<i>Type</i>	<i>Mutability</i>
upos::events::DataEvent		
Status:	<i>int32</i>	{ read-only }
upos::events::DirectIOEvent		
EventNumber:	<i>int32</i>	{ read-only }
Data:	<i>int32</i>	{ read-write }
Obj:	<i>object</i>	{ read-write }
upos::events::ErrorEvent		
ErrorCode:	<i>int32</i>	{ read-only }
ErrorCodeExtended:	<i>int32</i>	{ read-only }
ErrorLocus:	<i>int32</i>	{ read-only }
ErrorResponse	<i>int32</i>	{ read-write }
upos::events::StatusUpdateEvent		
Status:	<i>int32</i>	{ read-only }

General Information

The PIN Pad programmatic name is “PINPad”.

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:

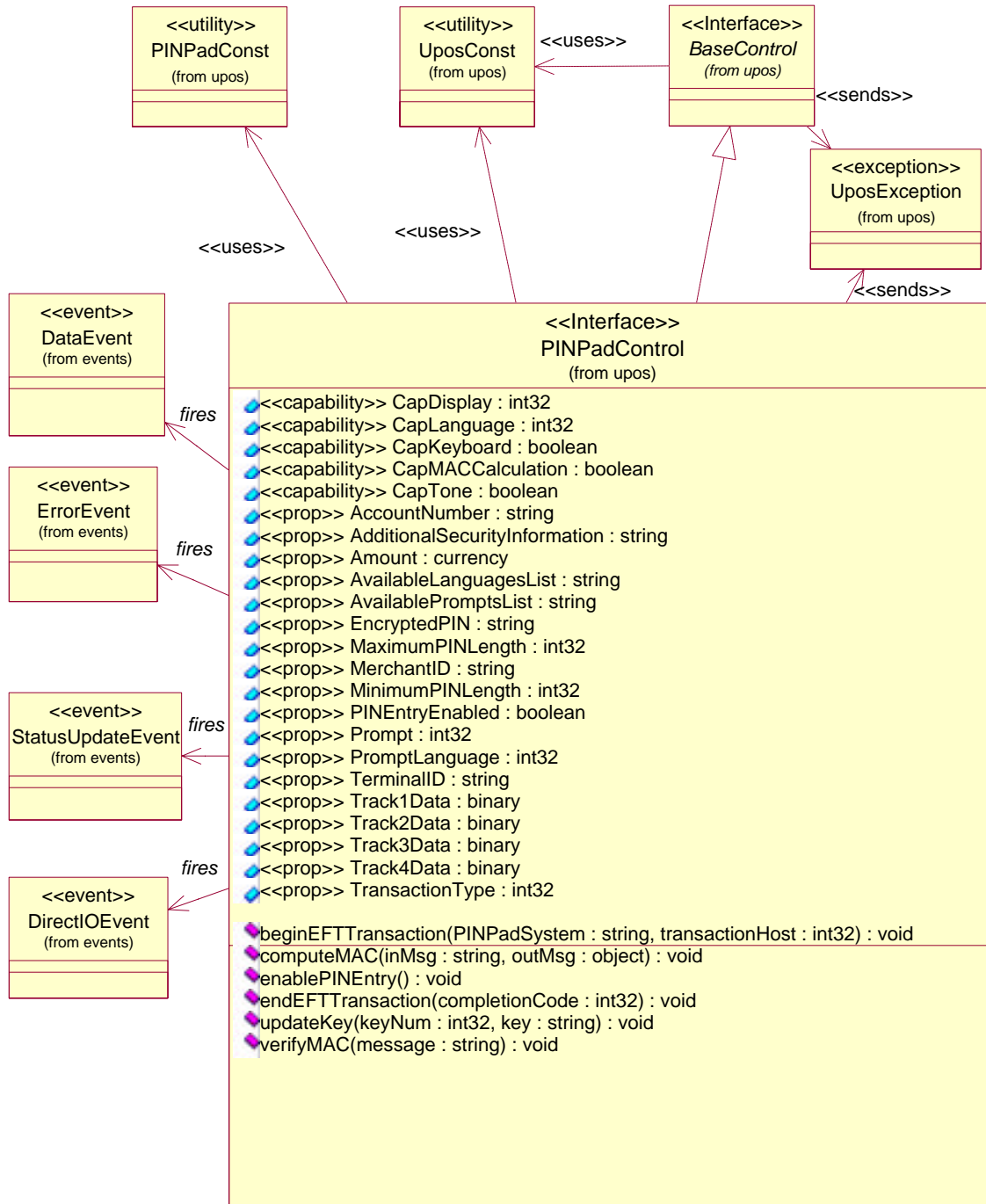
- Accept a PIN Entry at its keyboard and provide an Encrypted PIN to the application.

The PIN Pad Control may have the following additional capabilities:

- Compute Message Authentication Codes.
- Perform Key Updating in accordance with the selected PIN Pad Management System.
- Supports multiple PIN Pad Management Systems.
- Allow 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 Objects:

PIN Pad Class Diagram

The following diagram shows the relationships between the PIN Pad classes.



Feature 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 Pad 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 not 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 PIN Pad Control, an application should use the following general scenario for each EFT Transaction.

- Set the EFT transaction parameters (**AccountNumber**, **Amount**, **MerchantID**, **TerminalID**, **Track1Data**, **Track2Data**, **Track3Data**, **Track4Data**, and **TransactionType** properties) 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, use 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 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.

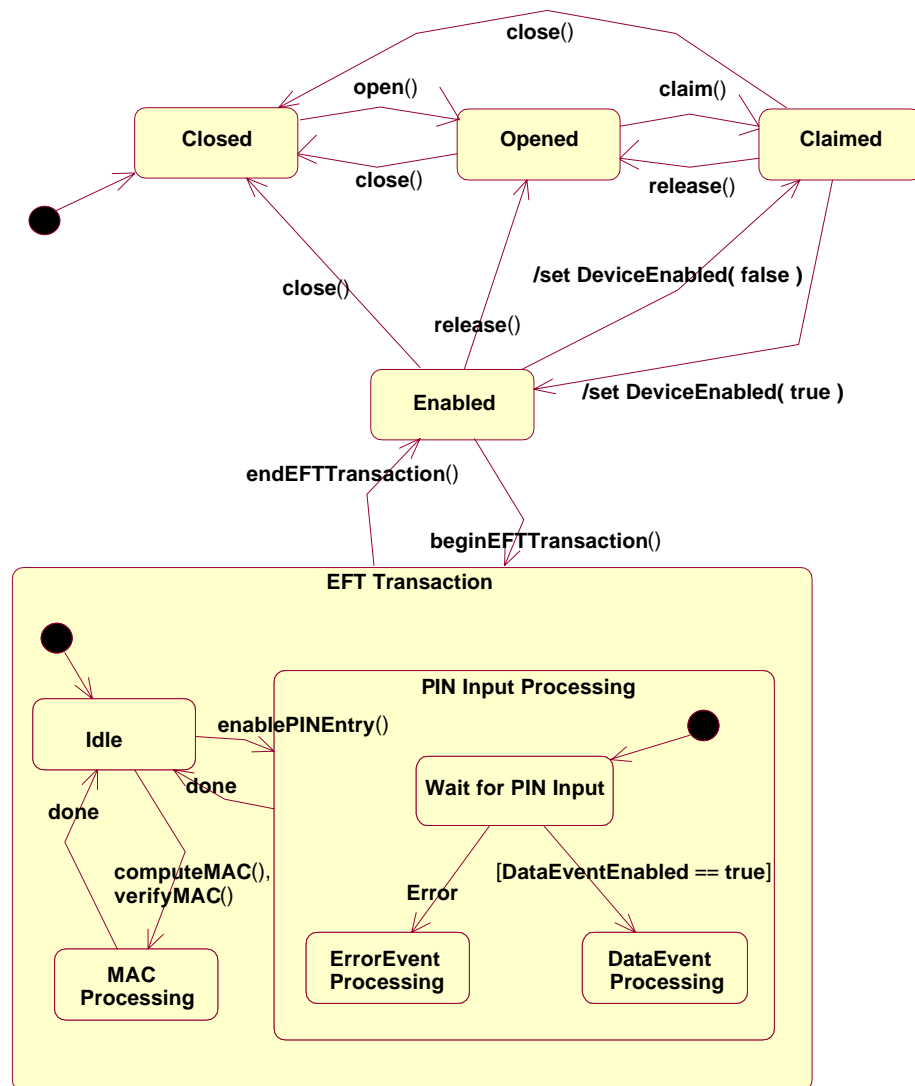
Device Sharing

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.

PIN Pad State Diagram

The following state diagram depicts the PIN Pad Control device model.



Properties (UML attributes)

AccountNumber Property

Syntax	AccountNumber: <i>string</i> { read-write, access after open }
Remarks	Holds the account number to be used for the current EFT transaction. The application must set this property before calling the beginEFTTransaction method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15. Some possible values of the exception’s <i>ErrorCode</i> property are:
Value	Meaning
E_ILLEGAL	An attempt was made to change this property after the beginEFTTransaction method has been called.

AdditionalSecurityInformation Property

Syntax	AdditionalSecurityInformation: <i>string</i> { read-only, access after open }
Remarks	Holds additional security/encryption information when a DataEvent 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.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

Amount Property

Syntax	Amount: <i>int32</i> { read-write, access after open }
Remarks	Holds the amount of the current EFT transaction. The application must set this property before calling the beginEFTTransaction method. This property is a monetary value stored using an implied four decimal places. For example, an actual value of 12345 represents 1.2345.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15. Some possible values of the exception’s <i>ErrorCode</i> property are:
Value	Meaning
E_ILLEGAL	An attempt was made to change this property after the beginEFTTransaction method has been called.

AvailableLanguagesList Property

Syntax	AvailableLanguagesList: <i>string</i> { read-only, access after open }
Remarks	<p>Holds a semi-colon separated list of a set of a “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. 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 CapLanguage is PPAD_LANG_NONE, then this property will be the empty string.</p> <p>This property is initialized by the open method.</p>
Errors	A UpoException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	PromptLanguage Property.

AvailablePromptsList Property

Syntax	AvailablePromptsList: <i>string</i> { read-only, access after open }														
Remarks	<p>Holds a comma-separated string representation of the supported values for the Prompt property.</p> <p>The full set of supported Prompt values are shown below:</p> <table border="1"> <thead> <tr> <th>Name (Value)</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>PPAD_MSG_ENTERPIN (1)</td><td>Enter pin number on the PIN Pad.</td></tr> <tr> <td>PPAD_MSG_PLEASEWAIT (2)</td><td>The system is processing. Wait.</td></tr> <tr> <td>PPAD_MSG_ENTERVALIDPIN (3)</td><td>The pin that was entered is not correct. Enter the correct pin number.</td></tr> <tr> <td>PPAD_MSG_RETRIESEXCEEDED (4)</td><td>The user has failed to enter the correct pin number and the maximum number of attempts has been exceeded.</td></tr> <tr> <td>PPAD_MSG_APPROVED (5)</td><td>The request has been approved.</td></tr> <tr> <td>PPAD_MSG_DECLINED (6)</td><td>The EFT Transaction Host has declined to perform the requested function.</td></tr> </tbody> </table>	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.
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.

PAD_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).

Value 1000 and above are reserved for device specific defined values.

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

Errors

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

CapDisplay Property

Syntax	CapDisplay: <i>int32</i> { read-only, access after open }												
Remarks	Defines the operations that the application may perform on the PIN Pad display.												
	<table> <tr> <th>Value</th><th>Meaning</th></tr> <tr> <td>PPAD_DISP_UNRESTRICTED</td><td>The application can use the PIN Pad display in an unrestricted manner to display messages. In this case, an associated Line Display Control Object is the interface to the PIN Pad display. The application must call Line Display methods to manipulate the display.</td></tr> <tr> <td>PPAD_DISP_PINRESTRICTED</td><td>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 Control Object while PIN Entry is enabled, the Line Display Control will throw a UposException with an associated <i>ErrorCode</i> of E_BUSY.</td></tr> <tr> <td>PPAD_DISP_RESTRICTED_LIST</td><td>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.</td></tr> <tr> <td>PPAD_DISP_RESTRICTED_ORDER</td><td>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.</td></tr> <tr> <td>PPAD_DISP_NONE</td><td>The PIN Pad does not have the PIN Pad display.</td></tr> </table>	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 Control Object 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 Control Object while PIN Entry is enabled, the Line Display Control will throw a UposException with an associated <i>ErrorCode</i> of 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.
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 Control Object 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 Control Object while PIN Entry is enabled, the Line Display Control will throw a UposException with an associated <i>ErrorCode</i> of 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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.												

CapKeyboard Property

Syntax	CapKeyboard: <i>boolean</i> { read-only, access after open }
Remarks	<p>If true, the application can use the PIN Pad to obtain input. The application will use an associated POS Keyboard Device Control object as the interface to the PIN Pad keyboard. Note that the associated POS Keyboard Control is effectively disabled while PINEntryEnabled is true.</p> <p>If false, the application cannot obtain input directly from the PIN Pad keyboard.</p> <p>This property is initialized by the open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapLanguage Property

Syntax	CapLanguage: <i>int32</i> { read-only, access after open }										
Remarks	<p>Defines the capabilities that the application has to select the language of pre-defined messages (e.g. English, French, Arabic etc.).</p> <table border="1"> <thead> <tr> <th>Value</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>PPAD_LANG_NONE</td><td>The PIN Pad supports no pre-defined prompt messages. The property will be set to this value if CapDisplay = PPAD_DISP_UNRESTRICTED. Any attempt to set the value of the PromptLanguage property will cause a UposException to be thrown with the associated <i>ErrorCode</i> of E_ILLEGAL.</td></tr> <tr> <td>PPAD_LANG_ONE</td><td>The PIN Pad supports pre-defined prompt messages in one language. Any attempt to set the value of the PromptLanguage property to other than the default value will cause a UposException to be thrown with the associated <i>ErrorCode</i> of E_ILLEGAL.</td></tr> <tr> <td>PPAD_LANG_PINRESTRICTED</td><td>The PIN Pad cannot change prompt languages during PIN Entry. The application must set the desired value into the PromptLanguage property before calling enablePINEntry. Any attempt to set the value of the PromptLanguage while PINEntryEnabled is true will cause a UposException to be thrown with the associated <i>ErrorCode</i> of E_BUSY.</td></tr> <tr> <td>PPAD_DISP_RESTRICTED_ORDER</td><td>The application can change the language of pre-defined prompt messages at anytime. The currently displayed message will change immediately.</td></tr> </tbody> </table> <p>This property is initialized by the open method.</p>	Value	Meaning	PPAD_LANG_NONE	The PIN Pad supports no pre-defined prompt messages. The property will be set to this value if CapDisplay = PPAD_DISP_UNRESTRICTED. Any attempt to set the value of the PromptLanguage property will cause a UposException to be thrown with the associated <i>ErrorCode</i> of E_ILLEGAL.	PPAD_LANG_ONE	The PIN Pad supports pre-defined prompt messages in one language. Any attempt to set the value of the PromptLanguage property to other than the default value will cause a UposException to be thrown with the associated <i>ErrorCode</i> of E_ILLEGAL.	PPAD_LANG_PINRESTRICTED	The PIN Pad cannot change prompt languages during PIN Entry. The application must set the desired value into the PromptLanguage property before calling enablePINEntry . Any attempt to set the value of the PromptLanguage while PINEntryEnabled is true will cause a UposException to be thrown with the associated <i>ErrorCode</i> of E_BUSY.	PPAD_DISP_RESTRICTED_ORDER	The application can change the language of pre-defined prompt messages at anytime. The currently displayed message will change immediately.
Value	Meaning										
PPAD_LANG_NONE	The PIN Pad supports no pre-defined prompt messages. The property will be set to this value if CapDisplay = PPAD_DISP_UNRESTRICTED. Any attempt to set the value of the PromptLanguage property will cause a UposException to be thrown with the associated <i>ErrorCode</i> of E_ILLEGAL.										
PPAD_LANG_ONE	The PIN Pad supports pre-defined prompt messages in one language. Any attempt to set the value of the PromptLanguage property to other than the default value will cause a UposException to be thrown with the associated <i>ErrorCode</i> of E_ILLEGAL.										
PPAD_LANG_PINRESTRICTED	The PIN Pad cannot change prompt languages during PIN Entry. The application must set the desired value into the PromptLanguage property before calling enablePINEntry . Any attempt to set the value of the PromptLanguage while PINEntryEnabled is true will cause a UposException to be thrown with the associated <i>ErrorCode</i> of E_BUSY.										
PPAD_DISP_RESTRICTED_ORDER	The application can change the language of pre-defined prompt messages at anytime. The currently displayed message will change immediately.										
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.										
See Also	PromptLanguage Property.										

CapMACCalculation Property

Syntax	CapMACCalculation: <i>boolean</i> { read-only , access after open }
Remarks	If true, the PIN Pad supports MAC calculation. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapTone Property

Syntax	CapTone: <i>boolean</i> { read-only , access after open }
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 open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

EncryptedPIN Property

Syntax	EncryptedPIN: <i>string</i> { read-only , access after open }
Remarks	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. 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). If the PIN Entry was cancelled, this property will contain the empty string.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

MaximumPINLength Property

Syntax	MaximumPINLength: <i>int32</i> { read-write , access after open }
Remarks	Holds the maximum acceptable number of digits in a PIN. This property must be set to a default value by the open method. If the application wishes to change this property, it should be set before the enablePINEntry method is called. Note that in some implementations, this value cannot be changed by the application.
Errors	A UposException 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:

Value	Meaning
E_ILLEGAL	An attempt was made to change this property after the enablePINEntry method has been called.

MerchantID Property

Syntax	MerchantID: <i>string</i> { read-write, access after open }
Remarks	Holds the Merchant ID, as it is known to the EFT Transaction Host. The application must set this property before calling the beginEFTTransaction method.
Errors	A UposException 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:

Value	Meaning
E_ILLEGAL	An attempt was made to change this property after the enablePINEntry method has been called.

MinimumPINLength Property

Syntax	MinimumPINLength: <i>int32</i> { read-only, access after open }
Remarks	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 enablePINEntry method is called. Note that in some implementations, this value cannot be changed by the application.
Errors	A UposException 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:

Value	Meaning
E_ILLEGAL	An attempt was made to change this property after the enablePINEntry method has been called.

PINEntryEnabled Property

Syntax	PINEntryEnabled: <i>boolean</i> { read-write, access after open }
Remarks	If true, the PIN entry operation is enabled. It is set when the enablePINEntry method is called. It will be set to false when the user has completed the PIN Entry operation or when the endEFTTransaction method has completed.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

Prompt Property

Syntax	Prompt: int32 { read-write, access after open }				
Remarks	<p>Holds the identifies a pre-defined message to be displayed on the PIN Pad. This property is used if CapDisplay is PPAD_DISP_RESTRICTED_LIST or PPAD_DISP_RESTRICTED_ORDER. It is also used during PIN Entry if CapDisplay has a value of PPAD_DISP_PINRESTRICTED. The AvailablePromptsList property lists the possible values for this property.</p> <p>This property is initialized by the open method.</p>				
Errors	<p>A UposException 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><tr><th>Value</th><th>Meaning</th></tr><tr><td>E_ILLEGAL</td><td>One of the following has occurred.<ul style="list-style-type: none">* An attempt was made to set the property to a value that is not supported by the PIN Pad Device Service.* An attempt was made to select prompt messages in an unacceptable order (if CapDisplay is PPAD_DISP_RESTRICTED_ORDER).</td></tr></table>	Value	Meaning	E_ILLEGAL	One of the following has occurred. <ul style="list-style-type: none">* An attempt was made to set the property to a value that is not supported by the PIN Pad Device Service.* An attempt was made to select prompt messages in an unacceptable order (if CapDisplay is PPAD_DISP_RESTRICTED_ORDER).
Value	Meaning				
E_ILLEGAL	One of the following has occurred. <ul style="list-style-type: none">* An attempt was made to set the property to a value that is not supported by the PIN Pad Device Service.* An attempt was made to select prompt messages in an unacceptable order (if CapDisplay is PPAD_DISP_RESTRICTED_ORDER).				
See Also	PromptLanguage Property.				

PromptLanguage Property

Syntax	PromptLanguage: <i>nls</i> { read-write, access after open }
Remarks	<p>Holds the “language definition” for the message to be displayed (as specified by the Prompt property). This property is used if the Prompt property is begin used. The exact effect of changing this property depends on the value of CapLanguage.</p> <p>A “language definition” consists of an ISO-639 language code and an ISO-3166 country code. 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>
Errors	<p>A UpoException 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>

Value	Meaning
E_ILLEGAL	<p>One of the following occurred.</p> <ul style="list-style-type: none"> * An attempt was made to set the property to a value that is not supported by the PIN Pad Device Service. * CapLanguage is PPAD_LANG_NONE. and an attempt was made to set the value of this property. * CapLanguage is PPAD_LANG_ONE and an attempt was made to set the value of this property to other than the default value.
E_BUSY	CapLanguage is PPAD_LANG_PINRESTRICTED and PINEntryEnabled is true.

See Also **CapLanguage** Property, **AvailableLanguagesList** Property.

TerminalID Property

Syntax	TerminalID: <i>string</i> { read-write, access after open }
Remarks	Holds the terminal ID, as it is known to the EFT Transaction Host. The application must set this property before calling the beginEFTTransaction method.
Errors	<p>A UpoException may be thrown when this property is accessed. For further information, see “Errors” on page 15.</p>

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

Value	Meaning
E_ILLEGAL	An attempt was made to change this property after the beginEFTTransaction method has been called.

Track1Data Property

Syntax	Track1Data: <i>binary</i> { read-write, access after open }				
Remarks	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 beginEFTTransaction method.				
Errors	<p>A UposException 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>Value</th><th>Meaning</th></tr></thead><tbody><tr><td>E_ILLEGAL</td><td>An attempt was made to change this property after the beginEFTTransaction method has been called.</td></tr></tbody></table>	Value	Meaning	E_ILLEGAL	An attempt was made to change this property after the beginEFTTransaction method has been called.
Value	Meaning				
E_ILLEGAL	An attempt was made to change this property after the beginEFTTransaction method has been called.				

Track2Data Property

Syntax	Track2Data: <i>binary</i> { read-write, access after open }				
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 beginEFTTransaction method.				
Errors	<p>A UposException 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>Value</th><th>Meaning</th></tr></thead><tbody><tr><td>E_ILLEGAL</td><td>An attempt was made to change this property after the beginEFTTransaction method has been called.</td></tr></tbody></table>	Value	Meaning	E_ILLEGAL	An attempt was made to change this property after the beginEFTTransaction method has been called.
Value	Meaning				
E_ILLEGAL	An attempt was made to change this property after the beginEFTTransaction method has been called.				

Track3Data Property

Syntax	Track3Data: <i>binary</i> { read-write, access after open }				
Remarks	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 beginEFTTransaction method.				
Errors	<p>A UposException 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>Value</th><th>Meaning</th></tr></thead><tbody><tr><td>E_ILLEGAL</td><td>An attempt was made to change this property after the beginEFTTransaction method has been called.</td></tr></tbody></table>	Value	Meaning	E_ILLEGAL	An attempt was made to change this property after the beginEFTTransaction method has been called.
Value	Meaning				
E_ILLEGAL	An attempt was made to change this property after the beginEFTTransaction method has been called.				

Track4Data Property**Added in Release 1.5**

Syntax	Track4Data: <i>binary</i> { read-write, access after open }
Remarks	<p>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 beginEFTTransaction method.</p> <p>To maintain compatibility with previous versions, the Control may also continue to store the JIS-II data in another Track<i>n</i>Data property. However, it should be noted that to ensure application portability, Track4Data should be used to access JIS-II data.</p>
Errors	<p>A UpoException may be thrown when this property is accessed. For further information, see “Errors” on page 15.</p>

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

Value	Meaning
E_ILLEGAL	An attempt was made to change this property after the beginEFTTransaction method has been called.

TransactionType Property

Syntax	TransactionType: <i>int32</i> { read-write, access after open }
Remarks	<p>Holds the type of the current EFT Transaction. The application must set this property before calling the beginEFTTransaction method.</p>

This property have one of the following values:

Value	Meaning
PPAD_TRANS_DEBIT	Debit (decrease) the specified account
PPAD_TRANS_CREDIT	Credit (increase) the specified account
PPAD_TRANS_INQ	(Balance) Inquiry
PPAD_TRANS_RECONCILE	Reconciliation/Settlement
PPAD_TRANS_ADMIN	Administrative Transaction

Errors	<p>A UpoException may be thrown when this property is accessed. For further information, see “Errors” on page 15.</p>
---------------	---

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

Value	Meaning
E_ILLEGAL	An attempt was made to change this property after the beginEFTTransaction method has been called.

Methods (UML operations)

beginEFTTransaction Method

Syntax **beginEFTTransaction** (*PINPadSystem*: *string*, *transactionHost*: *int32*) :
void { raises-exception, use after open-claim-enable }

Value	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>	Identifications particular EFT Transaction Host to be used for this transaction.

The *PINPadSystem* Parameter has one of the following values:

Value	Description
“M/S”	Master/Session (U.S.A 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 Initialize 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 UpoException 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
E_ILLEGAL	The requested PIN Pad Management System is not supported by the Control, or the requested EFT Transaction Host is an illegal value for the selected PIN Pad Management System.
E_BUSY	The PIN Pad is already performing an EFT transaction.

computeMAC Method

Syntax **computeMAC** (*inMsg*: *string*, *outMsg*: *object*) :
 void { raises-exception, use after beginEFTTransaction)

Value	Description
<i>inMsg</i>	The message that the application intends to send to an EFT Transaction.
<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 Computers 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 UpoException 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
E_DISABLED	A beginEFTTransaction method has not been performed.
E_BUSY	PINEntryEnabled is true. The PIN Pad cannot perform a MAC calculation during PIN Entry.

enablePINEntry Method

Syntax **enablePINEntry** ():
 void { raises-exception, use after beginEFTTransaction);

Remarks Enable 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 Control Object.

Errors A UpoException 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
E_DISABLED	A beginEFTTransaction method has not been performed.

endEFTTransaction Method

Syntax **endEFTTransaction** (*completionCode*: *int32*):
 void { raises-exception, use after beginEFTTransaction }

The *completionCode* is one of the following values:

Value	Description
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 UpoException may be thrown when this method is invoked. For further information, see “Errors” on page 15.

updateKey Method

Syntax **updateKey** (*keyNum*: *int32*, *key*: *string*):
 void { raises-exception, use after beginEFTTransaction }

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 UpoException 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
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 **verifyMAC** (*message*: *string*):
 void { raises-exception, use after beginEFTTransaction }

Parameter	Description
<i>message</i>	Contains a message received from an EFT Transaction Host.

Remarks Verify the MAC value in a message received from an EFT Transaction Host. This method throws a *UposException* if it can verify the message. Note that this method cannot be used while PIN Entry is enabled.

Errors A *UposException* 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
E_DISABLED	A beginEFTTransaction method has not been performed.
E_BUSY	PINEntryEnabled is true. The PIN Pad cannot perform a MAC verification during PIN Entry.

Events (UML interfaces)

DataEvent

<< event >> **upos::events::DataEvent**
Status: *int32* { read-only }

Description Notifies the application when a PIN Entry operation has completed.

Attributes This event contains the following attribute:

Attributes	Type	Description
<i>Status</i>	<i>int32</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 EncryptedPIN and AdditionalSecurityInformation 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).

Remarks 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.

See Also "Device Input Model" on page 17.

DirectIOEvent

```
<< event >> upos::events::DirectIOEvent
    EventNumber: int32 { read-only }
    Data: int32 { read-write }
    Obj: object { read-write }
```

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

Attributes This event contains the following attributes:

Attributes	Type	Description
<i>EventNumber</i>	<i>int32</i>	Event number whose specific values are assigned by the Service event.
<i>Data</i>	<i>int32</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 PIN Pad devices which may not have any knowledge of the Service's need for this event.

See Also "Events" on page 14, **directIO** Method

ErrorEvent

```
<< event >> upos::events::ErrorEvent
    ErrorCode: int32 { read-only }
    ErrorCodeExtended: int32 { read-only }
    ErrorLocus: int32 { read-only }
    ErrorResponse: int32 { read-only }
```

Description Notifies the application that an error was detected while trying to perform a PIN encryption function.

Attributes This event contains the following properties:

Attributes	Type	Description
<i>ErrorCode</i>	<i>int32</i>	Result code causing the error event. See a list of Error Codes on page 15.
<i>ErrorCodeExtended</i>	<i>int32</i>	Extended Error code causing the error event. If <i>ErrorCode</i> is E_EXTENDED, then see values below. Otherwise, it may contain a Service-specific value.
<i>ErrorLocus</i>	<i>int32</i>	Location of the error, and is set to EL_INPUT indicating that the error occurred while gathering or processing event-driven input.

ErrorResponse *int32* Error response, whose default value may be overridden by the application (i.e., this property is settable). See values below.

If *ErrorCode* is E_EXTENDED, then *ErrorCodeExtended* has one of the following values:

Value	Meaning
EPPAD_BAD_KEY	An Encryption Key is corrupted or missing.

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

Value	Meaning
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 values:

Value	Meaning
ER_CLEAR	Clear the buffered input data. The error state is exited. Default when locus is EL_INPUT.

Remarks Enqueued when an error is detected and the Service'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 Behavior Models" on page 9 and **ErrorReportingType** Property.

StatusUpdateEvent

<< event >> **upos::events::StatusUpdateEvent**
Status: *int32* { read-only }

Description Notifies the application that there is a change in the power status of a PIN Pad.

Attributes This event contains the following attribute:

Attributes	Type	Description
<i>Status</i>	<i>int32</i>	Indicates the status change, and has one of the following values: <i>Note that Release 1.3</i> added Power State Reporting with additional <i>Power reporting StatusUpdateEvent values</i> . See "StatusUpdateEvent" description on page 56.

Remarks Enqueued when the PIN Pad detects a power state change.

See Also "Events" on page 14.

Point Card Reader Writer

This Chapter defines the Point Card Reader Writer device category.

Summary

Properties (UML attributes)

<i>Common</i>	<i>Type</i>	<i>Mutability</i>	<i>Version^a</i>	<i>May Use After</i>
AutoDisable:	<i>boolean</i>	{ read-write }	1.5	Not Supported
CapPowerReporting:	<i>int32</i>	{ read-only }	1.5	open
CheckHealthText:	<i>string</i>	{ read-only }	1.5	open
Claimed:	<i>boolean</i>	{ read-only }	1.5	open
DataCount:	<i>int32</i>	{ read-only }	1.5	open
DataEventEnabled:	<i>boolean</i>	{ read-write }	1.5	open
DeviceEnabled:	<i>boolean</i>	{ read-write }	1.5	open & claim
FreezeEvents:	<i>boolean</i>	{ read-write }	1.5	open
OutputID:	<i>int32</i>	{ read-only }	1.5	open
PowerNotify:	<i>int32</i>	{ read-write }	1.5	open
PowerState:	<i>int32</i>	{ read-only }	1.5	open
State:	<i>int32</i>	{ read-only }	1.5	--
DeviceControlDescription:	<i>string</i>	{ read-only }	1.5	--
DeviceControlVersion:	<i>int32</i>	{ read-only }	1.5	--
DeviceServiceDescription:	<i>string</i>	{ read-only }	1.5	open
DeviceServiceVersion:	<i>int32</i>	{ read-only }	1.5	open
PhysicalDeviceDescription:	<i>string</i>	{ read-only }	1.5	open
PhysicalDeviceName:	<i>string</i>	{ read-only }	1.5	open

- a. The version representation provides the mechanism for recognizing when a change occurs to a property, method, or event. The Point Card Reader Writer definition was introduced in the UnifiedPOS version 1.5.

Properties (Continued)

<i>Specific:</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>	<i>May Use After</i>
CapBold:	<i>boolean</i>	{ read-only }	1.5	open
CapCardEntranceSensor:	<i>int32</i>	{ read-only }	1.5	open
CapCharacterSet:	<i>int32</i>	{ read-only }	1.5	open
CapCleanCard:	<i>boolean</i>	{ read-only }	1.5	open
CapClearPrint:	<i>boolean</i>	{ read-only }	1.5	open
CapDhigh:	<i>boolean</i>	{ read-only }	1.5	open
CapDwide:	<i>boolean</i>	{ read-only }	1.5	open
CapDwideDhigh:	<i>boolean</i>	{ read-only }	1.5	open
CapItalic:	<i>boolean</i>	{ read-only }	1.5	open
CapLeft90:	<i>boolean</i>	{ read-only }	1.5	open
CapPrint:	<i>boolean</i>	{ read-only }	1.5	open
CapPrintMode:	<i>boolean</i>	{ read-only }	1.5	open
CapRight90:	<i>boolean</i>	{ read-only }	1.5	open
CapRotate180:	<i>boolean</i>	{ read-only }	1.5	open
CapTracksToRead:	<i>int32</i>	{ read-only }	1.5	open
CapTracksToWrite:	<i>int32</i>	{ read-only }	1.5	open
CardState:	<i>int32</i>	{ read-only }	1.5	open
CharacterSet:	<i>int32</i>	{ read-write }	1.5	open, claim, & enable
CharacterSetList:	<i>string</i>	{ read-only }	1.5	open
FontTypeFaceList:	<i>string</i>	{ read-only }	1.5	open
LineChars:	<i>int32</i>	{ read-only }	1.5	open, claim, & enable
LineCharsList:	<i>string</i>	{ read-only }	1.5	open
LineHeight:	<i>int32</i>	{ read-only }	1.5	open, claim, & enable
LineSpacing:	<i>int32</i>	{ read-only }	1.5	open, claim, & enable
LineWidth:	<i>int32</i>	{ read-only }	1.5	open, claim, & enable
MapMode:	<i>int32</i>	{ read-only }	1.5	open, claim, & enable
MaxLine:	<i>int32</i>	{ read-only }	1.5	open, claim, & enable
PrintHeight:	<i>int32</i>	{ read-only }	1.5	open, claim, & enable
ReadState1:	<i>int32</i>	{ read-only }	1.5	open
ReadState2:	<i>int32</i>	{ read-only }	1.5	open
RecvLength1:	<i>int32</i>	{ read-only }	1.5	open, claim, & enable
RecvLength2:	<i>int32</i>	{ read-only }	1.5	open, claim, & enable
SidewaysMaxChars:	<i>int32</i>	{ read-only }	1.5	open
SidewaysMaxLines:	<i>int32</i>	{ read-only }	1.5	open

Properties (Continued)

<i>Specific:</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>	<i>May Use After</i>
TracksToRead:	<i>int32</i>	{ read-write }	1.5	open, claim, & enable
TracksToWrite:	<i>int32</i>	{ read-write }	1.5	open, claim, & enable
Track1Data:	<i>binary</i>	{ read-only }	1.5	open
Track2Data:	<i>binary</i>	{ read-only }	1.5	open
Track3Data:	<i>binary</i>	{ read-only }	1.5	open
Track4Data:	<i>binary</i>	{ read-only }	1.5	open
Track5Data:	<i>binary</i>	{ read-only }	1.5	open
Track6Data:	<i>binary</i>	{ read-only }	1.5	open
WriteState1:	<i>int32</i>	{ read-only }	1.5	open
WriteState2:	<i>int32</i>	{ read-only }	1.5	open
Write1Data:	<i>binary</i>	{ read-write }	1.5	open
Write2Data:	<i>binary</i>	{ read-write }	1.5	open
Write3Data:	<i>binary</i>	{ read-write }	1.5	open
Write4Data:	<i>binary</i>	{ read-write }	1.5	open
Write5Data:	<i>binary</i>	{ read-write }	1.5	open
Write6Data:	<i>binary</i>	{ read-write }	1.5	open

Methods (UML operations)

Common

Name

```
open ( logicalDeviceName: string ):  
    void { raises exception }  
close ():  
    void { raises exception, use after open }  
claim ( timeout: int32 ):  
    void { raises exception, use after open }  
release ():  
    void { raises exception, use after open, claim }  
checkHealth ( level: int32 ):  
    void { raises exception, use after open, claim, enable }  
clearInput ():  
    void { raises exception, use after open, claim }  
clearOutput ():  
    void { raises exception, use after open, claim }  
directIO ( command: int32, inout data: int32, inout obj: object ):  
    void { raises exception, use after open, claim }
```

Specific

Name

```
beginInsertion ( timeout: int32 ):  
    void { raises exception, use after open, claim, enable }  
beginRemoval ( timeout: int32 ):  
    void { raises exception, use after open, claim, enable }  
cleanCard ():  
    void { raises exception, use after open, claim, enable }  
clearPrintWrite ( kind: int32, hposition: int32, vposition: int32, width: int32,  
    height: int32 ):  
    void { raises exception, use after open, claim, enable }  
endInsertion ():  
    void { raises exception, use after open, claim, enable }  
endRemoval ():  
    void { raises exception, use after open, claim, enable }
```

printWrite (kind: *int32*, hposition: *int32*, vposition: *int32*, data: *string*):
 void { raises exception, use after open, claim, enable }

rotatePrint (rotation: *int32*):
 void { raises exception, use after open, claim, enable }

validateData (data: *string*):
 void { raises exception, use after open, claim, enable }

Events (UML interfaces)

<i>Name</i>	<i>Type</i>	<i>Mutability</i>
upos::events::DataEvent		
Status:	<i>int32</i>	{ read-only }
upos::events::DirectIOEvent		
EventNumber:	<i>int32</i>	{ read-only }
Data:	<i>int32</i>	{ read-write }
Obj:	<i>object</i>	{ read-write }
upos::events::ErrorEvent		
ErrorCode:	<i>int32</i>	{ read-only }
ErrorCodeExtended:	<i>int32</i>	{ read-only }
ErrorLocus:	<i>int32</i>	{ read-only }
ErrorResponse:	<i>int32</i>	{ read-write }
upos::events::OutputCompleteEvent		
OutputID:	<i>int32</i>	{ read-only }
upos::events::StatusUpdateEvent		
Status:	<i>int32</i>	{ read-only }

General Information

The Point Card Reader Writer programmatic name is “PointCardRW”.
This device was introduced in Version 1.5 of the specification.

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

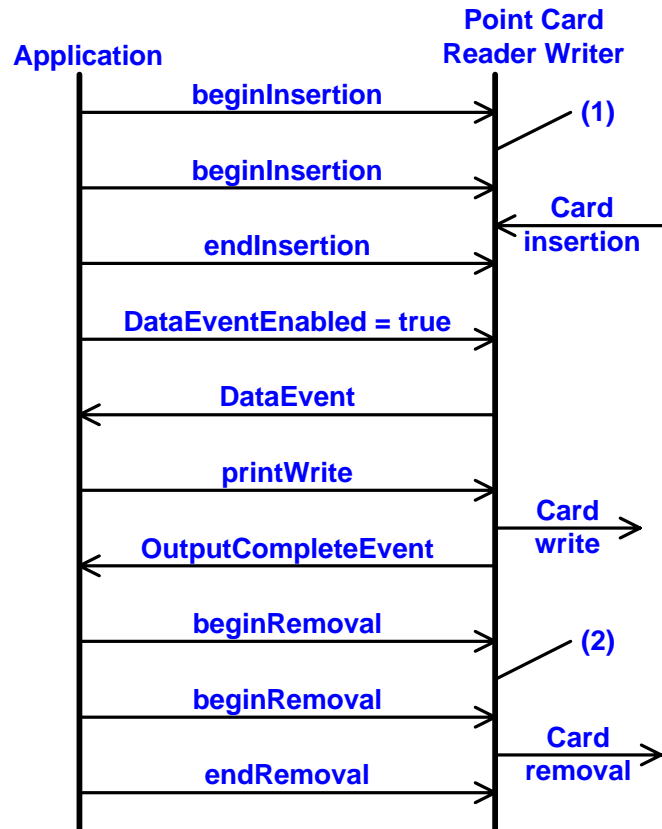
- An application must call **open** and **claim**, then set **DeviceEnabled** to true.
- When an application wants a card inserted, it calls the **beginInsertion** method, specifying a timeout value.
- If a card is not inserted before the timeout period elapses, the Point Card Reader Writer fires an 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.

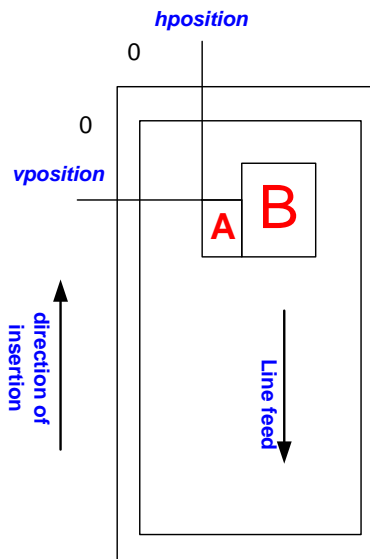


- (1) If the card is not inserted into the Point Card Reader Writer before the application specified timeout elapses, an exception is fired. The application needs to call **beginInsertion** again to confirm that a point card has been inserted or call **endInsertion** to cancel the card insertion. After 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 fired. 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.

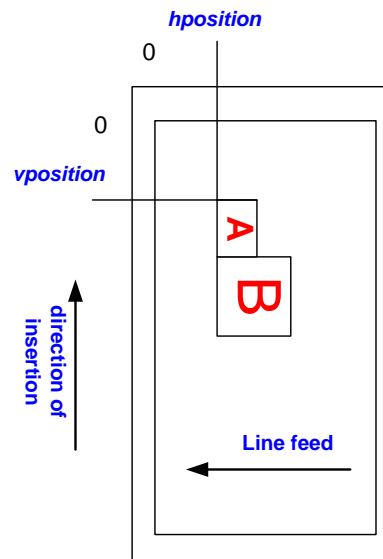
Printing Capability

- 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 printing



Vertical printing



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.

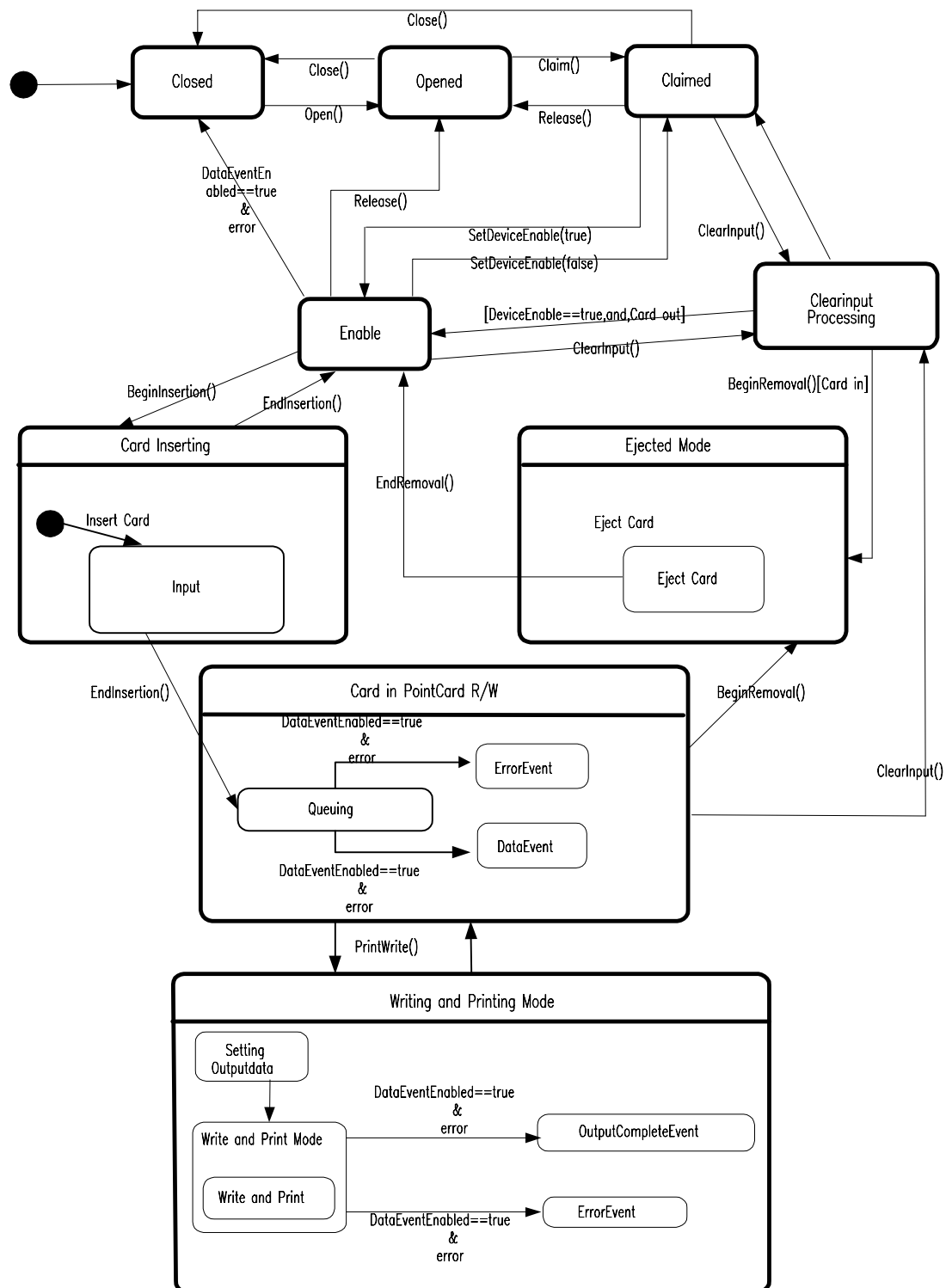
Print Mode Characteristics that are remembered until explicitly changed.

Name	Data	Remarks
Font typeface selection	ESC #fT	Selects a new typeface for the following data. Values for the character '#' are: 0 = Default typeface. 1 = Select first typeface from the FontTypefaceList property. 2 = Select second typeface from the FontTypefaceList property. And so on.

Print Line Characteristics that are reset at the end of each print method or by a “Normal” sequence.

Name	Data	Remarks
Bold	ESC bC	Prints in bold or double-strike.
Underline	ESC #uC	Prints with underline. The character ‘#’ is replaced by an ASCII decimal string telling the 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.

Point Card Reader Writer State Diagram



Properties (UML Attributes)

CapBold Property

Syntax	CapBold: <i>boolean</i> { read-only, access after open }
Remarks	<p>If true, then the Point Card Reader Writer can print bold characters, false if it cannot.</p> <p>This property is initialized by the open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapCardEntranceSensor Property

Syntax	CapCardEntranceSensor: <i>boolean</i> { read-only, access after open }
Remarks	<p>If true, then the Point Card Reader Writer has an entrance sensor, false if it does not.</p> <p>This property is initialized by the open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapCharacterSet Property

Syntax	CapCharacterSet: <i>int32</i> { read-only, access after open }												
Remarks	<p>Holds the default character set capability. It may be one of the following:</p> <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 CharacterSet property may be examined for additional information.</p> <p>This property is initialized by the open 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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.												

CapCleanCard Property

Syntax	CapCleanCard: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the Point Card Reader Writer supports cleaning under application control, false if it does not. This property is initialized by the open method.
Errors	A UpoException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapClearPrint Property

Syntax	CapClearPrint: <i>boolean</i> { read-only, access after open }
Remarks	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 open method.
Errors	A UpoException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapDhigh Property

Syntax	CapDhigh: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the Point Card Reader Writer can print double high characters, false if it cannot. This property is initialized by the open method.
Errors	A UpoException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapDwide Property

Syntax	CapDwide: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the Point Card Reader Writer can print double wide characters, false if it cannot. This property is initialized by the open method.
Errors	A UpoException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapDwideDhigh Property

Syntax	CapDwideDhigh: <i>boolean</i> { read-only, access after open }
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 open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapItalic Property

Syntax	CapItalic: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the Point Card Reader Writer can print italic characters, false if it cannot. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapLeft90 Property

Syntax	CapLeft90: <i>boolean</i> { read-only, access after open }
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 open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapPrint Property

Syntax	CapPrint: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the Point Card Reader Writer has printing capability; false if it does not. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapPrintMode Property

Syntax	CapPrintMode: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the Point Card Reader Writer can designate a printing start position with the MapMode property, false if it cannot. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapRight90 Property

Syntax	CapRight90: <i>boolean</i> { read-only, access after open }
Remarks	<p>If true, then the Point Card Reader Writer can print in a rotated 90° right mode, false if it cannot.</p> <p>This property is initialized by the open method.</p>
Errors	A UpoException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapRotate180 Property

Syntax	CapRotate180: <i>boolean</i> { read-only, access after open }
Remarks	<p>If true, then the Point Card Reader Writer can print in a rotated upside down mode, false if it cannot.</p> <p>This property is initialized by the open method.</p>
Errors	A UpoException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapTracksToRead Property

Syntax	CapTracksToRead: <i>int32</i> { read-only, access after open }														
Remarks	<p>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.</p> <p>For example, access to track 1 is possible when PCRW_TRACK1 is set.</p> <p>This property is initialized by the open method.</p> <table border="1"> <thead> <tr> <th>Value</th><th>Meaning</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>	Value	Meaning	PCRW_TRACK1	Track1	PCRW_TRACK2	Track2	PCRW_TRACK3	Track3	PCRW_TRACK4	Track4	PCRW_TRACK5	Track5	PCRW_TRACK6	Track6
Value	Meaning														
PCRW_TRACK1	Track1														
PCRW_TRACK2	Track2														
PCRW_TRACK3	Track3														
PCRW_TRACK4	Track4														
PCRW_TRACK5	Track5														
PCRW_TRACK6	Track6														
Errors	A UpoException may be thrown when this property is accessed. For further information, see “Errors” on page 15.														

CapTracksToWrite Property

- Syntax** **CapTracksToWrite:** *int32* { read-only, access after open }
- 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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CardState Property

- Syntax** **CardState:** *int32* { read-only, access after open }
- 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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

- See Also** **CapCardEntranceSensor** Property.

CharacterSet Property

Syntax **CharacterSet: *int32* { read-write, access after open-claim-enable }**

Remarks The character set for printing characters.

Value	Meaning
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 UpoException 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:

Value	Meaning
E_ILLEGAL	An invalid property value was specified.

See Also **CharacterSetList** Property.

CharacterSetList Property

Syntax **CharacterSetList: *string* { read-only, access after open }**

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

See Also **CharacterSet** Property.

FontTypefaceList Property

Syntax	FontTypefaceList: <i>string</i> { read-only, access after open }
Remarks	<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 (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 open method.</p>
Errors	A UpoException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	“Data Characters and Escape Sequences” on page 247.

LineChars Property

Syntax	LineChars: <i>int32</i> { read-write, access after open-claim-enable }				
Remarks	<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 LineChars may also update LineWidth, LineHeight, and LineSpacing, since the character pitch or font may be changed.</p> <p>The value of LineChars is initialized to the Point Card Reader Writer’s default line character width when the device is first enabled following the open method.</p>				
Errors	<p>A UpoException 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>Value</th><th>Meaning</th></tr></thead><tbody><tr><td>E_ILLEGAL</td><td>An invalid line character width was specified.</td></tr></tbody></table>	Value	Meaning	E_ILLEGAL	An invalid line character width was specified.
Value	Meaning				
E_ILLEGAL	An invalid line character width was specified.				
See Also	LineCharsList Property.				

LineCharsList Property

Syntax	LineCharsList: <i>string</i> { read-only, access after open }
Remarks	<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 open method.</p>
Errors	A UpoException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	LineChars Property.

LineHeight Property

Syntax	LineHeight: <i>int32</i> { read-write, access after open-claim-enable }
Remarks	<p>The Point Card Reader Writer print line height. If CapPrintMode is true, this is expressed in the unit of measure given by MapMode.</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 LineChars is changed, LineHeight is updated to the default line height for the selected width.</p> <p>The value of LineHeight is initialized to the Point Card Reader Writer’s default line height when the device is first enabled following the open method.</p>
Errors	A UpoException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

LineSpacing Property

Syntax	LineSpacing: <i>int32</i> { read-write, access after open-claim-enable }
Remarks	<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 CapPrintMode is true, line spacing is expressed in the unit of measure given by MapMode.</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 LineChars or LineHeight is changed, LineSpacing is updated to the default line spacing for the selected width or height.</p> <p>The value of LineSpacing is initialized to the Point Card Reader Writer’s default line spacing when the device is first enabled following the open method.</p>
Errors	A UpoException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

LineWidth Property

Syntax	LineWidth: <i>int32</i> { read-only, access after open-claim-enable }
Remarks	<p>The width of a line of LineChars characters. If CapPrintMode is true, expressed in the unit of measure given by MapMode.</p> <p>Setting LineChars may also update LineWidth.</p> <p>The value of LineWidth is initialized to the Point Card Reader Writer's default line width when the device is first enabled following the open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see "Errors" on page 15.

MapMode Property

Syntax	MapMode: <i>int32</i> { read-write, access after open-claim-enable }										
Remarks	<p>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:</p> <table> <tr> <th>Value</th><th>Meaning</th></tr> <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> </table> <p>Setting MapMode may also change LineHeight, LineSpacing, and LineWidth. The value of MapMode is initialized to PCRW_MM_DOTS when the device is first enabled following the open method.</p>	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.										
Errors	<p>A UposException 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> <tr> <th>Value</th><th>Meaning</th></tr> <tr> <td>E_ILLEGAL</td><td>An invalid mapping mode value was specified.</td></tr> </table>	Value	Meaning	E_ILLEGAL	An invalid mapping mode value was specified.						
Value	Meaning										
E_ILLEGAL	An invalid mapping mode value was specified.										

MaxLine Property

Syntax	MaxLine: <i>int32</i> { read-only, access after open-claim-enable }
Remarks	<p>When the CapPrintMode property is false, MaxLine contains the maximum printable line number.</p> <p>In the case where there is a double-high character in the same line, this is dependent upon the capability of the device.</p> <p>When the LineHeight property and/or the LineSpacing property change, the MaxLine property may be changed.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see "Errors" on page 15.
See Also	LineHeight Property.

PrintHeight Property

- Syntax** **PrintHeight: *int32* { read-only, access after open-claim-enable }**
- 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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
- See Also** **CapPrintMode** Property, **MapMode** Property.

ReadState1 Property

- Syntax** **ReadState1: *int32* { read-only, access after open }**
- 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:

Value	Meaning
SUCCESS	Successful read of the data.
EPCRW_START	It is a start sentinel error.
EPCRW_END	It is a end sentinel error.
EPCRW_PARITY	It is a parity error.
EPCRW_ENCODE	There is no encoding.
EPCRW_LRC	It is a LRC error.
EPCRW_VERIFY	It is a verify error.
E_FAILURE	It is other error.

- Errors** A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
- See Also** **ReadState2** Property.

ReadState2 Property

Syntax **ReadState2: *int32* { read-only, access after open }**

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.

Value	Meaning
SUCCESS	Successful read of the data.
EPCRW_START	It is a start sentinel error.
EPCRW_END	It is a end sentinel error.
EPCRW_PARITY	It is a parity error.
EPCRW_ENCODE	There is no encoding.
EPCRW_LRC	It is a LRC error.
EPCRW_VERIFY	It is a verify error.
E_FAILURE	It is other error.

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

See Also **ReadState1** Property.

RecvLength1 Property

Syntax **RecvLength1: *int32* { read-only, access after open-claim-enable }**

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

See Also **CapTracksToRead** property, **TracksToRead** property, **RecvLength2** Property.

RecvLength2 Property

Syntax **RecvLength2: *int32* { read-only, access after open-claim-enable }**

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

See Also **CapTracksToRead** property, **TracksToRead** property, **RecvLength1** Property.

SidewaysMaxChars Property

Syntax	SidewaysMaxChars: <i>int32</i> { read-only, access after open-claim-enable }
Remarks	<p>Holds the maximum number of characters that may be printed on each line in sideways mode.</p> <p>If the capabilities CapLeft90 and CapRight90 are both false, then SidewaysMaxChars is zero.</p> <p>Changing the properties LineHeight, LineSpacing, and LineChars may cause this property to change.</p> <p>This property is initialized when the device is first enabled following the open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	SidewaysMaxLines Property.

SidewaysMaxLines Property

Syntax	SidewaysMaxLines: <i>int32</i> { read-only, access after open-claim-enable }
Remarks	<p>Holds the maximum number of lines that may be printed in sideways mode.</p> <p>If the capabilities CapLeft90 and CapRight90 are both false, then SidewaysMaxLines is zero.</p> <p>Changing the properties LineHeight, LineSpacing, and LineChars may cause this property to change.</p> <p>This property is initialized when the device is first enabled following the open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	SidewaysMaxChars Property.

TracksToRead Property

Syntax	TracksToRead: <i>int32</i> { read-write, access after open-claim-enable }						
Remarks	<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 CapTracksToRead 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 open method.</p>						
Errors	<p>A UposException 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> <tr> <th>Value</th><th>Meaning</th></tr> <tr> <td>E_BUSY</td><td>This operation cannot be performed because asynchronous output is in progress.</td></tr> <tr> <td>E_ILLEGAL</td><td>An illegal track was defined. The track is not available for reading. Refer to CapTracksToRead.</td></tr> </table>	Value	Meaning	E_BUSY	This operation cannot be performed because asynchronous output is in progress.	E_ILLEGAL	An illegal track was defined. The track is not available for reading. Refer to CapTracksToRead .
Value	Meaning						
E_BUSY	This operation cannot be performed because asynchronous output is in progress.						
E_ILLEGAL	An illegal track was defined. The track is not available for reading. Refer to CapTracksToRead .						
See Also	CapTracksToRead Property.						

TracksToWrite Property

Syntax	TracksToWrite: <i>int32</i> { read-write, access after open-claim-enable }						
Remarks	<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 CapTracksToWrite 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 open method.</p>						
Errors	<p>A UposException 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> <tr> <th>Value</th><th>Meaning</th></tr> <tr> <td>E_BUSY</td><td>This operation cannot be performed because asynchronous output is in progress.</td></tr> <tr> <td>E_ILLEGAL</td><td>An illegal track was defined. The track is not available for writing. Refer to CapTracksToWrite.</td></tr> </table>	Value	Meaning	E_BUSY	This operation cannot be performed because asynchronous output is in progress.	E_ILLEGAL	An illegal track was defined. The track is not available for writing. Refer to CapTracksToWrite .
Value	Meaning						
E_BUSY	This operation cannot be performed because asynchronous output is in progress.						
E_ILLEGAL	An illegal track was defined. The track is not available for writing. Refer to CapTracksToWrite .						
See Also	CapTracksToWrite Property, printWrite Method.						

Track1Data Property

Syntax	Track1Data: <i>binary</i> { read-only, access after open }
Remarks	<p>Contains the track 1 data from the point card.</p> <p>This property contains track data between but not including the start and end sentinels.</p> <p>An empty string indicates that the track was not accessible.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

Track2Data Property

Syntax	Track2Data: <i>binary</i> { read-only, access after open }
Remarks	<p>Contains the track 2 data from the point card.</p> <p>This property contains track data between but not including the start and end sentinels.</p> <p>An empty string indicates that the track was not accessible.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

Track3Data Property

Syntax	Track3Data: <i>binary</i> { read-only, access after open }
Remarks	<p>Contains the track 3 data from the point card.</p> <p>This property contains track data between but not including the start and end sentinels.</p> <p>An empty string indicates that the track was not accessible.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

Track4Data Property

Syntax	Track4Data: <i>binary</i> { read-only, access after open }
Remarks	<p>Contains the track 4 data from the point card.</p> <p>This property contains track data between but not including the start and end sentinels.</p> <p>An empty string indicates that the track was not accessible.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

Track5Data Property

Syntax	Track5Data: <i>binary</i> { read-only, access after open }
Remarks	<p>Contains the track 5 data from the point card.</p> <p>This property contains track data between but not including the start and end sentinels.</p> <p>An empty string indicates that the track was not accessible.</p>
Errors	A UpoException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

Track6Data Property

Syntax	Track6Data: <i>binary</i> { read-only, access after open }
Remarks	<p>Contains the track 6 data from the point card.</p> <p>This property contains track data between but not including the start and end sentinels.</p> <p>An empty string indicates that the track was not accessible.</p>
Errors	A UpoException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

WriteState1 Property

Syntax **WriteState1: *int32* { read-only, access after open }**

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.

Value	Meaning
SUCCESS	Successful write of the data.
EPCRW_START	It is a start sentinel error.
EPCRW_END	It is a end sentinel error.
EPCRW_PARITY	It is a parity error.
EPCRW_ENCODE	There is not encoding.
EPCRW_LRC	It is a LRC error.
EPCRW_VERIFY	It is a verify error.
E_FAILURE	It is other error.

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

See Also **WriteState2** Property.

WriteState2 Property

Syntax **WriteState2: *int32* { read-only, access after open }**

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
SUCCESS	Successful write of the data.
EPCRW_START	It is a start sentinel error.
EPCRW_END	It is a end sentinel error.
EPCRW_PARITY	It is a parity error.
EPCRW_ENCODE	There is not encoding.
EPCRW_LRC	It is a LRC error.
EPCRW_VERIFY	It is a verify error.
E_FAILURE	It is other error.

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

See Also **WriteState1** Property.

Write1Data Property

Syntax **Write1Data: *binary* { read-write, access after open }**

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

Write2Data Property

Syntax	Write2Data: <i>binary</i> { read-write, access after open }
Remarks	The printWrite 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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

Write3Data Property

Syntax	Write3Data: <i>binary</i> { read-write, access after open }
Remarks	The printWrite 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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

Write4Data Property

Syntax	Write4Data: <i>binary</i> { read-write, access after open }
Remarks	The printWrite 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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

Write5Data Property

Syntax	Write5Data: <i>binary</i> { read-write, access after open }
Remarks	The printWrite 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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

Write6Data Property

Syntax	Write6Data: <i>binary</i> { read-write, access after open }
Remarks	The printWrite 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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

Methods (UML operations)

beginInsertion Method

Syntax **beginInsertion (timeout: *int32*):**
 void { raises-exception, use after open-claim-enable }

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 UposException 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
E_BUSY	This operation cannot be performed because asynchronous output is in progress.
E_ILLEGAL	The Point Card Reader Writer does not exist or an invalid <i>timeout</i> parameter was specified.
E_TIMEOUT	The specified time has elapsed without the point card being properly inserted.
E_EXTENDED	Refer to the definitions for <i>ErrorCodeExtended</i> in the Events section “ErrorEvent” on page 278.

See Also **endInsertion** Method, **beginRemoval** Method, **endRemoval** Method.

beginRemoval Method

Syntax **beginRemoval (timeout: *int32*):**
 void { raises-exception, use after open-claim-enable }

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 UposException 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
E_BUSY	This operation cannot be performed because asynchronous output is in progress.
E_ILLEGAL	The Point Card Reader Writer does not exist or an invalid <i>timeout</i> parameter was specified.
E_TIMEOUT	The specified time has elapsed without the point card being properly inserted.
E_EXTENDED	Refer to the definitions for <i>ErrorCodeExtended</i> in the Events section “ErrorEvent” on page 278.

See Also **CapCardEntranceSensor** Property, **CardState** Property, **beginInsertion** Method, **endInsertion** Method, **endRemoval** Method.

cleanCard Method

Syntax	cleanCard(): void { raises-exception, use after open-claim-enable }						
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	<p>A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 15.</p> <p>Some possible values of the exception’s <i>ErrorCode</i> property are:</p> <table><tr><th>Value</th><th>Meaning</th></tr><tr><td>E_ILLEGAL</td><td>The Point Card Reader Writer does not exist or CapCleanCard is false.</td></tr><tr><td>E_EXTENDED</td><td>Refer to the definitions for <i>ErrorCodeExtended</i> in the Events section “ErrorEvent” on page 278.</td></tr></table>	Value	Meaning	E_ILLEGAL	The Point Card Reader Writer does not exist or CapCleanCard is false.	E_EXTENDED	Refer to the definitions for <i>ErrorCodeExtended</i> in the Events section “ErrorEvent” on page 278.
Value	Meaning						
E_ILLEGAL	The Point Card Reader Writer does not exist or CapCleanCard is false.						
E_EXTENDED	Refer to the definitions for <i>ErrorCodeExtended</i> in the Events section “ErrorEvent” on page 278.						
See Also	CapCleanCard Property.						

clearPrintWrite Method

Syntax **clearPrintWrite (kind: *int32*, hposition: *int32*, vposition: *int32*, width: *int32*, height: *int32*);**
 void { raises-exception, use after open-claim-enable }

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 MapMode units if CapPrintMode is true.
<i>vposition</i>	The vertical start position for erasing the printing area. The value is in MapMode units if CapPrintMode is true.
<i>width</i>	The width used for erasing the printing area. The value is in MapMode units if CapPrintMode is true.
<i>height</i>	The height used for erasing the printing area. The value is in MapMode units if CapPrintMode 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 *UposException* 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
E_BUSY	This operation cannot be performed because asynchronous output is in progress.
E_EXTENDED	Refer to the definitions for <i>ErrorCodeExtended</i> in the Events section “ErrorEvent” on page 278.

See Also **CapClearPrint** Property, **CapPrint** Property, **CapPrintMode** Property, **MapMode** Property.

endInsertion Method

Syntax	endInsertion (): void { raises-exception, use after open-claim-enable }								
Remarks	<p>Called to end point card insertion processing.</p> <p>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.</p> <p>This method is paired with the beginInsertion method for controlling point card insertion.</p>								
Errors	<p>A UpoException may be thrown when this method is invoked. For further information, see “Errors” on page 15.</p> <p>Some possible values of the exception’s <i>ErrorCode</i> property are:</p> <table> <tr> <th>Value</th><th>Meaning</th></tr> <tr> <td>E_ILLEGAL</td><td>The Point Card Reader Writer is not in point card insertion mode.</td></tr> <tr> <td>E_FAILURE</td><td>A card is not inserted in the Point Card Reader Writer.</td></tr> <tr> <td>E_EXTENDED</td><td>Refer to the definitions for <i>ErrorCodeExtended</i> in the Events section “ErrorEvent” on page 278.</td></tr> </table>	Value	Meaning	E_ILLEGAL	The Point Card Reader Writer is not in point card insertion mode.	E_FAILURE	A card is not inserted in the Point Card Reader Writer.	E_EXTENDED	Refer to the definitions for <i>ErrorCodeExtended</i> in the Events section “ErrorEvent” on page 278.
Value	Meaning								
E_ILLEGAL	The Point Card Reader Writer is not in point card insertion mode.								
E_FAILURE	A card is not inserted in the Point Card Reader Writer.								
E_EXTENDED	Refer to the definitions for <i>ErrorCodeExtended</i> in the Events section “ErrorEvent” on page 278.								
See Also	beginInsertion Method, beginRemoval Method, endRemoval Method.								

endRemoval Method

Syntax	endRemoval (): void { raises-exception, use after open-claim-enable }								
Remarks	<p>Called to end point card removal processing.</p> <p>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.</p> <p>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.</p>								
Errors	<p>A UpoException may be thrown when this method is invoked. For further information, see “Errors” on page 15.</p> <p>Some possible values of the exception’s <i>ErrorCode</i> property are:</p> <table> <tr> <th>Value</th><th>Meaning</th></tr> <tr> <td>E_ILLEGAL</td><td>The Point Card Reader Writer is not in point card removal mode.</td></tr> <tr> <td>E_FAILURE</td><td>There is a card in the Point Card Reader Writer.</td></tr> <tr> <td>E_EXTENDED</td><td>Refer to the definitions for <i>ErrorCodeExtended</i> in the Events section “ErrorEvent” on page 278.</td></tr> </table>	Value	Meaning	E_ILLEGAL	The Point Card Reader Writer is not in point card removal mode.	E_FAILURE	There is a card in the Point Card Reader Writer.	E_EXTENDED	Refer to the definitions for <i>ErrorCodeExtended</i> in the Events section “ErrorEvent” on page 278.
Value	Meaning								
E_ILLEGAL	The Point Card Reader Writer is not in point card removal mode.								
E_FAILURE	There is a card in the Point Card Reader Writer.								
E_EXTENDED	Refer to the definitions for <i>ErrorCodeExtended</i> in the Events section “ErrorEvent” on page 278.								
See Also	beginInsertion Method, beginRemoval Method, endInsertion Method.								

printWrite Method

Syntax **printWrite (kind: *int32*, hposition: *int32*, vposition: *int32*, data: *string*):**
 void { raises-exception, use after open-claim-enable }

Parameter	Description
<i>kind</i>	Designates the effect of the point card. 1: Print 2: Write 3: Print+Write
<i>hposition</i>	The horizontal start position for printing. The value is in MapMode units if CapPrintMode is true.
<i>vposition</i>	The vertical start position for printing. The value is in MapMode units if CapPrintMode is true.
<i>data</i>	The data to be printed. Any escape sequences in the data are dependent upon the capabilities of the device.

Remarks This method will either print the specified data on the printing area of the point card, write data from the **WriteXData** properties to the magnetic tracks, or both. In order to print on a point card, the **CapPrint** property must be true. In order to write the magnetic tracks on a point card, the **WriteXData** properties for each desired track must be set to the desired value, the **TracksToWrite** property must be set to a bitmask indicating which tracks to write (see **TracksToWrite** for a complete description) and the **CapTracksToWrite** property must indicate that each tracks specified in **TracksToWrite** is legal.

Errors A UposException 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
E_ILLEGAL	There is no card in the Point Card Reader Writer.
E_EXTENDED	Refer to the definitions for <i>ErrorCodeExtended</i> in the Events section “ErrorEvent” on page 278.

See Also **CapPrint** Property, **CapPrintMode** Property, **CapTracksToWrite** Property, **MapMode** Property, **TracksToWrite** Property, **WriteXData** Property.

rotatePrint Method

Syntax **rotatePrint (rotation: *int32*):**
 void { raises-exception, use after open-claim-enable }

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 UpoException 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
-------	---------

E_BUSY	This operation cannot be performed because asynchronous output is in progress.
--------	--

E_ILLEGAL	The Point Card Reader Writer does not support the specified rotation.
-----------	---

E_EXTENDED	Refer to the definitions for <i>ErrorCodeExtended</i> in the Events section "ErrorEvent" on page 278.
------------	---

See Also "Data Characters and Escape Sequences" on page 247, **printWrite** Method.

validateData Method

Syntax **validateData (data: *string*):**
 void { raises-exception, use after open-claim-enable }

Parameter	Description
<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 UposException 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
E_ILLEGAL	Some of the data is not precisely supported by the device, but the Control can select valid alternatives.
E_FAILURE	Some of the data is not supported. No alternatives can be selected.

Cases which cause *ErrorCode* of E_ILLEGAL:

Escape Sequence	Condition
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 E_FAILURE to be returned are:

Escape Sequence	Condition
(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 247, **printWrite** Method.

Events (UML Interfaces)

DataEvent

<< event >> **upos::events::DataEvent**
Status: *int32* { read-only }

Description Fired to present input data from the device to the application.

Attributes This event contains the following attribute:

Attributes	Type	Description
<i>Status</i>	<i>int32</i>	The <i>Status</i> parameter contains zero.

Remarks The point card data is placed in each property before this event is delivered.

DirectIOEvent

<< event >> **upos::events::DirectIOEvent**
EventNumber: *int32* { read-only }
Data: *int32* { read-write }
Obj: *object* { read-write }

Description 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.

Attributes This event contains the following attributes:

Attributes	Type	Description
<i>EventNumber</i>	<i>int32</i>	Event number whose specific values are assigned by the Service.
<i>Data</i>	<i>int32</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.

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 point card devices which may not have any knowledge of the Service's need for this event.

See Also "Events" on page 14, **directIO** Method.

ErrorEvent

```
<< event >> upos::events::ErrorEvent
  ErrorCode: int32 { read-only }
  ErrorCodeExtended: int32 { read-only }
  ErrorLocus: int32 { read-only }
  ErrorResponse: int32 { read-write }
```

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

Attributes This event contains the following properties:

Attributes	Type	Description
<i>ErrorCode</i>	<i>int32</i>	Result code causing the error event. See a list of Error Codes on page15.
<i>ErrorCodeExtended</i>	<i>int32</i>	Extended Error code causing the error event. If <i>ErrorCode</i> is E_EXTENDED, then see values below. Otherwise, it may contain a Service-specific value.
<i>ErrorLocus</i>	<i>int32</i>	Location of the error. See values below.
<i>ErrorResponse</i>	<i>int32</i>	Error response, whose default value may be overridden by the application. (i.e., this property is settable). See values below.

If *ErrorCode* is E_EXTENDED, then *ErrorCodeExtended* has one of the following values:

Value	Meaning
EPCRW_READ	There was a read error.
EPCRW_WRITE	There was a write error.
EPCRW_JAM	There was a card jam.
EPCRW_MOTOR	There was a conveyance motor error.
EPCRW_COVER	The conveyance motor cover was open.
EPCRW_PRINTER	The printer has an error.
EPCRW_RELEASE	There is a card remaining in the entrance.
EPCRW_DISPLAY	There was a display indicator error.
EPCRW_NOCARD	There is no card in the reader.

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

Value	Meaning
EL_OUTPUT	Error occurred while processing asynchronous output.
EL_INPUT	Error occurred while gathering or processing event-driven input. No input data is available.
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
ER_RETRY	Typically valid only when locus is EL_OUTPUT. Retry the asynchronous output. The error state is exited. May be valid when locus is EL_INPUT. Default when locus is EL_OUTPUT.
ER_CLEAR	Clear the asynchronous output or buffered input data. The error state is exited. Default when locus is EL_INPUT.
ER_CONTINUEINPUT	Use only when locus is EL_INPUT_DATA. Acknowledges the error and directs the Control to continue processing. The Control 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 EL_INPUT. Default when locus is 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

<< event >> **upos::events::OutputCompleteEvent**
OutputID: int32 { read-only }

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

Attributes This event contains the following attribute:

Attributes	Type	Description
<i>OutputID</i>	<i>int32</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 Service has confirmation that it was processed by the device successfully.

See Also "Device Output Models" on page 20.

StatusUpdateEvent

<< event >> **upos::events::StatusUpdateEvent**
Status: int32 { read-only }

Description Notifies the application that there is a change in the status of the PointCard device.

Attributes This event contains the following attribute:

Attributes	Type	Description
<i>Status</i>	<i>int32</i>	Indicates a change in the power status of the unit.

If *Status* parameter has one of the following values:

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.

Note that Release 1.3 added Power State Reporting with additional *Power reporting StatusUpdateEvent values*. See "StatusUpdateEvent" description on page 56.

Remarks Fired when the entrance sensor status of the Point Card Reader Writer changes. If the capability **CapCardEntranceSensor** is false, then the device does not support status reporting, and this event will never be fired to report card insertion state changes.

See Also "Events" on page 14, **CapCardEntranceSensor** Property.

CHAPTER 15

POS Keyboard

This Chapter defines the POS Keyboard device category.

General Information

The POS Keyboard programmatic name is “POSKeyboard”.

This chapter is grandfathered in based on the OPOS and JavaPOS Version 1.4 specifications.

POS Power

This Chapter defines the POS Power device category.

Summary

Properties (UML attributes)

<i>Common</i>	<i>Type</i>	<i>Mutability</i>	<i>Version^a</i>	<i>May Use After</i>
AutoDisable:	<i>boolean</i>	{ read-write }	1.5	Not Supported
CapPowerReporting:	<i>int32</i>	{ read-only }	1.5	open
CheckHealthText:	<i>string</i>	{ read-only }	1.5	open
Claimed:	<i>boolean</i>	{ read-only }	1.5	open
DataCount:	<i>int32</i>	{ read-only }	1.5	Not Supported
DataEventEnabled:	<i>boolean</i>	{ read-write }	1.5	Not Supported
DeviceEnabled:	<i>boolean</i>	{ read-write }	1.5	open
FreezeEvents:	<i>boolean</i>	{ read-write }	1.5	open
OutputID:	<i>int32</i>	{ read-only }	1.5	Not Supported
PowerNotify:	<i>int32</i>	{ read-write }	1.5	open
PowerState:	<i>int32</i>	{ read-only }	1.5	open
State:	<i>int32</i>	{ read-only }	1.5	--
DeviceControlDescription:	<i>string</i>	{ read-only }	1.5	--
DeviceControlVersion:	<i>int32</i>	{ read-only }	1.5	--
DeviceServiceDescription:	<i>string</i>	{ read-only }	1.5	open
DeviceServiceVersion:	<i>int32</i>	{ read-only }	1.5	open
PhysicalDeviceDescription:	<i>string</i>	{ read-only }	1.5	open
PhysicalDeviceName:	<i>string</i>	{ read-only }	1.5	open

- a. The version representation provides the mechanism for recognizing when a change occurs to a property, method or event. This POSPower definition was introduced in UnifiedPOS version 1.5.

Properties (Continued)

<i>Specific</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>	<i>May Use After</i>
CapFanAlarm:	<i>boolean</i>	{ read-only }	1.5	open
CapHeatAlarm:	<i>boolean</i>	{ read-only }	1.5	open
CapQuickCharge:	<i>boolean</i>	{ read-only }	1.5	open
CapShutdownPOS:	<i>boolean</i>	{ read-only }	1.5	open
CapUPSChargeState:	<i>int32</i>	{ read-only }	1.5	open
EnforcedShutdownDelayTime:	<i>int32</i>	{ read-write }	1.5	open
PowerFailDelayTime:	<i>int32</i>	{ read-only }	1.5	open
QuickChargeMode:	<i>boolean</i>	{ read-only }	1.5	open
QuickChargeTime:	<i>int32</i>	{ read-only }	1.5	open
UPSChargeState:	<i>int32</i>	{ read-only }	1.5	open & enable

Methods (UML operations)**Common***Name***open (logicalDeviceName: *string*):**

void { raises exception }

close () :

void { raises exception, use after open }

claim (timeout: *int32*):

void { raises exception, use after open }

release ():

void { raises exception, use after open, claim }

checkHealth (level: *int32*):

void { raises exception, use after open, enable }

clearInput ():

void { }

Not supported**clearOutput ():**

void { }

Not supported**directIO (command: *int32*, inout data: *int32*, inout obj: *object*):**

void { raises exception, use after open }

Specific*Name***shutdownPOS ():**

void { raises exception, use after open, enable }

Events (UML interfaces)

<i>Name</i>	<i>Type</i>	<i>Mutability</i>
upos::events::DirectIOEvent		
EventNumber:	<i>int32</i>	{ read-only }
Data:	<i>int32</i>	{ read-write }
Obj:	<i>object</i>	{ read-write }
upos::events::StatusUpdateEvent		
Status:	<i>int32</i>	{ read-only }

General Information

The POS Power programmatic name is “POSPower”.

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.

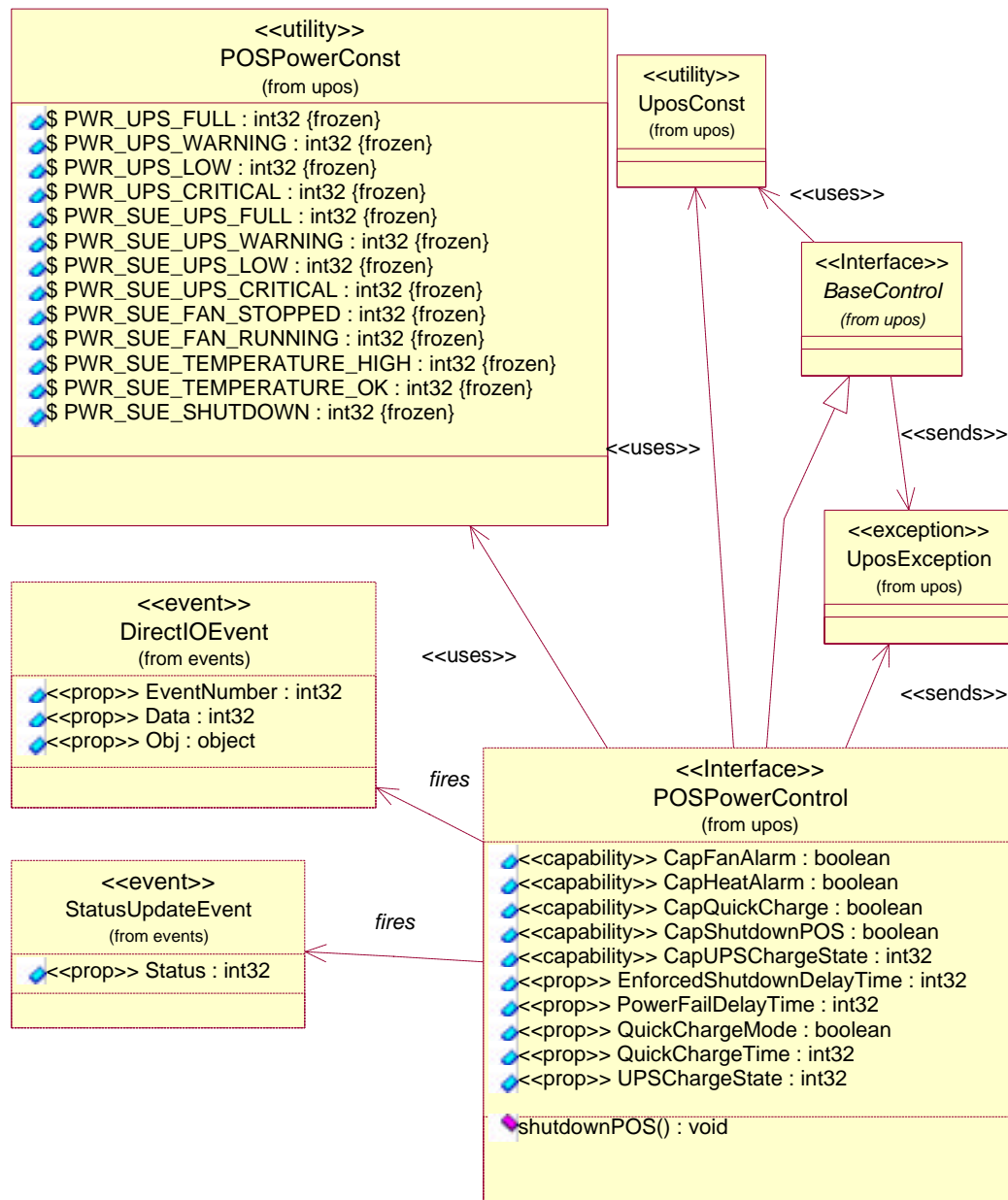
Power reporting only occurs while the device is open, enabled and power notification is switched on.

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.

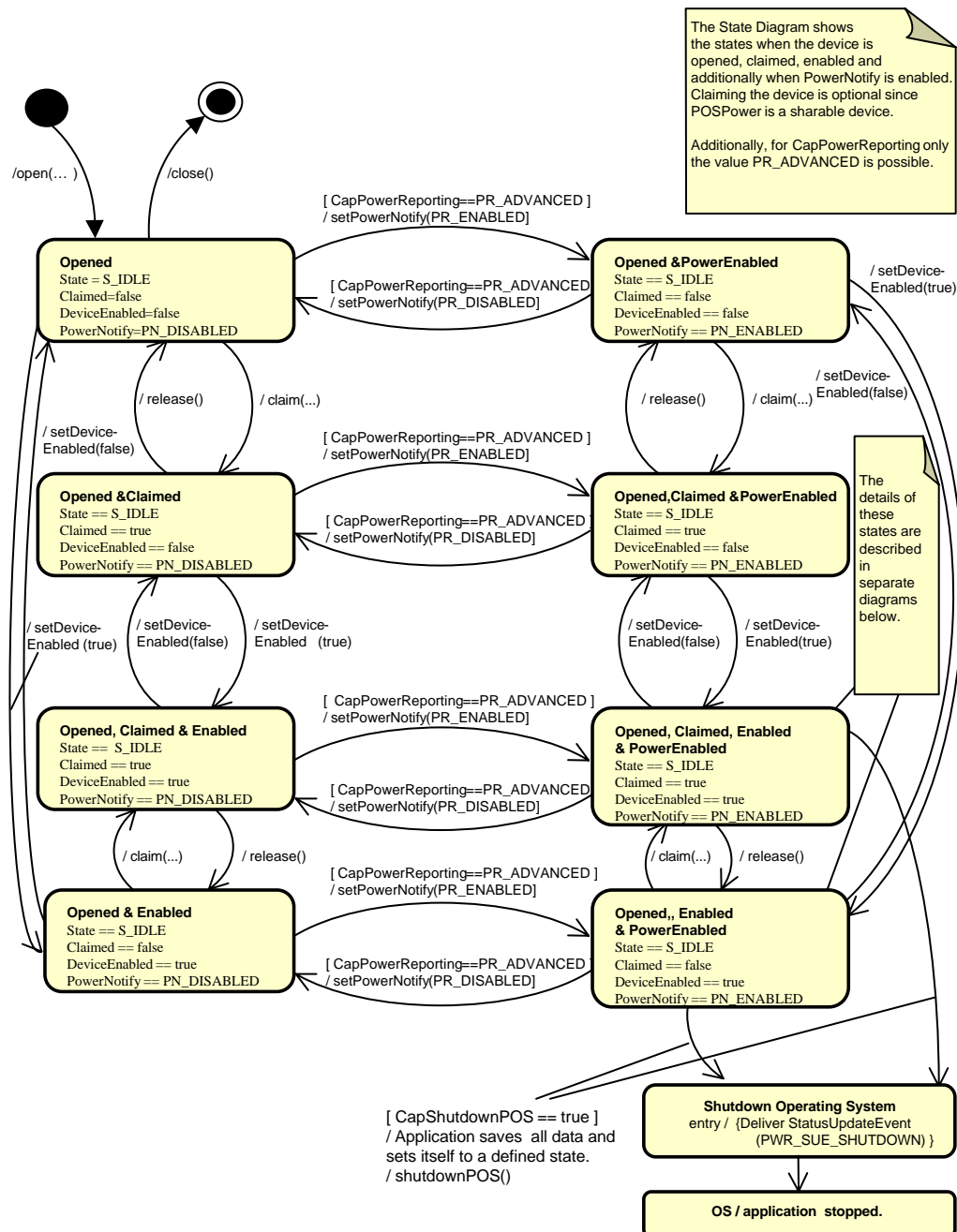
POSPower Class Diagram

The following diagram shows the relationships between the POSPower classes.



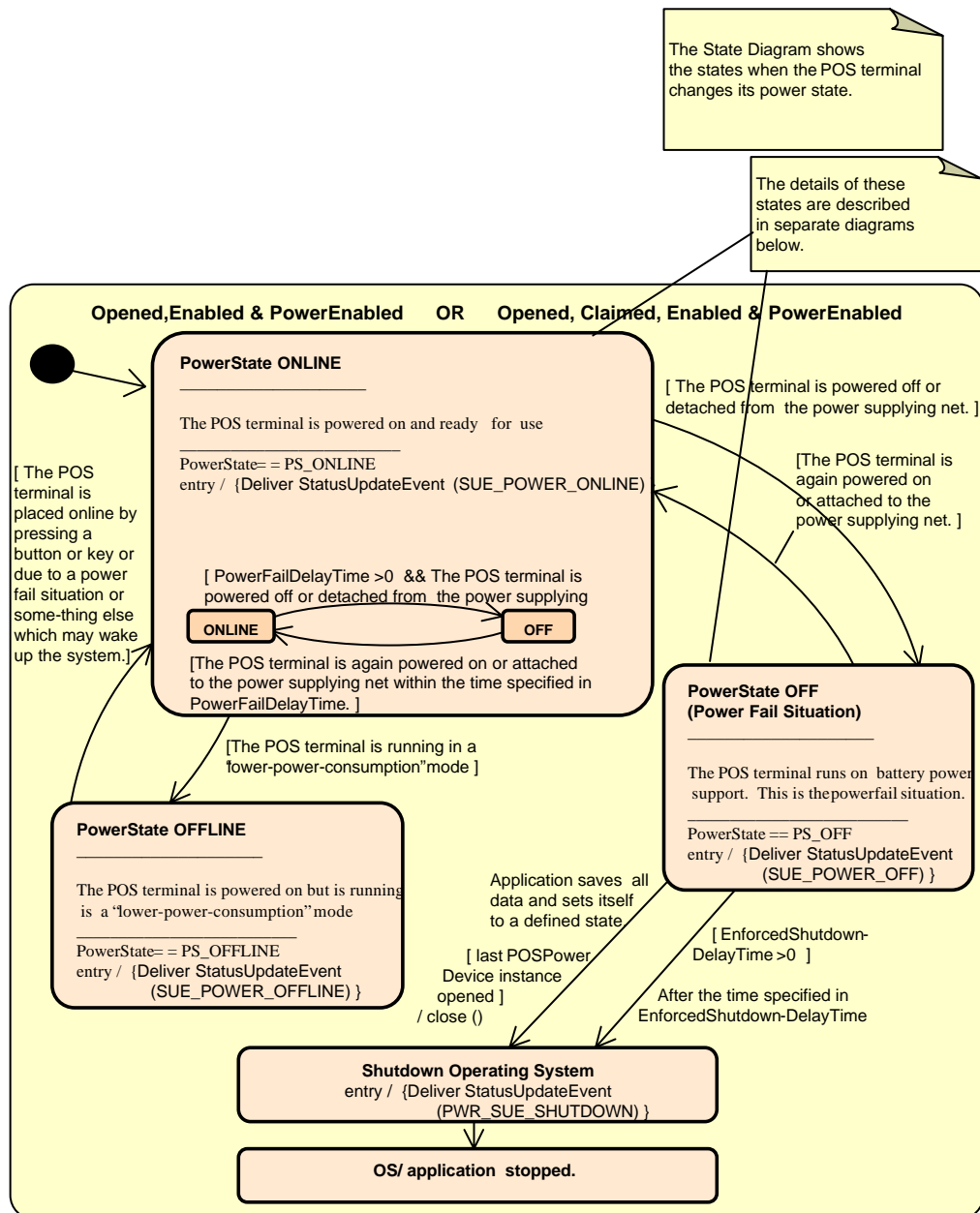
POSPower State Diagram

The following state diagram depicts the POSPower Control device model.



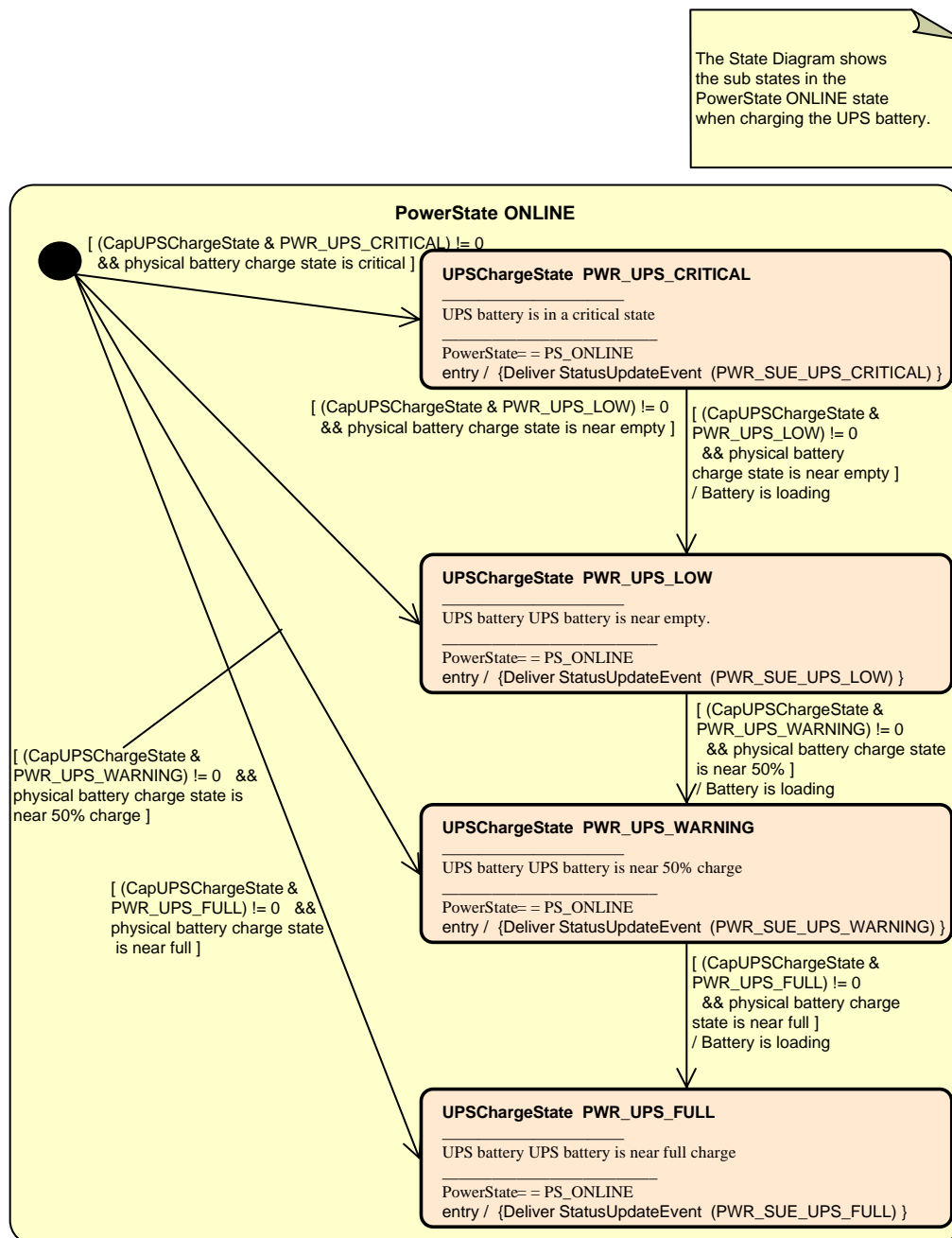
POSPower PowerState Diagram - part 1

The following state diagram depicts the POSPower Power States.



POSPower PowerState Diagram - part 2

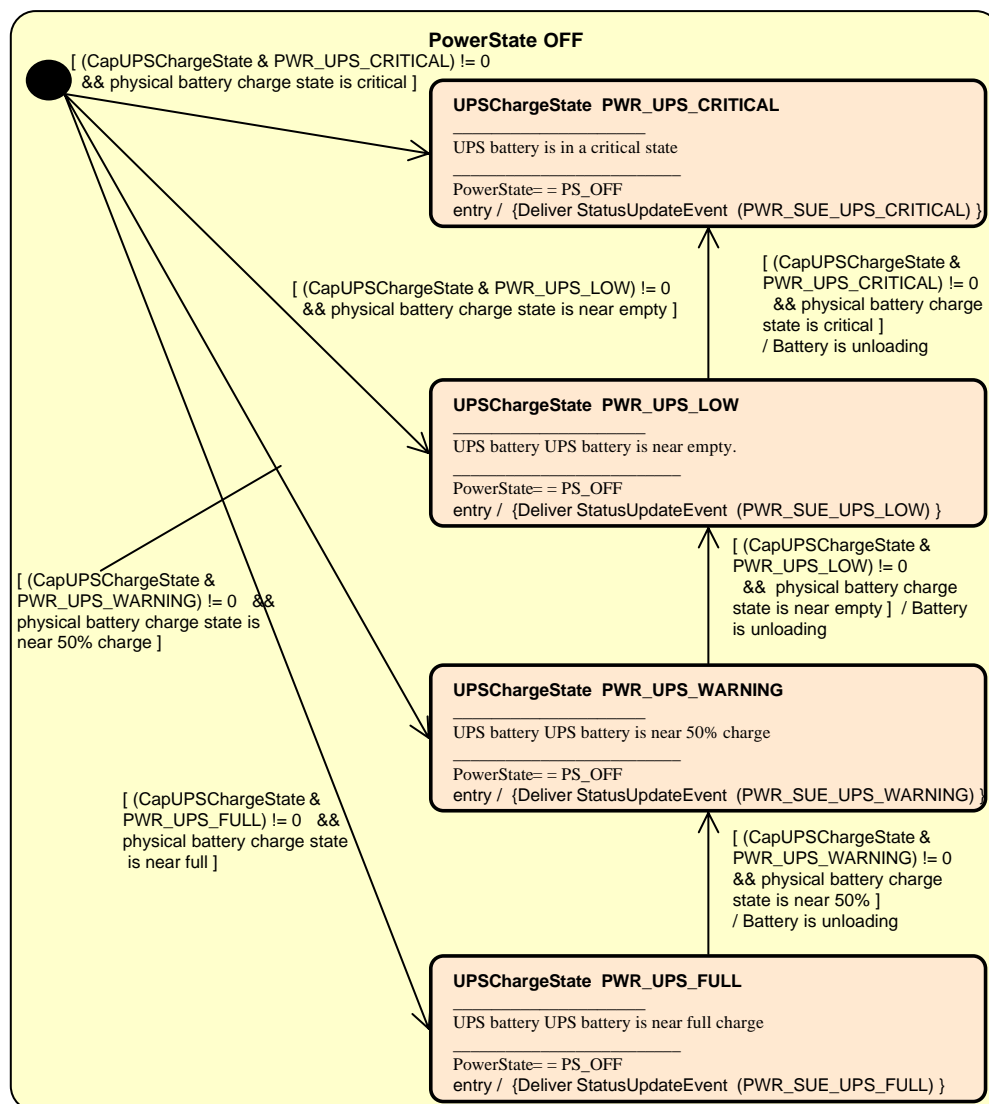
The following state diagram depicts the POSPower PowerState ONLINE.



POSPower PowerState Diagram - part 3

The following state diagram depicts the POSPower PowerState OFF.

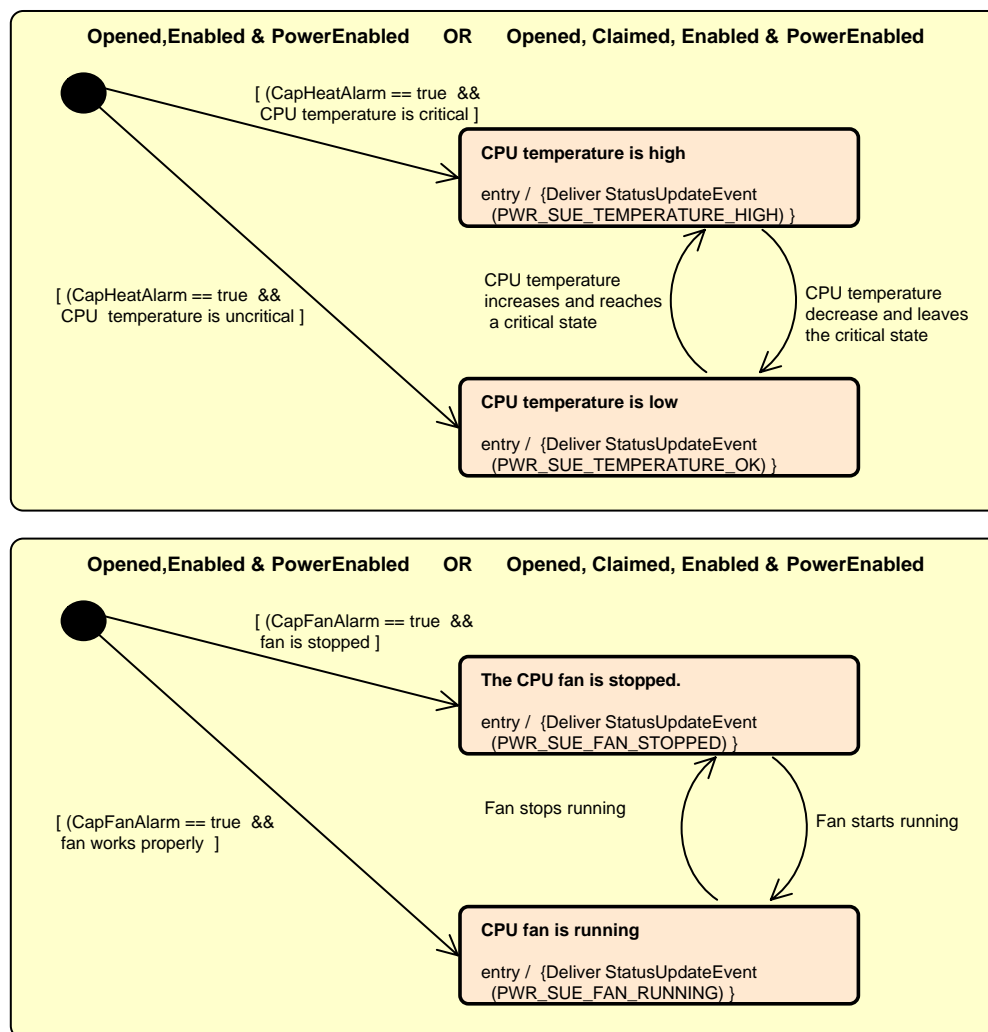
The State Diagram shows the sub states in the PowerState OFF state when unloading the UPS battery.



POSPower State chart Diagram for fan and temperature

The following state diagram depicts the handling of fan and temperature alarms.

The State Diagrams shows the states for handling high CPU temperature and stopped CPU fan.



Properties (UML attributes)

CapFanAlarm Property

Syntax	CapFanAlarm: <i>boolean</i> { read-only, access after open }
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 open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapHeatAlarm Property

Syntax	CapHeatAlarm: <i>boolean</i> { read-only, access after open }
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 open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapQuickCharge Property

Syntax	CapQuickCharge: <i>boolean</i> { read-only, access after open }
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 open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	QuickChargeMode Property, QuickChargeTime Property.

CapShutdownPOS Property

Syntax	CapShutdownPOS: <i>boolean</i> { read-only, access after open }
Remarks	If true the device is able to explicitly shut down the POS. Otherwise it is false. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	shutdownPOS Method.

CapUPSChargeState Property

Syntax	CapUPSChargeState: <i>int32</i> { read-only, access after open }										
Remarks	<p>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.</p> <table> <tr> <th>Value</th><th>Meaning</th></tr> <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> </table> <p>This property is initialized by the open method.</p>	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.										
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.										
See Also	UPSChargeState Property.										

EnforcedShutdownDelayTime Property

Syntax	EnforcedShutdownDelayTime: <i>int32</i> { read-write, access after open }
Remarks	<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 shutdownPOS method.</p> <p>Applications will be informed about an initiated automatic shutdown.</p> <p>This property is initialized by the open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	shutdownPOS Method.

PowerFailDelayTime Property

Syntax	PowerFailDelayTime: <i>int32</i> { read-only, access after open }
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 PowerFailDelayTime 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 open method.</p>
Errors	A <code>UposException</code> may be thrown when this property is accessed. For further information, see “Errors” on page 15.

QuickChargeMode Property

Syntax	QuickChargeMode: <i>boolean</i> { read-only, access after open }
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 CapQuickCharge is true.</p>
Errors	A <code>UposException</code> may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	CapQuickCharge Property, QuickChargeTime Property.

QuickChargeTime Property

Syntax	QuickChargeTime: <i>int32</i> { read-only, access after open }
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 CapQuickCharge is true.</p>
Errors	A <code>UposException</code> may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	CapQuickCharge Property, QuickChargeMode Property.

UPSChargeState Property

Syntax **UPSChargeState: *int32* { read-only, access after open, enable }**

Remarks This property holds the actual UPS charge state.

It has one of the following values:

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

See Also **CapUPSChargeState** Property.

Methods (UML operations)

shutdownPOS Method

Syntax	shutdownPOS (): void { raises exception, use after open, enable }				
Remarks	<p>Call to shut down the POS terminal. This method will always shut down the system independent of the system power state.</p> <p>If the POSPower is claimed, only the application which claimed the device is able to shut down the POS terminal.</p> <p>Applications will be informed about an initiated shutdown.</p> <p>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.</p> <p>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.</p>				
Errors	<p>A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 15</p> <p>Some possible values of the exception’s <i>ErrorCode</i> property are:</p> <table><tr><th>Value</th><th>Meaning</th></tr><tr><td>E_ILLEGAL</td><td>This method is not supported (see the CapShutdownPOS property)</td></tr></table>	Value	Meaning	E_ILLEGAL	This method is not supported (see the CapShutdownPOS property)
Value	Meaning				
E_ILLEGAL	This method is not supported (see the CapShutdownPOS property)				
See Also	CapShutdownPOS Property, EnforcedShutdownDelayTime Property.				

Events (UML Interfaces)

DirectIOEvent

```
<< event >> upos::events::DirectIOEvent
    EventNumber: int32 { read-only }
    Data: int32 { read-write }
    Obj: object { read-write }
```

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>int32</i>	Event number whose specific values are assigned by the Device Service.
<i>Data</i>	<i>int32</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 14, **directIO** Method.

StatusUpdateEvent

<< event >> **upos::events::StatusUpdateEvent**
Status: int32 { read-only }

Description Delivered when **UPSChargeState** changes or an alarm situation occurs.

Attributes This event contains the following attribute:

Attributes	Type	Description
<i>Status</i>	<i>int32</i>	See below.
The <i>Status</i> 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 CapUPSChargeState contains PWR_UPS_FULL.	
PWR_SUE_UPS_WARNING	UPS battery is near 50% charge. Can be returned if CapUPSChargeState 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 CapUPSChargeState contains PWR_UPS_LOW.	
PWR_SUE_UPS_CRITICAL	UPS is in critical state, and will in short time be disconnected. Can be returned if CapUPSChargeState contains PWR_UPS_CRITICAL.	
PWR_SUE_FAN_STOPPED	The CPU fan is stopped. Can be returned if CapFanAlarm is true.	
PWR_SUE_FAN_RUNNING	The CPU fan is running. Can be returned if CapFanAlarm is true.	
PWR_SUE_TEMPERATURE_HIGH	The CPU is running on high temperature. Can be returned if CapHeatAlarm is true.	
PWR_SUE_TEMPERATURE_OK	The CPU is running on normal temperature. Can be returned if CapHeatAlarm 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 56.

See Also **CapFanAlarm**, **CapHeatAlarm**, **CapUPSChargeState**, and **UPSChargeState** Properties.

POS Printer

This Chapter defines the POS Printer device category.

Summary

Properties (UML attributes)

<i>Common</i>	<i>Type</i>	<i>Mutability</i>	<i>Version^a</i>	<i>May Use After</i>
AutoDisable:	<i>boolean</i>	{ read-write }	1.3	Not Supported
CapPowerReporting:	<i>int32</i>	{ read-only }	1.3	open
CheckHealthText:	<i>string</i>	{ read-only }	1.3	open
Claimed:	<i>boolean</i>	{ read-only }	1.3	open
DataCount:	<i>int32</i>	{ read-only }	1.3	Not Supported
DataEventEnabled:	<i>boolean</i>	{ read-write }	1.3	Not Supported
DeviceEnabled:	<i>boolean</i>	{ read-write }	1.3	open & claim
FreezeEvents:	<i>boolean</i>	{ read-write }	1.3	open
OutputID:	<i>int32</i>	{ read-only }	1.3	open
PowerNotify:	<i>int32</i>	{ read-write }	1.3	open
PowerState:	<i>int32</i>	{ read-only }	1.3	open
State:	<i>int32</i>	{ read-only }	1.3	--
DeviceControlDescription:	<i>string</i>	{ read-only }	1.3	--
DeviceControlVersion:	<i>int32</i>	{ read-only }	1.3	--
DeviceServiceDescription:	<i>string</i>	{ read-only }	1.3	open
DeviceServiceVersion:	<i>int32</i>	{ read-only }	1.3	open
PhysicalDeviceDescription:	<i>string</i>	{ read-only }	1.3	open
PhysicalDeviceName:	<i>string</i>	{ read-only }	1.3	open

- a. The version representation provides the mechanism for recognizing when a change occurs to a property, method or event. This POS Printer definition was introduced in an existing standard and was not changed for the UnifiedPOS version 1.4.

Properties (Continued)

<i>Specific</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>	<i>May Use After</i>
CapCharacterSet:	<i>int32</i>	{ read-only }	1.3	open
CapConcurrentJrnRec:	<i>boolean</i>	{ read-only }	1.3	open
CapConcurrentJrnSlp:	<i>boolean</i>	{ read-only }	1.3	open
CapConcurrentRecSlp:	<i>boolean</i>	{ read-only }	1.3	open
CapCoverSensor:	<i>boolean</i>	{ read-only }	1.3	open
CapTransaction:	<i>boolean</i>	{ read-only }	1.3	open
CapJrnPresent:	<i>boolean</i>	{ read-only }	1.3	open
CapJrn2Color:	<i>boolean</i>	{ read-only }	1.3	open
CapJrnBold:	<i>boolean</i>	{ read-only }	1.3	open
CapJrnDhigh:	<i>boolean</i>	{ read-only }	1.3	open
CapJrnDwide:	<i>boolean</i>	{ read-only }	1.3	open
CapJrnDwideDhigh:	<i>boolean</i>	{ read-only }	1.3	open
CapJrnEmptySensor:	<i>boolean</i>	{ read-only }	1.3	open
CapJrnItalic:	<i>boolean</i>	{ read-only }	1.3	open
CapJrnNearEndSensor:	<i>boolean</i>	{ read-only }	1.3	open
CapJrnUnderline:	<i>boolean</i>	{ read-only }	1.3	open
CapJrnCartridgeSensor:	<i>int32</i>	{ read-only }	1.5	open
CapJrnColor:	<i>int32</i>	{ read-only }	1.5	open
CapRecPresent:	<i>boolean</i>	{ read-only }	1.3	open
CapRec2Color:	<i>boolean</i>	{ read-only }	1.3	open
CapRecBarCode:	<i>boolean</i>	{ read-only }	1.3	open
CapRecBitmap:	<i>boolean</i>	{ read-only }	1.3	open
CapRecBold:	<i>boolean</i>	{ read-only }	1.3	open
CapRecDhigh:	<i>boolean</i>	{ read-only }	1.3	open
CapRecDwide:	<i>boolean</i>	{ read-only }	1.3	open
CapRecDwideDhigh:	<i>boolean</i>	{ read-only }	1.3	open
CapRecEmptySensor:	<i>boolean</i>	{ read-only }	1.3	open
CapRecItalic:	<i>boolean</i>	{ read-only }	1.3	open
CapRecLeft90:	<i>boolean</i>	{ read-only }	1.3	open
CapRecNearEndSensor:	<i>boolean</i>	{ read-only }	1.3	open
CapRecPapercut:	<i>boolean</i>	{ read-only }	1.3	open
CapRecRight90:	<i>boolean</i>	{ read-only }	1.3	open
CapRecRotate180:	<i>boolean</i>	{ read-only }	1.3	open
CapRecStamp:	<i>boolean</i>	{ read-only }	1.3	open
CapRecUnderline:	<i>boolean</i>	{ read-only }	1.3	open
CapRecCartridgeSensor:	<i>int32</i>	{ read-only }	1.5	open
CapRecColor:	<i>int32</i>	{ read-only }	1.5	open
CapRecMarkFeed:	<i>int32</i>	{ read-only }	1.5	open

Properties (Continued)

<i>Specific (continued)</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>	<i>May Use After</i>
CapSlpPresent:	<i>boolean</i>	{ read-only }	1.3	open
CapSlpFullslip:	<i>boolean</i>	{ read-only }	1.3	open
CapSlp2Color:	<i>boolean</i>	{ read-only }	1.3	open
CapSlpBarCode:	<i>boolean</i>	{ read-only }	1.3	open
CapSlpBitmap:	<i>boolean</i>	{ read-only }	1.3	open
CapSlpBold:	<i>boolean</i>	{ read-only }	1.3	open
CapSlpDhigh:	<i>boolean</i>	{ read-only }	1.3	open
CapSlpDwide:	<i>boolean</i>	{ read-only }	1.3	open
CapSlpDwideDhigh:	<i>boolean</i>	{ read-only }	1.3	open
CapSlpEmptySensor:	<i>boolean</i>	{ read-only }	1.3	open
CapSlpItalic:	<i>boolean</i>	{ read-only }	1.3	open
CapSlpLeft90:	<i>boolean</i>	{ read-only }	1.3	open
CapSlpNearEndSensor:	<i>boolean</i>	{ read-only }	1.3	open
CapSlpRight90:	<i>boolean</i>	{ read-only }	1.3	open
CapSlpRotate180:	<i>boolean</i>	{ read-only }	1.3	open
CapSlpUnderline:	<i>boolean</i>	{ read-only }	1.3	open
CapSlpBothSidesPrint:	<i>boolean</i>	{ read-only }	1.5	open
CapSlpCartridgeSensor:	<i>int32</i>	{ read-only }	1.5	open
CapSlpColor:	<i>int32</i>	{ read-only }	1.5	open
AsyncMode:	<i>boolean</i>	{ read-write }	1.3	open
CartridgeNotify:	<i>int32</i>	{ read-write }	1.5	open
CharacterSet:	<i>int32</i>	{ read-write }	1.3	open, claim, & enable
CharacterSetList:	<i>string</i>	{ read-only }	1.3	open
CoverOpen:	<i>boolean</i>	{ read-only }	1.3	open, claim, & enable
ErrorLevel:	<i>int32</i>	{ read-only }	1.3	open
ErrorStation:	<i>int32</i>	{ read-only }	1.3	open
ErrorString:	<i>string</i>	{ read-only }	1.3	open
FontTypefaceList:	<i>string</i>	{ read-only }	1.3	open
FlagWhenIdle:	<i>boolean</i>	{ read-write }	1.3	open
MapMode:	<i>int32</i>	{ read-write }	1.3	open
RotateSpecial:	<i>int32</i>	{ read-write }	1.3	open
JrnLineChars:	<i>int32</i>	{ read-write }	1.3	open, claim, & enable
JrnLineCharsList:	<i>string</i>	{ read-only }	1.3	open
JrnLineHeight:	<i>int32</i>	{ read-write }	1.3	open, claim, & enable
JrnLineSpacing:	<i>int32</i>	{ read-write }	1.3	open, claim, & enable
JrnLineWidth:	<i>int32</i>	{ read-only }	1.3	open, claim, & enable

Properties (Continued)

<i>Specific (continued)</i>	<i>Type</i>	<i>Mutability</i>	<i>Version</i>	<i>May Use After</i>
JrnLetterQuality:	<i>boolean</i>	{ read-write }	1.3	open, claim, & enable
JrnEmpty:	<i>boolean</i>	{ read-only }	1.3	open, claim, & enable
JrnNearEnd:	<i>boolean</i>	{ read-only }	1.3	open, claim, & enable
JrnCartridgeState:	<i>int32</i>	{ read-only }	1.5	open, claim, & enable
JrnCurrentCartridge:	<i>int32</i>	(read-write }	1.5	open, claim, & enable
RecLineChars:	<i>int32</i>	{ read-write }	1.3	open, claim, & enable
RecLineCharsList:	<i>string</i>	{ read-only }	1.3	open
RecLineHeight:	<i>int32</i>	{ read-write }	1.3	open, claim, & enable
RecLineSpacing:	<i>int32</i>	{ read-write }	1.3	open, claim, & enable
RecLineWidth:	<i>int32</i>	{ read-only }	1.3	open, claim, & enable
RecLetterQuality:	<i>boolean</i>	{ read-write }	1.3	open, claim, & enable
RecEmpty:	<i>boolean</i>	{ read-only }	1.3	open, claim, & enable
RecNearEnd:	<i>boolean</i>	{ read-only }	1.3	open, claim, & enable
RecSidewaysMaxLines:	<i>int32</i>	{ read-only }	1.3	open, claim, & enable
RecSidewaysMaxChars:	<i>int32</i>	{ read-only }	1.3	open, claim, & enable
RecLinesToPaperCut:	<i>int32</i>	{ read-only }	1.3	open, claim, & enable
RecBarCodeRotationList:	<i>string</i>	{ read-only }	1.3	open
RecCartridgeState:	<i>int32</i>	{ read-only }	1.5	open, claim, & enable
RecCurrentCartridge:	<i>int32</i>	{ read-write }	1.5	open, claim, & enable
SlpLineChars:	<i>int32</i>	{ read-write }	1.3	open, claim, & enable
SlpLineCharsList:	<i>string</i>	{ read-only }	1.3	open
SlpLineHeight:	<i>int32</i>	{ read-write }	1.3	open, claim, & enable
SlpLineSpacing:	<i>int32</i>	{ read-write }	1.3	open, claim, & enable
SlpLineWidth:	<i>int32</i>	{ read-only }	1.3	open, claim, & enable
SlpLetterQuality:	<i>boolean</i>	{ read-write }	1.3	open, claim, & enable
SlpEmpty:	<i>boolean</i>	{ read-only }	1.3	open, claim, & enable
SlpNearEnd:	<i>boolean</i>	{ read-only }	1.3	open, claim, & enable
SlpSidewaysMaxLines:	<i>int32</i>	{ read-only }	1.3	open, claim, & enable
SlpSidewaysMaxChars:	<i>int32</i>	{ read-only }	1.3	open, claim, & enable
SlpMaxLines:	<i>int32</i>	{ read-only }	1.3	open, claim, & enable
SlpLinesNearEndToEnd:	<i>int32</i>	{ read-only }	1.3	open, claim, & enable
SlpBarCodeRotationList:	<i>string</i>	{ read-only }	1.3	open
SlpPrintSide:	<i>int32</i>	{ read-only }	1.5	open, claim, & enable
SlpCartridgeState:	<i>int32</i>	{ read-only }	1.5	open, claim, & enable
SlpCurrentCartridge:	<i>int32</i>	{ read-write }	1.5	open, claim, & enable

Methods (UML operations)

Common

Name

open (logicalDeviceName: *string*):
 void { raises exception }

close ():
 void { raises exception, use after open }

claim (timeout: *int32*):
 void { raises exception, use after open }

release ():
 void { raises exception, use after open, claim }

checkHealth (level: *int32*):
 void { raises exception, use after open, claim, enable }

clearInput (): **Not supported**
 void { raises exception, use after open }

clearOutput ():
 void { raises exception, use after open, claim }

directIO (command: *int32*, inout data: *int32*, inout obj: *object*):
 void { raises exception, use after open }

Specific

beginInsertion (timeout: *int32*):
 void { raises exception, use after open, claim, enable }

beginRemoval (timeout: *int32*):
 void { raises exception, use after open, claim, enable }

changePrintSide (side: *int32*):
 void { raises exception, use after open, claim, enable }

cutPaper (percentage: *int32*):
 void { raises exception, use after open, claim, enable }

endInsertion ():
 void { raises exception, use after open, claim, enable }

endRemoval ():
 void { raises exception, use after open, claim, enable }

markFeed (side: *int32*):
 void { raises exception, use after open, claim, enable }

**printBarcode (station: *int32*, data: *string*, symbology: *int32*, height: *int32*,
 width: *int32*, alignment: *int32*, textPosition: *int32*):**
 void { raises exception, use after open, claim, enable }

printBitmap (station: *int32*, fileName: *string*, width: *int32*, alignment: *int32*):
 void { raises exception, use after open, claim, enable }

```

printImmediate ( station: int32, data: string ):
    void { raises exception, use after open, claim, enable }

printNormal ( station: int32, data: string ):
    void { raises exception, use after open, claim, enable }

printTwoNormal ( station: int32, data1: string, data2: string ):
    void { raises exception, use after open, claim, enable }

rotatePrint ( station: int32, rotation: int32 ):
    void { raises exception, use after open, claim, enable }

setBitmap ( bitmapNumber: int32, station: int32, fileName: string, width:
    int32, alignment: int32 ):
    void { raises exception, use after open, claim, enable }

setLogo ( location: int32, data: string ):
    void { raises exception, use after open, claim, enable }

transactionPrint ( station: int32, control: int32 ):
    void { raises exception, use after open, claim, enable }

validateData ( station: int32, data: string ):
    void { raises exception, use after open, claim, enable }

```

Events (UML interfaces)

<i>Name</i>	<i>Type</i>	<i>Mutability</i>
upos::events::DirectIOEvent		
EventNumber:	<i>int32</i>	{ read-only }
Data:	<i>int32</i>	{ read-write }
Obj:	<i>object</i>	{ read-write }
upos::events::ErrorEvent		
ErrorCode:	<i>int32</i>	{ read-only }
ErrorCodeExtended:	<i>int32</i>	{ read-only }
ErrorLocus:	<i>int32</i>	{ read-only }
ErrorResponse	<i>int32</i>	{ read-write }
upos::events::OutputCompleteEvent		
OutputID:	<i>int32</i>	{ read-only }
upos::events::StatusUpdateEvent		
Status:	<i>int32</i>	{ read-only }

General Information

The POS Printer programmatic name is “POSPrinter”.

The POS Printer Service 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 Service 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.)

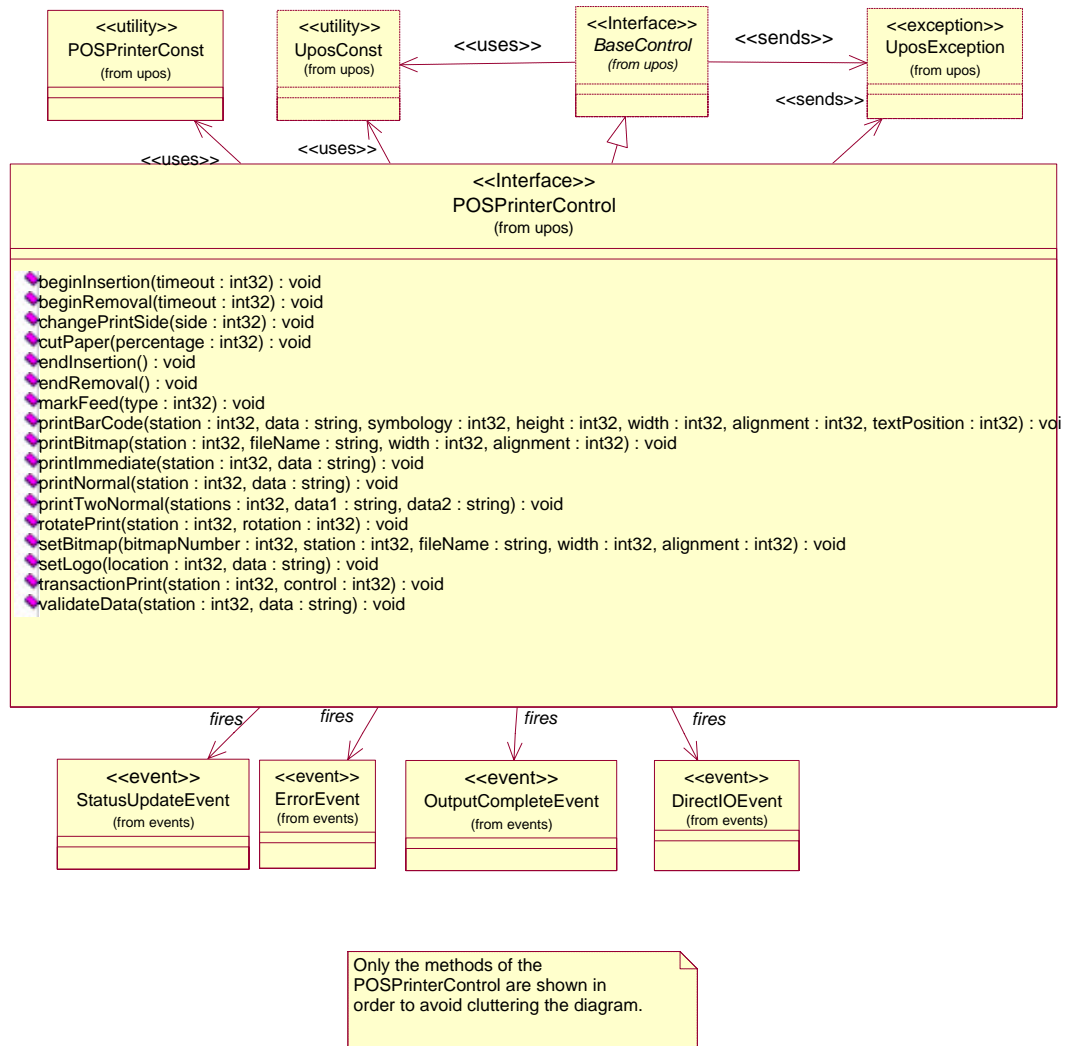
The POS printer may have several additional capabilities. See the capabilities properties for specific information.

The following capabilities are not addressed in this version of the specification. A 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.

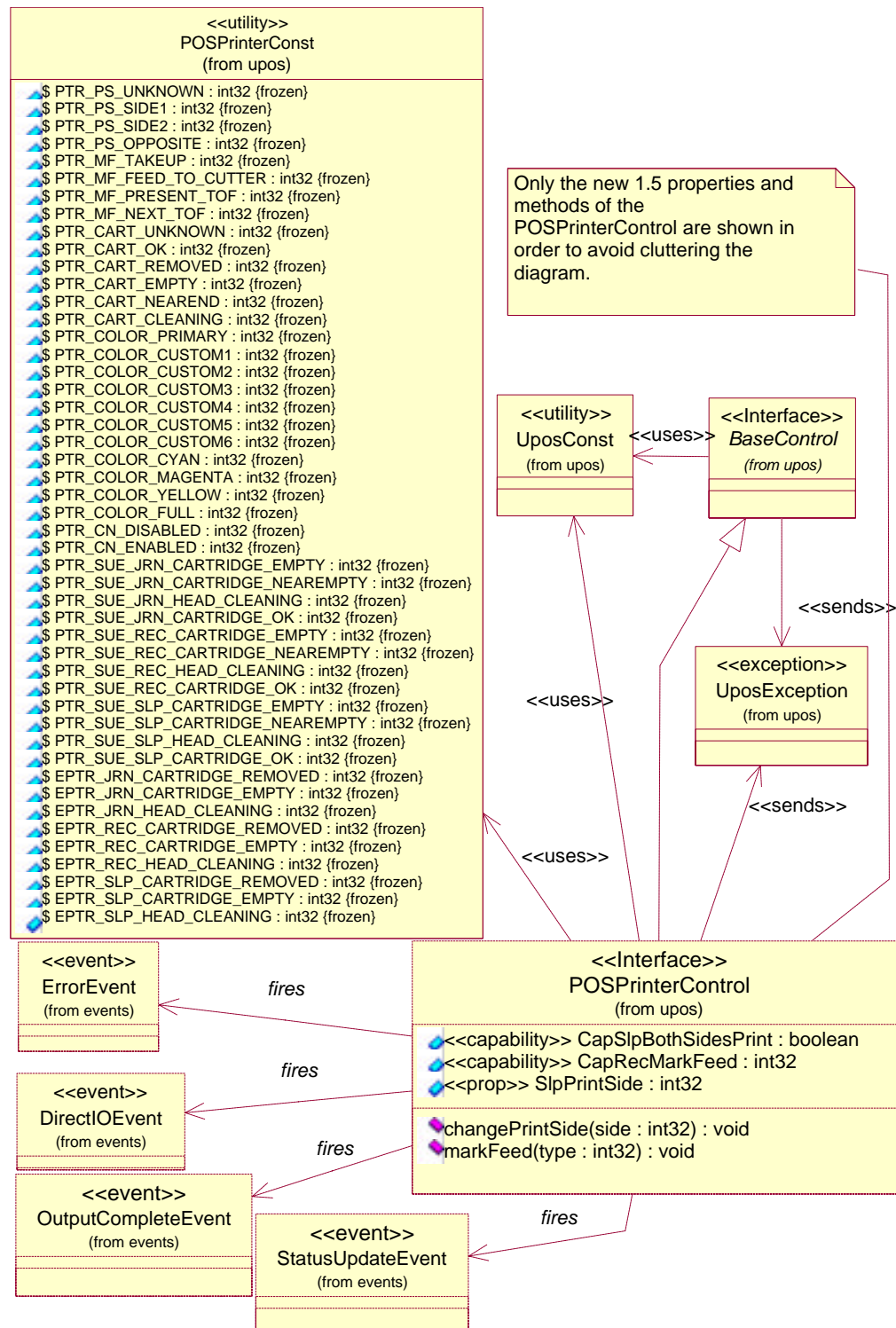
POS Printer Class Diagram

The following diagram shows the relationships between the POS Printer classes.



POS Printer Class Diagram - Version 1.5 Updates

The following diagram shows the relationships between the POS Printer classes that were updated in version 1.5 of the specification.



Model

The POS Printer follows the general device behavior 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: **cutPaper**, **markFeed**, **printBarCode**, **printBitmap**, **printNormal**, **printTwoNormal**, **rotatePrint**, and **transactionPrint**. When **AsyncMode** is false, then these methods are performed synchronously.
- When **AsyncMode** is true, then these methods operate as follows:
 - The Service 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 raise 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 raised 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 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 raised, 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 *E_EXTENDED* and then setting the associated *ErrorCodeExtended* to one of the following error conditions:
EPTR_JRN_EMPTY,
EPTR_REC_EMPTY,
EPTR_SLP_EMPTY,
EPTR_COVER_OPEN,
EPTR_JRN_CARTRIDGE_REMOVED,
EPTR_REC_CARTRIDGE_REMOVED,
EPTR_SLP_CARTRIDGE_REMOVED,
EPTR_JRN_CARTRIDGE_EMPTY,
EPTR_REC_CARTRIDGE_EMPTY,
EPTR_SLP_CARTRIDGE_EMPTY,
EPTR_JRN_HEAD_CLEANING,
EPTR_REC_HEAD_CLEANING, or
EPTR_SLP_HEAD_CLEANING.
- Other printer errors are reported by setting the exception's (or **ErrorEvent**'s) *ErrorCode* to *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:

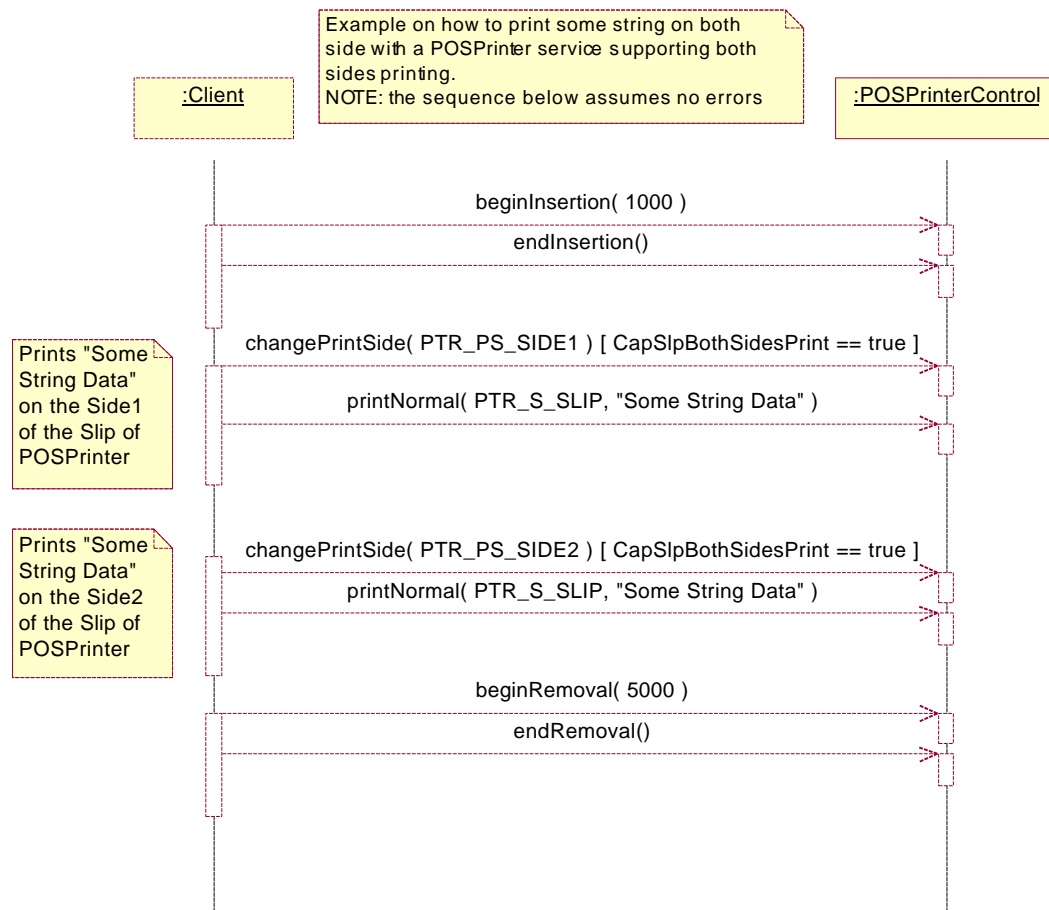
- The application must claim the device before enabling it.
- The application must claim and enable the device before accessing many printer-specific properties.
- The application must claim and enable the device before calling methods that manipulate the device.
- See the “Summary” table for precise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

POS Printer State Diagram

To be added in a future release.

"Both sides printing" sequence Diagram

The following sequence diagram is a representation of the typical usage of the "Both sides printing" feature.



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 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 RecLinesToPaperCut lines. The character '#' is defined by the "Paper cut" escape sequence.
Feed, Paper cut, and Stamp	ESC #sP	Cuts and stamps receipt paper, after feeding the paper by the RecLinesToPaperCut lines. The character '#' is defined by the "Paper cut" escape sequence.
Fire stamp	ESC sL	Fires the stamp solenoid, which usually contains a graphical store emblem.
Print bitmap	ESC #B	Prints the pre-stored bitmap. The character '#' is replaced by the bitmap number. See setBitmap 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 FontTypefaceList property. 2 = Select second typeface from the FontTypefaceList property. And so on.

Print Line Characteristics that are reset at the end of each print method or by a “Normal” sequence.

Name	Data	Remarks
Bold	ESC bC	Prints in bold or double-strike.
Underline	ESC #uC	Prints with underline. The character ‘#’ is replaced by an ASCII decimal string telling the 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. (See Note below.)
SubScript	ESC tbC	Prints SubScript characters. (See Note below.)
SuperScript	ESC tpC	Prints SuperScript characters. (See Note below.)
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 Version 1.5 and later.

POS Printer State Diagrams (Low Level)

Purpose:

The Low level state diagrams show a simplified, implementable flow of the POSPrinter.

They are intended to be used by Device service implementers as an example of how a Service may be designed. It uses multiple threads of execution to separate initiation of requests (via the POSPrinter APIs) with their processing and event delivery.

They are also intended to be used by application developers to show more details on processing of their API calls than can be given in the high level state diagram.

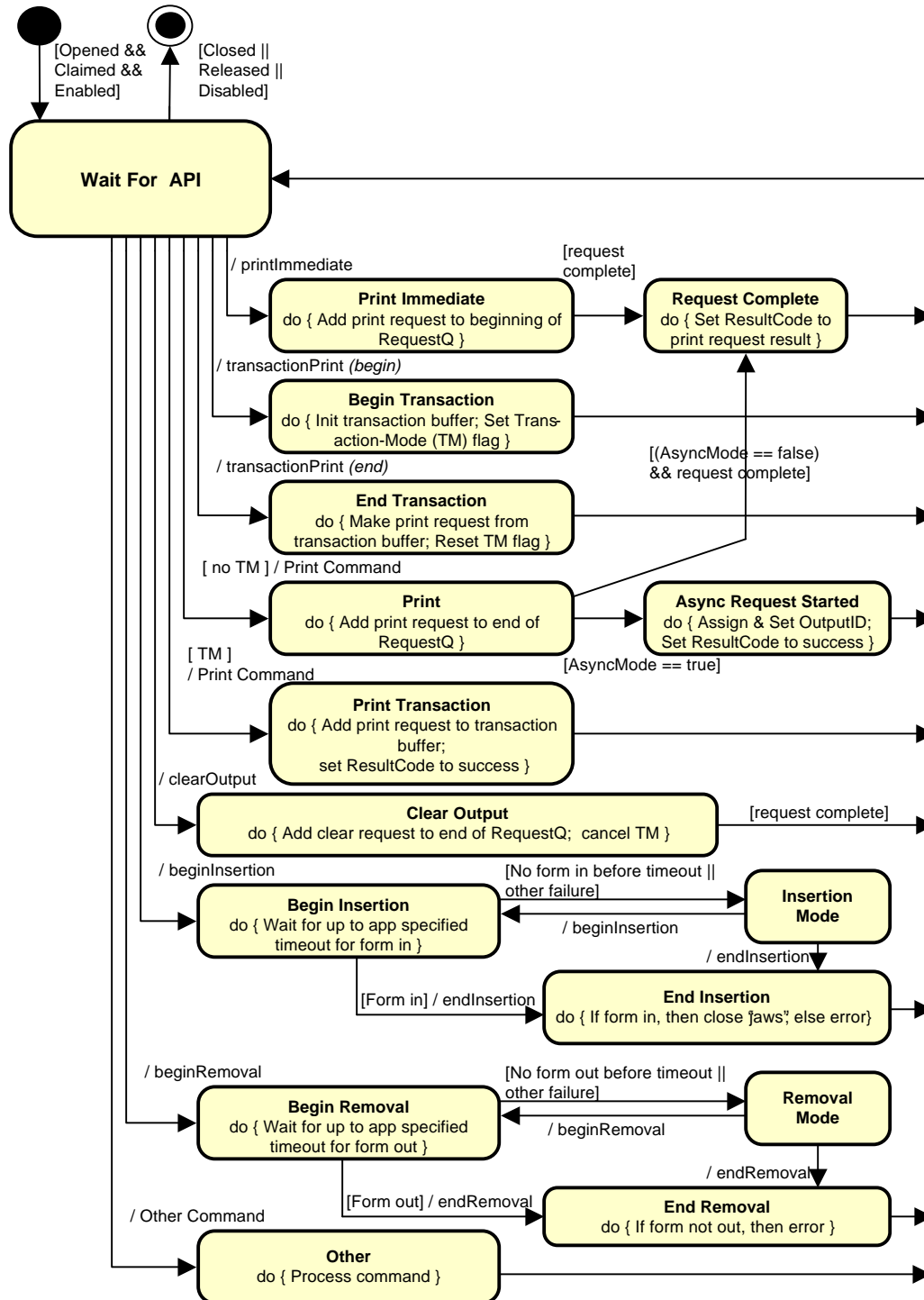
These diagrams assume:

- A separate request thread that processes print request.
Print requests are placed on a request queue (RequestQ) for the request thread to access. The request thread has some mechanism to report request completion and results.
- A separate event thread that delivers events.
Events are placed on an event queue (EventQ) for the event thread to access. The event thread has some mechanism to report error event results.

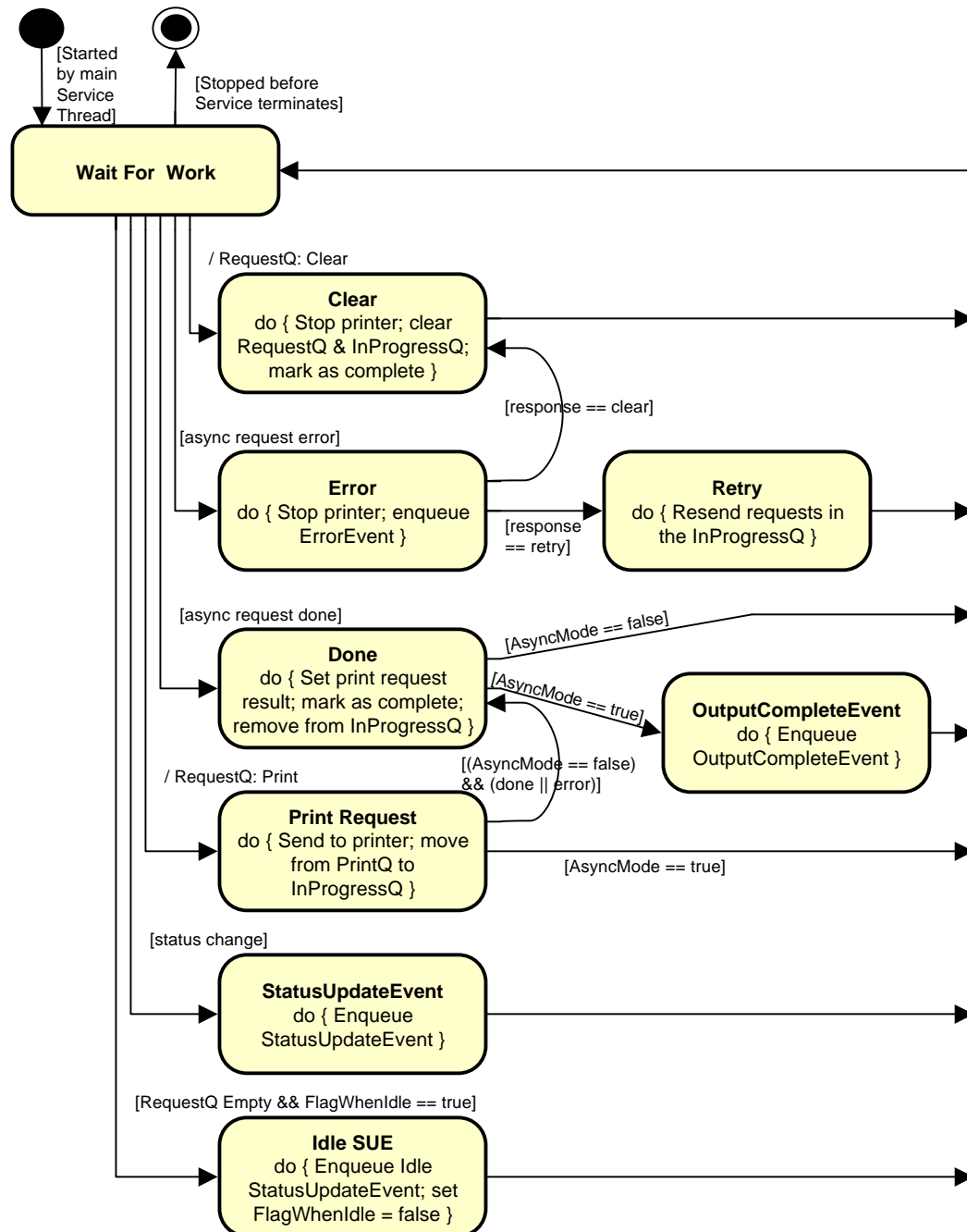
Print Commands: changePrintSide, cutPaper, markFeed, printBarCode, printBitmap, printNormal, printTwoNormal, rotatePrint.

Not Shown: Validation of APIs. If an API fails during validation, then it may return an error result and return prematurely to the “Wait for API” state.

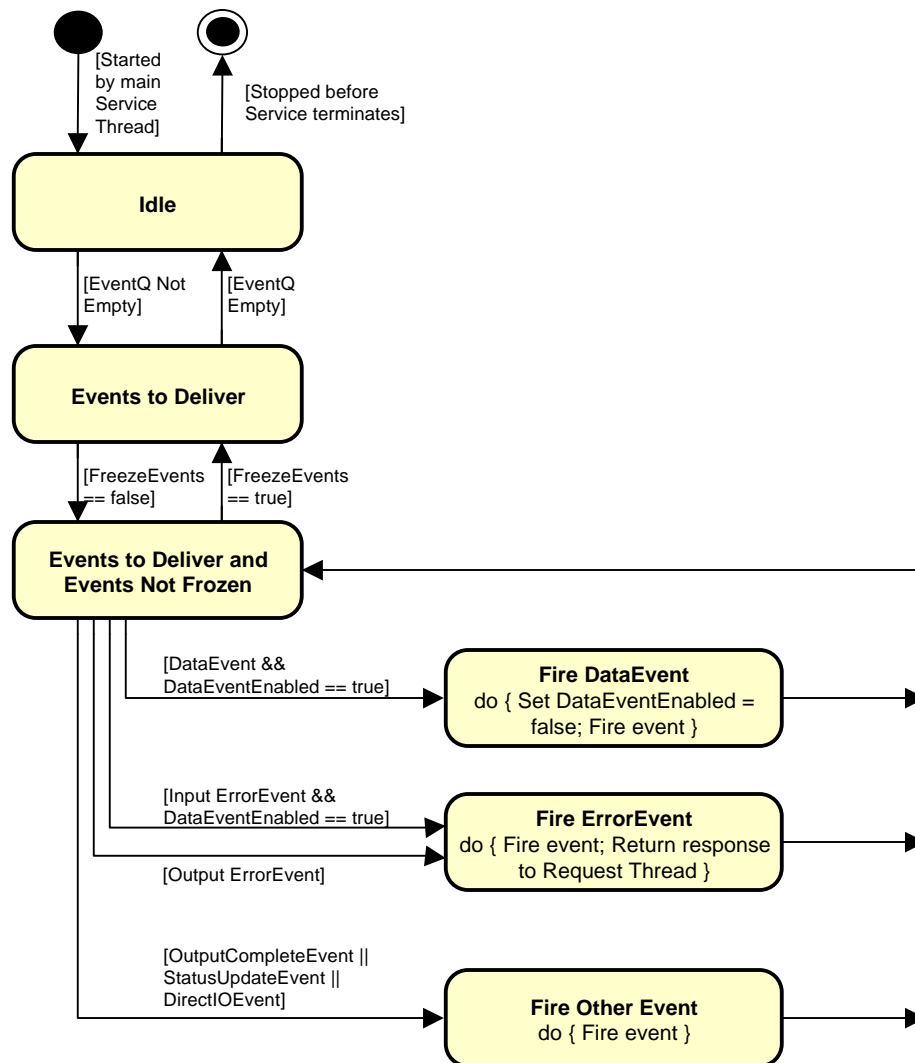
POS Printer State Diagram (Low Level): API



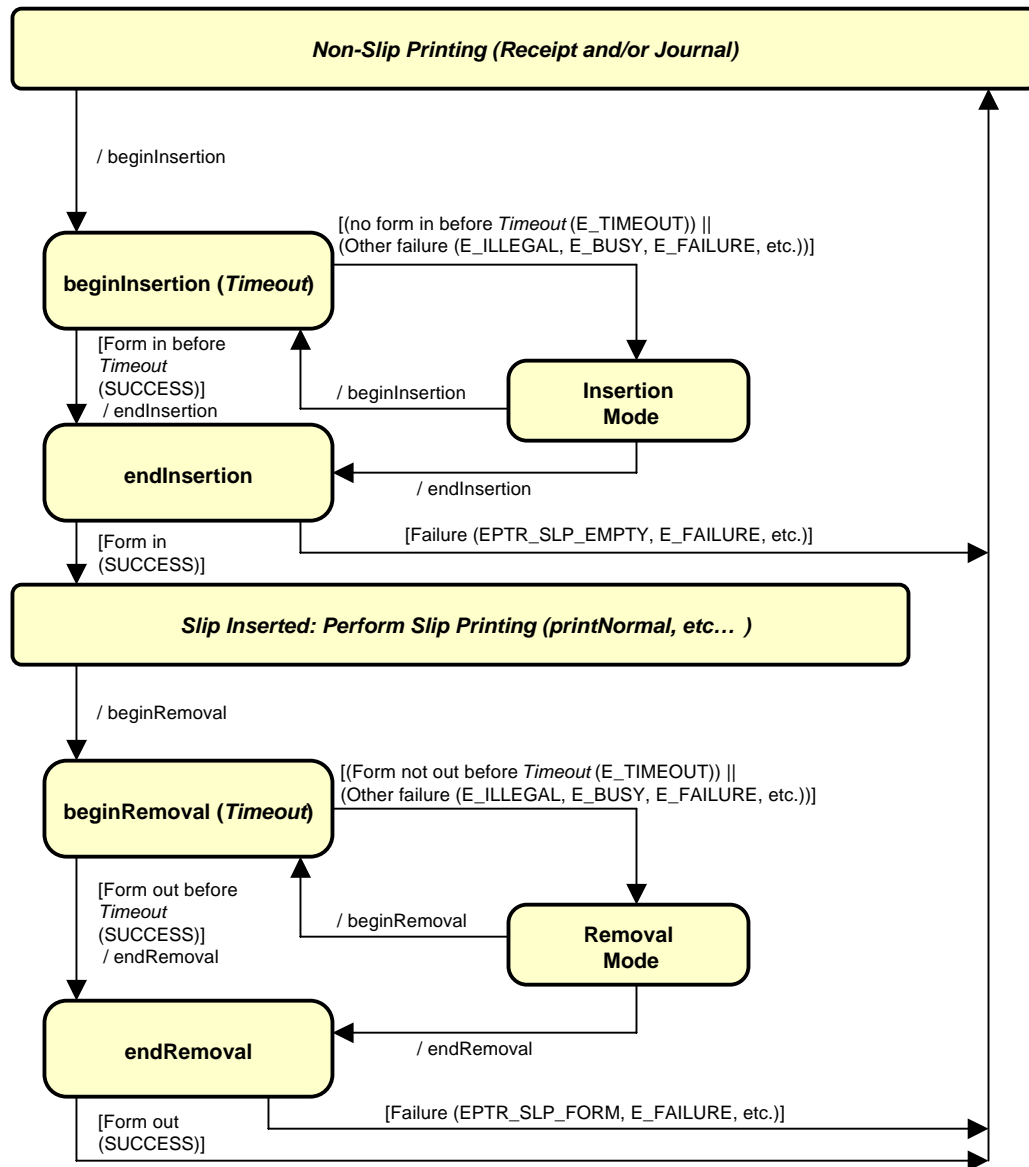
POS Printer State Diagram (Low Level): Request Thread



POS Printer State Diagram (Low Level): Event Delivery Thread



POS Printer Slip Handling State Diagram



Properties (UML attributes)

AsyncMode Property

Syntax	AsyncMode: <i>boolean</i> { read-write, access after open }
Remarks	<p>If true, then the print methods cutPaper, markFeed, printBarCode, printBitmap, printNormal, printTwoNormal, rotatePrint, and transactionPrint will be performed asynchronously. If false, they will be printed synchronously.</p> <p>This property is initialized to false by the open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapCharacterSet Property

Updated in Release 1.5

Syntax	CapCharacterSet: <i>int32</i> { read-only, access after open }												
Remarks	<p>Holds the default character set capability. It has one of the following values:</p> <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 CharacterSet property may be examined for additional information.</p> <p>This property is initialized by the open 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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.												
See Also	CharacterSet Property.												

CapConcurrentJrnRec Property

Syntax	CapConcurrentJrnRec: <i>boolean</i> { read-only , access after open }
Remarks	<p>If true, then the Journal and Receipt stations can print at the same time. The printTwoNormal method may be used with the PTR_TWO_RECEIPT_JOURNAL and PTR_S_JOURNAL_RECEIPT station 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">• Higher likelihood of error, such as greater chance of paper jams when moving between the stations.• Higher performance when each station is printed separately. <p>This property is initialized by the open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapConcurrentJrnSlp Property

Syntax	CapConcurrentJrnSlp: <i>boolean</i> { read-only , access after open }
Remarks	<p>If true, then the Journal and Slip stations can print at the same time. The printTwoNormal 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 beginInsertion/endInsertion followed by print requests to the Slip followed by beginRemoval/endRemoval before printing on the Journal. Non-concurrent printing may be required for reasons such as:</p> <ul style="list-style-type: none">• Physical constraints, such as the Slip form being placed in front of the Journal station.• Higher likelihood of error, such as greater chance of paper jams when moving between the stations.• Higher performance when each station is printed separately. <p>This property is initialized by the open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapConcurrentRecSlp Property

Syntax	CapConcurrentRecSlp: <i>boolean</i> { read-only, access after open }
Remarks	<p>If true, then the Receipt and Slip stations can print at the same time. The printTwoNormal 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 beginInsertion/endInsertion followed by print requests to the Slip followed by beginRemoval/endRemoval before printing on the Receipt. Non-concurrent printing may be required for reasons such as:</p> <ul style="list-style-type: none"> • Physical constraints, such as the Slip form being placed in front of the Receipt station. • Higher likelihood of error, such as greater chance of paper jams when moving between the stations. • Higher performance when each station is printed separately. <p>This property is initialized by the open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapCoverSensor Property

Syntax	CapCoverSensor: <i>boolean</i> { read-only, access after open }
Remarks	<p>If true, then the printer has a “cover open” sensor.</p> <p>This property is initialized by the open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapJrn2Color Property

Syntax	CapJrn2Color: <i>boolean</i> { read-only, access after open }
Remarks	<p>If true, then the journal can print dark plus an alternate color.</p> <p>This property is initialized by the open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapJrnBold Property

Syntax	CapJrnBold: <i>boolean</i> { read-only, access after open }
Remarks	<p>If true, then the journal can print bold characters.</p> <p>This property is initialized by the open method.</p>
Errors	A UpoException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapJrnCartridgeSensor Property

Added in Release 1.5

Syntax	CapJrnCartridgeSensor: <i>int32</i> { read-only, access after open }										
Remarks	<p>This bit mapped parameter is used to indicate the presence of Journal Cartridge monitoring sensors.</p> <p>If CapJrnPresent is false, this property is “0”. Otherwise it is a logical OR combination of any of the following values:</p> <table> <tr> <th>Value</th><th>Meaning</th></tr> <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> </table> <p>Note that the above mentioned values are arranged according to their priority level.</p> <p>This property is initialized by the open method.</p>	Value	Meaning	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.
Value	Meaning										
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.										
Errors	A UpoException may be thrown when this property is accessed. For further information, see “Errors” on page 15.										
See Also	JrnCartridgeState Property, JrnCurrentCartridge Property, CartridgeNotify Property.										

CapJrnColor Property***Added in Release 1.5*****Syntax** **CapJrnColor: *int32* { read-only, access after open }****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:

Value	Meaning
PTR_COLOR_PRIMARY	Supports Primary Color (Usually Black)
PTR_COLOR_CUSTOM1	Supports 1 st Custom Color (Secondary Color, usually Red)
PTR_COLOR_CUSTOM2	Supports 2 nd Custom Color
PTR_COLOR_CUSTOM3	Supports 3 rd Custom Color
PTR_COLOR_CUSTOM4	Supports 4 th Custom Color
PTR_COLOR_CUSTOM5	Supports 5 th Custom Color
PTR_COLOR_CUSTOM6	Supports 6 th 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 UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapJrnDhigh Property

Syntax	CapJrnDhigh: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the journal can print double high characters. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapJrnDwide Property

Syntax	CapJrnDwide: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the journal can print double wide characters. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapJrnDwideDhigh Property

Syntax	CapJrnDwideDhigh: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the journal can print double high / double wide characters. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapJrnEmptySensor Property

Syntax	CapJrnEmptySensor: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the journal has an out-of-paper sensor. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapJrnItalic Property

Syntax	CapJrnItalic: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the journal can print italic characters. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapJrnNearEndSensor Property

Syntax	CapJrnNearEndSensor: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the journal has a low paper sensor. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapJrnPresent Property

Syntax	CapJrnPresent: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the journal print station is present. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapJrnUnderline Property

Syntax	CapJrnUnderline: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the journal can underline characters. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapRec2Color Property

Syntax	CapRec2Color: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the receipt can print dark plus an alternate color. This property is initialized by the open method.
Errors	A UpoException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapRecBarCode Property

Syntax	CapRecBarCode: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the receipt has bar code printing capability. This property is initialized by the open method.
Errors	A UpoException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapRecBitmap Property

Syntax	CapRecBitmap: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the receipt can print bitmaps. This property is initialized by the open method.
Errors	A UpoException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapRecBold Property

Syntax	CapRecBold: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the receipt can print bold characters. This property is initialized by the open method.
Errors	A UpoException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapRecCartridgeSensor Property***Added in Release 1.5*****Syntax** **CapRecCartridgeSensor: *int32* { read-only, access after open }****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:

Value	Meaning
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 UposException 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**Added in Release 1.5**

Syntax **CapRecColor: *int32* { read-only, access after open }**

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:

Value	Meaning
PTR_COLOR_PRIMARY	Supports Primary Color (Usually Black)
PTR_COLOR_CUSTOM1	Supports 1 st Custom Color (Secondary Color, usually Red)
PTR_COLOR_CUSTOM2	Supports 2 nd Custom Color
PTR_COLOR_CUSTOM3	Supports 3 rd Custom Color
PTR_COLOR_CUSTOM4	Supports 4 th Custom Color
PTR_COLOR_CUSTOM5	Supports 5 th Custom Color
PTR_COLOR_CUSTOM6	Supports 6 th 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 UpoException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapRecDhigh Property

Syntax **CapRecDhigh: *boolean* { read-only, access after open }**

Remarks If true, then the receipt can print double high characters.

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

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

CapRecDwide Property

Syntax	CapRecDwide: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the receipt can print double wide characters. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapRecDwideDhigh Property

Syntax	CapRecDwideDhigh: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the receipt can print double high /double wide characters. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapRecEmptySensor Property

Syntax	CapRecEmptySensor: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the receipt has an out-of-paper sensor. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapRecItalic Property

Syntax	CapRecItalic: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the receipt can print italic characters. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapRecLeft90 Property

Syntax	CapRecLeft90: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the receipt can print in a rotated 90° left mode. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapRecMarkFeed Property***Added in Release 1.5***

Syntax	CapRecMarkFeed: <i>int32</i> { read-only, access after open }										
Remarks	<p>This parameter indicates the type of mark sensed paper handling available.</p> <p>CapRecMarkFeed is a logical OR combination of the following values. (The values are identical to those used with the markFeed method.)</p> <table> <tr> <th>Value</th><th>Meaning</th></tr> <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> </table> <p>If CapRecMarkFeed equals "0", mark sensed paper handling is not supported.</p> <p>This property is initialized by the open method.</p>	Value	Meaning	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.
Value	Meaning										
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.										
Errors	A UpoException may be thrown when this property is accessed. For further information, see "Errors" on page 15.										
See Also	markFeed Method.										

CapRecNearEndSensor Property

Syntax	CapRecNearEndSensor: <i>boolean</i> { read-only, access after open }
Remarks	<p>If true, then the receipt has a low paper sensor.</p> <p>This property is initialized by the open method.</p>
Errors	A UpoException may be thrown when this property is accessed. For further information, see "Errors" on page 15.

CapRecPapercut Property

Syntax	CapRecPapercut: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the receipt can perform paper cuts. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapRecPresent Property

Syntax	CapRecPresent: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the receipt print station is present. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapRecRight90 Property

Syntax	CapRecRight90: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the receipt can print in a rotated 90° right mode. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapRecRotate180 Property

Syntax	CapRecRotate180: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the receipt can print in a rotated upside down mode. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapRecStamp Property

Syntax	CapRecStamp: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the receipt has a stamp capability. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapRecUnderline Property

Syntax	CapRecUnderline: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the receipt can underline characters. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapSlp2Color Property

Syntax	CapSlp2Color: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the slip can print dark plus an alternate color. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapSlpBarCode Property

Syntax	CapSlpBarCode: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the slip has bar code printing capability. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapSlpBitmap Property

Syntax	CapSlpBitmap: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the slip can print bitmaps. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapSlpBold Property

Syntax	CapSlpBold: <i>boolean</i> { read-only , access after open }
Remarks	If true, then the slip can print bold characters. This property is initialized by the open method.
Errors	A UpoException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapSlpBothSidesPrint Property **Added in Release 1.5**

Syntax	CapSlpBothSidesPrint: <i>boolean</i> { read-only , access after open }
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 open method.
Errors	A UpoException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapSlpCartridgeSensor Property **Added in Release 1.5**

Syntax	CapSlpCartridgeSensor: <i>int32</i> { read-only , access after open }										
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: <table border="1" data-bbox="467 1176 1385 1575"> <thead> <tr> <th>Value</th><th>Meaning</th></tr> </thead> <tbody> <tr> <td>PTR_CART_REMOVED</td><td>There is a function to indicate the Cartridge has been removed.</td></tr> <tr> <td>PTR_CART_EMPTY</td><td>There is a function to indicate the Cartridge is empty.</td></tr> <tr> <td>PTR_CART_CLEANING</td><td>There is a function to indicate head is being cleaned.</td></tr> <tr> <td>PTR_CART_NEAREND</td><td>There is a function to indicate the color Cartridge is near end.</td></tr> </tbody> </table> <p>Note that the above mentioned values are arranged according to their priority level.</p> <p>This property is initialized by the open method.</p>	Value	Meaning	PTR_CART_REMOVED	There is a function to indicate the Cartridge has been removed.	PTR_CART_EMPTY	There is a function to indicate the Cartridge is empty.	PTR_CART_CLEANING	There is a function to indicate head is being cleaned.	PTR_CART_NEAREND	There is a function to indicate the color Cartridge is near end.
Value	Meaning										
PTR_CART_REMOVED	There is a function to indicate the Cartridge has been removed.										
PTR_CART_EMPTY	There is a function to indicate the Cartridge is empty.										
PTR_CART_CLEANING	There is a function to indicate head is being cleaned.										
PTR_CART_NEAREND	There is a function to indicate the color Cartridge is near end.										
Errors	A UpoException 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***Added in Release 1.5***

Syntax	CapSlpColor: <i>int32</i> { read-only, access after open }																								
Remarks	<p>This capability indicates the availability of Slip printing color cartridges.</p> <p>If CapSlpPresent is false, this property is “0”. Otherwise, this property indicates the supported color cartridges.</p> <p>CapSlpColor is a logical OR combination of any of the following values:</p> <table> <tr> <th>Value</th><th>Meaning</th></tr> <tr> <td>PTR_COLOR_PRIMARY</td><td>Supports Primary Color (Usually Black)</td></tr> <tr> <td>PTR_COLOR_CUSTOM1</td><td>Supports 1st Custom Color (Secondary Color, usually Red)</td></tr> <tr> <td>PTR_COLOR_CUSTOM2</td><td>Supports 2nd Custom Color</td></tr> <tr> <td>PTR_COLOR_CUSTOM3</td><td>Supports 3rd Custom Color</td></tr> <tr> <td>PTR_COLOR_CUSTOM4</td><td>Supports 4th Custom Color</td></tr> <tr> <td>PTR_COLOR_CUSTOM5</td><td>Supports 5th Custom Color</td></tr> <tr> <td>PTR_COLOR_CUSTOM6</td><td>Supports 6th Custom Color</td></tr> <tr> <td>PTR_COLOR_CYAN</td><td>Supports Cyan Color for full color printing</td></tr> <tr> <td>PTR_COLOR_MAGENTA</td><td>Supports Magenta Color for full color printing</td></tr> <tr> <td>PTR_COLOR_YELLOW</td><td>Supports Yellow Color for full color printing</td></tr> <tr> <td>PTR_COLOR_FULL</td><td>Supports Full Color.</td></tr> </table> <p>This property is initialized by the open method.</p>	Value	Meaning	PTR_COLOR_PRIMARY	Supports Primary Color (Usually Black)	PTR_COLOR_CUSTOM1	Supports 1 st Custom Color (Secondary Color, usually Red)	PTR_COLOR_CUSTOM2	Supports 2 nd Custom Color	PTR_COLOR_CUSTOM3	Supports 3 rd Custom Color	PTR_COLOR_CUSTOM4	Supports 4 th Custom Color	PTR_COLOR_CUSTOM5	Supports 5 th Custom Color	PTR_COLOR_CUSTOM6	Supports 6 th 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.
Value	Meaning																								
PTR_COLOR_PRIMARY	Supports Primary Color (Usually Black)																								
PTR_COLOR_CUSTOM1	Supports 1 st Custom Color (Secondary Color, usually Red)																								
PTR_COLOR_CUSTOM2	Supports 2 nd Custom Color																								
PTR_COLOR_CUSTOM3	Supports 3 rd Custom Color																								
PTR_COLOR_CUSTOM4	Supports 4 th Custom Color																								
PTR_COLOR_CUSTOM5	Supports 5 th Custom Color																								
PTR_COLOR_CUSTOM6	Supports 6 th 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.																								
Errors	A UpoException may be thrown when this property is accessed. For further information, see “Errors” on page 15.																								

CapSlpDhigh Property

Syntax	CapSlpDhigh: <i>boolean</i> { read-only, access after open }
Remarks	<p>If true, then the slip can print double high characters.</p> <p>This property is initialized by the open method.</p>
Errors	A UpoException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapSlpDwide Property

Syntax	CapSlpDwide: <i>boolean</i> { read-only , access after open }
Remarks	If true, then the slip can print double wide characters. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapSlpDwideDhigh Property

Syntax	CapSlpDwideDhigh: <i>boolean</i> { read-only , access after open }
Remarks	If true, then the slip can print double high / double wide characters. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapSlpEmptySensor Property

Syntax	CapSlpEmptySensor: <i>boolean</i> { read-only , access after open }
Remarks	If true, then the slip has a “slip in” sensor. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapSlpFullslip Property

Syntax	CapSlpFullslip: <i>boolean</i> { read-only , access after open }
Remarks	If true, then the slip is a full slip station. It can print full-length forms. If false, then the slip is a “validation” type station. This usually limits the number of print lines, and disables access to the receipt and/or journal stations while the validation slip is being used. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapSlpItalic Property

Syntax	CapSlpItalic: <i>boolean</i> { read-only , access after open }
Remarks	If true, then the slip can print italic characters. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapSlpLeft90 Property

Syntax	CapSlpLeft90: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the slip can print in a rotated 90° left mode. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapSlpNearEndSensor Property

Syntax	CapSlpNearEndSensor: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the slip has a “slip near end” sensor. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapSlpPresent Property

Syntax	CapSlpPresent: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the slip print station is present. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapSlpRight90 Property

Syntax	CapSlpRight90: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the slip can print in a rotated 90° right mode. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapSlpRotate180 Property

Syntax	CapSlpRotate180: <i>boolean</i> { read-only, access after open }
Remarks	If true, then the slip can print in a rotated upside down mode. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapSlpUnderline Property

Syntax	CapSlpUnderline: <i>boolean</i> { read-only , access after open }
Remarks	If true, then the slip can underline characters. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CapTransaction Property

Syntax	CapTransaction: <i>boolean</i> { read-only , access after open }
Remarks	If true, then printer transactions are supported by each station. This property is initialized by the open method.
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

CartridgeNotify Property***Added in Release 1.5***

Syntax	CartridgeNotify: <i>int32</i> { read-write , access after open }
Remarks	Contains the type of cartridge state notification selected by the application. The CartridgeNotify values are:

Value	Meaning
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 StatusUpdateEvents will be fired, and JrnCartridgeState , RecCartridgeState , and SlpCartridgeState may not be set.
PTR_CN_ENABLED	The Control will fire cartridge state notification StatusUpdateEvents and update JrnCartridgeState , RecCartridgeState and SlpCartridgeState , beginning when DeviceEnabled is set true. The level of functionality depends upon CapJrnCartridgeSensor , CapRecCartridgeSensor and CapSlpCartridgeSensor .

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 `UposException` 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:

Value	Meaning
E_ILLEGAL	One of the following errors occurred: The device is already enabled. CapJrnCartridgeSensor , CapRecCartridgeSensor , and CapSlpCartridgeSensor = “0”.

See Also **CapJrnCartridgeSensor** Property, **CapRecCartridgeSensor** Property, **CapSlpCartridgeSensor** Property, **JrnCartridgeState** Property, **RecCartridgeState** Property, **SlpCartridgeState** Property.

CharacterSet Property

Updated in Release 1.5

Syntax `CharacterSet: int32 { read-write, access after open-claim-enable }`

Remarks Holds the character set for printing characters. It has one of the following values:

Value	Meaning
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.

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

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

See Also **CharacterSetList** Property.

CharacterSetList Property

Syntax	CharacterSetList: <i>string</i> { read-only, access after open }
Remarks	<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 open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	CharacterSet Property.

CoverOpen Property

Syntax	CoverOpen: <i>boolean</i> { read-only, access after open-claim-enable }
Remarks	<p>If true, then the printer’s cover is open.</p> <p>If CapCoverSensor 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>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

ErrorLevel Property

Syntax	ErrorLevel: <i>int32</i> { read-only, access after open }								
Remarks	<p>Holds the severity of the error condition. It has one of the following values:</p> <table border="1"> <thead> <tr> <th>Value</th><th>Meaning</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 ErrorEvent. When the error is cleared, then the property is changed to PTR_EL_NONE.</p>	Value	Meaning	PTR_EL_NONE	No error condition is present.	PTR_EL_RECOVERABLE	A recoverable error has occurred. (Example: Out of paper.)	PTR_EL_FATAL	A non-recoverable error has occurred. (Example: Internal printer failure.)
Value	Meaning								
PTR_EL_NONE	No error condition is present.								
PTR_EL_RECOVERABLE	A recoverable error has occurred. (Example: Out of paper.)								
PTR_EL_FATAL	A non-recoverable error has occurred. (Example: Internal printer failure.)								
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.								

ErrorStation Property

Syntax	ErrorStation: <i>int32</i> { read-only , access after open }										
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:</p> <table> <tr> <td>PTR_S_JOURNAL</td><td>PTR_S_RECEIPT</td></tr> <tr> <td>PTR_S_SLIP</td><td>PTR_S_JOURNAL_RECEIPT</td></tr> <tr> <td>PTR_S_JOURNAL_SLIP</td><td>PTR_S_RECEIPT_SLIP</td></tr> <tr> <td>PTR_TWO_RECEIPT_JOURNAL</td><td>PTR_TWO_SLIP_JOURNAL</td></tr> <tr> <td>PTR_TWO_SLIP_RECEIPT</td><td></td></tr> </table> <p>This property is only valid if the ErrorLevel is not equal to PTR_EL_NONE. It is set just before delivering an ErrorEvent.</p>	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	
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											
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.										

ErrorString Property

Syntax	ErrorString: <i>string</i> { read-only , access after open }
Remarks	<p>Holds a vendor-supplied description of the current error.</p> <p>This property is set just before delivering an ErrorEvent. If no description is available, the property is set to an empty string. When the error is cleared, then the property is changed to an empty string.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

FlagWhenIdle Property

Syntax	FlagWhenIdle: <i>boolean</i> { read-write , access after open }
Remarks	<p>If true, a StatusUpdateEvent 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 clearOutput method or by an ErrorEvent handler.</p> <p>If the State is already set to S_IDLE when this property is set to true, then a StatusUpdateEvent 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 open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

FontTypefaceList Property

Syntax	FontTypefaceList: <i>string</i> { read-only , access after open }
Remarks	<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 open method.</p>
Errors	A UpoException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	“Data Characters and Escape Sequences” on page 316.

JrnCartridgeState Property***Added in Release 1.5***

Syntax	JrnCartridgeState: <i>int32</i> { read-only , access after open-claim-enable }										
Remarks	<p>This property contains the status of the currently selected Journal cartridge (ink, ribbon or toner).</p> <p>It contains one of the following values:</p> <table> <tr> <th>Value</th><th>Meaning</th></tr> <tr> <td>PTR_CART_UNKNOWN</td><td> <p>Cannot determine the cartridge state, for one of the following reasons:</p> <p>CapJrnCartridgeSensor = “0”.</p> <p>Device does not support cartridge state reporting.</p> <p>CartridgeNotify = PTR_CN_DISABLED.</p> <p>Cartridge state notifications are disabled.</p> <p>DeviceEnabled = FALSE.</p> <p>Cartridge state monitoring does not occur until the device is enabled.</p> </td></tr> <tr> <td>PTR_CART_REMOVED</td><td>The cartridge selected by JrnCurrentCartridge has been removed.</td></tr> <tr> <td>PTR_CART_EMPTY</td><td>The cartridge selected by JrnCurrentCartridge is empty.</td></tr> <tr> <td>PTR_CART_CLEANING</td><td>The head selected by JrnCurrentCartridge is being cleaned.</td></tr> </table>	Value	Meaning	PTR_CART_UNKNOWN	<p>Cannot determine the cartridge state, for one of the following reasons:</p> <p>CapJrnCartridgeSensor = “0”.</p> <p>Device does not support cartridge state reporting.</p> <p>CartridgeNotify = PTR_CN_DISABLED.</p> <p>Cartridge state notifications are disabled.</p> <p>DeviceEnabled = FALSE.</p> <p>Cartridge state monitoring does not occur until the device is enabled.</p>	PTR_CART_REMOVED	The cartridge selected by JrnCurrentCartridge has been removed.	PTR_CART_EMPTY	The cartridge selected by JrnCurrentCartridge is empty.	PTR_CART_CLEANING	The head selected by JrnCurrentCartridge is being cleaned.
Value	Meaning										
PTR_CART_UNKNOWN	<p>Cannot determine the cartridge state, for one of the following reasons:</p> <p>CapJrnCartridgeSensor = “0”.</p> <p>Device does not support cartridge state reporting.</p> <p>CartridgeNotify = PTR_CN_DISABLED.</p> <p>Cartridge state notifications are disabled.</p> <p>DeviceEnabled = FALSE.</p> <p>Cartridge state monitoring does not occur until the device is enabled.</p>										
PTR_CART_REMOVED	The cartridge selected by JrnCurrentCartridge has been removed.										
PTR_CART_EMPTY	The cartridge selected by JrnCurrentCartridge is empty.										
PTR_CART_CLEANING	The head selected by JrnCurrentCartridge is being cleaned.										

PTR_CART_NEAREND The cartridge selected by **JrnCurrentCartridge** is near end.

PTR_CART_OK The cartridge selected by **JrnCurrentCartridge** 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 UposException 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

Added in Release 1.5

Syntax **JrnCurrentCartridge: *int32* { read-write, access after open-claim-enable }**

Remarks This property specifies the currently selected Journal cartridge.

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

This value is guaranteed to be one of the color cartridges specified by the **CapJrnColor** property. (PTR_COLOR_FULL can not be set.)

Setting **JrnCurrentCartridge** may also update **JrnCartridgeState**.

Errors A UposException 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:

Value	Meaning
E_ILLEGAL	An invalid property value was specified.

See Also **JrnCartridgeState** Property.

JrnEmpty Property

Syntax **JrnEmpty: *boolean* { read-only, access after open-claim-enable }**

Remarks If true, the journal is out of paper. If false, journal paper is present.

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

See Also **JrnNearEnd** Property.

JrnLetterQuality Property

Syntax	JrnLetterQuality: <i>boolean</i> { read-write, access after open-claim-enable }
Remarks	<p>If true, prints in high quality mode. If false, prints in high speed mode.</p> <p>This property advises the 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 JrnLineWidth, JrnLineHeight, and JrnLineSpacing if MapMode is PTR_MM_DOTS. (See the footnote at MapMode.)</p> <p>This property is initialized to false when the device is first enabled following the open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

JrnLineChars Property

Syntax	JrnLineChars: <i>int32</i> { read-write, access after open-claim-enable }
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 is less than or equal to the maximum value allowed for the printer, then the width is set to the specified value. If the exact width cannot be supported, then subsequent lines will be printed with a character size that most closely supports the specified characters per line. (For example, if set to 36 and the printer can print either 30 or 40 characters per line, then the Service should select the 40 characters per line size and print only up to 36 characters per line.)</p> <p>If the character width 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 Service cannot support the request.)</p> <p>Setting this property may also update JrnLineWidth, JrnLineHeight, and JrnLineSpacing, 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 open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	JrnLineCharsList Property.

JrnLineCharsList Property

Syntax	JrnLineCharsList: <i>string</i> { read-only, access after open }
Remarks	<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 open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	JrnLineChars Property.

JrnLineHeight Property

Syntax	JrnLineHeight: <i>int32</i> { read-write, access after open-claim-enable }
Remarks	<p>Holds the journal print line height. Expressed in the unit of measure given by MapMode.</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 JrnLineChars 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 open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

JrnLineSpacing Property

Syntax	JrnLineSpacing: <i>int32</i> { read-write, access after open-claim-enable }
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 MapMode.</p> <p>If changed to a spacing that can be supported by the printer, then the line spacing is set to this value. If the spacing cannot be supported, then the spacing is set to the closest supported value.</p> <p>When JrnLineChars or JrnLineHeight 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 open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see "Errors" on page 15.

JrnLineWidth Property

Syntax	JrnLineWidth: <i>int32</i> { read-only, access after open-claim-enable }
Remarks	<p>Holds the width of a line of JrnLineChars characters. Expressed in the unit of measure given by MapMode.</p> <p>Setting JrnLineChars 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 open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see "Errors" on page 15.

JrnNearEnd Property

Syntax	JrnNearEnd: <i>boolean</i> { read-only, access after open-claim-enable }
Remarks	<p>If true, the journal paper is low. If false, journal paper is not low.</p> <p>If CapJrnNearEndSensor is false, then this property is always false.</p> <p>This property is initialized and kept current while the device is enabled.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see "Errors" on page 15.
See Also	JrnEmpty Property.

MapMode Property

Syntax **MapMode:** *int32* { read-write, access after open }

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:

Value	Meaning
PTR_MM_DOTS	The printer's dot width. This width may be different for each printer station. ¹
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 UpoException may be thrown when this property is accessed. For further information, see "Errors" on page 15.

¹. From the POS Printer perspective, the exact definition of a "dot" is not significant. It is a Printer/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 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

- Syntax** **RecBarCodeRotationList:** *string* { read-only, access after open }
- 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:

Value	Meaning
0	Bar code may be printed in the normal orientation.
R90	Bar code may be rotated 90° to the right.
L90	Bar code may be rotated 90° to the left.
180	Bar code may be rotated 180° - upside down.

For example, if the string is “0,180”, then the printer can print normal bar codes and upside down bar codes.

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

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

- See Also** **RotateSpecial** Property, **printBarCode** Method.

RecCartridgeState Property**Added in Release 1.5**

- Syntax** **RecCartridgeState:** *int32* { read-only, access after open-claim-enable }
- Remarks** This property contains the status of the currently selected Receipt cartridge (ink, ribbon or toner).
- It contains one of the following values:

Value	Meaning
PTR_CART_UNKNOWN	Cannot determine the cartridge state, for one of the following reasons: CapRecCartridgeSensor = “0”. Device does not support cartridge state reporting. CartridgeNotify = PTR_CN_DISABLED. Cartridge state notifications are disabled. DeviceEnabled = FALSE. Cartridge state monitoring does not occur until the device is enabled.
PTR_CART_REMOVED	The cartridge selected by RecCurrentCartridge has been removed.
PTR_CART_EMPTY	The cartridge selected by RecCurrentCartridge is empty.

PTR_CART_CLEANING	The head selected by RecCurrentCartridge is being cleaned.
PTR_CART_NEAREND	The cartridge selected by RecCurrentCartridge is near end.
PTR_CART_OK	The cartridge selected by RecCurrentCartridge 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 UposException 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

Added in Release 1.5

Syntax **RecCurrentCartridge: int32 { read-write, access after open-claim-enable }**

Remarks This property specifies the currently selected Receipt cartridge.

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

This value is guaranteed to be one of the color cartridges specified by the **CapRecColor** property. (PTR_COLOR_FULL can not be set.)

Setting **RecCurrentCartridge** may also update **RecCartridgeState**.

Errors A UposException 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:

Value	Meaning
E_ILLEGAL	An invalid property value was specified.

See Also **RecCartridgeState** Property.

RecEmpty Property

Syntax **RecEmpty: boolean { read-only, access after open-claim-enable }**

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

See Also **RecNearEnd** Property.

RecLetterQuality Property

Syntax	RecLetterQuality: <i>boolean</i> { read-write, access after open-claim-enable }
Remarks	<p>If true, prints in high quality mode. If false, prints in high speed mode.</p> <p>This property advises the Service that either high quality or high speed printing is desired. For example:</p> <ul style="list-style-type: none"> Printers with bi-directional print capability may be placed in unidirectional mode for high quality, so that column alignment is more precise. Bitmaps may be printed in a high-density graphics mode for high-quality, and in a low-density mode for high speed. <p>Setting this property may also update RecLineWidth, RecLineHeight, and RecLineSpacing if MapMode is PTR_MM_DOTS. (See the footnote at MapMode.)</p> <p>This property is initialized to false when the device is first enabled following the open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

RecLineChars Property

Syntax	RecLineChars: <i>int32</i> { read-write, access after open-claim-enable }
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 is less than or equal to the maximum value allowed for the printer, then the width is set to the specified value. If the exact width cannot be supported, then subsequent lines will be printed with a character size that most closely supports the specified characters per line. (For example, if set to 36 and the printer can print either 30 or 40 characters per line, then the Service should select the 40 characters per line size and print only up to 36 characters per line.)</p> <p>If the character width 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 Service cannot support the request.)</p> <p>Setting this property may also update RecLineWidth, RecLineHeight, and RecLineSpacing, 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 open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	RecLineCharsList Property.

RecLineCharsList Property

Syntax	RecLineCharsList: <i>string</i> { read-only, access after open }
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 open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	RecLineChars Property.

RecLineHeight Property

Syntax	RecLineHeight: <i>int32</i> { read-write, access after open-claim-enable }
Remarks	<p>Holds the receipt print line height, expressed in the unit of measure given by MapMode.</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 RecLineChars 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 open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	RecLineChars Property.

RecLineSpacing Property

Syntax	RecLineSpacing: <i>int32</i> { read-write, access after open-claim-enable }
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 MapMode.</p> <p>If changed to a spacing that can be supported by the printer, then the line spacing is set to this value. If the spacing cannot be supported, then the spacing is set to the closest supported value.</p> <p>When RecLineChars or RecLineHeight are changed, 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 open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see "Errors" on page 15.

RecLinesToPaperCut Property

Syntax	RecLinesToPaperCut: <i>int32</i> { read-only, access after open-claim-enable }
Remarks	<p>Holds the number of lines that must be advanced before the receipt paper is cut.</p> <p>If CapRecPapercut is true, then this is the line count before reaching the paper cut mechanism. Otherwise, this is the line count before the manual tear-off bar.</p> <p>Changing the properties RecLineChars, RecLineHeight, and RecLineSpacing may cause this property to change.</p> <p>This property is initialized when the device is first enabled following the open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see "Errors" on page 15.

RecLineWidth Property

Syntax	RecLineWidth: <i>int32</i> { read-only, access after open-claim-enable }
Remarks	<p>Holds the width of a line of RecLineChars characters, expressed in the unit of measure given by MapMode.</p> <p>Setting RecLineChars 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 open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see "Errors" on page 15.

RecNearEnd Property

Syntax	RecNearEnd: <i>boolean</i> { read-only, access after open-claim-enable }
Remarks	<p>If true, the receipt paper is low. If false, receipt paper is not low.</p> <p>If CapRecNearEndSensor is false, then this property is always false.</p> <p>This property is initialized and kept current while the device is enabled.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	RecEmpty Property.

RecSidewaysMaxChars Property

Syntax	RecSidewaysMaxChars: <i>int32</i> { read-only, access after open-claim-enable }
Remarks	<p>Holds the maximum number of characters that may be printed on each line in sideways mode.</p> <p>If CapRecLeft90 and CapRecRight90 are both false, then this property is zero.</p> <p>Changing the properties RecLineHeight, RecLineSpacing, and RecLineChars may cause this property to change.</p> <p>This property is initialized when the device is first enabled following the open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	RecSidewaysMaxLines Property.

RecSidewaysMaxLines Property

Syntax	RecSidewaysMaxLines: <i>int32</i> { read-only, access after open-claim-enable }
Remarks	<p>Holds the maximum number of lines that may be printed in sideways mode.</p> <p>If CapRecLeft90 and CapRecRight90 are both false, then this property is zero.</p> <p>Changing the properties RecLineHeight, RecLineSpacing, and RecLineChars may cause this property to change.</p> <p>This property is initialized when the device is first enabled following the open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	RecSidewaysMaxChars Property.

RotateSpecial Property

Syntax	RotateSpecial: <i>int32</i> { read-write, access after open }										
Remarks	<p>Holds the rotation orientation for bar codes. It has one of the following values:</p> <table> <tr> <th>Value</th><th>Meaning</th></tr> <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> </table> <p>This property is initialized to PTR_RP_NORMAL by the open method.</p>	Value	Meaning	PTR_RP_NORMAL	Print subsequent bar codes in normal orientation.	PTR_RP_RIGHT90	Rotate printing 90° to the right (clockwise)	PTR_RP_LEFT90	Rotate printing 90° to the left (counter-clockwise)	PTR_RP_ROTATE180	Rotate printing 180°, that is, print upside-down
Value	Meaning										
PTR_RP_NORMAL	Print subsequent bar codes in normal orientation.										
PTR_RP_RIGHT90	Rotate printing 90° to the right (clockwise)										
PTR_RP_LEFT90	Rotate printing 90° to the left (counter-clockwise)										
PTR_RP_ROTATE180	Rotate printing 180°, that is, print upside-down										
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.										
See Also	printBarCode Method.										

SlpBarCodeRotationList Property

Syntax	SlpBarCodeRotationList: <i>string</i> { read-only, access after open }										
Remarks	<p>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:</p> <table> <tr> <th>Value</th><th>Meaning</th></tr> <tr> <td>0</td><td>Bar code may be printed in the normal orientation.</td></tr> <tr> <td>R90</td><td>Bar code may be rotated 90° to the right.</td></tr> <tr> <td>L90</td><td>Bar code may be rotated 90° to the left.</td></tr> <tr> <td>180</td><td>Bar code may be rotated 180° - upside down.</td></tr> </table> <p>For example, if the string is “0,180”, then the printer can print normal bar codes and upside down bar codes.</p> <p>This property is initialized by the open method.</p>	Value	Meaning	0	Bar code may be printed in the normal orientation.	R90	Bar code may be rotated 90° to the right.	L90	Bar code may be rotated 90° to the left.	180	Bar code may be rotated 180° - upside down.
Value	Meaning										
0	Bar code may be printed in the normal orientation.										
R90	Bar code may be rotated 90° to the right.										
L90	Bar code may be rotated 90° to the left.										
180	Bar code may be rotated 180° - upside down.										
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.										
See Also	RotateSpecial Property, printBarCode Method.										

SlpCartridgeState Property***Added in Release 1.5*****Syntax** **SlpCartridgeState: *int32* { read-only, access after open-claim-enable }****Remarks** This property contains the status of the currently selected Slip cartridge (ink, ribbon or toner).

It contains one of the following values:

Value	Meaning
PTR_CART_UNKNOWN	Cannot determine the cartridge state, for one of the following reasons: CapSlpCartridgeSensor = "0". Device does not support cartridge state reporting. CartridgeNotify = PTR_CN_DISABLED. Cartridge state notifications are disabled. DeviceEnabled = FALSE. Cartridge state monitoring does not occur until the device is enabled.
PTR_CART_REMOVED	The cartridge selected by SlpCurrentCartridge has been removed.
PTR_CART_EMPTY	The cartridge selected by SlpCurrentCartridge is empty.
PTR_CART_CLEANING	The head selected by SlpCurrentCartridge is being cleaned.
PTR_CART_NEAREND	The cartridge selected by SlpCurrentCartridge is near end.
PTR_CART_OK	The cartridge selected by SlpCurrentCartridge 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 UpoException 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***Added in Release 1.5*****Syntax** **SlpCurrentCartridge: *int32* { read-write, access after open-claim-enable }**

Remarks This property specifies the currently selected slip cartridge.

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

This value is guaranteed to be one of the color cartridges specified by the **CapSlpColor** property. (PTR_COLOR_FULL can not be set.)

Setting **SlpCurrentCartridge** may also update **SlpCartridgeState**.

Errors A UposException 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:

Value	Meaning
E_ILLEGAL	An invalid property value was specified.

See Also **RecCartridgeState** Property.

SlpEmpty Property**Syntax** **SlpEmpty: *boolean* { read-only, access after open-claim-enable }**

Remarks If true, a slip form is not present. If false, a slip form is present.

If **CapSlpEmptySensor** is false, then this property is always false.

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

Note

The “slip empty” sensor should be used primarily to determine whether a form has been inserted before printing, and can be monitored to determine whether a form is still in place. This sensor is usually placed one or more print lines above the slip print head.

However, the “slip near end” sensor (when present) should be used to determine when nearing the end of the slip. This sensor is usually placed one or more print lines below the slip print head.

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

See Also **SlpNearEnd** Property.

SlpLetterQuality Property

Syntax	SlpLetterQuality: <i>boolean</i> { read-write, access after open-claim-enable }
Remarks	<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"> • Printers with bi-directional print capability may be placed in unidirectional mode for high quality, so that column alignment is more precise. • Bitmaps may be printed in a high-density graphics mode for high-quality, and in a low-density mode for high speed. <p>Setting this property may also update SlpLineWidth, SlpLineHeight, and SlpLineSpacing if MapMode is PTR_MM_DOTS. (See the footnote at MapMode.)</p> <p>This property is initialized to false when the device is first enabled following the open method.</p>
Errors	A UpoException may be thrown when this property is accessed. For further information, see “Errors” on page 15.

SlpLineChars Property

Syntax	SlpLineChars: <i>int32</i> { read-write, access after open-claim-enable }
Remarks	<p>Holds the number of characters that may be printed on a slip line.</p> <p>If changed to a line character width that is less than or equal to the maximum value allowed for the printer, then the width is set to the specified value. If the exact width cannot be supported, then subsequent lines will be printed with a character size that most closely supports the specified characters per line. (The Service should print the requested characters in the column positions closest to the side of the slip table at which the slip is aligned. (For example, if the operator inserts the slip with the right edge against the table side and if SlpLineChars is set to 36 and the printer prints 60 characters per line, then the Service should add 24 spaces at the left margin and print the characters in columns 25 through 60.)</p> <p>If the character width is greater than the maximum value allowed for the printer, then an exception is thrown. (For example, if set to 65 and the printer can only print 60 characters per line, then the Service cannot support the request.)</p> <p>Setting this property may also update SlpLineWidth, SlpLineHeight, and SlpLineSpacing, 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 open method.</p>
Errors	A UpoException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	SlpLineCharsList Property.

SlpLineCharsList Property

Syntax	SlpLineCharsList: <i>string</i> { read-only, access after open }
Remarks	<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 open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	SlpLineChars Property.

SlpLineHeight Property

Syntax	SlpLineHeight: <i>int32</i> { read-write, access after open-claim-enable }
Remarks	<p>Holds the slip print-line height, expressed in the unit of measure given by MapMode.</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 SlpLineChars 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 open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	SlpLineChars Property.

SlpLinesNearEndToEnd Property.

Syntax	SlpLinesNearEndToEnd: <i>int32</i> { read-only, access after open-claim-enable }
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 SlpLineHeight, SlpLineSpacing, or SlpLineChars properties may cause this property to change.</p> <p>This property is initialized when the device is first enabled following the open method.</p>
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	SlpEmpty Property, SlpNearEnd Property.

SlpLineSpacing Property

Syntax	SlpLineSpacing: <i>int32</i> { read-write, access after open-claim-enable }
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 MapMode.</p> <p>If changed to a spacing that can be supported by the printer, then the line spacing is set to this value. If the spacing cannot be supported, then the spacing is set to the closest supported value.</p> <p>When SlpLineChars or SlpLineHeight 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 open method.</p>
Errors	A UpoException may be thrown when this property is accessed. For further information, see "Errors" on page 15.

SlpLineWidth Property

Syntax	SlpLineWidth: <i>int32</i> { read-only, access after open-claim-enable }
Remarks	<p>Holds the width of a line of SlpLineChars characters, expressed in the unit of measure given by MapMode.</p> <p>Setting SlpLineChars 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 open method.</p>
Errors	A UpoException may be thrown when this property is accessed. For further information, see "Errors" on page 15.

SlpMaxLines Property

Syntax	SlpMaxLines: <i>int32</i> { read-only, access after open-claim-enable }
Remarks	<p>Holds the maximum number of lines that can be printed on a form.</p> <p>When CapSlpFullslip is true, then this property will be zero, indicating an unlimited maximum slip length. When CapSlpFullslip is false, then this value will be non-zero.</p> <p>Changing the SlpLineHeight, SlpLineSpacing, or SlpLineChars properties may cause this property to change.</p> <p>This property is initialized when the device is first enabled following the open method.</p>
Errors	A UpoException may be thrown when this property is accessed. For further information, see "Errors" on page 15.

SlpNearEnd Property

Syntax	SlpNearEnd: <i>boolean</i> { read-only, access after open-claim-enable }
Remarks	<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 CapSlpNearEndSensor 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>Note</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/>
Errors	A <code>UposException</code> may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	SlpEmpty Property, SlpLinesNearEndToEnd Property.

SlpPrintSide Property***Added in Release 1.5***

Syntax	SlpPrintSide: <i>int32</i> { read-only, access after open-claim-enable }								
Remarks	<p>This property holds the current side of the slip document on which printing will occur.</p> <p>If the Slip is not selected, the value of the property is PTR_PS_UNKNOWN.</p> <p>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.</p> <p>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.</p> <p>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.</p> <p>It contains one of the following values:</p> <table> <tr> <th>Value</th><th>Meaning</th></tr> <tr> <td>PTR_PS_UNKNOWN</td><td>Slip is not inserted.</td></tr> <tr> <td>PTR_PS_SIDE1</td><td>Default Print side. (After slip paper insertion, printer can print this side immediately.)</td></tr> <tr> <td>PTR_PS_SIDE2</td><td>The other side of the document to print on. (Reverse side of default.)</td></tr> </table> <p>This property is initialized and kept current while the device is enabled.</p>	Value	Meaning	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.)
Value	Meaning								
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.)								
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.								
See Also	CapSlpBothSidesPrint Property, changePrintSide Method.								

SlpSidewaysMaxChars Property

Syntax	SlpSidewaysMaxChars: <i>int32</i> { read-only, access after open-claim-enable }
Remarks	<p>Holds the maximum number of characters that may be printed on each line in sideways mode.</p> <p>If CapSlpLeft90 and CapSlpRight90 are both false, then this property is zero.</p> <p>Changing the properties SlpLineHeight, SlpLineSpacing, and SlpLineChars may cause this property to change.</p> <p>This property is initialized when the device is first enabled following the open method.</p>
Errors	A <code>UposException</code> may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	SlpSidewaysMaxLines Property.

SlpSidewaysMaxLines Property

Syntax	SlpSidewaysMaxLines: <i>int32</i> { read-only, access after open-claim-enable }
Remarks	<p>Holds the maximum number of lines that may be printed in sideways mode.</p> <p>If CapSlpLeft90 and CapSlpRight90 are both false, then this property is zero.</p> <p>Changing the properties SlpLineHeight, SlpLineSpacing, and SlpLineChars may cause this property to change.</p> <p>This property is initialized when the device is first enabled following the open method.</p>
Errors	A <code>UposException</code> may be thrown when this property is accessed. For further information, see “Errors” on page 15.
See Also	SlpSidewaysMaxChars Property.

Methods (UML operations)

beginInsertion Method

Syntax **beginInsertion (timeout: *int32*):**
 void { raises-exception, use after open-claim-enable }

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 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, an exception is raised. 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, an exception is raised with an *ErrorCode* of 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 UposException 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
E_BUSY	Cannot perform request while output is in progress.
E_ILLEGAL	The slip station does not exist (see the CapSlpPresent property) or an invalid <i>timeout</i> parameter was specified.
E_TIMEOUT	The specified time has elapsed without the form being properly inserted.

See Also **endInsertion** Method, **beginRemoval** Method, **endRemoval** Method.

beginRemoval Method

Syntax **beginRemoval (timeout: *int32*):**
 void { raises-exception, use after open-claim-enable }

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 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, an exception is raised. 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, an exception is raised with an *ErrorCode* of 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 UposException 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
E_BUSY	Cannot perform request while output is in progress.
E_ILLEGAL	The slip station does not exist (see the CapSlpPresent property) or an invalid <i>timeout</i> parameter was specified.
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 **changePrintSide (side: *int32*):**
 void { raises-exception, use after open-claim-enable }

The *side* parameter indicates the side on which to print. Valid values are:

Value	Description
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 beginInsertion/endInsertion 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 SlpPrintSide 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 UpoException 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
E_BUSY	Cannot be performed while output is in progress. (Can only apply if AsyncMode is false.)
E_ILLEGAL	One of the following errors occurred: * The slip station does not exist (see the CapSlpPresent property) * the printer does not support both sides printing (see the CapSlpBothSidesPrint property) * an invalid <i>side</i> parameter was specified
E_EXTENDED	<i>ErrorCodeExtended</i> = EPTR_COVER_OPEN:

The printer cover is open.

(Can only apply if **AsyncMode** is false.)

ErrorCodeExtended = EPTR_SLP_EMPTY:

The slip station was specified, but a form is not inserted.

(Can only apply if **AsyncMode** is false.)

See Also **CapSlpBothSidesPrint** Property, **CapSlpPresent** Property, **SlpPrintSide** Property, **cutPaper** Method.

cutPaper Method

Syntax **cutPaper (percentage: *int32*):**
 void { raises-exception, use after open-claim-enable }

Parameter	Description
<i>percentage</i>	The percentage of paper to cut.
	The constant identifier PTR_CP_FULLCUT or the value 100 causes a full paper cut. Other values request a partial cut percentage.

Remarks 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:

Value	Meaning
100	Full cut.
90	Leave only a small portion of paper for very easy final separation.
70	Perforate the paper for final separation that is somewhat more difficult and unlikely to occur by accidental handling.
50	Partial perforation of the paper.

The Service 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 `UposException` 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
E_BUSY	Cannot perform while output is in progress. (Can only apply if AsyncMode is false.)
E_ILLEGAL	An invalid percentage was specified, the receipt station does not exist (see the CapRecPresent property), or the receipt printer does not have paper cutting ability (see the CapRecPapercut property).
E_EXTENDED	<i>ErrorCodeExtended</i> = EPTR_COVER_OPEN: The printer cover is open. (Can only apply if AsyncMode is false.) <i>ErrorCodeExtended</i> = EPTR_REC_EMPTY: The receipt station is out of paper. (Can only apply if AsyncMode is false.)

See Also “Data Characters and Escape Sequences” on page 316.

endInsertion Method

Syntax	endInsertion (): void { raises-exception, use after open-claim-enable }								
Remarks	<p>Ends form insertion processing.</p> <p>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, an exception is raised with its <i>ErrorCodeExtended</i> property set to EPTR_SLP_EMPTY.</p> <p>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.</p>								
Errors	<p>A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 15.</p> <p>Some possible values of the exception’s <i>ErrorCode</i> property are:</p> <table> <tr> <th>Value</th><th>Meaning</th></tr> <tr> <td>E_BUSY</td><td>Cannot perform request while output is in progress.</td></tr> <tr> <td>E_ILLEGAL</td><td>The printer is not in slip insertion mode.</td></tr> <tr> <td>E_EXTENDED</td><td> <i>ErrorCodeExtended</i> = EPTR_COVER_OPEN: The device was taken out of insertion mode while the printer cover was open. <i>ErrorCodeExtended</i> = EPTR_SLP_EMPTY: The device was taken out of insertion mode without a form being inserted. </td></tr> </table>	Value	Meaning	E_BUSY	Cannot perform request while output is in progress.	E_ILLEGAL	The printer is not in slip insertion mode.	E_EXTENDED	<i>ErrorCodeExtended</i> = EPTR_COVER_OPEN: The device was taken out of insertion mode while the printer cover was open. <i>ErrorCodeExtended</i> = EPTR_SLP_EMPTY: The device was taken out of insertion mode without a form being inserted.
Value	Meaning								
E_BUSY	Cannot perform request while output is in progress.								
E_ILLEGAL	The printer is not in slip insertion mode.								
E_EXTENDED	<i>ErrorCodeExtended</i> = EPTR_COVER_OPEN: The device was taken out of insertion mode while the printer cover was open. <i>ErrorCodeExtended</i> = EPTR_SLP_EMPTY: The device was taken out of insertion mode without a form being inserted.								
See Also	beginInsertion Method, beginRemoval Method, endRemoval Method.								

endRemoval Method

Syntax	endRemoval (): void { raises-exception, use after open-claim-enable }								
Remarks	<p>Ends form removal processing.</p> <p>When called, the printer is taken out of form removal or ejection mode. If a form is present, an exception is raised with its <i>ErrorCodeExtended</i> property set to EPTR_SLP_FORM.</p> <p>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.</p>								
Errors	<p>A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 15.</p> <p>Some possible values of the exception’s <i>ErrorCode</i> property are:</p> <table> <tr> <th>Value</th><th>Meaning</th></tr> <tr> <td>E_BUSY</td><td>Cannot perform request while output is in progress.</td></tr> <tr> <td>E_ILLEGAL</td><td>The printer is not in slip removal mode.</td></tr> <tr> <td>E_EXTENDED</td><td><i>ErrorCodeExtended</i> = EPTR_SLP_FORM: The device was taken out of removal mode while a form was still present.</td></tr> </table>	Value	Meaning	E_BUSY	Cannot perform request while output is in progress.	E_ILLEGAL	The printer is not in slip removal mode.	E_EXTENDED	<i>ErrorCodeExtended</i> = EPTR_SLP_FORM: The device was taken out of removal mode while a form was still present.
Value	Meaning								
E_BUSY	Cannot perform request while output is in progress.								
E_ILLEGAL	The printer is not in slip removal mode.								
E_EXTENDED	<i>ErrorCodeExtended</i> = EPTR_SLP_FORM: 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 **markFeed (type: *int32*):**
 void { raises-exception, use after open-claim-enable }

The *type* parameter indicates the type of mark sensed paper handling. Valid values are:

Value	Description
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.

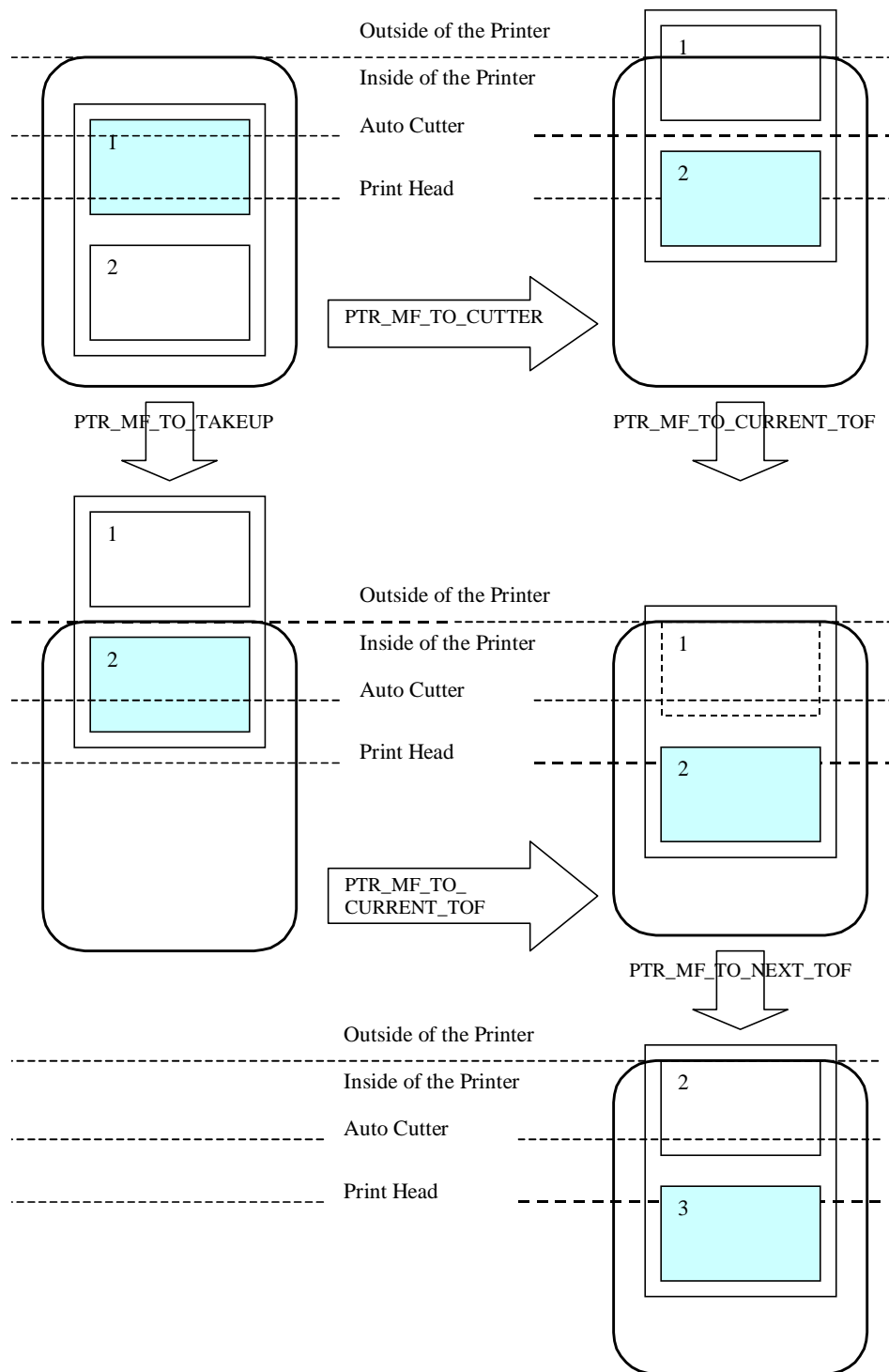
If *type* is PTR_MF_TO_TAKEUP, the printer will feed the mark sensed paper so that the present form is moved so that it can be manually removed by the operator.

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	<p>A UposException may be thrown when this method is invoked. For further information, see “Errors” on page 15.</p> <p>Some possible values of the exception’s <i>ErrorCode</i> property are:</p> <table> <tr> <th data-bbox="475 401 548 428">Value</th><th data-bbox="764 401 870 428">Meaning</th></tr> <tr> <td data-bbox="475 459 581 487">E_BUSY</td><td data-bbox="764 459 1385 525">Cannot be performed while output is in progress. (Can only apply if AsyncMode is false.)</td></tr> <tr> <td data-bbox="475 535 626 562">E_ILLEGAL</td><td data-bbox="764 535 1385 630">The receipt print station does not support the given mark sensed paper handling function. (Refer to the CapRecMarkFeed property)</td></tr> <tr> <td data-bbox="475 640 656 667">E_EXTENDED</td><td data-bbox="764 640 1385 840"> <p><i>ErrorCodeExtended</i> = EPTR_COVER_OPEN: The printer cover is open. (Can only apply if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EPTR_REC_EMPTY: The receipt paper is empty. (Can only apply if AsyncMode is false.)</p> </td></tr> </table>	Value	Meaning	E_BUSY	Cannot be performed while output is in progress. (Can only apply if AsyncMode is false.)	E_ILLEGAL	The receipt print station does not support the given mark sensed paper handling function. (Refer to the CapRecMarkFeed property)	E_EXTENDED	<p><i>ErrorCodeExtended</i> = EPTR_COVER_OPEN: The printer cover is open. (Can only apply if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EPTR_REC_EMPTY: The receipt paper is empty. (Can only apply if AsyncMode is false.)</p>
Value	Meaning								
E_BUSY	Cannot be performed while output is in progress. (Can only apply if AsyncMode is false.)								
E_ILLEGAL	The receipt print station does not support the given mark sensed paper handling function. (Refer to the CapRecMarkFeed property)								
E_EXTENDED	<p><i>ErrorCodeExtended</i> = EPTR_COVER_OPEN: The printer cover is open. (Can only apply if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EPTR_REC_EMPTY: The receipt paper is empty. (Can only apply if AsyncMode is false.)</p>								
See Also	CapRecMarkFeed Property.								

printBarCode Method

Syntax **printBarCode** (*station: int32*, *data: string*, *symbolology: int32*, *height: int32*, *width: int32*, *alignment: int32*, *textPosition: int32*):
void { raises-exception, use after open-claim-enable }

Parameter	Description
<i>station</i>	The printer station to be used. May be either PTR_S_RECEIPT or PTR_S_SLIP.
<i>data</i>	Character string to be bar coded.
<i>symbolology</i>	Bar code symbol type to use. See values below.
<i>height</i>	Bar code height. Expressed in the unit of measure given by MapMode .
<i>width</i>	Bar code width. Expressed in the unit of measure given by MapMode .
<i>alignment</i>	Placement of the bar code. See values below.
<i>textPosition</i>	Placement of the readable character string. See values below.

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

Value	Meaning
PTR_BC_LEFT	Align with the left-most print column.
PTR_BC_CENTER	Align in the center of the station.
PTR_BC_RIGHT	Align with the right-most print column.
<i>Other Values</i>	Distance from the left-most print column to the start of the bar code. Expressed in the unit of measure given by MapMode .

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

Value	Meaning
PTR_BC_TEXT_NONE	No text is printed. Only print the bar code.
PTR_BC_TEXT_ABOVE	Print the text above the bar code.
PTR_BC_TEXT_BELOW	Print the text below the bar code.

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

Value	Meaning
<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 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 UpoException 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
E_ILLEGAL	One of the following parameter 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, or The RotateSpecial rotation is not supported
E_BUSY	Cannot perform while output is in progress. (Can only apply if AsyncMode is false.)
E_EXTENDED	<i>ErrorCodeExtended</i> = EPTR_COVER_OPEN: The printer cover is open. (Can only apply if AsyncMode is false.) <i>ErrorCodeExtended</i> = EPTR_REC_EMPTY: The receipt station was specified but is out of paper. (Can only apply if AsyncMode is false.) <i>ErrorCodeExtended</i> = EPTR_REC_CARTRIDGE_REMOVED: A receipt cartridge has been removed. (Can only apply if AsyncMode is false.) <i>ErrorCodeExtended</i> = EPTR_REC_CARTRIDGE_EMPTY: A receipt cartridge is empty. (Can only apply if AsyncMode is false.) <i>ErrorCodeExtended</i> = EPTR_REC_HEAD_CLEANING: A receipt cartridge head is being cleaned.

(Can only apply if **AsyncMode** is false.)

ErrorCodeExtended = EPTR_SLP_EMPTY:

The slip station was specified, but a form is not inserted.

(Can only apply if **AsyncMode** is false.)

ErrorCodeExtended =

EPTR_SLP_CARTRIDGE_REMOVED:

A slip cartridge has been removed.

(Can only apply if **AsyncMode** is false.)

ErrorCodeExtended =

EPTR_SLP_CARTRIDGE_EMPTY:

A slip cartridge is empty.

(Can only apply if **AsyncMode** is false.)

ErrorCodeExtended =

EPTR_SLP_HEAD_CLEANING:

A slip cartridge head is being cleaned.

(Can only apply if **AsyncMode** is false.)

printBitmap Method

Syntax **printBitmap (station: *int32*, fileName: *string*, width: *int32*, alignment: *int32*):**
 void { raises-exception, use after open-claim-enable }

Parameter	Description
<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:

Value	Meaning
PTR_BM_ASIS	Print the bitmap with one bitmap pixel per printer dot.
<i>Other Values</i>	Bitmap width expressed in the unit of measure given by MapMode .

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

Value	Meaning
PTR_BM_LEFT	Align with the left-most print column.
PTR_BM_CENTER	Align in the center of the station.
PTR_BM_RIGHT	Align with the right-most print column.
<i>Other Values</i>	Distance from the left-most print column to the start of the bitmap. Expressed in the unit of measure given by MapMode .

Remarks 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 UpoException 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
E_BUSY	Cannot perform while output is in progress. (Can only apply if AsyncMode is false.)
E_ILLEGAL	One of the following parameter 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

E_NOEXIST	<i>fileName</i> was not found.
E_EXTENDED	<p><i>ErrorCodeExtended</i> = EPTR_TOOBIG: The bitmap is either too wide to print without transformation, or it is too big to transform.</p> <p><i>ErrorCodeExtended</i> = EPTR_COVER_OPEN: The printer cover is open. (Can only apply if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EPTR_BADFORMAT: The specified file is either not a bitmap file, or it is in an unsupported format.</p> <p><i>ErrorCodeExtended</i> = EPTR_REC_EMPTY: The receipt station was specified but is out of paper. (Can only apply if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EPTR_REC_CARTRIDGE_REMOVED: A receipt cartridge has been removed. (Can only apply if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EPTR_REC_CARTRIDGE_EMPTY: A receipt cartridge is empty. (Can only apply if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EPTR_REC_HEAD_CLEANING: A receipt cartridge head is being cleaned. (Can only apply if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EPTR_SLP_EMPTY: The slip station was specified, but a form is not inserted. (Can only apply if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EPTR_SLP_CARTRIDGE_REMOVED: A slip cartridge has been removed. (Can only apply if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EPTR_SLP_CARTRIDGE_EMPTY: A slip cartridge is empty. (Can only apply if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EPTR_SLP_HEAD_CLEANING: A slip cartridge head is being cleaned. (Can only apply if AsyncMode is false.)</p>

printImmediate Method

Syntax **printImmediate (station: *int32*, data: *string*):**
 void { raises-exception, use after open-claim-enable }

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 line feeds (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
Line Feed (10)	Print any data in the line buffer, and feed to the next print line. (A Carriage Return is not required in order to cause the line to be printed.)
Carriage Return (13)	If a Carriage Return immediately precedes a Line Feed, or if the line buffer is empty, then it is ignored. Otherwise, the line buffer is printed and the printer does not feed to the next print line. On some printers, print without feed may be directly supported. On others, a print may always feed to the next line, in which case the Service will print the line buffer and perform a reverse line feed if supported. If the printer does not support either of these features, then Carriage Return acts like a Line Feed. The validateData method may be used to determine whether a Carriage Return without Line Feed is possible, and whether a reverse line feed is required to support it.

Errors A `UpoSException` 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
E_ILLEGAL	The specified <i>station</i> does not exist. (See the CapJrnPresent , CapRecPresent , and CapSlpPresent properties.)
E_EXTENDED	<p><i>ErrorCodeExtended</i> = EPTR_COVER_OPEN: The printer cover is open.</p> <p><i>ErrorCodeExtended</i> = EPTR_JRN_EMPTY: The journal station was specified but is out of paper.</p> <p><i>ErrorCodeExtended</i> = EPTR_JRN_CARTRIDGE_REMOVED: A journal cartridge has been removed. (Can only apply if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EPTR_JRN_CARTRIDGE_EMPTY: A journal cartridge is empty. (Can only apply if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EPTR_JRN_HEAD_CLEANING: A journal cartridge head is being cleaned. (Can only apply if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EPTR_REC_EMPTY: The receipt station was specified but is out of paper. (Can only apply if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EPTR_REC_CARTRIDGE_REMOVED: A receipt cartridge has been removed. (Can only apply if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EPTR_REC_CARTRIDGE_EMPTY: A receipt cartridge is empty. (Can only apply if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EPTR_REC_HEAD_CLEANING: A receipt cartridge head is being cleaned. (Can only apply if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EPTR_SLP_EMPTY: The slip station was specified, but a form is not inserted. (Can only apply if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EPTR_SLP_CARTRIDGE_REMOVED: A slip cartridge has been removed. (Can only apply if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EPTR_SLP_CARTRIDGE_EMPTY: A slip cartridge is empty. (Can only apply if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EPTR_SLP_HEAD_CLEANING: A slip cartridge head is being cleaned. (Can only apply if AsyncMode is false.)</p>

See Also **printNormal** Method, **printTwoNormal** Method.

printNormal Method

Syntax **printNormal (station: *int32*, data: *string*):**
 void { raises-exception, use after open-claim-enable }

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 line feeds (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
Line Feed (10)	Print any data in the line buffer, and feed to the next print line. (A Carriage Return is not required in order to cause the line to be printed.)
Carriage Return (13)	<p>If a Carriage Return immediately precedes a Line Feed, or if the line buffer is empty, then it is ignored.</p> <p>Otherwise, the line buffer is printed and the printer does not feed to the next print line. On some printers, print without feed may be directly supported. On others, a print may always feed to the next line, in which case the Service 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.</p> <p>The validateData method may be used to determine whether a Carriage Return without Line Feed is possible, and whether a reverse line feed is required to support it.</p>

Errors A `UposException` 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
E_ILLEGAL	The specified <i>station</i> does not exist. (See the CapJrnPresent , CapRecPresent , and CapSlpPresent properties.)
E_BUSY	Cannot perform while output is in progress.(Can only apply if AsyncMode is false.)
E_EXTENDED	<p><i>ErrorCodeExtended</i> = EPTR_COVER_OPEN: The printer cover is open.</p> <p><i>ErrorCodeExtended</i> = EPTR_JRN_EMPTY: The journal station was specified but is out of paper.</p> <p><i>ErrorCodeExtended</i> = EPTR_JRN_CARTRIDGE_REMOVED: A journal cartridge has been removed. (Can only apply if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EPTR_JRN_CARTRIDGE_EMPTY: A journal cartridge is empty. (Can only apply if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EPTR_JRN_HEAD_CLEANING: A journal cartridge head is being cleaned. (Can only apply if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EPTR_REC_EMPTY: The receipt station was specified but is out of paper. (Can only apply if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EPTR_REC_CARTRIDGE_REMOVED: A receipt cartridge has been removed. (Can only apply if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EPTR_REC_CARTRIDGE_EMPTY: A receipt cartridge is empty. (Can only apply if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EPTR_REC_HEAD_CLEANING: A receipt cartridge head is being cleaned. (Can only apply if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EPTR_SLP_EMPTY: The slip station was specified, but a form is not inserted. (Can only apply if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EPTR_SLP_CARTRIDGE_REMOVED: A slip cartridge has been removed. (Can only apply if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EPTR_SLP_CARTRIDGE_EMPTY: A slip cartridge is empty. (Can only apply if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EPTR_SLP_HEAD_CLEANING: A slip cartridge head is being cleaned. (Can only apply if AsyncMode is false.)</p>

See Also **printImmediate** Method, **printTwoNormal** Method.

printTwoNormal Method

Syntax **printTwoNormal (stations: *int32*, data1: *string*, data2: *string*):**
 void { raises-exception, use after open-claim-enable }

Parameter	Description
-----------	-------------

<i>stations</i>	<p>Release 1.2 The printer stations to be used may be: PTR_S_JOURNAL_RECEIPT, PTR_S_JOURNAL_SLIP, or PTR_S_RECEIPT_SLIP.</p> <p>Release 1.3 and later: Select one of the following:</p> <table border="1"> <thead> <tr> <th><i>stations</i> Parameter</th> <th>First Station</th> <th>Second Station</th> </tr> </thead> <tbody> <tr> <td>PTR_TWO_RECEIPT_JOURNAL</td> <td>Receipt</td> <td>Journal</td> </tr> <tr> <td>PTR_TWO_SLIP_JOURNAL</td> <td>Slip</td> <td>Journal</td> </tr> <tr> <td>PTR_TWO_SLIP_RECEIPT</td> <td>Slip</td> <td>Receipt</td> </tr> </tbody> </table>	<i>stations</i> Parameter	First Station	Second Station	PTR_TWO_RECEIPT_JOURNAL	Receipt	Journal	PTR_TWO_SLIP_JOURNAL	Slip	Journal	PTR_TWO_SLIP_RECEIPT	Slip	Receipt
<i>stations</i> Parameter	First Station	Second Station											
PTR_TWO_RECEIPT_JOURNAL	Receipt	Journal											
PTR_TWO_SLIP_JOURNAL	Slip	Journal											
PTR_TWO_SLIP_RECEIPT	Slip	Receipt											
<i>data1</i>	The characters to be printed on the first station. May consist of printable characters and escape sequences as listed in the “Print Line” table under “Data Characters and Escape Sequences” on page 316. The characters must all fit on one printed line, so that the printer may attempt to print on both stations simultaneously.												
<i>data2</i>	The characters to be printed on the second station. (Restrictions are the same as for <i>data1</i> .) If this string is the empty string (“”), then print the same data as <i>data1</i> . On some printers, using this format may give additional increased print performance.												

Release 1.2

The printer stations to be used may be:

PTR_S_JOURNAL_RECEIPT, PTR_S_JOURNAL_SLIP, or
 PTR_S_RECEIPT_SLIP.

Release 1.3 and later:

Select one of the following:

<i>stations</i> Parameter	First Station	Second Station
PTR_TWO_RECEIPT_JOURNAL	Receipt	Journal
PTR_TWO_SLIP_JOURNAL	Slip	Journal
PTR_TWO_SLIP_RECEIPT	Slip	Receipt

The characters to be printed on the first station. May consist of printable characters and escape sequences as listed in the “Print Line” table under “Data Characters and Escape Sequences” on page 316. The characters must all fit on one printed line, so that the printer may attempt to print on both stations simultaneously.

The characters to be printed on the second station. (Restrictions are the same as for *data1*.) If this string is the empty string (“”), then print the same data as *data1*. On some printers, using this format may give additional increased print performance.

Remarks Prints two strings on two print stations simultaneously. When supported, this may give increased print performance.

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

Release 1.2

Documentation release 1.2 was not sufficiently clear as to the meaning of “first” and “second” station so Service implementations varied between the following:

- Assign stations based on order within the constants. For example, PTR_S_JOURNAL_RECEIPT prints *Data1* on the journal and *Data2* on the receipt.
- Assign stations based upon physical device characteristics or internal print order.

Due to this inconsistency, the application should use the new constants if the Control and Service versions indicate Release 1.3 or later.

Release 1.3 and later

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 `UposException` 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
E_ILLEGAL	The specified <i>stations</i> do not support concurrent printing. (See the CapConcurrentJrnRec , CapConcurrentJrnSlp , and CapConcurrentRecSlp properties.)
E_BUSY	Cannot perform while output is in progress. (Can only apply if AsyncMode is false.)
E_EXTENDED	<p><i>ErrorCodeExtended</i> = EPTR_COVER_OPEN: The printer cover is open.</p> <p><i>ErrorCodeExtended</i> = EPTR_JRN_EMPTY: The journal station was specified but is out of paper.</p> <p><i>ErrorCodeExtended</i> = EPTR_JRN_CARTRIDGE_REMOVED: A journal cartridge has been removed. (Can only apply if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EPTR_JRN_CARTRIDGE_EMPTY: A journal cartridge is empty. (Can only apply if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EPTR_JRN_HEAD_CLEANING: A journal cartridge head is being cleaned. (Can only apply if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EPTR_REC_EMPTY: The receipt station was specified but is out of paper. (Can only apply if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EPTR_REC_CARTRIDGE_REMOVED: A receipt cartridge has been removed. (Can only apply if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EPTR_REC_CARTRIDGE_EMPTY: A receipt cartridge is empty. (Can only apply if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EPTR_REC_HEAD_CLEANING: A receipt cartridge head is being cleaned. (Can only apply if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EPTR_SLP_EMPTY: The slip station was specified, but a form is not inserted. (Can only apply if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EPTR_SLP_CARTRIDGE_REMOVED: A slip cartridge has been removed. (Can only apply if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EPTR_SLP_CARTRIDGE_EMPTY: A slip cartridge is empty. (Can only apply if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EPTR_SLP_HEAD_CLEANING: A slip cartridge head is being cleaned. (Can only apply if AsyncMode is false.)</p>

See Also

printNormal Method

rotatePrint Method

Syntax **rotatePrint (station: *int32*, rotation: *int32*):**
 void { raises-exception, use after open-claim-enable }

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 *rotation* 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 Service, depending on the printer capabilities) until **rotatePrint** is called with *rotation* 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 `UposException` 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
E_ILLEGAL	The specified <i>station</i> does not exist (see the CapJrnPresent , CapRecPresent , and CapSlpPresent properties), or the <i>station</i> does not support the specified rotation (see the station’s rotation capability properties).
E_BUSY	Cannot perform while output is in progress. (Can only apply if AsyncMode is false.)
E_EXTENDED	<p><i>ErrorCodeExtended</i> = EPTR_COVER_OPEN: The printer cover is open. (Can only apply if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EPTR_REC_EMPTY: The receipt station was specified but is out of paper. (Can only apply if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EPTR_REC_CARTRIDGE_REMOVED: A receipt cartridge has been removed. (Can only apply if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EPTR_REC_CARTRIDGE_EMPTY: A receipt cartridge is empty. (Can only apply if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EPTR_REC_HEAD_CLEANING: A receipt cartridge head is being cleaned. (Can only apply if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EPTR_SLP_EMPTY: The slip station was specified, but a form is not inserted. (Can only apply if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EPTR_SLP_CARTRIDGE_REMOVED: A slip cartridge has been removed. (Can only apply if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EPTR_SLP_CARTRIDGE_EMPTY: A slip cartridge is empty. (Can only apply if AsyncMode is false.)</p> <p><i>ErrorCodeExtended</i> = EPTR_SLP_HEAD_CLEANING: A slip cartridge head is being cleaned. (Can only apply if AsyncMode is false.)</p>

See Also “Data Characters and Escape Sequences” on page 316.

setBitmap Method

Syntax **setBitmap (bitmapNumber: *int32*, station: *int32*, fileName: *string*, width: *int32*, alignment: *int32*):**
void { raises-exception, use after open-claim-enable }

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 printBitmap for values.
<i>alignment</i>	Placement of the bitmap. See printBitmap 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 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 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 `UposException` 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
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 * <i>width</i> is too big * <i>alignment</i> is invalid or too big
E_NOEXIST	<i>fileName</i> was not found.
E_EXTENDED	<i>ErrorCodeExtended</i> = EPTR_TOOBIG: The bitmap is either too wide to print without transformation, or it is too big to transform. <i>ErrorCodeExtended</i> = 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” on page 316, **printBitmap** Method.

setLogo Method

Syntax **setLogo** (*location*: *int32*, *data*: *string*):
 void { raises-exception, use after open-claim-enable }

Parameter	Description
<i>location</i>	The logo to be set. May be PTR_L_TOP or PTR_L_BOTTOM.
<i>data</i>	The characters that produce the logo. May consist of printable characters, escape sequences, carriage returns (13 decimal), and line feeds (10 decimal).

Remarks 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 `UposException` 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
E_ILLEGAL	An invalid <i>location</i> was specified.

See Also “Data Characters and Escape Sequences” on page 316.

transactionPrint Method

Syntax **transactionPrint (station: *int32*, control: *int32*):**
 void { raises-exception, use after open-claim-enable }

Parameter	Description
<i>station</i>	The printer station to be used. May be PTR_S_JOURNAL, PTR_S_RECEIPT, or PTR_S_SLIP.
<i>control</i>	Transaction control. See values below:
Value	Meaning
PTR_TP_TRANSACTION	Begin a transaction.
PTR_TP_NORMAL	End a transaction by printing the buffered data.

Remarks Enters or exits transaction mode.

If *control* is PTR_TP_TRANSACTION, then transaction mode is entered. Subsequent calls to **printNormal**, **cutPaper**, **rotatePrint**, **printBarCode**, and **printBitmap** will buffer the print data (either at the printer or the Service, 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 enqueued.)

If *control* is PTR_TP_NORMAL, then transaction mode is exited. If some data was buffered by calls to the methods **printNormal**, **cutPaper**, **rotatePrint**, **printBarCode**, and **printBitmap**, then the buffered data is printed. The entire transaction is treated as one message. This method is performed synchronously if **AsyncMode** is false, and asynchronously if **AsyncMode** is true.

Calling the **clearOutput** method cancels transaction mode. Any buffered print lines are also cleared.

Errors A UposException 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
E_ILLEGAL	The specified <i>station</i> does not exist (see the CapJrnPresent , CapRecPresent , and CapSlpPresent properties), or CapTransaction is false.
E_BUSY	Cannot perform while output is in progress. (Can only apply if AsyncMode is false and <i>control</i> is PTR_TP_NORMAL.)

E_EXTENDED *ErrorCodeExtended* = EPTR_COVER_OPEN:
The printer cover is open.
(Can only apply if **AsyncMode** is false and *control* is PTR_TP_NORMAL.)
ErrorCodeExtended = EPTR_JRN_EMPTY:
The journal station was specified but is out of paper.
ErrorCodeExtended = EPTR_JRN_CARTRIDGE_REMOVED:
A journal cartridge has been removed.
(Can only apply if **AsyncMode** is false.)
ErrorCodeExtended = EPTR_JRN_CARTRIDGE_EMPTY:
A journal cartridge is empty.
(Can only apply if **AsyncMode** is false.)
ErrorCodeExtended = EPTR_JRN_HEAD_CLEANING:
A journal cartridge head is being cleaned.
(Can only apply if **AsyncMode** is false.)
ErrorCodeExtended = EPTR_REC_EMPTY:
The receipt station was specified but is out of paper.
(Can only apply if **AsyncMode** is false.)
ErrorCodeExtended = EPTR_REC_CARTRIDGE_REMOVED:
A receipt cartridge has been removed.
(Can only apply if **AsyncMode** is false.)
ErrorCodeExtended = EPTR_REC_CARTRIDGE_EMPTY:
A receipt cartridge is empty.
(Can only apply if **AsyncMode** is false.)
ErrorCodeExtended = EPTR_REC_HEAD_CLEANING:
A receipt cartridge head is being cleaned.
(Can only apply if **AsyncMode** is false.)
ErrorCodeExtended = EPTR_SLP_EMPTY:
The slip station was specified, but a form is not inserted.
(Can only apply if **AsyncMode** is false.)
ErrorCodeExtended = EPTR_SLP_CARTRIDGE_REMOVED:
A slip cartridge has been removed.
(Can only apply if **AsyncMode** is false.)
ErrorCodeExtended = EPTR_SLP_CARTRIDGE_EMPTY:
A slip cartridge is empty.
(Can only apply if **AsyncMode** is false.)
ErrorCodeExtended = EPTR_SLP_HEAD_CLEANING:
A slip cartridge head is being cleaned.
(Can only apply if **AsyncMode** is false.)

validateData Method

Syntax **validateData (station: *int32*, data: *string*):**
 void { raises-exception, use after open-claim-enable }

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 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 UpoException 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
E_ILLEGAL	Some of the data is not precisely supported by the printer station, but the Service can select valid alternatives.
E_FAILURE	Some of the data is not supported. No alternatives can be selected.

Cases which cause *ErrorCode* of E_ILLEGAL:

Escape Sequence	Condition
Paper cut	The percentage ‘#’ is not precisely supported: Service will select the closest supported value.
Feed and Paper cut	The percentage ‘#’ is not precisely supported: Service will select the closest supported value.
Feed, Paper cut, and Stamp	The percentage ‘#’ is not precisely supported: Service will select the closest supported value.
Feed units	The unit count ‘#’ is not precisely supported: Service will select the closest supported value.
Feed reverse	The line count ‘#’ is too large: Service will select the maximum supported value.
Underline	The thickness ‘#’ is not precisely supported: Service will select the closest supported value.
Shading	The percentage ‘#’ is not precisely supported: Service will select the closest supported value.
Scale horizontally	The scaling factor ‘#’ is not supported: Service will select the closest supported value.
Scale vertically	The scaling factor ‘#’ is not supported: Service will select the closest supported value.

Alternate Color	The color '#' is not supported: Service will select the closest supported value.
RGB Color	The color '#' is not supported: Service will select the closest supported value.
Data	Condition

<i>data1</i> CR <i>data2</i> LF	(Where CR is a Carriage Return and LF is a Line Feed) In order to print data <i>data1</i> and remain on the same line, the Service will print with a line advance, then perform a reverse line feed. The data <i>data2</i> will then overprint <i>data1</i> .
---	--

Cases which will cause *ErrorCode* of 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.
Alternate color	Not supported.
RGB color	Not supported.
Reverse video	Not supported.
SubScript	Not supported.
SuperScript	Not supported.
Shading	Not supported.
Single high & wide	Not supported.
Double wide	Not supported.
Double high	Not supported.
Double high & wide	Not supported.
Data	Condition

<i>data1</i> CR <i>data2</i> LF	(Where CR is a Carriage Return and LF is a Line Feed) Not able to print data and remain on the same line. The data <i>data1</i> will print on one line, and the data <i>data2</i> will print on the next line.
---	---

See Also "Data Characters and Escape Sequences" on page 316.

Events (UML interfaces)

DirectIOEvent

<< event >> **upos::events::DirectIOEvent**

EventNumber: *int32* { read-only }

Data: *int32* { read-write }

Obj: *object* { read-write }

Description Provides Service information directly to the application. This event provides a means for a vendor-specific POS Printer 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>int32</i>	Event number whose specific values are assigned by the Service.
<i>Data</i>	<i>int32</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.

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 POS Printer devices which may not have any knowledge of the Service's need for this event.

See Also "Events" on page 14, **directIO** Method.

ErrorEvent

```
<< event >> upos::events::ErrorEvent
    ErrorCode: int32 { read-only }
    ErrorCodeExtended: int32 { read-only }
    ErrorLocus: int32 { read-only }
    ErrorResponse: int32 { read-write }
```

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

Attributes This event contains the following properties:

Attributes	Type	Description
<i>ErrorCode</i>	<i>int32</i>	Result code causing the error event. See a list of Error Codes on page 15.
<i>ErrorCodeExtended</i>	<i>int32</i>	Extended Error code causing the error event. If <i>ErrorCode</i> is E_EXTENDED, then see values below. Otherwise, it may contain a Service-specific value.
<i>ErrorLocus</i>	<i>int32</i>	Location of the error, and is set to EL_OUTPUT indicating that the error occurred while processing asynchronous output.
<i>ErrorResponse</i>	<i>int32</i>	Error response, whose default value may be overridden by the application (i.e., this property is settable). See values below.

If *ErrorCode* is E_EXTENDED, then *ErrorCodeExtended* has one of the following values:

Value	Meaning
EPTR_COVER_OPEN	The printer cover is open.
EPTR_JRN_EMPTY	The journal station is out of paper.
EPTR_REC_EMPTY	The receipt station is out of paper.
EPTR_SLP_EMPTY	A form is not inserted in the slip station.
EPTR_JRN_CARTRIDGE_REMOVED:	A journal cartridge has been removed.
EPTR_JRN_CARTRIDGE_EMPTY:	A journal cartridge is empty.
EPTR_JRN_HEAD_CLEANING:	A journal cartridge head is being cleaned.
EPTR_REC_CARTRIDGE_REMOVED:	A receipt cartridge has been removed.
EPTR_REC_CARTRIDGE_EMPTY:	A receipt cartridge is empty.
EPTR_REC_HEAD_CLEANING:	A receipt cartridge head is being cleaned.

EPTR_SLP_CARTRIDGE_REMOVED:

A slip cartridge has been removed.

EPTR_SLP_CARTRIDGE_EMPTY:

A slip cartridge is empty.

EPTR_SLP_HEAD_CLEANING:

A slip cartridge head is being cleaned.

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
ER_CLEAR	Clear the asynchronous output or buffered output data. The error state is exited.
ER_RETRY	Retry the asynchronous output. The error state is exited. The default.

Remarks Enqueued when an error is detected and the Service'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 20, "Device States" on page 25

OutputCompleteEvent

<< event >> **upos::events::OutputCompleteEvent**
OutputID: int32 { read-only }

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

Attributes This event contains the following attribute:

Attributes	Type	Description
<i>OutputID</i>	<i>int32</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 Service has confirmation that it was processed by the device successfully.

See Also "Device Output Models" on page 20.

StatusUpdateEvent

<< event >> **upos::events::StatusUpdateEvent**
Status: *int32* { read-only }

Description Notifies the application that a printer has had an operation status change.

Attributes This event contains the following attribute:

Attributes	Type	Description
<i>Status</i>	<i>int32</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 S_IDLE. The FlagWhenIdle 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” on page 56.

Release 1.5 and later – Cartridge State Reporting

If **CartridgeNotify** = PTR_CN_ENABLED, **StatusUpdateEvents** with the following *status* parameter values may be fired.

Value	Meaning
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 14.

CHAPTER 18

Remote Order Display

This Chapter defines the Remote Order Display device category.

General Information

The Remote Order Display programmatic name is “RemoteOrderDisplay”.

This chapter is grandfathered in based on the OPOS and JavaPOS Version 1.4 specifications.

This device had minor revisions for Version 1.5, and only the changes are included in this specification.

Properties (UML attributes)

CharacterSet Property

Updated in Release 1.5

Syntax	CharacterSet: <i>int32</i> { read-write, access after open-claim-enable }												
Remarks	<p>Holds the character set for displaying characters for the video unit indicated by CurrentUnitID. When CapSelectCharacterSet is true, this property can be set to one of the following values:</p> <table><tr><th>Value</th><th>Meaning</th></tr><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>ROD_CS_UNICODE</td><td>The character set supports UNICODE. The value of this constant is 997.</td></tr><tr><td>ROD_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>ROD_CS_ANSI</td><td>The ANSI character set. The value of this constant is 999.</td></tr></table> <p>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.</p> <p>This is updated during the selectCharacterSet method.</p>	Value	Meaning	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.	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.
Value	Meaning												
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.												
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.												
Errors	A UposException may be thrown when this property is accessed. For further information, see “Errors” on page 15.												
See Also	CurrentUnitID Property, CharacterSetList Property, CapSelectCharacterSet Property, selectCharacterSet method.												

CHAPTER 19

Scale

This Chapter defines the Scale device category.

General Information

The Scale programmatic name is “Scale”.

This chapter is grandfathered in based on the OPOS and JavaPOS Version 1.4 specifications.

CHAPTER 20

Scanner (Bar Code Reader)

This Chapter defines the Scanner device category.

General Information

The Scanner programmatic name is “Scanner”.

This chapter is grandfathered in based on the OPOS and JavaPOS Version 1.4 specifications.

CHAPTER 21

Signature Capture

This Chapter defines the Signature Capture device category.

General Information

The Signature Capture programmatic name is “SignatureCapture”.

This chapter is grandfathered in based on the OPOS and JavaPOS Version 1.4 specifications.

CHAPTER 22

Tone Indicator

This Chapter defines the Tone Indicator device category.

General Information

The Tone Indicator programmatic name is “ToneIndicator”.

This chapter is grandfathered in based on the OPOS and JavaPOS Version 1.4 specifications.

Change History

Release Version 1.4

Version 1.4 is the first release of the UnifiedPOS standard. It derives its release version number from the corresponding OPOS and JavaPOS standard version numbers 1.4. In an attempt to prevent confusion, all peripheral device classifications that are present in the version 1.4 standard of OPOS and JavaPOS are “grandfathered” into this first release of UnifiedPOS standard.

The Chapters that are shown in this standard shall be used as guidelines for future peripheral device classifications to be included in subsequent versions of the standards. Therefore, one can be assured that if they have version 1.4 of the UnifiedPOS standard it will be the basis for the version 1.4 of the OPOS or JavaPOS standard. This cross-linking of standard version numbers will be maintained in the future.

Release Version 1.5

Version 1.5 of this specification contains several new chapters (devices) and updates to existing chapters that provide clarifications and corrections to Version 1.4. These are detailed below, with links to the corresponding pages and/or chapters as appropriate.

- Updated the Version and issue date on the front page. See “Version 1.5 July 31, 2000” on pag e1.
- Updated the Table of Contents to reflect additional chapters and headings. “Table of Contents” on page i
- Updated the “Table of extensions to UML for UnifiedPOS.” on page 6.
- Updated the Package Diagram. See “Package Diagram” on page 7.
- Added another condition that causes the Device to exit the Error state. See “The Device exits the Error state when one of the following occurs:” on page 19.
- Updated the Power State Diagram. See “Power State Diagram” on pag e22.
- Updated the Device State Diagram. See “Device State Diagram” on pag e26.
- Updated, throughout the specification, the mutability of the **DirectIOEvent** attributes *Data* and *Obj* to reflect the fact that they are read-write.
- Updated, throughout the specification, the mutability of the **ErrorEvent** attribute *ErrorResponse* to reflect the fact that it is read-write.

- Updated the case of the first letter of all Properties, and Event Attributes to uppercase to make consistent throughout the specification.
- Added the Base Control Class Diagram. See “The following diagram shows the relationships between the Common classes.” on page 32.
- Updated the Event Interfaces Diagram. See “upos::events interfaces” on page 51.
- Updated the Bump Bar chapter header to remove the “example” status. See “Chapter 2 Bump Bar” on page 57.
- Updated the Bump Bar Class Diagram. See “Bump Bar Class Diagram” on page 62.
- Updated the Bump Bar State Diagram. See “Bump Bar State Diagram” on page 66.
- Added a new chapter describing the Cash Changer, including 1.5 specific updates. See “Chapter 3 Cash Changer” on page 83.
- Added a new chapter describing the Cash Drawer, including 1.5 specific updates. See “Chapter 4 Cash Drawer” on page 115.
- Added a new chapter describing the CAT, including 1.5 specific updates. See “Chapter 5 CAT - Credit Authorization Terminal” on page 125.
- Added a new chapter describing the MSR. See “Chapter 12 MSR - Magnetic Stripe Reader” on page 181.
- Updated the MSR chapter to include Track 4 handling for JIS-II type cards. See various additions within the MSR chapter.
- Updated the MSR chapter to include a typical usage sequence diagram. See “MSR Usage Diagram” on page 190.
- Added a new chapter describing the PIN Pad, including 1.5 specific updates. See “Chapter 13 PIN Pad” on page 207.
- Added a new chapter describing the Point Card Reader Writer. See “Chapter 14 Point Card Reader Writer” on page 235.
- Added a new chapter describing the POS Power. See “Chapter 16 POS Power” on page 283.
- Added a new chapter describing the POS Printer. See “Chapter 17 POS Printer” on page 301.
- Updated the POS Printer chapter to include “both sides printing” support, including a new Property, Method, and sequence diagram. See ““Both sides printing” sequence Diagram” on page 315. See “CapSlpBothSidesPrint Property Added in Release 1.5” on page 338. See “changePrintSide Method Added in Release 1.5” on page 369.
- Added a new Appendix C describing Hardware References. See “Appendix C Additional Hardware References” on page C-1.
- Made minor typographical and formatting changes as necessary.

A P P E N D I X B

Additional Software References

UML References

The following additional is a list of additional material that may prove helpful for the understanding of the Unified Modeling Language which is used for the basis of peripheral device modeling in this standard. They are listed in alphabetical order and not according to a ranking on usefulness.

Web Location References

Official On-line UML Documentation at:

<http://www.rational.com/uml/resources/documentation/>

Object Management Group at:

<http://www.omg.org>

Reading Material References

- 1) [Booch98] Booch, G. et al, Unified Modeling Language User Guide, Addison Wesley Longman, Inc., 1998, ISBN 0201571684
- 2) Eriksson, H. and Penker, M., UML Toolkit, John Wiley & Sons, Inc., 1997, ISBN 0471191612
- 3) Fowler, M. and Scott, K., UML Distilled: Applying the Standard Object Modeling Language, Addison Wesley Longman, Inc., 1997, ISBN 0201325632
- 4) Harmon, P. and Watson, M., Understanding UML: The Developer's Guide, Morgan Kaufmann Pubs., Inc., 1997, ISBN 1558604650
- 5) Muller, P., Instant UML, Wrox Press Ltd., 1997, ISBN 1861000871
- 6) Quatrani, T., foreword by Booch, G., Visual Modeling with Rational Rose & UML, Addison Wesley Longman, Inc., 1997, ISBN 0201310163
- 7) Rumbaugh, J. et al, The Unified Modeling Language Reference Manual, Addison Wesley Longman, Inc., 1998, ISBN 020130998X
- 8) Si Alhir, S., UML In a Nutshell, O'Reilly & Associates, Inc., 1998, ISBN 1565924487

- 9) Warmer, J. and Kleppe, A., The Object Constraint Language: Precise Modeling with UML, Addison Wesley Longman, Inc., 1998, ISBN 0201379406

A P P E N D I X C

Additional Hardware References

This appendix contains a list of additional material that may prove helpful for the understanding of the UnifiedPOS hardware environment.

USB PlusPower Connector

Web Location References

Official On-line Documentation for the USB PlusPower connector is available at:

<http://www.eia.org>

Reading Material References

1) EIA-700BAAD, Detail Specification for Shielded Rectangular Connector(s) For Universal Serial Bus PlusPower Connector(s) Type "A", EIA Engineering Publications Office, 2500 Wilson Boulevard, Arlington, Virginia, 22201.

